Sur certaines méthodes raisonnées pour l'apprentissage par renforcement profond

Léonard Blier

Some Principled Methods for Deep Reinforcement Learning

Sur certaines méthodes raisonnées pour l'apprentissage par renforcement profond

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580 :
Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et sciences du numérique Référent : Faculté des Sciences d'Orsay

Thèse préparée dans l'équipe TAU, LISN (Inria, CNRS, Université Paris-Saclay)
et au sein de Facebook Artificial Intelligence Research (Paris)
sous la direction de
Marc SCHOENAUER, Directeur de Recherche INRIA Saclay
Yann Ollivier, Research Scientist, Facebook AI Research

Thèse soutenue à Paris-Saclay, le 28 Avril 2022, par

Léonard Blier

Composition du jury

Gael Varoquaux Directeur de recherche, soda, Inria	Président
Doina Precup Associate Professor, McGill University Research Team Lead DeepMind Montreal	Rapporteure & Examinatrice
Bruno Scherrer Chargé de recherche Inria BIGS (Inria) IECL (Université de Lorraine)	Rapporteur & Examinateur
Tristan Cazenave Professeur d'Université, LAMSADE, Université Paris-Dauphine, PSL	Examinateur
Elisabeth Gassiat Professeure d'Université, Laboratoire de Mathématiques, Université Paris-Saclay	Examinatrice
Emilie Kaufmann Chargée de recherche, CRIStAL (CNRS), Scool (Inria)	Examinatrice
Marc Schoenauer Directeur de Recherche, TAU, Inria	Directeur de thèse
Yann Ollivier Research Scientist, Facebook AI Research	Co-directeur de thèse

THESE DE DOCTORAT

NNT : 2022UPASG040

Acknowledgments

I allow myself, for these few words of acknowledgments only, to use my own language.

Je souhaite d'abord remercier Bruno Scherrer et Doina Precup d'avoir accepté de rapporter ma thèse, et de l'attention qu'ils y ont consacré. Je les remercie également, de même que Gaël Varoquaux, Emilie Kaufmann, Tristan Cazenave et Elisabeth Gassiat d'avoir accepté de faire partie de mon jury. Je veux ensuite remercier Marc Schoenauer. La relation d'encadrement aura sans doute été plutôt inhabituelle, et je suis particulièrement reconnaissant de la confiance et de la liberté qu'il m'a accordé, ainsi que de son efficacité à chaque fois que j'en ai eu besoin.

Je remercie Yann pour ces presque cinq ans de travail ensemble, son encadrement quotidien, ses idées fourmillantes, sa vision si riche qu'il m'a transmise du machine learning, cette ambition scientifique, au delà des publications, que nous avons je crois partagée, et pour sa compréhension dans les périodes personnelles difficiles.

Je tiens ensuite à remercier Corentin, dont j'ai tant appris lors de nos différentes projets communs. La thèse est un travail solitaire, et notre collaboration en duo en restera la plus belle période. Par la suite, son amitié et son soutien dans les difficultés, scientifiques ou non, m'aura été essentiel.

Je remercie l'équipe TAU pour son accueil, Balthazar Donon pour ses conseils pour la soutenance, et Pierre Wolinski pour notre collaboration. Je remercie bien sûr toute l'équipe de FAIR. En particulier Nicolas Usunier, François Charton, pour leur aide régulière, Ahmed Touati, Hervé Jégou, Alessandro Lazaric pour nos nombreuses discussions éclairantes, Patricia pour son aide. Evidemment, je remercie l'exceptionelle équipe des CIFREs qui aura été si importante au quotidien pendant (hélas seulement) la moitié de la thèse, et notamment ceux qui étaient mes ainés, m'ont si bien intégré et tant appris.

Je profite de cette occasion particulière pour remercier quelques personnes qui ont grandement contribué dans ma vie à mon désir de me tourner vers les sciences, les mathématiques, puis le machine learning. Michel, par qui j'ai découvert que les mathématiques n'étaient pas que scolaires, mais bien au contraire belles et créatives, qu'elles ne restraignaient pas l'esprit à des problèmes abstraits mais l'ouvraient au monde. Je pense également à Delphine et Lucas et aux après-midis à faire des expériences, à Jean Michel qui m'a suggéré alors que

j'étais au collège d'apprendre à programmer, à Catherine Theurkauff pour sa transmission au lycée de la poesie et du plaisir des maths. Je pense à David, qui il y a bientôt 15 ans m'a montré ses expériences de machine learning: j'ai été fasciné qu'un *"simple"* algorithme mathématique arrive à reproduire des processus que je pensais réservés au cerveau humain. Quelques années plus tard, pour mon TIPE, David m'a suggéré un sujet, sur l'algorithme NMF. J'ai passé sur ce projet un temps démesuré, mais surtout, j'ai trouvé une voie que je n'allais plus quitter.

Plusieurs stages m'ont confirmé dans cette vocation, et je tiens à remercier Jonathan Taylor pour un premier stage passionnant à Stanford qui m'a convaincu de faire une thèse. Je remercie Gaël encore, pour m'avoir plusieurs fois guidé, puis pour m'avoir lui-même accueilli en stage dans l'équipe Parietal où j'ai tellement appris techniquement et scientifiquement, et je suis très heureux qu'il participe à mon jury aujourd'hui. Enfin, je remercie Charles Ollion pour une merveilleuse expérience à Heuritech, dans le monde des start-up que je retrouve aujourd'hui, où j'ai découvert le deep learning. Charles m'a envoyé écouter quelques conférences, et notamment un jour par hasard un exposé de Yann Ollivier. C'est suite à cela que je lui ai écrit un mail, et ce fut le début de cette aventure de thèse.

Je n'aurais pas traversé ces années sans mes proches qui m'ont entouré, et je tiens à dire à quel point je leur en suis reconnaissant. Je remercie mon père, Farida, et Leïla, pour leur soutien et leur compréhension malgré les tempêtes. Je remercie ma mère et Michel, pour toute la confiance qu'ils me donnent, pour être mes piliers, et mon refuge. Je remercie tous mes amis qui m'ont soutenu, ont écouté mes angoisses, ou m'ont changé les idées. Alexis qui m'aura accepté comme colocataire éphémère, Jonathan, resté si proche malgré la distance, Corentin encore, Matthieu, Théis, Nina, Matthias, Clémence, Théo, pour tant de beaux moments, de jeux, de rires, et même un confinement enchanté passé ensemble, Elie, Marion, Arthur, Charlotte, Mendes, Emile, Alexis, Alex, Hélène, Julie, Mathilde et Thomas, Katyusha et Thomas, pour ces soirées, ces verres, ces échanges précieux. J'ai souvent mesuré pendant ces années la chance que j'avais d'être si bien entouré. Du fond du coeur, merci à tous.

Enfin, je remercie Salomé, qui plus que personne a partagé l'intensité des ces montagnes russes, et grace à qui dans les hauts et les bas, la vie est restée si passionnante, surprenante, intense, douce, et heureuse.

Contents

Part I

Introduction

Chapter 1

A short introduction to Reinforcement Learning

In this chapter, we will give a short introduction to Reinforcement Learning (RL). Every chapter in this thesis except Chapters 3 and 4 are studying aspects of RL. The purpose of this chapter is to introduce the fundamental tools which will be used throughout this thesis, and some of the most standard algorithms for reinforcement learning in general continuous state spaces, using function approximators. Additionally, we will state some theoretical properties of these standard algorithms, without every mathematical details, which are similar to the theoretical guarantees we will discuss for our algorithms.

1.1 The Reinforcement Learning problem

Reinforcement Learning (RL) is the learning setting in which a learning agent interacts with its environment. The agent can *observe* the environment, and from his observations take some *actions*. After taking an *action*, the agent gets a new *observation* together with a *reward*. The *reward* is a positive or negative scalar signal. The agent's objective is to gather as much reward as possible.

Many application can be formalized in that way, and in recent years RL approaches have provided impressive results in a variety of domains, achieving superhuman performance with no expert knowledge in perfect information zero-sum games (Silver et al., 2017), reaching top player level in video games (OpenAI 2018b, Mnih et al. 2015), or learning dexterous manipulation from scratch without demonstrations (OpenAI, 2018a).

Reinforcement Learning raise additional issues compared to supervised or unsupervised learning. Here are a few simple examples, highlighting some of the fundamental questions in RL:

Exploration-exploitation dilemma Alice has to decide every day in which restaurant to have lunch, from a list of restaurant. Every day, she takes an *action* (choosing a restaurant for lunch) and gets a *reward* (how much she enjoyed the restaurant).

Because every restaurant is changing their menu every day, it is not easy to be sure which one is the *best* one (in average): even after tasting a disappointing meal in some place, it would be a wrong strategy to never go back there again, as it could have been a one-time mistake. On the contrary, always going to the same restaurant because it is *good* might prevent Alice to discover a better one.

This specific setting is usually called the *bandit* setting, and the problem of finding the best tradeoff between always going to the currently preferred restaurant and exploring new possibilities is called the *exploration-exploitation* tradeoff. In this thesis, exploration strategies will be discussed in Chapter 5 for *near-continuous time* environments.

Credit assignment Bob tries to improve his skills at the game of Go by playing with Chloe. His *actions* are all the *moves* authorized by the game. After every action, he *observes* the board after Chloe's move. He only gets a *reward* at the end of the game if he wins.

In order to improve his skills, Bob will need at the end of the game to understand which of his moves made him win or lose. This can be extremely difficult, as a game can be a few hundreds moves long, and the only reward signal is at the end of the game.

This specific problem is usually called the *credit assignment problem*. One possible way of tackling the *credit assignment problem* is via *policy evaluation*, which informally consists in measuring the *advantage* of choosing an action compared to an other (or going to a *state* compared to an other). In this thesis, all Part IV will focus on policy evaluation, as well as Chapter 16 in the context of *multi-goal RL*.

Model based Reinforcement Learning Dennis is participating in a billiard game competition. Before every shot, his *observation* is the current position of every ball. His *action* is choosing the angle, velocity and rotation for hitting the white ball. The *reward* is obtained at the end of the game,

In order to choose her *action*, Dennis has to wonder: what will happen if I play this shot? For each possible shot, he can try to *predict* the balls positions after the shot. Then, decide which shot will achieve the best final configuration.

That way of selecting the action is called *model-based RL*. First, Dennis learned a *model*: a way to predict the future state of the environment given its current state and an action. Then, she can use that model for *planning*: select the action which optimizes the final position *according to the model*.

Unsupervised Reinforcement Learning Elise is playing a game in a maze. At the beginning of every game, she starts at a random point in the maze, and has to reach a treasure as fast as possible, which is always at the same spot of the maze. The *actions* are the moves in the maze, the *observation* is the current position in the maze, and the *reward* is obtained when reaching the treasure.

During her first game, Elise doesn't know where the treasure is, and she might explore the maze randomly for a long time without receiving *learning signal* to improve her strategy. Still, she is acquiring knowledge, even without any reward: in order to go faster in the next games, he can build a map of the maze. Hence, for the second run she will be able to use that map to reach the treasure faster.

Learning from the environment even when no reward is observed is usually called *unsupervised reinforcement learning*. In Part IV of this thesis, we will study unsupervised techniques via the successor states operator.

In this chapter, we will present the standard formalization of RL with Markov Decision Process. Then, we will present some of the most important algorithms in RL.

1.2 The Reinforcement Learning framework

We now introduce the standard Reinforcement Learning framework with Markov Decision Processes (Sutton and Barto, 2018, Chapter 3), and define related mathematical tools.

Markov Decision Process An environment $\mathcal{M}$ described as a Markov Decision Process (MDP) is defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where:

- $\mathcal{S}$ is the state-space of the environment. It can be finite, discrete, or continuous. At step $t \in \mathbb{N}$, the agent will observe the current state s_t
- $\mathcal{A}$ is the action space. As for the state space, it can be finite, discrete, or continuous. At step $t \in \mathbb{N}$, the agent will execute an action a_t in the environment

- $P(\mathrm{d}s'|s,a)$ is the transition probability operator, which describes the probability distribution of the next state s' starting from s and after action a.
- $\mathcal{R}(r|s)$ reward probability density in state s. We define for every state $s \in \mathcal{S}$: $R(s) := \mathbb{E}_{r\sim\mathcal{R}(r|s)}[r]$ the expected reward in state s.
- γ is the discount factor, with $0 \leqslant \gamma < 1$.

In the finite case, $P_{sas'}$ can be viewed as a tensor of size $S \times A \times S$ where $S := |\mathcal{S}|$ and $A := |\mathcal{A}|$. In the general case, for each $s \in \mathcal{S}$, $P(\mathrm{d}s'|s,a)$ is a *probability measure* on s' that depends on s. From now on, we use the notation $P(\mathrm{d}s'|s,a)$. The measure formalism will be important in Parts IV and V. Formally, we take the setting from (Hairer, 2010). The state space $\mathcal{S}$ is assumed to be a complete, separable metric space (*Polish space*), such as a finite or countable space or $\mathbb{R}^n$. It is equipped with its Borel σ-algebra (the σ-algebra generated by all open sets). This guarantees that integration behaves as expected. $P(s,\mathrm{d}s')$ is assumed to be a *Markov kernel*, namely, a measurable map from $\mathcal{S}$ to probability measures over $\mathcal{S}$.

Formalizing the RL problem in MDPs means assuming the *Markov hypothesis*: the next state distribution $P(\mathrm{d}s'|s,a)$ depends only on the current state s and the current action a, but not of past states or past actions. This setting does not cover many standard environments: consider a car-racing environment in which the agent only observe the current frame. The next state of the car depends not only on the observed position but on the current velocity, which cannot be deduced from a single frame. Hence, the environment does not satisfy the Markov hypothesis. The *Partially Observable MDP* setting (POMDP) (Sutton and Barto, 2018; Spaan, 2012) extends the MDP setting, by assuming the agent only observes a function $o_t := f(s_t)$. In the car-racing example, the full state s_t would contain both the current frame and the current velocity, but the observation o_t would only contain the current frame. Some standard algorithms designed for MDPs can be applied in the POMDP setting via Recurrent Neural Networks keeping a memory of past steps (Hausknecht and Stone, 2015). In this thesis, we will only consider the MDP setting.

Policy The agent's behavior in the environment is defined via its policy. Formally, a policy $\pi(a|s)$ defines a probability distribution over the action space $\mathcal{A}$ for every state s. If $s_0 \in \mathcal{S}$ is a state, we say that τ is a trajectory is sampled from s_0 with policy π if $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$ with, for all $t \geqslant 0$: $a_t \sim \pi(\mathrm{d}a|s_t)$, $r_t \sim \mathcal{R}(.|s_t)$, $s_{t+1} \sim P(\mathrm{d}s|s_t, a_t)$. We will write $\tau \sim \mathbb{P}(\tau|s_0,\pi)$.

Discounted return For a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$ we define the discounted *return* $G(\tau)$ as:

$$G(\tau) := \sum_{t\geqslant 0} \gamma^t r_t \tag{1.2.1}$$

The return of a trajectory represents the amount of reward earned for that trajectory. This quantity is well-defined if the reward is bounded.

If $\rho_0(\mathrm{d}s_0)$ is a distribution for initial states, we can define the expected discounted return for policy π:

$$J(\pi) := \mathbb{E}_{s_0\sim\rho_0,\tau\sim\mathbb{P}(\tau|s_0,\pi)}\left[\sum_{t\geqslant 0}\gamma^t r_t\right] \tag{1.2.2}$$

Because the reward probability distribution only depends on the current state, we equivalently have:

$$J(\pi) := \mathbb{E}_{s_0\sim\rho_0,\tau\sim\mathbb{P}(\tau|s_0,\pi)}\left[\sum_{t\geqslant 0}\gamma^t R(s_t)\right] \tag{1.2.3}$$

where the stochastic reward r_s is replaced by its expectation $R(s) = \mathbb{E}_{r_s\sim\mathcal{R}(.|s)}[r_s]$.

The reinforcement learning problem is then to find the policy π^* maximizing $J(\pi)$. Such a policy exists (Sutton and Barto, 2018, Section 3.6).

Value function The value function for policy π in state s, $V^\pi(s)$, is defined as the expected discounted return for trajectories in $\mathcal{M}$ starting from s:

$$V^\pi(s) := \mathbb{E}_{a_{t+1}\sim\pi(.|s_t),s_{t+1}\sim P(.|s_t,a_t),r_t\sim\mathcal{R}(.|s_t)}\left[\sum_{t\geqslant 0}\gamma^t r_t|s_0=s\right] \tag{1.2.4}$$

The value function measures how much (discounted) reward the agent will gather by following the policy π starting from a state s. The problem of estimating the value function is called *policy evaluation*, and is a necessary step for multiple algorithms. Policy evaluation will be the main topic of Part IV in this thesis.

The expected discounted return with $\rho_0(\mathrm{d}s_0)$ for initial distribution can be computed from the value function as:

$$J(\pi) = \mathbb{E}_{s_0\sim\rho_0}\left[V^\pi(s_0)\right] \tag{1.2.5}$$

where the stochastic reward r_s is replaced by its expectation.

We say that a policy π_1 dominates a policy π_2 if, from every starting state, using π_1 will gather more reward than using π_2: $\forall s\in\mathcal{S}\quad,\quad V^{\pi_1}(s)\geqslant V^{\pi_2}(s)$ for every state s. In particular, if π_1 dominates π_2, then $J(\pi_1)\geqslant J(\pi_2)$. A policy π^* is the *optimal policy* if, for every policy π, π^* dominates π.

The action-value function Q^π Similarly, we can define the action-value function $Q^\pi(s,a)$ for every state-action $(s,a)\in\mathcal{S}\times\mathcal{A}$ pair, as:

$$Q^\pi(s,a) := \mathbb{E}_{a_{t+1}\sim\pi(.|s_t),s_{t+1}\sim P(.|s_{t,a_t}),r_t\sim\mathcal{R}(.|s_t)}\left[\sum_{t\geqslant 0}\gamma^t r_t|s_0=s,a_0=a\right] \tag{1.2.6}$$

$$= \mathbb{E}_{a_{t+1}\sim\pi(.|s_t),s_{t+1}\sim P(.|s_{t,a_t})}\left[\sum_{t\geqslant 0}\gamma^t R(s_t)|s_0=s,a_0=a\right] \tag{1.2.7}$$

The action-value function measures how much (discounted) reward the agent will gather by following the policy π starting from a state s and doing action a. The value function can be computed from the Q function as:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}\left[Q^\pi(s,a)\right] \tag{1.2.8}$$

Markov Reward Process If $\mathcal{M}$ is a MDP and π a fixed policy, we can define a corresponding *Markov Reward Process* (MRP). Informally, a MRP is a *Markov Process*, in which we additionally get a scalar reward at every step. We define the transition probability operator for policy π: $P^\pi(\mathrm{d}s'|s) = \int_a \pi(\mathrm{d}a|s)P(\mathrm{d}s'|s,a)$. The MRP for policy π is defined as $\mathcal{M}^\pi := \langle\mathcal{S},P^\pi,\mathcal{R},\gamma\rangle$. A *trajectory* τ in the MRP is a sequence $(s_0,r_0,s_1,r_1,...)$. The expected discounted return for policy $J(\pi)$, or the value function V^π, does not depend on actions, and are well-defined in the setting of MRPs. Therefore, when we describe algorithms for policy evaluation, we will use the MRP formalization for simplicity: it allows simplifying the notations when the policy is fixed.

1.3 Model based reinforcement learning

A first family of RL approach are *model-based* approach. Informally, the algorithms are usually two-steps: first, estimate a *model* $\hat{P}(s'|s,a)$ of the transition probability operator, and a model $\hat{R}(s)$ of the reward. Then, find a policy optimally solving the *estimated* environment $\hat{P}$. Such an approach allows answering the question: *What would happen if I perform this action in this state?*. This approach is especially relevant in some settings:

- **The tabular setting:** In finite environment, it is possible to estimate P by storing a matrix $\hat{P}$, and estimating it online by counting transitions. This setting has been studied a lot (Jaksch et al., 2010; Azar et al., 2017; Pananjady and Wainwright, 2019).

- **Optimal control:** An other similar setting is optimal control. In that setting, the model is usually supposed to be known from external knowledge, from a physics analysis of the environment (Bertsekas, 2012).

In the general MDP setting, in continuous state spaces, it is possible to learn a *model* $\hat{P}$ with function approximations: we define a parametric family of model $P_\theta(\mathrm{d}s'|s,a)$, and optimize the parameter θ such that $P_\theta \approx P$. With the model P_θ, it is possible to sample artificial trajectories (sometimes called *dream* trajectories (Ha and Schmidhuber, 2018)): for every starting state s_0 and sequence of actions $(a_0, a_1, ...)$, we can *sample* trajectories $\tilde{\tau} = (\tilde{s}_0, \tilde{s}_1, ...)$ with $\tilde{s}_0 := s_0$, and $\tilde{s}_{t+1} \sim P_\theta(.|s_t, a_t)$, and consider the estimated return $\tilde{R}((a_0, a_1, ...)) = \mathbb{E}_{\tilde{s}_1, \tilde{s}_2, ...}\left[\sum_{t \geqslant 0} \gamma^t R(\tilde{s}_t)\right]$ (here, we assume for simplicity we know the reward function $R(s)$).

If a sufficiently good model is found, it is possible to do *planning*: from a starting state s, we can select the next *real* actions performed in the environment by finding the best sequence of action in the *estimated* environment: finding $\arg\max_{a_0, a_1, ...} \tilde{R}((a_0, a_1, ...))$.

Model based methods were used successfully in several cases in Deep RL context, for example for board games with MuZero (Schrittwieser et al., 2020), control (Ebert et al., 2018; Nagabandi et al., 2018), or some video games (Ha and Schmidhuber, 2018; Racanière et al., 2017).

Some of the main technical difficulties of model-based RL are the following:

- Even with an *accurate* model, planning can be complex. Indeed, it requires maximizing $\arg\max_{a_0, a_1, ...} \tilde{R}((a_0, a_1, ...))$. If the action space $\mathcal{A}$ is finite, and if we only compute the $\arg\max$ for T steps, the number of possible sequences still grows as $O(|\mathcal{A}|^T)$. Moreover, if the environment is stochastic, it might be necessary to sample multiple trajectories for every sequence of actions, because of variance.

- Learning an *accurate* model is hard. Moreover, because of compounding errors, if P_θ is slightly wrong, the prediction at step T sampled according to P_θ^T might be completely incorrect. In such case, planning might be a wrong strategy.

In this thesis, we will only study *model-free* algorithms. In Part IV, we will study the successor states operator, which can be understood as a intermediate object between model-based and model-free: it does not allow to predict the next step, but it predicts a distribution over states representing the *future* of the trajectory. We now present standard methods for model-free RL, such as policy gradient, actor critic, and Q-learning.

1.4 Policy evaluation

Finding the optimal policy means maximizing the value function $V^\pi(s)$ (as a function of π) for all state. Hence, we would like to estimate this quantity. This step is sometimes called *policy evaluation*. Informally, many algorithm iterate between two steps:

1. *Evaluate* the current policy π, by estimating $V^\pi(s)$ or $Q^\pi(s,a)$

2. Improve the policy in order to select more often actions a with high action-value function $Q(s,a)$, or to spend more time in states with higher value $V(s)$. Then, go back to step 1.

Hence, policy evaluation is a crucial step towards policy optimization. In this section, we will present some of the most important techniques for policy evaluation.

If the state space $\mathcal{S}$ is finite, we can use a *tabular* estimate: store a vector $\hat{V}(s)$ of values for every state. If $\mathcal{S}$ is a continuous state space, storing all the values is not possible anymore. In that case, we define a parametric model $V_\theta(s)$ (typically a neural network), where $\theta \in \Theta$ is the parameter vector. The goal is then to find a parameter θ^* which minimizes the error $\|V_{\theta^*} - V^\pi\|$, for a given metric $\|.\|$.

In this text, we will only consider methods which can be used with *any* function approximation. We only assume we are able to compute values $V_\theta(s)$ for every input s, as well as the derivatives $\partial_\theta V_\theta(s)$ for every input s.

1.4.1 Monte-Carlo approach for policy evaluation

The first approach for policy evaluation is the Monte Carlo estimation (Sutton and Barto, 2018). Assume we are able to interact with the environment to sample trajectories $\tau = (s_0, r_0, s_1, r_1, ...)$, sampled by using the policy π in the environment. Then, $G(\tau) = \sum_{t \geqslant 0} \gamma^t r_t$ is an unbiased estimate of $V^\pi(s_0)$. If $V_\theta(s)$ is our current approximation of the value function, we can define the stochastic update $\widehat{\delta\theta}(\tau)$ on the parameter θ

$$\widehat{\delta\theta}(\tau) := \partial_\theta \left(V_\theta(s_0) - G(\tau)\right)^2 \tag{1.4.1}$$

Then, we can update θ with $\theta_{t+1} := \theta_t - \eta_t \widehat{\delta\theta}(\tau_t)$. This translates the policy evaluation problem into a supervised learning problem: each trajectory going through a state s gives a sample of $V^\pi(s)$.

One of the main limitation of this approach is its variance. Estimating the value function of a state with a Monte-Carlo approach requires a lot of trajectories going through that state, as the algorithm does not generalize between states, except via the inductive bias of the function approximation. A more sample efficient method can be obtained by leveraging the structure of the value function, through the Bellman equation.

1.4.2 Temporal Difference algorithm

The value function at state s can be related to the value function at the next state via the Bellman equation. Here is an informal derivation of this relation:

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_{\tau=(s_0,s_1,s_2,...)}\left[\sum_{t\geqslant 0} \gamma^t R(s_t) | s_0 = s\right] \\
&= R(s) + \gamma \mathbb{E}_{\tau=(s_0,s_1,...)\sim\mathcal{M}^\pi}\left[\sum_{t\geqslant 1} \gamma^t R(s_t) | s_0 = s\right] \\
&= R(s) + \gamma \mathbb{E}_{s'\sim P^\pi(.|s)}\left[\mathbb{E}_{\tau=(s_2,...)\sim\mathcal{M}^\pi}\left[\sum_{t\geqslant 0} \gamma^t R(s_t) | s_0 = s'\right]\right] \\
&= R(s) + \gamma \mathbb{E}_{s'\sim P^\pi(.|s)}\left[V^\pi(s')\right]
\end{aligned} \tag{1.4.2}$$

The result is formalized in the following proposition. This result is well-known (Jaakkola et al., 1994), and we don't not give formal details in this chapter. Still, we formalized this statement, as some of our theorems in this thesis can be interpreted as extensions of this proposition. This remark also applies to the next proposition. We denote by $B(\mathcal{S})$ the set of bounded measurable functions on $\mathcal{S}$:

Proposition 1.1. *Let T be the Bellman operator on $B(\mathcal{S})$ defined as:*

$$(T \cdot V)(s) = R(s) + \gamma \mathbb{E}_{s'\sim P^\pi(.|s)}\left[V(s')\right] \tag{1.4.3}$$

Then, T is γ-contractive, and its unique fixed point is the true value function V^π. In particular, for any initial value V_0, the sequence defined as $V_{t+1} := T \cdot V_t$ converges to V^π.

We know that for any V_0, the sequence defined as $V_{t+1} := T \cdot V_t$ converge to V^π. We now want to approximate this sequence with function approximations. The strategy is the following: If V_θ is a current estimate of V^π, we consider $V^{\text{tar}} = R + \gamma P V_\theta$, a target estimate for V defined via the Bellman equation, and we move θ in order to move V_θ closer to V^{tar}. The temporal difference update obtained with that strategy is defined in the following proposition (which is also well-known), which also formalizes that this estimate is *unbiased*:

Proposition 1.2. *Let V_θ be a current estimate of V^π. Consider $V^{\text{tar}} = R + \gamma P V_\theta$, a target estimate for V defined via the Bellman equation.*

Let (s, r, s') be a sample of the environment such that $s \sim \rho$ (where ρ is a probability distribution over states), $r \sim \mathcal{R}(.|s)$ and $s' \sim P(s'|s)$. We define $\widehat{\delta\theta}_{\text{TD}}(s, r, s')$ as:

$$\widehat{\delta\theta}_{\text{TD}}(s, r, s') := \partial_\theta \frac{1}{2}(V_\theta(s) - (r + \gamma V_{\bar{\theta}}(s'))^2 \tag{1.4.4}$$

$$= \partial_\theta V_\theta(s)(V_\theta(s) - r - \gamma V_\theta(s')) \tag{1.4.5}$$

where $\bar{\theta} = \theta$ but is not differentiated through ∂_θ. Then $\widehat{\delta\theta}_{\text{TD}}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}_{s\sim\rho, r\sim\mathcal{R}(.|s), s'\sim P(s,\mathrm{d}s')}\left[\widehat{\delta\theta}_{\text{TD}}(s, r, s')\right] = \frac{1}{2}\partial_\theta \|V_\theta - V^{\text{tar}}\|_\rho^2 \tag{1.4.6}$$

where the norm $\|.\|_\rho$ is defined as $\|f\|_\rho^2 := \mathbb{E}_{s\sim\rho}\left[f^2(s)\right]$. In particular, the true value function V^π is a fixed point of this update: if $V_\theta = V^\pi$, then $\mathbb{E}\left[\widehat{\delta\theta}_{\text{TD}}\right] = 0$.

With Proposition 1.2, we can now define an algorithm for policy evaluation with function approximations: When observing (s_t, r_t, s_{t+1}) at step t, we define $\theta_{t+1} := \theta_t - \eta\widehat{\delta\theta}(s, r, s')$. This strategy to derive algorithms is more general, and will be used many times in this thesis in Parts IV and V. We now describe more generally this strategy.

A general strategy to derive unbiased algorithm We are interested about *unbiased* algorithms, as such methods have some interesting theoretical properties. Here, we briefly present in the case of value functions, what we mean with *unbiased* methods in this text:

- First, we define an *operator* T on $B(\mathcal{S})$ the space of functions over the state space $\mathcal{S}$, such that its iterates are guarantee to converge to the true value function V^π: for any initialization $V_0 \in B(\mathcal{S})$, if we define the sequence $V_{t+1} := T \cdot V_t$, we have $V_t \to_{t\to\infty} V^\pi$. Moreover, we want V^π to be the only fixed point of the operator T in $B(\mathcal{S})$.

- Then, we approximate the sequence V_t with function approximation. Let θ_t be our current parameter at step t. We define $\theta_{t+1} := \theta_t - \eta\widehat{\delta\theta}$, where $\widehat{\delta\theta}$ is a stochastic update computed with a currently observed trajectory/observation/transition. We require the algorithm to be unbiased, which means that the expected update $\mathbb{E}[\widehat{\delta\theta}]$ (where the expectation is with respect to the observed trajectory/observation/transition) is a gradient step towards the target $V^{\text{tar}} := T \cdot V_{\theta_t}$:
$$\mathbb{E}[\widehat{\delta\theta}] = \partial_\theta \|V_\theta - V^{\text{tar}}\|^2 \tag{1.4.7}$$
where $\|.\|$ is a given norm.

The Monte-Carlo approach described in Section 1.4.1 is a trivial case of this approach, with T the constant operator $T \cdot V := V^\pi$ for every V. The TD approach defined in Section 1.4.2 also corresponds to this strategy with T the Bellman operator.

There is no general guarantee to converge to V^π. Our theoretical guarantee is that with such an algorithm, V^π is a fixed point. Indeed, if there is θ^* such that $V_{\theta^*} = V^\pi$, then at $\theta = \theta^*$ we have $V^{\text{tar}} = T \cdot V^\pi = V^\pi$, hence θ^* is already minimizing the distance $\|V_\theta - V^{\text{tar}}\|$ and the expected update at $\theta = \theta^*$ is $\mathbb{E}[\widehat{\delta\theta}] = 0$.

Moreover, if the parametric family V_θ is overparametrized[1], then V^π is the *unique* fixed point of the algorithm. In practice, the overparametrized hypothesis is not satisfied, except in the tabular setting, but as we use highly expressive deep learning models, the overparametrized setting is a relevant viewpoint.

[1]Here, we say that the parametric family V_θ is overparametrized if the function $\theta \mapsto V_\theta$ is surjective from Θ to $B(\mathcal{S})$, and if for every θ, the map $\partial_\theta V_\theta$ is surjective for any θ

TD(n) and TD(λ) The TD update defined in Proposition 1.2 is called TD with 1-step: it estimates the value at step t using the value estimate at step $t+1$. This approach can be generalized to TD with n-step: when observing a sub-trajectory $(s_t, r_t, s_{t+1}, r_{t+1}, ..., s_{t+n})$, we can define a target estimate with n-step as:

$$V_{\text{tar}} = r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n V_\theta(s_{t+n}), \tag{1.4.8}$$

then use this value to obtain a value update:

$$\widehat{\delta\theta}_{\text{TD}(n)}(s_t, r_t, s_{t+1}, r_{t+1}, ..., s_{t+n}) := \partial_\theta \frac{1}{2}(V_\theta(s) - V_{\text{tar}})^2 \tag{1.4.9}$$

$$= \partial_\theta V_\theta(s) \left(V_\theta(s) - \left(r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n V_\theta(s_{t+n})\right)\right) \tag{1.4.10}$$

The update $\widehat{\delta\theta}_{\text{TD}(n)}$ is an unbiased estimate of the n-step Bellman. If T is the Bellman operator defined in Proposition 1.1, then we define, $V^{\text{tar}} := T^n \cdot V_\theta = T \cdot ... \cdot T \cdot V = \sum_{t=0}^{n-1} \gamma^t P^t R + \gamma^n P^n V_\theta$,

$$\mathbb{E}\left[\widehat{\delta\theta}_{\text{TD}}(s_t, r_t, s_{t+1}, r_{t+1}, ..., s_{t+n})\right] = \frac{1}{2}\partial_\theta \|V_\theta - V^{\text{tar}}\|_\rho^2 \tag{1.4.11}$$

When increasing n, the variance of the update $\widehat{\delta\theta}_{\text{TD}(n)}$ increases, but the expected target $V^{\text{tar}} := T^n \cdot V_\theta$ gets closer to the true value function V^π. Taking the limit $n = \infty$, the target is exactly the value function, and $\widehat{\delta\theta}_{\text{TD}(n)}$ corresponds to the Monte-Carlo update defined in Section 1.4.1.

We now introduce briefly TD(λ), and its forward and backward views. Assume we have access to an infinite trajectory $\tau = (s_0, r_0, s_1, r_1, ...)$. Then, we can update the value of s_0 with all TD(n) updates for all $n \geqslant 1$, using a geometric averaging of these updates:

$$\widehat{\delta\theta}_{\text{TD}(\lambda)}(\tau) = (1-\lambda) \sum_{n \geqslant 1} \lambda^{n-1} \widehat{\delta\theta}_{\text{TD}(n)}(s_0, r_0, s_1, r_1, ..., s_n) \tag{1.4.12}$$

This update is the forward view of TD(λ) (Sutton and Barto, 2018, Section 12.1), and can be understood as a mixture of updates with low variance (n low) and updates with a higher variance but a more accurate target (large n). In practice, we don't observe infinite trajectories, and truncate the sum in equation (1.4.12).

There is an *backward* view of TD(λ), via eligibility traces (Sutton and Barto, 2018, Section 12.2). Consider for simplicity a finite environment. We observing a trajectory, $\tau = (s_0, r_0, s_1, r_1, ...)$, we update online an *eligibility traces* vector $e_t(s)$ with: $e_0(s) = 0$ for all states, and at step t:

$$e_t \leftarrow \gamma \lambda e_{t-1} \tag{1.4.13}$$

$$e_t(s_t) \leftarrow e_t(s_t) + 1 \tag{1.4.14}$$

Hence, e_t is the empirical discounted measure of observed state in the current trajectory. Then, the TD(λ) algorithm via eligibility traces updates the value function as:

$$\hat{V}_{t+1} := V_t + \eta_t e_t (V_\theta(s_t) - r_t - \gamma V_\theta(s_{t+1})) \tag{1.4.15}$$

This update propagates the Bellman error to all states observed in the past. In can be used in a true online setting, but is equivalent in expectation to the forward view defined in Equation (1.4.12) (van Seijen et al., 2016). The TD(λ) algorithm with eligibility traces will be discussed in Chapter 12.

1.5 Policy gradient and Actor-Critic algorithms

Knowing how to evaluate a policy, we will now describe policy improvement algorithms. We first describe policy gradient methods. With these methods, we consider a parametric policy $\pi_{\theta^\pi}(a|s)$, where $\theta^\pi \in \Theta^\pi$ is a vector parameter. The goal is the to find θ^π maximizing $J(\pi_{\theta^\pi})$.

The main difficulty is that we cannot compute easily the gradient $\partial_{\theta^\pi} J(\pi_{\theta^\pi})$. Computing that gradient means estimating: *If I change my policy slightly, how would my trajectory evolve?* However, we are not able to answer that question. One of the reason is that we don't have access to the transition operator P, and can't know precisely what would happen if we behave differently.

Evolution strategies We know how to estimate $J(\pi_{\theta^\pi})$ for every θ^π, by sampling trajectories with policy π_{θ^π}. Therefore, it is possible to use Evolution Strategies (ES), a class of *black box optimization algorithms* (Beyer and Schwefel, 2004; Sun et al., 2009). Informally, in many Evaluation Strategies (such as CMAES (Hansen, 2016)) the principle is to approximate a gradient descent-like method, by estimating multiple parameters ${\theta^\pi}_1, ..., {\theta^\pi}_k$ in the same neighbourhood, then deduce from this estimate a direction for improvement. These methods have been used in Reinforcement Learning (Salimans et al., 2017; Ha and Schmidhuber, 2018). Still, an issue of these methods is that the number of trajectories required to learn roughly scales with the number of parameters. Hence, for complex environments requiring large architecture, the number of parameter will become a limitation.

REINFORCE and the policy gradient theorem The policy gradient algorithms are a way to leverage the structure of the RL problem in order to estimate the gradient $\partial_{\theta^\pi} J(\pi_{\theta^\pi})$, only using estimates of $J(\pi_{\theta^\pi})$ and the gradient of the probability (or the log-probability) of choosing action a in state s: $\partial_{\theta^\pi} \pi_{\theta^\pi}(a|s)$. We can use the following relation (Williams, 1992):

$$\partial_{\theta^\pi} J(\pi_{\theta^\pi}) = \mathbb{E}_{s\sim\nu^\pi, a\sim\pi_{\theta^\pi}(.|s), \tau\sim\mathbb{P}(\tau|s,a)} \left[\partial_{\theta^\pi} \log \pi_{\theta^\pi}(a|s)\, G(\tau)\right] \tag{1.5.1}$$

where $\tau = (s_0, a_0, r_0, s_1, ...) \sim \mathbb{P}(\tau|s_0)$ if $a_t \sim \pi(.|s_t)$, $s_{t+1} \sim P(.|s_t, a_t)$, and where $\nu^\pi(\mathrm{d}s)$ is the discounted occupancy measure starting from $\rho_0(\mathrm{d}s)$. For finite state space[2], ν^π is defined as:

$$\nu^\pi(s) = (1-\gamma)\mathbb{E}_{s_0\sim\rho_0} \sum_{t\geqslant 0} \gamma^t \mathbb{P}(S_t = s|s_0 = s) \tag{1.5.2}$$

Therefore, it is possible to define an unbiased estimate of the policy-gradient. The update defined by estimating equation (1.5.1) defines the REINFORCE algorithm (Sutton et al., 2000). Similarly to our discussion on policy evaluation, $G(\tau)$ has high variance, hence the REINFORCE algorithm has high variance. Another viewpoint on this issue is with *credit-assignment*: when observing a trajectory τ with high (resp. low) return, REINFORCE propagates the information uniformly to every actions, increasing (resp. decreasing) the probability of choosing this action. Understanding which action caused the high/low return would allow to improve the policy faster. We now present the actor-critic algorithm, which uses estimates of the value-function to tackle the *credit assignment* problem, and reduce the policy gradient estimate variance, to the cost of bias introduced by the error between the true value function and the current estimate.

Actor-Critic We now consider *actor-critic* methods. We both learn an *actor* network π_{θ^π}, and a *critic*. Here, we will consider learning only the value function $V_\theta(s)$. The first technique to reduce variance is to use a *baseline*. If $b(s)$ (called a *baseline*) is a function of states which does not depend on action, we can show that: $\mathbb{E}_{s\sim\nu^\pi, a\sim\pi_{\theta^\pi}(.|s), \tau\sim\mathbb{P}(\tau|s,a)} \left[\partial_{\theta^\pi} \log \pi_{\theta^\pi}(a|s)\, b(s)\right] = 0$. We use $b(s) = V^\pi(s)$, and, we can estimate the policy gradient as:

$$\partial_{\theta^\pi} J(\pi_{\theta^\pi}) = \mathbb{E}_{s\sim\nu^\pi, a\sim\pi_{\theta^\pi}(.|s), \tau\sim\mathbb{P}(\tau|s,a)} \left[\partial_{\theta^\pi} \log \pi_{\theta^\pi}(a|s)\, (G(\tau) - V^\pi(s))\right] \tag{1.5.3}$$

Moreover we have:

$$\mathbb{E}_{a\sim\pi_{\theta^\pi}(.|s), s'\sim P(.|s,a), \tau\sim\mathbb{P}(\tau|s,a)} \left[G(\tau)\right] = \mathbb{E}_{a\sim\pi_{\theta^\pi}(.|s), s'\sim P(.|s,a)} \left[R(s) + \gamma V^\pi(s')\right] \tag{1.5.4}$$

By plugging (1.5.4) into (1.5.3), we obtain the following proposition, which is well-known (Sutton et al., 2000). Similarly to the previous statements, we do not give all mathematical details

[2] A formal definition of ν^π for general (continuous) state spaces can also be given, and is related to the successor state operator studied in Part IV.

and a proof, but state this proposition, in order to show the similarities with other formal statements in this thesis.

Proposition 1.3. *Let (s, a, r, s') be a sample of the environment such that $s \sim \nu^\pi$, $a \sim \pi(.|s), r \sim \mathcal{R}(.|s)$, and $s' \sim P(.|s, a)$. We define $\widehat{\delta\theta}_{\mathrm{AC}}(s, a, r, s')$ as:*

$$\widehat{\delta\theta}_{\mathrm{AC}}(s, a, r, s') := \log \pi_{\theta^\pi}(a|s)\left(r_s + \gamma V^\pi(s') - V^\pi(s)\right) \tag{1.5.5}$$

Then, $\widehat{\delta\theta}_{\mathrm{AC}}$ is an unbiased policy gradient update:

$$\mathbb{E}_{s\sim\nu^\pi, a\sim\pi_{\theta^\pi}(.|s), s'\sim P(.|s,a)}\left[\widehat{\delta\theta}_{\mathrm{AC}}(s, a, r, s')\right] = \partial_{\theta^\pi} J(\pi_{\theta^\pi}) \tag{1.5.6}$$

The actor-critic update $\widehat{\delta\theta}_{\mathrm{AC}}(s, a, r, s')$ is *on-policy*: it requires that the action a is sampled according to the current policy π to be unbiased. Hence, once a transition is used for an update and the policy changed, the transition cannot be used again. More recent policy gradient approaches (Schulman et al., 2015, 2017) partly relax this constraint, if the current policy is still reasonably close to the policy used to obtain the transition.

1.6 Q-learning methods

We now present *Q-learning* methods. Its principle is slightly different from policy gradient approaches. Instead of learning an estimate V^π of the current policy π, it directly estimate the *optimal* Q-function $Q^* := Q^{\pi^*}$, where π^* is the optimal policy. Once we know Q^*, we can easily obtain the optimal policy as $\pi^*(.|s) = \arg\max_a Q^*(s, a)$. One of the advantages of Q-learning methods is that they are off-policy: they can learn from trajectories sampled from a policy different from the current policy. Q-learning methods use the *optimal Bellman equation*, formalized in the following statement (Jaakkola et al., 1994):

Proposition 1.4. *Let T be the operator on $B(\mathcal{S} \times \mathcal{A})$ defined as:*

$$(T \cdot Q)(s, a) = R(s) + \gamma \mathbb{E}_{s'\sim P(.|s,a)} \max_{a'} Q(s', a') \tag{1.6.1}$$

Then, T is γ-contractive and its unique fixed point in $B(\mathcal{S} \times \mathcal{A})$ is the optimal Q function Q^*. In particular, for any initial value $Q_0 \in B(\mathcal{S} \times \mathcal{A})$, the sequence defined as $Q_{t+1} := T \cdot Q_t$ converges to Q^*.

As for policy evaluation, we can now *approximate* the sequence Q_t with function approximations. The following theorem is a well-known fact, and is formalized in order to show the similarity with our methods. If the action space is finite, it corresponds to the Deep-Q Network algorithm (Mnih et al., 2013), and can be adapted with continuous action with DDPG (Lillicrap et al., 2016).

Proposition 1.5. *Let $Q_\theta(s, a)$ be a current estimate of Q^*, we define $Q^{\mathrm{tar}} := (T \cdot Q_\theta) = R + \gamma \mathbb{E}_{s'\sim} \max_{a'} Q(s', a')$.*

Let (s, a, r, s') be a sample of the environment such that $s \sim \rho_{\mathrm{expl}}, r \sim \mathcal{R}(.|s), a \sim \pi_{\mathrm{expl}}, s' \sim P(.|s, a)$, where π_{expl} is can be any policy, and is called the exploration *policy. We define $\widehat{\delta\theta}_{\mathrm{DQN}}(s, a, r, s')$ as:*

$$\widehat{\delta\theta}_{\mathrm{DQN}}(s, a, r, s') := \frac{1}{2}\partial_\theta \left(Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\bar{\theta}}(s', a')\right)^2 \tag{1.6.2}$$

$$= (\partial_\theta Q_\theta(s, a)) \times \left(Q_\theta(s, a) - r - \gamma \max_{a'} Q_\theta(s', a')\right) \tag{1.6.3}$$

Then $\widehat{\delta\theta}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}\left[\widehat{\delta\theta}_{\mathrm{DQN}}(s, a, r, s')\right] = \frac{1}{2}\partial_\theta \|Q_\theta - Q^{\mathrm{tar}}\|_{\mathrm{expl}} \tag{1.6.4}$$

where the norm $\|.\|_{\text{expl}}$ *is defined as* $\|f(s,a)\|_{\text{expl}} = \mathbb{E}_{s\sim\rho_{\text{expl}},a\sim\pi_{\text{expl}}(.|s)}\left[f(s,a)^2\right]$.

In particular, the true optimal Q-function Q^* *is a fixed point of this update: if* $Q_\theta = Q^*$*, then* $\mathbb{E}\left[\widehat{\delta\theta}_{\text{DQN}}(s,a,r,s')\right] = 0$.

In this chapter, we introduced the main algorithms for Reinforcement Learning, as well as some of their properties which will be linked to our work in this thesis. In the next chapter, we will give an overview of the content of this thesis, highlighting our main contributions.

Chapter 2

An overview of this thesis

In this chapter, we give an overview of this thesis. We give a summary of every chapter, highlighting our main contributions. In Section 2.1, corresponding to Chapter 3, we present an information theory viewpoint on the complexity of deep learning models. Then, in Section 2.2, corresponding to Part III, we present two published papers, both developing mathematical tools for robustness in deep RL. The first one (Chapter 4) describes *Alrao* (All learning rates at once) an optimization method, not specific to RL, but designed to work in many setting, including non-stationary RL problems, without hyperparameter tuning. The second one (Chapter 5) studies near continuous-time environments, and show how to design robust algorithms in that setting. Section 2.3 corresponds to Part IV (Chapters 6-12) and present our work on the successor state operator, for policy evaluation. Finally, Section 2.4 corresponds to Part V (Chapter 13-17) and present how we applied tools developed for the successor state operator to the setting of multi-goal reinforcement learning.

2.1 The description length of deep learning models

In Chapter 3, we present the following published paper:

> Blier, L. and Ollivier, Y. (2018). The description length of deep learning models. In *Advances in Neural Information Processing Systems*

This was the first of this thesis. Its story is worth mentioning. As I was starting to work with Yann Ollivier, he asked me to read about the information theory viewpoint (Solomonoff inference, Minimum Description Length) for machine learning.

In information theory and Minimum Description Length (MDL), learning a good model of the data is recast as using the model to losslessly transmit the data in as few bits as possible. Consider an image classification setting with a dataset (for example CIFAR10) $\mathcal{D} = \{(x_1, y_1), ..., (x_N, y_N)\}$ where the x_i are the images and the y_i are the labels. The classification problem is recast as follows: Alice has the entire dataset $\mathcal{D}$, but Bob only has the images $\{x_1, ..., x_N\}$, and Alice wants to transmit the labels. The first possibility is to directly send a file containing the labels $\{y_1, ..., y_N\}$ without leveraging the fact that Bob has the images. But it is possible to do better, by sending a *model* predicting the labels from the images, together with the list of errors of the model. A more *complex* model might make less errors, hence compress the data more, but the model must be transmitted as well. The overall codelength can be understood as a combination of quality-of-fit of the model (compressed data length), together with the cost of encoding (transmitting) the model itself. The MDL viewpoint is that the *best model* is the model achieving the best trade-off between *complexity* and accuracy of the model, measured with compression bounds.

MDL is based on Occam's razor, and on Chaitin's hypothesis that "*comprehension is compression*" (Chaitin, 2007): any regularity in the data can be exploited both to compress it and to make predictions. This is ultimately rooted in Solomonoff's general theory of inference

(Solomonoff, 1964), whose principle is to favor models that correspond to the "shortest program" to produce the training data, based on its Kolmogorov complexity (Li and Vitányi, 2008). If no structure is present in the data, no compression to a shorter program is possible.

After a few weeks reading about these topics, I came back to Yann. I told him that while this viewpoint is elegant, it is *obviously wrong*: indeed, the hypothesis is that the best model for prediction is also the best model for compression. But the success of deep learning *proves* that this is untrue. I took the classification problem introduced above, with the example of CIFAR10. In that case, the *baseline* for compression is encoding directly the labels of the dataset, without any model using the images, which costs $N \times \log_2(\text{Number of classes}) = 50,000 \times \log_2 10 =$ 166 kbits. This means that the MDL viewpoint would only select a model if it is able to encode the labels with *less* than 166 kbits. But the best model for prediction on that dataset are deep learning models with millions of parameters, and encoding the dataset with such a model requires taking into account the cost of encoding the model weights in a way. The conclusion was quite clear to me: it is impossible to encode a dataset of 166 kbits using a model with millions of parameters, hence: *The success of deep learning proves that the MDL viewpoint is wrong, and that the best model for prediction are not the best models for compression.* Yann did not agree with my conclusion, and told me that it was actually possible to compress such a dataset with a large model, even taking into account the cost of encoding the weights, using variational techniques introduced by (Hinton and Van Camp, 1993). I went back home, and tried to think of other ways to encode a dataset with a deep learning model.

It turned out we were *both* wrong (but to be honest, especially me): it is actually possible to compress the labels of a dataset like CIFAR10 in less than 166 kbits (we reached 35 kbits) with a state-of-the-art network (in our case with a VGG19 which has more than 10M parameters), but not with variational methods.

We now describe our contributions. First, we study variational methods for deep learning.

- We show that the traditional method to estimate MDL codelengths in deep learning, variational inference (Hinton and Van Camp, 1993), yields surprisingly inefficient codelengths for deep models, despite explicitly minimizing this criterion. This might explain why variational inference as a regularization method often does not reach optimal test performance.

This contribution applies for standard variational techniques used while we were working on this project (Blei et al., 2017), but could be different for more recent techniques such as normalizing flows (Rezende and Mohamed, 2015; Kobyzev et al., 2020).

Then, we introduce new practical ways to compute tight compression bounds in deep learning models, based on the MDL toolbox (Grünwald, 2007):

- We show that *prequential coding* on top of standard learning, yields much better codelengths than variational inference, correlating better with test set performance. Thus, despite their many parameters, deep learning models do compress the data well, even when accounting for the cost of describing the model.

The principle of prequential code is the following: First, Alice sends to Bob the description of a network architecture and an optimization algorithm (which is only a few lines of code), and a file containing the first 100 labels of the dataset. Then, Alice and Bob both train the network with these 100 labels. Alice can now use this trained network as a *model* (which will not be very accurate) to encode 100 next labels. Now they both have 200 labels and can train a better model, which will be used for the next 100 labels. As more labels are transmitted, the model becomes more accurate and sending more labels is less expensive. This compression scheme leverages the generalization ability of deep learning networks, even with very limited datasets. These introduced techniques lead to the main contribution of this work:

- Deep learning models, even with a large number of parameters, compress the data well: from an information theory point of view, *the number of parameters is not an obstacle to compression.* This is consistent with Chaitin's hypothesis that "*comprehension is compression*".

Finally, prequential codes also lead to a model selection strategy:

- With prequential codes, we obtain a model selection strategy, suitable for a deep learning framework, not using cross validation, hence useful for small datasets settings. This technique was especially used for probing in NLP (Voita and Titov, 2020).

2.2 Mathematical approaches towards robustness in Deep Reinforcement Learning

In this Section, we introduce two published papers developing mathematical tools for robustness in deep reinforcement learning. First, *All learning rates at once* (Alrao) is an optimization method, not specific to RL, but designed to work in many setting, including non-stationary RL problems, without hyperparameter tuning. The second one studies near continuous-time environments, such as continuous control environments, many video games, ... It first shows that standard approaches such as DQN, DQQPG fail to learn in that setting when the time discretization decreases (or equivalently the number of action/observation per second increases), and then show how to design robust algorithms in that setting.

2.2.1 Learning with all learning rates at once

In Chapter 4, we present the published paper, which is joint work with Pierre Wolinski and Yann Ollivier:

> Blier, L., Wolinski, P., and Ollivier, Y. (2019). Learning with Random Learning Rates. In *ECML PKDD 2019 - European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*

This project started from discussions on how to design a learning system able both to adapt very quickly to new observed patterns, but also to learn complex patterns which might require a slow learning. I was starting to work in Reinforcement Learning at that time. In RL, the learning setting is *non-stationary*: the distribution of observations can change while the agent is improving its policy, and the learning methods has to be able to adapt to all of these regimes. My co-author Pierre Wolinski was interested in topics in AutoML (Guyon et al., 2016). From this viewpoint, a learning system has to be able to adapt to multiple learning settings without any hyperparameter tuning, such as settings which would require small or large learning rates. Many methods were designed to directly set optimal per-parameter learning rates (Tieleman and Hinton, 2012; Kingma and Ba, 2015), such as the popular Adam optimizer, but they still require some hyperparameter tuning.

This discussion lead to the idea of a learning system which would be a mixture of *slow learning units*, and *fast learning units*.

- We implemented this idea directly in deep learning models, by using different learning rates for different *neurons*, sampled across multiple order of magnitudes, hence leveraging redundancy in the network. We call this method *All Learning Rates At Once* algorithm (Alrao).

Alrao departs from the usual philosophy of trying to find the "right" learning rates; instead we take advantage of the overparameterization of network-based models to produce a diversity of behaviors from which good network outputs can be built.

We then experiment Alrao in multiple settings. Experimentally, we were interested to see if Alrao was working *out-of-the-box*, without any hyperparameter tuning, and how it would compare to SGD with an optimally selected learning rate.

- Surprisingly, Alrao does manage to learn well over a range of problems from image classification, text prediction, and reinforcement learning. In our tests, Alrao's performance is always close to that of SGD with the optimal learning rate, without any tuning. Alrao

combines performance with *robustness*: not a single run failed to learn with the default learning rate range we used. In contrast, our parameter-free baseline, Adam with default hyperparameters, is not reliable across the board.

2.2.2 Making Deep Q-learning methods robust to time discretization

In Chapter 5, we present the published paper:

> Tallec, C., Blier, L., and Ollivier, Y. (2019). Making Deep Q-learning methods robust to time discretization. In *ICML 2019 - Thirty-sixth International Conference on Machine Learning*

In this paper we study the sensitivity of Deep Reinforcement Learning (DRL) techniques to time discretization, such as what happens when an agent receives 50 observations and is expected to take 50 actions per second instead of 10. In principle, decreasing time discretization, or equivalently shortening reaction time, should only improve agent performance. Robustness to time discretization is especially relevant in *near-continuous* environments, which includes most continuous control environments, robotics, and many video games.

This work is a contribution to the problem of robustness of DRL techniques. Despite the impressive results of DRL techniques in a variety of domains (Silver et al., 2017; OpenAI, 2018b; Mnih et al., 2015; OpenAI, 2018a), these approaches are sensitive to a number of factors, including hyperparameterization, implementation details or small changes in the environment parameters (Henderson et al., 2017; Zhang et al., 2018). This sensitivity, along with sample inefficiency, largely prevents DRL from being applied in real world settings. Notably, high sensitivity to environment parameters prevents transfer from imperfect simulators to real world scenarios. In this work, the goal is to mitigate one of these sensitivity factors: the time discretization.

Our first contribution is to show that standard approaches based on estimation of state-action value functions, such as *Deep Q-learning* (DQN, Mnih et al. 2015) and *Deep deterministic policy gradient* (DDPG, Lillicrap et al. 2015) are not at all robust to changes in time discretization:

- We show experimentally that when the time discretization decreases in standard environment, DQN and DDPG are unable to learn at all.

Intuitively, as the discretization timestep decreases, the effect of individual actions on the total return decreases too: $Q^*(s, a)$ is the value of playing action a then playing optimally, and if a is only maintained for a very short time its advantage over other actions will be accordingly small. If the discretization timestep becomes infinitesimal, the effect of every individual action vanishes. Hence, the Q-function $Q(s, a))$ collapses to the value function $V(s)$ and does not bear any information on the ranking of actions. We say that there is no continuous-time Q-function.

- Building on (Baird, 1994), we formalize these statement, in the framework of continuous-time RL, and show that there is no continuous-time Q-function, hence the poor performance of Q-learning with small time steps (Theorem 5.2). More precisely, Thus that standard Q-learning is ill-behaved in this setting.

We then looked for an algorithm that would be as invariant as possible to changing the discretization timestep. Such an algorithm should remain viable when this timestep is small, and in particular admit a continuous-time limit when the discretization timestep goes to 0. This leads to the algorithm *Deep Advantage Updating* (Algorithm 4). Our first contributions are to show that while there is no continuous Q-function, there is a continuous *advantage* function, which is possible to learn.:

- First, we formally show that while the Q-function $Q_{\delta t}$ collapses when the time discretization $\delta t \to 0$, the rescaled advantage:

$$A_{\delta t}(s, a) = \frac{Q_{\delta t}(s, a) - V_{\delta t}(s)}{\delta t} \tag{2.2.1}$$

  converge to a *continuous time limit advantage function* $A(s, a)$ (Theorem 5.3).

- We then learn together models of the value function $V(s)$ and the continuous time limit advantage function $A(s, a)$. We formally show that there are Bellman equations on V and A, and we use it for policy optimization, with the *Deep Advantage Updating* (DAU) algorithm.

In order to define a time-discretization invariant algorithm, it is also necessary to define an invariant exploration strategy:

- We formally show that an ε-greedy exploration strategy collapse to *no exploration at all* when the time-discretization goes to 0 (Theorem 5.4).
- We derive a time-discretization invariant exploration scheme, both for discrete and continuous actions.

The principle is the following, inspired by Lillicrap et al. (2016) in the case of continuous actions, and extended to deterministic actions. For continuous actions, if $a_t = \pi(s_t)$ is the action selected by the current deterministic policy π at step t, we actually perform action $\tilde{a}_t := a_t + z_t$, where z_t is a time-discretization invariant random process defined via an Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930).

Finally, we also show how we can define a time-discretization invariant optimization procedure:

- We show that with a SGD algorithm, the learning rate needs to be of order $O(\delta t)$. If it is *larger*, the algorithm will diverge, if it is *smaller*, the parameters stay at their initial values (Theorem 5.5).

We then provide experiments comparing DQN or DPPG to DAU:

- We empirically show that standard Q-learning methods are not robust to changes in time discretization in continuous control environments (Brockman et al., 2016), exhibiting degraded performance, while our algorithm demonstrates substantial robustness.

2.3 Policy evaluation via the successor states operator

In Part IV, we present our work on policy evaluation via the successor states operator. This part is mainly based on the following preprint, with improved and additional results:

> Blier, L., Tallec, C., and Ollivier, Y. (2021). Learning successor states and goal-dependent values: A mathematical viewpoint. *arXiv preprint arXiv:2101.07123*

In an environment with a very sparse reward, learning the value function as described in Section 1.4.2 is a hard problem. At the beginning of training, no learning will occur until a reward is observed. This highlight the fact that not all the observed information is used. Leveraging this information might lead to better sample efficiency.

The *successor state operator* of a Markov reward process is an object that expresses the value functions of all possible reward functions for a given, fixed policy. Here we offer a formal treatment of these objects in both finite and continuous spaces. We present several learning algorithms and associated results. There are multiple motivations for this approach:

First, learning the successor state operator is done without reward signals and extracts information from every observed transition, illustrating an *unsupervised reinforcement learning* approach. Successor state lie in between model-free and model-based reinforcement learning approaches, providing a representation of the future of a state without having to synthesize future states or unrolling synthetic trajectories.

Then, successor states exploit multiple relationships between how to reach different states. Similarly to the value function, it satisfies a Bellman equation. Additionally, it also satisfies two other fixed point equations: a *backward* Bellman equation and a *Bellman–Newton* equation, expressing path compositionality in the Markov process. These new relation allow us to

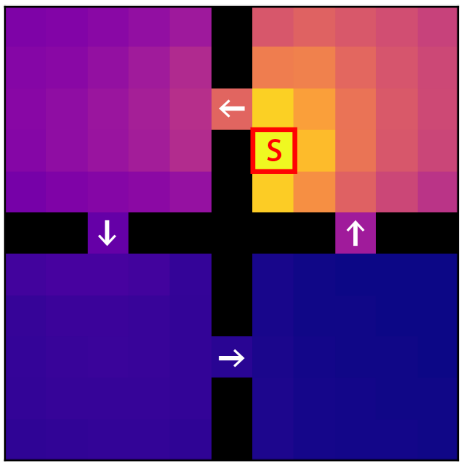


Figure 2.1: A learned successor states with function approximation in a maze. We represent the successor measure $M(s, s')$, where the state s marked in red is the starting state. The arrows represent doors which can only be passed in one direction.

generalize from observed trajectories in several ways, potentially leading to more sample efficiency.

Finally, the study of the successor states operator and its algorithms allow us to derive, in Section 2.4, unbiased methods in the setting of multi-goal RL, dealing with the issue of extremely sparse rewards.

2.3.1 The successor states operator

The *successor state operator* $M^\pi(s, \mathrm{d}s')$ of a Markov reward process is an object that linearly *transforms* a reward function into the corresponding value function. In particular, it expresses the value functions of all possible reward functions for a given, fixed policy. For finite spaces, the entries $M_{ss'}$ of the successor state matrix describe the expected discounted time spent in state s' by a trajectory starting at s Dayan (1993) (see Figure 2.1):

$$M^\pi_{ss'} = \sum_{t \geqslant 0} \gamma^t \mathbb{P}(S_t = s' | S_0 = s) \tag{2.3.1}$$

Equivalently:

$$M^\pi_{ss'} = \mathbb{E}_{a_t \sim \pi(a|s_t), s_{t+1} \sim P(s'|s_t, a_t)} \left[\sum_{t \geqslant 0} \gamma^t \mathbb{1}_{s_t = s'} | s_0 = s \right]. \tag{2.3.2}$$

The entry $M_{ss'}$ is also the value function at s if the reward is 1 at s' and 0 everywhere else. As such, M contains the information about *reaching* every state in the environment, not just those states providing a reward.

For a fixed policy π, the value function V^π depends linearly on the reward: in a finite state space, for any reward function, represented as a vector R over states, its associated value function is

$$V^\pi(s) = (M^\pi R)(s) = \sum_{s_2} M^\pi_{ss_2} R_{s_2}. \tag{2.3.3}$$

This equation will allow us to derive policy evaluation algorithms via successor states. First, we will estimate models of the successor state operator M^π. As M^π does not depend on the reward, we can learn it in an *unsupervised* way, without observing any reward. Contrary to standard value function learning algorithms, the algorithm can start to learn before observing

any reward. Then, we can use the successor states model to *compute* a model of the value function once the reward function is observed, using (2.3.3).

In the following, we study how to learn a model of the successor state operator and how to use it. In finite environments with tabular models, this was studied by Dayan (1993), and we will extend this principle to general continuous state spaces, with function approximators. In Chapter 6, we introduce the successor states operator, and its proper definitions of in continuous state space:

- We formally define the successor states operator in general state spaces (Theorem 6.1), extending the discrete case of Dayan (1993). For continuous states, this involves some measure theory: the successor state operator defines for every state s a measure over successor states s':

$$M^\pi(s, \mathrm{d}s') = (\mathrm{Id} - \gamma P)^{-1}(s, \mathrm{d}s') = \sum_{t \geqslant 0} \gamma^t (P^\pi)^t (s, \mathrm{d}s') \tag{2.3.4}$$

  We relate the value function V^π to the successor states operator M^π and the reward R in general state spaces (Proposition 6.3) via:

$$V^\pi(s) = (M^\pi \cdot R)(s) = \int_{s_2} M^\pi(s, \mathrm{d}s_2) R(s_2) \tag{2.3.5}$$

We then describe how to represent the successor operator $M^\pi(s, \mathrm{d}s')$ with function approximators, as a density $m_\theta(s, s')$ with respect to a reference measure $\varphi(\mathrm{d}s')$:

$$M_\theta(s, \mathrm{d}s') := m_\theta(s, s') \rho(\mathrm{d}s') \tag{2.3.6}$$

The reference measure ρ is a probability distribution, such that we are able to sample $s \sim \rho$. We do not require more knowledge on ρ, such as knowing its probability density function, ... Typically, ρ can be the distribution of states observed along trajectories sampled with π with an initial state $s_0 \sim \rho_0$. Unless specified, there are no hypothesis on ρ. For $m_\theta(s_1, s_2)$, we can use any parametric family of functions. In practice, we will use deep learning models.

The successor states operator is related but not equivalent to *successor features* (Kulkarni et al., 2016; Borsa et al., 2018; Barreto et al., 2018; Zhang et al., 2017b; Hansen et al., 2020). Given a *feature* function φ over the state space $\mathcal{S}$, the *successor feature* is the expectation of the cumulated, discounted future values of φ given the starting point s_0 of a trajectory (s_t) is

$$\mathbb{E}\left[\sum_{t \geqslant 0} \gamma^t \varphi(s_t)\right] = \sum_{t \geqslant 0} \gamma^t (P^t \varphi)(s_0) = (M\varphi)(s_0). \tag{2.3.7}$$

Thus, the successor representation of a state s is obtained by applying M to some user-chosen function φ: $M^\pi \cdot \varphi$. In practice the function φ is learned together with the successor feature. Still, in order to prevent convergence to the trivial solution $\varphi = 0$, an additional loss (such as pixel reconstruction) has to be added. On the contrary, the successor states operator does not depend on a given function φ, and can be learned without adding any information independent of the dynamic.

The next steps are then to describe how this density model $m_\theta(s, s')$ can be learned, and used for policy evaluation.

2.3.2 TD algorithms for deep successor states

Once the successor states operator is properly defined, the goal is now to *learn* it. We consider a *model* $M_\theta(s_1, \mathrm{d}s_2) := m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ parameterized by its density $m_\theta(s_1, s_2)$ with respect to a measure ρ, as introduced previously. In Chapter 7, we derive a temporal difference algorithm for learning $m_\theta(s_1, s_2)$. We extend the standard temporal difference approach with function approximators described in Section 1.4.2 to the successor state.

Following this strategy, our first contribution is define such an operator T for the successor state operator:

- We define the (forward) Bellman operator for successor operators:

$$T \cdot M := \mathrm{Id} + \gamma P \cdot M, \tag{2.3.8}$$

  which corresponds to the standard Bellman equation for value functions, but for successor states.

  We show that that the Bellman operator is γ-contractive, and that its unique fixed point is the true successor states operator M^π (Theorem 7.1 and Proposition 7.2).

Once such an operator is defined, we can derive a stochastic Temporal Difference estimate for successor states:

- We define the stochastic update $\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2)$, where we assume (s, s') is a transition observed in the Markov Process and s_2 is an independent state. The update is:

$$\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2) := \partial_\theta m_\theta(s, s) + \partial_\theta m_\theta(s, s_2)\left(\gamma m_\theta(s', s_2) - m_\theta(s, s_2)\right) \tag{2.3.9}$$

  We then define the corresponding Forward TD algorithm for successor states operator (Algorithm 8): informally, at step t, if our current parameter is θ_t, when observing a transition (s_t, s'_t) in the environment, we sample $s_2 \sim \rho$ and define:

$$\theta_{t+1} = \theta_t - \eta_t \widehat{\delta\theta}_{\text{F-TD}}(s_t, s'_t, s_2) \tag{2.3.10}$$

  We prove that $\widehat{\delta\theta}_{\text{F-TD}}$ is an unbiased estimate of the Bellman error (Theorem 7.5): if we define the target $M^{\text{tar}} := T \cdot M_\theta = \mathrm{Id} + \gamma P \cdot M_\theta$, we have:

$$\mathbb{E}_{s,s',s_2}\left[\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2)\right] = \frac{1}{2}\partial_\theta \|M_\theta - M^{\text{tar}}\|^2 \tag{2.3.11}$$

  where formal definition of the probability laws in the expectation, and of the norm $\|.\|$ are given in the corresponding sections.

The strategy used here is similar to standard policy evaluation for the value function with function approximations described in Section 1.4.2, and will be used to derive many algorithms in this thesis: First, we define a contractive *operator* T such that its unique fixed point is M^π. Hence, for any initialization $M_0(s, \mathrm{d}s')$, if we define the sequence $M_{t+1} := T \cdot M_t$, we have $M_t \to_{t\to\infty} M^\pi$. Then, we approximate the sequence M_t with function approximation. We define a model $M_\theta(s, \mathrm{d}s') = m_\theta(s, s')\rho(\mathrm{d}s')$ as introduced above. When observing a trajectory/observation/transition, we compute a stochastic update $\widehat{\delta\theta}$, such that it is an unbiased gradient step towards the target $M^{\text{tar}} := T \cdot M_{\theta_t}$: $\mathbb{E}[\widehat{\delta\theta}] = \frac{1}{2}\partial_\theta \|M_\theta - M^{\text{tar}}\|^2$, and we update θ with $\theta_{t+1} := \theta_t - \eta\widehat{\delta\theta}$.

As explained for V-function in Section 1.4.2, with such an algorithm, M^π is guaranteed to be a fixed point: if there is θ^* such that $M_{\theta^*} = M^\pi$, then $\mathbb{E}[\widehat{\delta\theta}] = 0$. Moreover, if the parametric family M_θ is overparametrized then M^π is the *unique* fixed point of the algorithm.

Experimentally, we demonstrate that we are able to approximate the successor states operator in simple continuous environments with the Forward TD algorithm, with deep neural networks.

We know that several variants of TD are used in practice for learning the value function. We derive similar variants in the context of successor states operator:

- We show how to compute an unbiased forward temporal difference update with an additional target network (Theorem 7.6).

- We define a $TD(n)$ update $\widehat{\delta\theta}_{\text{TD}(n)}$ for the successor states operator. Similarly to the forward TD update, the TD(n) update is an unbiased estimate of the n-step Bellman error (Theorem 7.7).

- We show how to define and learn the successor state-action operator $M^\pi(s, a, \mathrm{d}s')$, which takes into account the first action (similarly to the Q function) (Definition 7.13, Theorems 7.14 and 7.15).

Forward Temporal Difference for successor states corresponds to standard Temporal Difference for the value function, but on the successor states operator. We will now present new algorithms for learning the successor states operator, without any equivalent for the value function, leveraging the additional information contained in the object.

2.3.3 The Backward Temporal Difference Algorithm

Informally, the *forward* TD algorithm is using that, for every target state s_{tar}, if we observe a transition $s \to s'$ (hence s and s' are close to each other), then their value functions must be close. In this section, we present the *backward* temporal difference algorithm, which is using the opposite point of view: for every starting state s_{start}, if we observe a transition $s \to s'$, then the value function of s_{start} if the reward is localized in s must be close to the value function of the same state s_{start} if the reward is localized in s'.

Similarly to forward Temporal Difference, this relation can be formalized as a fixed point equation over the successor states operator:

- We define the backward Bellman operator for successor states operator as:

$$M \mapsto \mathrm{Id} + \gamma M \cdot P \tag{2.3.12}$$

  We show that this operator is γ-contractive and that its unique fixed point is the true successor states operator M^π (Theorem 8.1 and Proposition 8.2).

Once this operator is defined, we obtain a algorithm for function approximators similarly to forward temporal difference:

- We define the stochastic update $\widehat{\delta\theta}_{\text{B-TD}}(s_2, s, s')$, where we assume (s, s') is a transition observed in the Markov Process and s_1 is a state sampled independently. We then define the corresponding Backward TD algorithm for successor states operator.

  We prove that $\widehat{\delta\theta}_{\text{B-TD}}$ is an unbiased estimate of the backward Bellman error (Theorem 8.3).

Finally, we analyze the backward TD algorithm from the *backward process* viewpoint. Informally, if P is the transition matrix of a Markov process, the *backward* process is the process corresponding to (infinite) trajectories $(..., s_{-2}, s_{-1}, s_0, s_1, s_2, ...)$ in the reversed order $(..., s_2, s_1, s_0, s_{-1}, s_{-2}, ...)$. Viewing the backward TD algorithm from this viewpoint lead to the following contribution:

- We show that the backward TD update on operators is equivalent to the forward TD update applied to the backward process (Theorem 8.5).

While the updates $\widehat{\delta\theta}_{\text{B-TD}}$ and $\widehat{\delta\theta}_{\text{F-TD}}$ are not equal, there is therefore a strong relation between them. Still, the forward TD update has an equivalent update on the value function while the backward TD update does not. In the next section, we introduce second-order methods for successor states operator learning.

2.3.4 Second-order methods for successor states

The Bellman–Newton operator and path concatenation In order to introduce our second-order approaches, let us first give an interpretation of M^π, forward and backward TD in terms of *paths* in the environment. In the finite environment case, we can express $M^\pi_{ss'}$ as a

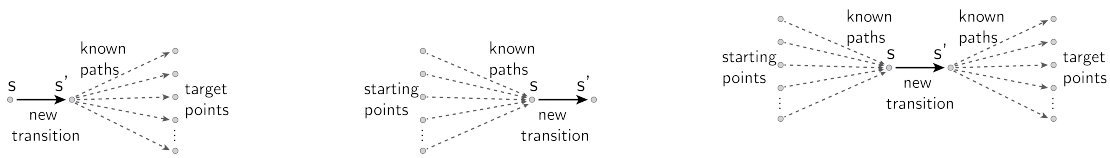


Figure 2.2: Combining paths: forward TD, backward TD, and path composition (Bellman–Newton).

sum over all paths from s to s' in the environment

$$M^{\pi}_{ss'} = (\mathrm{Id} - \gamma P^{\pi})^{-1}_{s_1 s_2} = \sum_{t \geqslant 0} \gamma^t (P^{\pi})^t_{s_1 s_2}$$
$$= \sum_{t \geqslant 0} \gamma^t \sum_{\substack{\text{path}(s_0, s_1, \ldots, s_t) \\ \text{with } s_0 = s \text{ and } s_t = s'}} P^{\pi}_{s_0 s_1} P^{\pi}_{s_1 s_2} \ldots P^{\pi}_{s_{t-1} s_t} . = \sum_{\substack{p \text{ path from} \\ s \text{ to } s'}} \gamma^{\mathtt{length}(p)} \mathbb{P}(p)$$

where $\mathbb{P}((s_0, \ldots, s_t)) = P^{\pi}_{s_0 s_1} \ldots P^{\pi}_{s_{t-1} s_t}$. Therefore, $M^{\pi}_{ss'}$ is the *sum* of all paths from s to s', discounted by their length, and weighted by their probabilities.

From this viewpoint, we can give an other interpretation of forward and backward TD: When a transition $s \to s'$ is observed, for every target state s^{tar} forward TD builds *new* paths from s to s^{tar} by concatenating the transition (s, s') to all known paths from s' to s^{tar} (Figure 2.2, left). On the contrary, when $s \to s'$ is observed, for every starting state s^{start}, backward TD builds new paths from s^{start} to s' by concatenating all known paths from s^{start} to s with the transition (s, s') (Figure 2.2, middle).

This discussion naturally leads to a *third* algorithm: when observing a transition (s, s'), it is possible to build new path from any starting point s^{start} to any target state s^{tar} by concatenating all paths from s^{start} to s, to the transition (s, s'), to all paths from s^{tar} (Figure 2.2, right). Informally, instead of increasing the length of all known paths by 1 at every step, this would double the length of all known paths at every step.

In Chapter 9, we define the Bellman-Newton equation and the corresponding algorithm, corresponding the path composition strategy defined above:

- We define the Bellman-Newton operator:

$$M \mapsto 2M - M \cdot (\mathrm{Id} - \gamma P^{\pi}) \cdot M \tag{2.3.13}$$

  and show that the true successor states operator M^{π} is a fixed point of the Bellman–Newton operator (Proposition 9.4).

  We show that the update defined via the Bellman–Newton equation $M_{t+1} := 2M_t - M_t \cdot (\mathrm{Id} - \gamma P^{\pi}) \cdot M_t$ corresponds to the path concatenation strategy described above (Theorem 9.6). It also corresponds to the Newton method for matrix inversion (Pan and Schreiber, 1991), which explains the name given to the Bellman Newton method method.

Using the Bellman–Newton operator, we then define a Bellman–Newton update for successor states with function approximation:

- We define the stochastic update $\widehat{\delta\theta}_{\text{BN}}(s_1, s, s', s_2)$, where we assume (s, s') is a transition observed in the Markov Process and s_1 and s_2 are states sampled independently. We then define the corresponding Bellman–Newton algorithm for successor states operator (Algorithm 9).

  We prove that $\widehat{\delta\theta}_{\text{BN}}$ is an unbiased estimate of the Bellman–Newton error (Theorem 9.5).

Experimentally, this update raises multiple issues. First, values of $m(s_1, s_2)$ can reach several order of magnitudes. Typically, when $s_1 \approx s_2$ the values can go to infinity, but be of order $O(1)$ in every other cases, and these two regimes needs to be learned accurately (this first

issue is shared by every algorithm learning the successor state operator). Then, the update $\widehat{\delta\theta}_{\mathrm{BN}}(s_1, s, s', s_2)$ requires the sampling of one transition (s, s'), and two additional states s_1 and s_2. On the contrary, Forward TD and Backward TD only require the sampling of a single additional state. Hence, the Bellman–Newton update has a variance. Finally, it is known that Newton's methods can be numerically unstable. Hence, the high variance becomes important, as the method can diverge. To counterbalance this variance, we can use smaller learning rates, but this reduces the efficiency of the method.

These issues makes a vanilla implementation of the Bellman–Newton algorithm not efficient in continuous environments with function approximations. In Chapter 11 introduced below, we present a possible solution to this issue, via low-rank parametrization. We will see that this approach allow us to reduce variance, and still be close to a Bellman–Newton approach in some cases.

Still, we prove that in tabular cases, the Bellman–Newton algorithm approximates the *process estimation* algorithm (Theorem 9.2), defined as follows: in a tabular setting, we can learn an estimate $\hat{P}$ of the transition matrix P^π by keeping a *frequency matrix* of every transition starting from every state. Then, we can define an estimate of M^π as $\hat{M} := (\mathrm{Id} - \gamma \hat{P})^{-1}$. If we also learn $\hat{R}$ a model of the reward, we can define a model of the value function as $\hat{V} := \hat{M}\hat{R}$. This algorithm trivially converges to M^π and V^π when the number of samples goes to infinity, because the matrix inverse is continuous. It corresponds to the *Least Squares Temporal Difference* algorithm in the literature (Bradtke and Barto, 1996).

This strategy is only possible in the tabular setting, and has no direct equivalent in the function approximation setting. Indeed, in a continuous environment, if we learn a model $P_\theta(s_1, \mathrm{d}s_2)$ of the transition operator, we can't directly compute the inverse $(\mathrm{Id} - \gamma \hat{P})^{-1}$. Using a continuous model $P_\theta(s_1, \mathrm{d}s_2)$ would still be possible, for instance by sampling trajectories according to P_θ. This is related to model-based methods, and has known technical issues, discussed in Section 1.3.

- In the tabular setting, the Bellman–Newton algorithm approximates the *process estimation* algorithm (Theorem 9.2), while never estimating the process directly.

In the next Section, we study theoretically the convergence properties of the process estimation algorithm.

2.3.5 A non-asymptotic convergence bound for policy evaluation via process estimation

In order to better understand how much the Bellman–Newton algorithm can potentially improve the sample efficiency of policy evaluation, we study in more details in Chapter 10 the *tabular case*.

We therefore study the sample efficiency of policy evaluation with the *process estimation* algorithm, which corresponds to the LSTD algorithm (Bradtke and Barto, 1996) in its very specific tabular case. We consider the i.i.d data model: we assume the process is ergodic and has an invariant measure ρ. Then, we observe *independent* transitions (s_t, r_t, s'_t) of the environment such that $s_t \sim \rho$, $r_t \sim \mathcal{R}(.|s_t)$, $s'_t \sim P(.|s_t)$, and we consider the $L_1(\rho)$ norm on $\mathcal{S}$ defined as $\|f\|_{L_1(\rho)} = \sum_{s \in \mathcal{S}} \rho(s)|f(s)|$. We also assume that the reward r is bounded by $R_{\max}$.

We consider the process estimation algorithm defined in the previous section: we learn $\hat{P}_t$ as a frequency matrix: $\hat{P}_t = \frac{n_{ss'}}{n_s}$ where $n_{ss'}$ is the number of times the transition (s, s') was observed, and $n_s = \sum_{s_2} n_{ss_2}$. Similarly, we learn $\hat{R}_t$ as $\hat{R}_t(s) = \frac{1}{n_s} \sum_{k \leqslant t | s_k = s} r_k$. Then, we estimate the value function as $\hat{V}_t = (\mathrm{Id} - \gamma \hat{P}_t)^{-1} \hat{R}_t$. This estimate clearly converges to the true value function V^π, as $\hat{P}_t \to P^\pi$, $\hat{R}_t \to R$ almost surely, and the inverse is continuous. We are interested into measuring the sample efficiency of this approach.

We provide a convergence bound on $\hat{V}_t$ with the process estimation algorithm for the $L_1(\rho)$ norm. The most interesting feature this new convergence bound is that it does not depend on the number of states, or of the measure of infrequently visited states. Hence the result is non-vacuous even if some states are almost-never observed, or for a very large number of states,

or even for a discrete infinite state space. Up to our knowledge, it is the first convergence bound for policy evaluation which shows that we can provably learn the value function in finite time, even with an arbitrarily large (or infinite) state space. This is a desirable result: if a state s is *almost never observed* for the measure ρ, an estimate $V(s)$ will clearly be inaccurate, but because we consider the $L_1(\rho)$ or $L_2(\rho)$ norms which weights the error in state s with ρ_s, this error is s should be controlled.

We use the following quantity $\Lambda_t(\rho P)$, introduced by Cohen et al. (2020) in the context of discrete distribution learning:

$$\Lambda_t(\rho P) := \sum_{(s,s')|\rho_s P_{ss'} < 1/t} \rho_s P_{ss'} + \frac{1}{\sqrt{t}} \sum_{(s,s')|\rho_s P_{ss'} \geqslant 1/t} \sqrt{\rho_s P_{ss'}} \tag{2.3.14}$$

The sequence $\Lambda_t(\rho P)$ is decreasing when t increases. We easily have $\Lambda_t(\rho P) \geqslant \frac{1}{\sqrt{t}}$. Moreover, if $\mathcal{S}$ is finite, then we have $\Lambda_t(\rho P) \leqslant \sqrt{\frac{E}{t}}$ where E is the number of *edge* in the graph ((s, s') is an edge if $P^{\pi}_{ss'} > 0$). The equality corresponds to an environment in which the probability distribution $P(.|s)$ is uniform for every state s. More generally, the quantity $\Lambda_t(\rho P)$ is lower if when the distribution is short tail. Interestingly, the quantity $\Lambda_t(\rho P)$ is still well defined when $\mathcal{S}$ is infinite, and can handle a large number of states with low probability.

- We consider the process estimate $\hat{P}_t$ defined above, obtained by keeping a *frequency matrix* of every transition starting from every state, and similarly a tabular model $\hat{R}_t$ of the reward. We define the value estimate as $\hat{V}_t := (\mathrm{Id} - \gamma \hat{P})^{-1} \hat{R}_t$. Then, after t i.i.d. observations ($s \sim \rho$, $s' \sim P_{ss'}$), we have with probability $1 - \delta$ (Theorem 10.2):

$$\|\hat{V}_t(s) - V(s)\|_{L_1(\rho)} \leqslant \frac{R_{\max}}{(1-\gamma)^2} \left(10\Lambda_t(\rho P) + 9\sqrt{\frac{\log \frac{4}{\delta}}{t}} \right) \tag{2.3.15}$$

For a finite environment, we can use that $\Lambda_t(\rho P) \leqslant \sqrt{\frac{E}{t}}$ and the bound corresponds to

$$\|\hat{V}_t(s) - V(s)\|_{L_1(\rho)} \leqslant \frac{R_{\max}}{(1-\gamma)^2 \sqrt{t}} \left(10\sqrt{E} + 9\sqrt{\log \frac{4}{\delta}} \right). \tag{2.3.16}$$

We compare these convergence to known results for policy evaluation. In particular, we consider the results from Bhandari et al. (2018) for Temporal Difference for the norm $L_2(\rho)$ under the same i.i.d observation model, and the results from Pananjady and Wainwright (2019), for an algorithm equivalent to SSIPE (called the *plug-in* in their work), for the L_∞ norm, under the *synchronous* observation model (at every step, a transition from every state is observed).

From these comparisons, our bounds raises a few interesting properties. First, they are remarkably simple, and only depend on γ, $R_{\max}$, and the number of edges in the graph (or $\Lambda_t(\rho P)$). Then, it is, to our knowledge the first bound for policy evaluation, for i.i.d. (or trajectory) data, which is non-vacuous even when some states are hardly ever visited ($\rho(s)$ is very small), or when the number of states goes to infinity.

2.3.6 Matrix Factorization and the Forward-Backward parametrization

Finally, we get back to continuous environments, and study a specific parametric model for the successor state operator, in order to mitigate the variance issue of the Bellman–Newton method. We consider the model $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ with the particular choice:

$$m_\theta(s_1, s_2) = \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2) \rangle = \sum_{i=1}^{r} \left(F_{\theta_F}(s_1)\right)_i \left(B_{\theta_B}(s_2)\right)_i \tag{2.3.17}$$

where $F\colon S \to \mathbb{R}^r$ and $B\colon S \to \mathbb{R}^r$ are two learnable functions from the state space to some representation space $\mathbb{R}^r$, parameterized by $\theta = (\theta_F, \theta_B)$. This provides an approximation of M by a rank-r operator. Such a factorization is used for instance in (Schaul et al., 2015) for the goal-dependent Q-function (up to the factor ρ). Intuitively, F is a "*forward*" representation of states and B a "*backward*" representation: if the future of s_1 matches the past of s_2, then $M(s_1, \mathrm{d}s_2)$ is large.

The forward-TD and backward-TD algorithms introduced above can be directly applied to the FB parametrization, simply by considering $m_\theta(s_1, s_2)$ as any function approximator. Actually, it is also possible to *mix* the forward and backward updates: if we consider θ_B as fixed, we can use the forward or backward TD algorithms on the model $\theta_F \mapsto \langle F_{\theta_F}(.), B_{\theta_B}(.)\rangle$. This defines forward and backward updates for F_{θ_F}, and we can define similarly forward and backward updates for B_{θ_B}. We can then mix these updates and use independently a forward/backward update for F and a potentially different update for B_{θ_B}.

- We define the *mixed* algorithms *forward-forward*, *forward-backward*, *backward-forward* and *backward-backward* for the FB parametrization. We show that the true successor state operator M^π is a fixed point of every of these four algorithms (Theorem ref11.1).

We then study in more details the *forward-backward* algorithm. Indeed, this algorithm has two interesting properties. The first one is about variance. We learn online estimates $\hat{\Sigma}_F$ and $\hat{\Sigma}_B$ of the $r \times r$ covariance matrices Σ_F and Σ_B defined as $\Sigma_F := \mathbb{E}_{s_1 \sim \rho} F(s_1) F(s_1)^\top$ and $\Sigma_B := \mathbb{E}_{s_1 \sim \rho} B(s_1) B(s_1)^\top$, for example by computing a moving average of the matrices $F(s)F(s)^\top$ for every observed state. Then, we define a FB update with reduced variance:

- Knowing estimates $\hat{\Sigma}_F$ and $\hat{\Sigma}_B$ of Σ_F and Σ_B, we define an update $\widehat{\delta\theta}_{\text{fb-TD}}(s, s', \hat{\Sigma}_F, \hat{\Sigma}_B)$, where (s, s') is supposed to be an observed transition in the process (Algorithm 10).

  Contrary to the Forward or Backward TD algorithm defined for any model m_θ, this update does not require the sampling of an additional state s_2.

  We prove that, if we our estimates of the covariance matrices are correct ($\tilde{\Sigma}_F = \Sigma_F$ and $\tilde{\Sigma}_B = \Sigma_B$), the update $\widehat{\delta\theta}_{\text{fb-TD}}$ is an unbiased estimate of the Forward Bellman error gradient for F and of the Backward Bellman error for B, but with lower variance (Theorem 11.2).

The second interesting property is a relation between the fixed points of this method and the SVD. We know that the optimal low-rank approximation of an operator for the L_2 norm corresponds to a truncated SVD. We have the following result:

- We show that the fixed point of the forward-backward TD algorithm are truncated SVDs of rank r of the true successor states operator M^π for the norm $L_2(\rho)$ (Theorem 11.3).

This statement is necessary but not sufficient to show that the algorithm will converge to the optimal low-rank representation. In practice, we observe that this algorithm converges to the optimal low-rank representation of the successor state operator in simple environments.

Additionally, these representations might be useful for other purposes. Once state (or state-actions) representation are computed, they can be used directly as input of a more simple policy (Ha and Schmidhuber, 2018). They can also be used to derive a bonus for exploration (Machado et al., 2019). These FB representations only depend on the dynamics and not on other signals (such as pixels), which can be irrelevant for the task and biased representation learning toward ignoring the most important information.

Finally we show that the Forward-Backward algorithm is related to the Bellman–Newton update defined in the previous section:

- In a limited setting (tabular, in a reversible process in which the uniform measure is the invariant measure of the process), we show that for small learning rates, the Forward-Backward update is equivalent to Bellman–Newton update (Theorem 11.6).

This result is interesting, as it suggests that the FB algorithm might share with the Bellman–Newton algorithm the relation with implicit process estimation, hence its sample efficiency, without the variance issue of Bellman–Newton.

In the next section, we finally describe several methods to learn the value function V^π via the successor states operator.

2.3.7 Learning value functions via successor states models

In Chapter 12, we describe several methods to learn a model $V_\varphi(s)$ of the value function V^π once we are able to learn a model of the successor states operator. We mainly define two approaches: first, by using the equation $V^\pi(s) = (M^\pi \cdot R)(s)$ and estimating the integral $\int_{s_2} \rho(\mathrm{d}s_2) m_\theta(s, s_2) R(s_2)$. Then, by using m_θ as a way to propagate the Bellman error of Temporal Difference in the environment, similarly to TD(λ) with eligibility traces.

Estimating the value function via $V^\pi = M^\pi \cdot R$ We know that:

$$V^\pi(s) = (M^\pi \cdot R)(s) = \int_{s_2} M^\pi(s, \mathrm{d}s_2) R(s_2) \tag{2.3.18}$$

Therefore, after learning a model $m_\theta(s_1, s_2)$, we might want to use the model:

$$\hat{V}(s) := \int_{s_2} \rho(\mathrm{d}s_2) m_\theta(s, s_2) R(s_2) \tag{2.3.19}$$

. We introduce three cases in which we can estimate the integral (2.3.19):

- If the reward is sparse and located in a single known state s^{tar}, equation (2.3.19) corresponds to:
$$\hat{V}(s) = m_\theta(s, s^{\mathrm{tar}}) \tag{2.3.20}$$
up to a multiplicative factor independent of s. Hence, we can directly use $m_\theta(s, s^{\mathrm{tar}})$ as an estimate of the value function V^π.

- If the reward is dense, we consider a model $V_\varphi(s)$ (such as a neural network), and optimize the parameter φ such that $V_\varphi \approx \hat{V}$. We store a *buffer* of tuples (s, r_s) where s is an observed state and r_s the reward observed in s. Then, we can learn V_φ in a supervised way: by sampling (s, r_r) in the buffer and an additional independent state s_1, and reduce the empirical loss $(V_\varphi(s_1) - m_\theta(s_1, s) r_s)^2$

  This approach reduces the problem of policy evaluation to a supervised learning problem (once a model m_θ is learned)

- If the reward is dense and we additionally use the Forward-Backward parametrization described in the previous section ($m_\theta(s_1, s_2) := \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2) \rangle$ where $F(s)$ and $B(s)$ are low rank representations in $\mathbb{R}^k$), we have: $\hat{V}(s) = \mathbb{E}_{s_2 \sim \rho} [\langle F_{\theta_F}(s), B_{\theta_B}(s_2) \rangle] = \langle F_{\theta_F}(s), b \rangle$ where $b := \mathbb{E}_{s_2 \sim \rho} [B_{\theta_B}(s_2)]$. In that case, we can estimate b with an online averaging $\hat{b} \in \mathbb{R}^k$ of state representations ($\hat{b} := \frac{1}{t} \sum_{i=1}^t B_{\theta_B}(s_i)$), and then use the following estimate for the value function: $\hat{V}(s) = \langle F_{\theta_F}(s), \hat{b} \rangle$. With this approach, there is no need to learn an additional parametric model V_φ of the value function.

Using M to propagate the Bellman error in the environment: expected value update via process estimation, and expected TD(λ) update We now consider an other approach for policy evaluation via the successor states operator, in which the successor state model m_θ is used to propagate the Bellman error in the environment, or in other words to improve the *credit assignment* when observing a transition (s, r, s').

First, we derive this method from the expected value update via the process estimation approach in the tabular case, introduced in Section 2.3.4 of this overview, defined as $\hat{V}_t := \hat{M}_t \hat{R}_t$, where $\hat{M}_t := (\mathrm{Id} - \gamma \hat{P}_t)^{-1}$ and $\hat{P}_t$ is the frequency matrix of observed transitions.

- We show that, in expectation over the transition observed at step t (s_t, r, s_{t+1}), conditionally to the current estimates V_t, M_t, we have $\mathbb{E}_{s_t \sim \rho, s'_t \sim P(.|s_t), r \sim \mathcal{R}(.|s)} [V_{t+1}] = V_t + \frac{1}{t}\delta V + o(1/t)$, where $\delta V = M_t(R + \gamma P V_t - V_t)$ (Theorem 12.1).

Informally, this equation means that when observing a transition (s, r, s'), the bellman error $(r_s + \gamma \hat{V}_t(s') - \hat{V}_t(s))$ is propagated to every state s_1 with weight $\hat{M}_t(s_1, s)$. Hence, $M_t(s_1, s)$ propagates the *credit* in the entire environment. We then generalize this update for function approximations:

- Once we know a model $m_\theta(s_1, s_2)$ of the successor states density, we define the stochastic update $\widehat{\delta\varphi}_{\text{prop-TD}}(s, s', r, s_1)$ for the value function V_φ where we assume (s, r, s') is a transition observed in the Markov Process and s_1 is a state sampled independently, as: $\widehat{\delta\varphi}_{\text{prop-TD}}(s, s', r_s, s_1) = \partial_\varphi V_\varphi(s_1) m_\theta(s_1, s) \left(r_s + \gamma V_\varphi(s') - V_\varphi(s)\right)$ (Algorithm 13).

  We prove that $\widehat{\delta\varphi}_{\text{prop-TD}}$ is an unbiased estimate of $\|V_\varphi - V^{\text{tar}}\|^2_\rho$, where $V^{\text{tar}} := V_\varphi + \delta V$ and δV is defined as in the tabular expected value update via process estimation: $\delta V = M_\theta(R + \gamma P V_\varphi - V_\varphi)$ (Theorem 12.2).

Hence, this method can be seen as an approximation of the online update of the value function for the process estimation method, with function approximations.

We then show in Section 12.2.3 that this update corresponds to an estimate of the expected eligibility traces update in TD(λ). Eligibility traces, introduced in Section 1.4.2 are a way to improve credit assignment by propagating the Bellman error to the states recently visited in the current trajectory. We show that our approach is tackling credit assignment by propagating the Bellman error to all states which could have been visited from the current state s, according to the distribution of *predecessor* states, which is equivalent to the expected traces for a state s.

- We prove that the TD(λ) update with eligibility traces and the update $\widehat{\delta\varphi}_{\text{prop-TD}}$ estimating the value update via process estimation are both approximating the expected eligibility traces (Theorem 12.3).

This method is closely related to *expected eligibility traces* (van Hasselt et al., 2020), and *source traces* (Pitis, 2018), both discussed in Section 12.2.3.

One of our issues while working on this project was to find the proper experimental setup. As discussed in Section 2.3.4, learning a model $m_\theta(s_1, s_2)$ of the successor state operator raises multiple technical issues. Additionally, we cannot measure the *direct* efficiency of learning the successor states operator to improve our method: we first need to compute a value function, then to plug this estimate into an other RL algorithm such as actor critic, and observe the policy improvement. Hence, our measure of progress was very indirect.

We wanted to focus on a simple setup, but still with continuous state space. The simplest case for deriving the value function from the successor state operator, in a continuous state space, is when the reward is sparse and localized in a known target state s^{tar}. Unfortunately, this is not a frequent setting in standard environments. Still, this setup is quite similar to multi-goal RL, in which the reward is localized in a known *goal state* g. The main difference is that in multi-goal RL, the agent learns a *goal-dependent* policy $\pi(a|s, g)$, whose objective is to *reach* g, while in our setup, the policy was not goal-dependent $\pi(a|s)$.

We extended our approach for the successor state operator with a fixed policy to the multi-goal RL setting. In the next Section, we describe how this approach allows us to derive unbiased Q-learning methods and actor-critic methods for multi-goal RL, dealing with the issue of sparse rewards.

2.4 Unbiased methods for multi-goal RL

In Part V, we present our work on multi-goal reinforcement learning problems. This part is mainly based on the following preprint:

> Blier, L. and Ollivier, Y. (2021). Unbiased methods for multi-goal reinforcement learning. *arXiv preprint arXiv:2106.08863*

In the last part of the thesis, we study *Multi-goal* reinforcement learning is a specific setting of RL where the agent learns a *goal-dependent policy* $\pi(a|s,g)$, whose objective is to *reach* a goal g in the environment. In this introduction we will consider only states as goals ($g \in \mathcal{S}$), but in Part V the setting is more general.

This setting is not the same the one used in the previous part, for policy evaluation via the successor state operator. In the previous part, via M^π, we are estimating the value function of a *non-goal dependent* policy $\pi(.|s)$ for reaching any goal g. In this part, the policy is different for every goal. Still, we are able to translate some of the tools developed for the successor states for the goal oriented setting. Let $\mathcal{M}$ be a multi-goal environment with state space $\mathcal{S}$ and a goal dependent reward $R(s,g)$. Typically, in a discrete environment, the goal dependent reward is defined as $R(s,g) = \mathbb{1}_{s=g}$, which is the sparse reward, non-zero only if the goal is reached. We try to optimize a policy $\pi(.|s,g)$.

We define the *augmented* environment $\tilde{\mathcal{M}}$ with state space $\tilde{\mathcal{S}} := \mathcal{S} \times \mathcal{S}$, in which the current state $\tilde{s} \in \tilde{\mathcal{S}}$ is defined as $\tilde{s} := (s,g)$, the tuple containing the state in the *original* process s and the currently aimed goal g. In the *augmented* environment, the goal-dependent reward $R(s,g)$ becomes the non-goal oriented reward $\tilde{R}(\tilde{s}) := R(s,g)$, and similarly the goal-oriented policy $\pi(a|s,g)$ becomes the non-goal oriented policy $\tilde{\pi}(a|\tilde{s})$. In that setting, estimating the *multi-goal value function* $V^{\pi(.|.,g)}(s,g)$ is now equivalent to estimating the value function in the augmented environment $\tilde{V}^{\tilde{\pi}}(\tilde{s})$, if the reward is a sparse reward in g. Estimating the value function for every sparse goal is now very similar to estimating the successor states operator in the augmented environment $\tilde{\mathcal{M}}$.

The first known approach for multi-goal RL is with *Universal Value Function Approximators* (UVFA) (Schaul et al., 2015), which extend the classical Q-learning and Temporal Difference (TD) algorithms to the multi-goal setting. It learns the goal-conditioned value-function $V^\pi(s,g)$ or Q-function $Q^*(s,a,g)$ for every state-goal pair, with function approximation, via a TD algorithm.

Still, UVFA requires observing rewards, and no learning occurs until a reward is observed. In continuous state spaces, the reward is usually defined as $R_\varepsilon(s,g) = \mathbb{1}_{\|s-g\| \leqslant \varepsilon}$. When $\varepsilon \to 0$, the probability of reaching the reward with a stochastic policy goes to 0, and UVFA can't learn. In practice, UVFA fails in many high dimensional environments, when the probability of reaching the target goal is low and the agent almost never gets any learning signal. We call this phenomena the issue of *vanishing rewards*.

The most popular method in that setting is *Hindsight Experience Replay* (HER) (Andrychowicz et al., 2017). It leverages information between goals via the following principle: trajectories aiming at a goal g but reaching a goal g' can be used for learning exactly as if the trajectory had been aiming at g' from start. This strategy has proved successful in practice and removes the issue of *vanishing rewards*, but is known to be *biased* (Manela and Biess, 2021; Lanka and Wu, 2018).

In the following, we first study some of HER's theoretical properties. Then, we will derive a Q-learning algorithm and an actor-critic algorithm for multi-goal environments.

2.4.1 A study of Hindsight Experience Replay's bias

While HER has proved successful in practice, it is known to be *biased* (Manela and Biess, 2021; Lanka and Wu, 2018), which means it could converge in some settings to low-return policy. This bias corresponds to a well-known psychological bias (Fischhoff, 1975). In their request for research for robotic multi-goal environments, Plappert et al. (2018) list the necessity for an unbiased version of HER, as such bias can lead to low-return policies.

In chapter 14, we study HER's bias. First, we confirm theoretically that HER is biased:

- We define *counter-example* environments, such that it highlights HER's bias. Theoretically, we prove that HER cannot converge to the true optimal Q-function Q^* in these environments (Theorem 14.2). Empirically, we show that in such environments, HER converges to a low-return policy.

Our second contribution on HER is a *positive* result. We show that despite its bias in general settings, HER is mathematically well-grounded in deterministic environments:

- We show that HER is actually *unbiased* in deterministic environments (Theorem 14.1).

This result vindicates HER for deterministic environments: HER leverages the structure of multi-goal environments, is not vanishing when the rewards are sparse, and is mathematically well-grounded. This covers many usual environments such as robotic environments.

2.4.2 Multi-goal RL via infinitely sparse rewards

While HER is well-founded in deterministic environments, it is biased in the stochastic case and can learn low-return policies. We now introduce unbiased methods for multi-goal RL in the general setting, including stochastic environments, removing the issue of *vanishing rewards*. We first introduce the setting used to derive our algorithms.

In continuous state spaces, the goal-oriented reward is usually defined as:

$$R_\varepsilon(s,g) = \mathbb{1}_{\|s-g\|\leqslant\varepsilon}. \tag{2.4.1}$$

When $\varepsilon \to 0$, the probability of reaching the reward with a stochastic policy goes to 0, and for any stochastic policy, the value function $V_\varepsilon^\pi(s,g)$ converges to 0 as well. This is the *vanishing rewards* issue. To avoid this issue, we need a scaling factor, and consider the reward $\frac{1}{\lambda(\varepsilon)}R_\varepsilon(s,g)$, with $\lambda(\varepsilon)$ the volume of the ball of size ε in $\mathcal{S}$. When $\varepsilon \to 0$, this rescaled reward *converges* to the *Dirac reward*:

$$R(s,\mathrm{d}g) := \delta_s(\mathrm{d}g), \tag{2.4.2}$$

where δ_x is the Dirac measure at x. Intuitively, the Dirac reward $R(s,\mathrm{d}g)$ is infinite if the goal is reached ($s = g$) and 0 elsewhere. Formally, the reward is not a function but a *measure* on the goal space $\mathcal{G}$ parametrized by the state s.

However, even after such a scaling, the UVFA update still vanishes with high probability for small ε (this just scales things by $1/\lambda(\varepsilon)$). We will build algorithms that work directly in the limit $\varepsilon = 0$: replacing the sparse reward $R_\varepsilon(s,g)$ by the *infinitely sparse* reward $R(s,\mathrm{d}g) = \delta_s(\mathrm{d}g)$ will allow us to leverage the Dirac structure to remove the vanishing rewards issue.

The first step is to understand this setting mathematically. In Chapter 13, we formally define multi-goal RL with infinitely sparse rewards, and check that it *corresponds asymptotically* to the original problem with reward R_ε:

- We properly define the infinitely sparse reward via Dirac measures, and show that it corresponds to the *limit* of the reward R_ε.

  Under continuity assumptions, we define the corresponding *expected return with infinitely sparse rewards* $J(\pi)$, and show that its corresponds to the limit of the limit of the return $J_\varepsilon(\pi)$ with reward R_ε when $\varepsilon \to 0$: $J_\varepsilon(\pi) \to_{\varepsilon\to 0} J(\pi)$ (Theorem 13.8).

  Under these assumptions, we show that if $\pi_1(.|s,g)$ and $\pi_2(.|s,g)$ are two goal-conditioned policies, π_1 is *better* than π_2 with infinitely sparse rewards if and only if π_1 is *asymptotically better* than π_2 for reward R_ε when $\varepsilon \to 0$ (Theorem 13.7).

There results allow us to work directly with *infinitely sparse rewards*, even for solving the original problem with reward R_ε, when ε is small. Counter-intuitively, replacing *sparse* rewards by *infinitely sparse* rewards solves the vanishing issue. Instead of waiting an observation of the reward, we can algebraically compute the reward contribution in the updates, leveraging our knowledge on the Dirac function, an obtain non-vanishing algorithms.

In the following sections, we describe how to design Q-learning and actor-critic methods, with no vanishing reward issue, via infinitely sparse rewards.

2.4.3 Unbiased actor-critic for multi-goal RL

In chapter 16, we describe actor critic methods for multi-goal RL via infinitely sparse rewards. We consider a parametric goal-conditioned policy $\pi_{\theta^\pi}(a|s,g)$. Our goal is to maximize $\theta^\pi \mapsto J(\pi_{\theta^\pi})$, by computing stochastic estimates of $\partial_{\theta^\pi} J(\pi_{\theta^\pi})$. Our goal is to adapt the standard policy gradient theorem stated in Proposition 1.3 in the case of goal-oriented environments: $\partial_{\theta^\pi} J(\pi_{\theta^\pi}) = \mathbb{E}_{s,a,s'}\left[\partial_{\theta^\pi} \log \pi_{\theta^\pi}(a|s)\left(r_s + \gamma V^\pi(s') - V^\pi(s)\right)\right]$. This requires learning a model of the multi-goal value function.

Actually, we show that while the value function satisfies a Bellman equation, it is not directly possible to estimate an unbiased estimate of the Bellman error gradient on the value function. This can be explained because of the double dependency of the value function as a function of the goal: the goal both defines the policy and the reward. These two effects need to be separated in order to get an unbiased estimate of Bellman error gradient. We mitigate this issue by introducing the *successor goal operator* $M^\pi(s, g_1, \mathrm{d}g_2)$. This object is very similar to the successor states operator, and describes the expected discounted time spent in the goal g_2 if following the policy g_1. We have the following relation between the goal-conditioned value measure and the successor goal operator:

- Under continuity assumptions, we can compute the goal conditioned value measure from the successor goal operator, using:

$$V^\pi(s, \mathrm{d}g) = M^\pi(s, g, \mathrm{d}g) \tag{2.4.3}$$

  Hence, if we are able to learn a model $M_\theta(s, g_1, \mathrm{d}g_2) = m_\theta(s, g_1, g_2)\rho(\mathrm{d}g_2)$ for the successor goal operator, we naturally obtain a model $V_\theta(s, \mathrm{d}g) = m_\theta(s, g, g)\rho(\mathrm{d}g)$ of the goal-conditioned value measure (Theorem 13.5).

Therefore, our objective is now to describe how to learn an unbiased estimate of the successor goal operator $M^\pi(s, g_1, \mathrm{d}g_2)$. This can be done by applying some of our results derived for the successor goals operator:

- We define a γ-contractive Bellman operator for the successor goals operator, show that its unique fixed point is the true successor goals operator M^π, and derive an unbiased estimate of the Bellman error's gradient for function approximators (Theorems 16.1 and 13.2). This algorithm removes the vanishing reward issue.

Finally, we are able to derive an unbiased actor-critic algorithm for goal-conditioned policy. This actor-critic update is an extension of the standard actor-critic update defined in Proposition 1.3, but for multi-goal environments:

- We define the actor-critic update $\widehat{\delta\theta}_{\delta\text{-AC}}(s, a, s', g)$

$$\widehat{\delta\theta}_{\delta\text{-AC}}(s, a, s', g) := \partial_\theta \log \pi_\theta(a|s, g)\left(\gamma m_{\theta_M}(s', g, g) - m_{\theta_M}(s, g, g)\right) \tag{2.4.4}$$

  where we assume (s, a, s') is a transition observed while aiming for goal g.

  We show that with an accurate model m_{θ_M} of the successor goals operator, $\widehat{\delta\theta}_{\delta\text{-AC}}(s, a, s', g)$ is an unbiased estimate of the policy gradient $\partial_{\theta^\pi} J(\pi_{\theta^\pi})$ (Theorem 16.2).

2.4.4 Unbiased Q-learning for multi-goal RL

In Chapter 15, we derive an unbiased Q-learning algorithm with infinitely sparse rewards, solving the issue of *vanishing rewards*. Our approach is similar to the strategy described for the successor state operator: first, we define a contractive operator on the space of action-value measures such that its fixed point is the $Q^*(s, a, \mathrm{d}g)$, then we use this operator to define an unbiased Q-learning method with function approximators:

- We formally define the *optimal action-value measure* $Q^*(s, a, \mathrm{d}g)$, and the *optimal Bellman operator for action-value measure*:

$$Q(s, a, .) \mapsto \delta_s(.) + \gamma \mathbb{E}_{s' \sim P(.|s,a)} \left[\sup_{a'} Q(s', a', .) \right], \tag{2.4.5}$$

  We show that if we define the sequence $Q_{t+1} := T \cdot Q_t$, then $Q_t \to_{t \to \infty} Q^*$, similarly to standard result on the Q-function and the optimal Bellman operator.

Once we defined Q^* and the optimal Bellman operator, we can derive a Q-learning algorithm with function approximations, similarly to our approach for learning the successor state operator. We represent a model $Q_\theta(s, a, \mathrm{d}g) := q_\theta(s, a, g)\rho(\mathrm{d}g)$, where ρ is a reference measure on goals:

- We define the stochastic update $\widehat{\delta\theta}_{\delta\text{-DQN}}(s, a, s', g)$, where we assume (s, a, s') is a transition observed in the Markov Process and g is an independent goal:

$$\widehat{\delta\theta}_{\delta\text{-DQN}}(s, a, s', g) := \partial_\theta q_\theta(s, a, s) + \partial_\theta q_\theta(s, a, g) \left(\gamma \max_{a'} q_\theta(s', a', g) - q_\theta(s, a, g) \right) \tag{2.4.6}$$

  We then define the corresponding Q-learning algorithm with infinitely sparse rewards (Algorithm 14). This update removes the issue of vanishing rewards.

  We prove that $\widehat{\delta\theta}_{\delta\text{-DQN}}$ is an unbiased estimate of the optimal Bellman error (Theorem 15.2).

This algorithm can be used with discrete actions such as DQN (Mnih et al., 2013) or with continuous actions such as DDPG (Lillicrap et al., 2015). It is off-policy, hence can be used with any exploration strategy. Informally, the first term is leveraging that, when observing the state s, we have an information on the Q-function on how to reach s. The second term propagates the Q-value on how to reach the target goal g.

Experimentally, we demonstrate that the algorithms using infinitely sparse rewards improves performance of the corresponding method (UVFA) using sparse reward R_ε. In environments designed to exhibit the HER bias issue, we show that HER is unable to learn while unbiased methods can learn the optimal policy. Still, these methods do not perform as well as HER in some standard environments, and are unable to learn at all in more complex environments.

One of the issues of these methods is variance. The Dirac rewards remove the *infinite* variance of vanishing rewards in UVFA when $\varepsilon \to 0$ (first term of (2.4.6)). But this does not change the way the reward is propagated to other states (second term of (2.4.6)). Selecting goals g more correlated to the state s as in HER could also be helpful, but this is not obvious to do without re-introducing HER-style bias.

To conclude, in Part V, we prove that there exist unbiased goal-oriented RL algorithms which do not vanish when rewards become sparse: it is possible to deal with sparse rewards in RL directly via the infinitely sparse reward limit, although this does not solve all variance issues. We also prove that another multi-goal method, HER, is unbiased and has the correct fixed point in all deterministic environments

Part II

An information theory viewpoint on the complexity of Deep Learning Models

Chapter 3

The Description Length of Deep Learning Models

In this chapter, we present the following published paper:

Blier, L. and Ollivier, Y. (2018). The description length of deep learning models. In *Advances in Neural Information Processing Systems*

3.1 Introduction

Deep learning has achieved remarkable results in many different areas (LeCun et al., 2015). Still, the ability of deep models not to overfit despite their large number of parameters is not well understood. To quantify the complexity of these models in light of their generalization ability, several metrics beyond parameter-counting have been measured, such as the number of degrees of freedom of models (Gao and Jojic, 2016), or their intrinsic dimension (Li et al., 2018). These works concluded that deep learning models are significantly simpler than their numbers of parameters might suggest.

In information theory and Minimum Description Length (MDL), learning a good model of the data is recast as using the model to losslessly transmit the data in as few bits as possible. More complex models will compress the data more, but the model must be transmitted as well. The overall codelength can be understood as a combination of quality-of-fit of the model (compressed data length), together with the cost of encoding (transmitting) the model itself. For neural networks, the MDL viewpoint goes back as far as (Hinton and Van Camp, 1993), which used a variational technique to estimate the joint compressed length of data and parameters in a neural network model.

Compression is strongly related to generalization and practical performance. Standard sample complexity bounds (VC-dimension, PAC-Bayes...) are related to the compressed length of the data in a model, and any compression scheme leads to generalization bounds (Blum and Langford, 2003). Specifically for deep learning, (Arora et al., 2018) showed that compression leads to generalization bounds (see also (Dziugaite and Roy, 2017)). Several other deep learning methods have been inspired by information theory and the compression viewpoint. In unsupervised learning, autoencoders and especially variational autoencoders (Kingma and Welling, 2013) are compression methods of the data (Ollivier, 2014). In supervised learning, the information bottleneck method studies how the hidden representations in a neural network compress the inputs while preserving the mutual information between inputs and outputs (Tishby and Zaslavsky, 2015; Shwartz-Ziv and Tishby, 2017; Achille and Soatto, 2017).

MDL is based on Occam's razor, and on Chaitin's hypothesis that "*comprehension is compression*" (Chaitin, 2007): any regularity in the data can be exploited both to compress it and to make predictions. This is ultimately rooted in Solomonoff's general theory of inference (Solomonoff, 1964) (see also, e.g., (Hutter, 2007; Schmidhuber, 1997)), whose principle is to

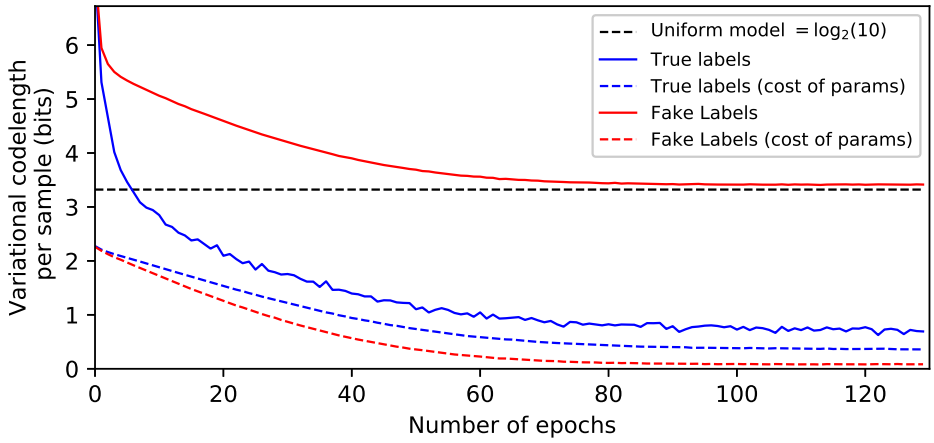


Figure 3.1: **Fake labels cannot be compressed** Measuring codelength while training a deep model on MNIST with true and fake labels. The model is an MLP with 3 hidden layers of size 200, with RELU units. With ordinary SGD training, the model is able to overfit random labels. The plot shows the effect of using variational learning instead, and reports the variational objective (encoding cost of the training data, see Section 3.3.3), on true and fake labels. We also isolated the contribution from parameter encoding in the total loss (KL term in (3.3.2)). With true labels, the encoding cost is below the uniform encoding, and half of the description length is information contained in the weights. With fake labels, on the contrary, the encoding cost converges to a uniform random model, with no information contained in the weights: there is no mutual information between inputs and outputs.

favor models that correspond to the "shortest program" to produce the training data, based on its Kolmogorov complexity (Li and Vitányi, 2008). If no structure is present in the data, no compression to a shorter program is possible.

The problem of overfitting fake labels is a nice illustration: convolutional neural networks commonly used for image classification are able to reach 100% accuracy on random labels on the train set (Zhang et al., 2017a). However, measuring the associated compression bound (Fig. 3.1) immediately reveals that these models do not *compress* fake labels (and indeed, theoretically, they cannot, see Appendix 3.A), that no information is present in the model parameters, and that no learning has occurred.

In this work we explicitly measure how much current deep models actually compress data. As seen above, this may clarify several issues around generalization and measures of model complexity. Our contributions are:

- We show that the traditional method to estimate MDL codelengths in deep learning, variational inference (Hinton and Van Camp, 1993), yields surprisingly inefficient codelengths for deep models, despite explicitly minimizing this criterion. This might explain why variational inference as a regularization method often does not reach optimal test performance.

- We introduce new practical ways to compute tight compression bounds in deep learning models, based on the MDL toolbox (Grünwald, 2007; Rissanen, 2007). We show that *prequential coding* on top of standard learning, yields much better codelengths than variational inference, correlating better with test set performance. Thus, despite their many parameters, deep learning models do compress the data well, even when accounting for the cost of describing the model.

3.2 Probabilistic Models, Compression, and Information Theory

Imagine that Alice wants to efficiently transmit some information to Bob. Alice has a dataset $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$ where $x_1, ..., x_n$ are some inputs and $y_1, ..., y_n$ some labels. We do not assume that these data come from a "true" probability distribution. Bob also has the data $x_1, ..., x_n$, but he does not have the labels. This describes a *supervised learning* situation in which the inputs x may be publicly available, and a prediction of the labels y is needed. How can deep learning models help with data encoding? One key problem is that Bob does not necessarily know the precise, trained model that Alice is using. So some explicit or implicit transmission of the model itself is required.

We study, in turn, various methods to encode the labels y, with or without a deep learning model. Encoding the labels knowing the inputs is equivalent to estimating their mutual information (Section 3.2.4); this is distinct from the problem of practical network compression (Section 3.3.2) or from using neural networks for lossy data compression. Our running example will be image classification on the MNIST (LeCun et al., 1998a) and CIFAR10 (Krizhevsky, 2009) datasets.

3.2.1 Definitions and notation

Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ the output (label) space. In this work, we only consider classification tasks, so $\mathcal{Y} = \{1, ..., K\}$. The dataset is $\mathcal{D} := \{(x_1, y_1), ..., (y_n, x_n)\}$. Denote $x_{k:l} := (x_k, x_{k+1}, ..., x_{l-1}, x_l)$. We define a *model* for the supervised learning problem as a conditional probability distribution $p(y|x)$, namely, a function such that for each $x \in \mathcal{X}$, $\sum_{y \in \mathcal{Y}} p(y|x) = 1$. A *model class*, or *architecture*, is a set of models depending on some parameter θ: $\mathcal{M} = \{p_\theta, \theta \in \Theta\}$. The *Kullback–Leibler divergence* between two distributions is $\mathrm{KL}(\mu \| \nu) = \mathbb{E}_{X \sim \mu}[\log_2 \frac{\mu(x)}{\nu(x)}]$.

3.2.2 Models and codelengths

We recall a basic result of compression theory (Shannon, 1948).

Proposition 3.1 (Shannon–Huffman code)**.** *Suppose that Alice and Bob have agreed in advance on a model p, and both know the inputs $x_{1:n}$. Then there exists a code to transmit the labels $y_{1:n}$ losslessly with codelength (up to at most one bit on the whole sequence)*

$$L_p(y_{1:n}|x_{1:n}) = -\sum_{i=1}^{n} \log_2 p(y_i|x_i) \tag{3.2.1}$$

This bound is known to be optimal if the data are independent and coming from the model p (Mackay, 2003). The one additional bit in the Shannon–Huffman code is incurred only once for the whole dataset (Mackay, 2003). With large datasets this is negligible. Thus, from now on we will systematically omit the $+1$ as well as admit non-integer codelengths (Grünwald, 2007). We will use the terms *codelength* or *compression bound* interchangeably.

This bound is exactly the categorical *cross-entropy loss* evaluated on the model p. Hence, trying to minimize the description length of the outputs over the parameters of a model class is equivalent to minimizing the usual classification loss.

Here we do not consider the practical implementation of compression algorithms: we only care about the theoretical *bit length* of their associated encodings. We are interested in measuring the amount of information contained in the data, the mutual information between input and output, and how it is captured by the model. Thus, we will directly work with codelength functions.

An obvious limitation of the bound (3.2.1) is that Alice and Bob both have to know the model p in advance. This is problematic if the model must be learned from the data.

3.2.3 Uniform encoding

The uniform distribution $p_{\text{unif}}(y|x) = \frac{1}{K}$ over the K classes does not require any learning from the data, thus no additional information has to be transmitted. Using $p_{\text{unif}}(y|x)$ (3.2.1) yields a codelength

$$L^{\text{unif}}(y_{1:n}|x_{1:n}) = n \log_2 K \tag{3.2.2}$$

This *uniform encoding* will be a sanity check against which to compare the other encodings in this text. For MNIST, the uniform encoding cost is $60000 \times \log_2 10 = 199\,\text{kbits}$. For CIFAR, the uniform encoding cost is $50000 \times \log_2 10 = 166\,\text{kbits}$.

3.2.4 Mutual information between inputs and outputs

Intuitively, the only way to beat a trivial encoding of the outputs is to use the mutual information (in a loose sense) between the inputs and outputs.

This can be formalized as follows. Assume that the inputs and outputs follow a "true" joint distribution $q(x, y)$. Then any transmission method with codelength L satisfies (Mackay, 2003)

$$\mathbb{E}_q[L(y|x)] \geqslant H(y|x) \tag{3.2.3}$$

Therefore, the gain (per data point) between the codelength L and the trivial codelength $H(y)$ is

$$H(y) - \mathbb{E}_q[L(y|x)] \leqslant H(y) - H(y|x) = I(y; x) \tag{3.2.4}$$

the mutual information between inputs and outputs (Mackay, 2003).

Thus, the gain of *any* codelength compared to the uniform code is limited by the amount of mutual information between input and output. (This bound is reached with the true model $q(y|x)$.) Any successful compression of the labels is, at the same time, a direct estimation of the mutual information between input and output. The latter is the central quantity in the Information Bottleneck approach to deep learning models (Shwartz-Ziv and Tishby, 2017).

Note that this still makes sense without assuming a true underlying probabilistic model, by replacing the mutual information $H(y) - H(y|x)$ with the "absolute" mutual information $K(y) - K(y|x)$ based on Kolmogorov complexity K (Li and Vitányi, 2008).

3.3 Compression Bounds via Deep Learning

Various compression methods from the MDL toolbox can be used on deep learning models. (Note that a given model can be stored or encoded in several ways, some of which may have large codelengths. A good model in the MDL sense is one that admits at least one good encoding.)

3.3.1 Two-Part Encodings

Alice and Bob can first agree on a model class (such as "neural networks with two layers and 1,000 neurons per layer"). However, Bob does not have access to the labels, so Bob cannot train the parameters of the model. Therefore, if Alice wants to use such a parametric model, the parameters themselves have to be transmitted. Such codings in which Alice first transmits the parameters of a model, then encodes the data using this parameter, have been called *two-part codes* (Grünwald, 2007).

Definition 3.2 (Two-part codes)**.** Assume that Alice and Bob have first agreed on a model class $(p_\theta)_{\theta\in\Theta}$. Let $L_{\text{param}}(\theta)$ be any encoding scheme for parameters $\theta \in \Theta$. Let θ^* be any parameter. The corresponding *two-part codelength* is

$$L^{\text{2-part}}_{\theta^*}(y_{1:n}|x_{1:n}) := L_{\text{param}}(\theta^*) + L_{p_{\theta^*}}(y_{1:n}|x_{1:n}) = L_{\text{param}}(\theta^*) - \sum_{i=1}^{n} \log_2 p_{\theta^*}(y_i|x_i) \tag{3.3.1}$$

Table 3.1: **Compression bounds via Deep Learning.** Compression bounds given by different codes on two datasets, MNIST and CIFAR10. The *Codelength* is the number of bits necessary to send the labels to someone who already has the inputs. This codelength *includes* the description length of the model. The *compression ratio* for a given code is the ratio between its codelength and the codelength of the uniform code. The *test accuracy* of a model is the accuracy of its predictions on the test set. For 2-part and network compression codes, we report results from (Han et al., 2015a) and (Xu et al., 2017b), and for the intrinsic dimension code, results from (Li et al., 2018). The values in the table for these codelengths and compression ratio are lower bounds, only taking into account the codelength of the weights, and not the codelength of the data encoded with the model (the final loss is not always available in these publications). For variational and prequential codes, we selected the model and hyperparameters providing the best compression bound.

Code	MNIST			CIFAR10		
	Codelength (kbits)	Comp. Ratio	Test Acc	Codelength (kbits)	Comp. Ratio	Test Acc
Uniform	199	1.	10%	166	1.	10%
float32 2-part	> 8.6Mb	$> 45.$	98.4%	> 428Mb	$> 2500.$	**92.9%**
Network compr.	> 400	$> 2.$	98.4%	> 14Mb	$> 83.$	**93.3%**
Intrinsic dim.	> 9.28	> 0.05	90%	$> 92,8$	> 0.56	70%
Variational	22.2	0.11	98.2%	89.0	0.54	66,5%
Prequential	**4.10**	**0.02**	**99.5%**	**45.3**	0.27	**93.3%**

An obvious possible code L_{param} for θ is the standard float32 binary encoding for θ, for which $L_{\text{param}}(\theta) = 32\dim(\theta)$. In deep learning, two-part codes are widely inefficient and much worse than the uniform encoding (Graves, 2011). For a model with 1 million parameters, the two-part code with float32 binary encoding will amount to 32 Mbits, or 200 times the uniform encoding on CIFAR10.

3.3.2 Network Compression

The practical encoding of trained models is a well-developed research topic, e.g., for use on small devices such as cell phones. Such encodings can be seen as two-part codes using a clever code for θ instead of encoding every parameter on 32 bits. Possible strategies include training a *student layer* to approximate a well-trained network (Ba and Caruana, 2014; Romero et al., 2015), or pipelines involving retraining, pruning, and quantization of the model weights (Han et al., 2015a,b; Simonyan and Zisserman, 2014; Louizos et al., 2017; See et al., 2016; Ullrich et al., 2017).

Still, the resulting codelengths (for compressing the labels given the data) are way above the uniform compression bound for image classification (Table 3.1).

Another scheme for network compression, less used in practice but very informative, is to sample a random low-dimensional affine subspace in parameter space and to optimize in this subspace (Li et al., 2018). The number of parameters is thus reduced to the dimension of the subspace and we can use the associated two-part encoding. (The random subspace can be transmitted via a pseudorandom seed.) Our methodology to derive compression bounds from (Li et al., 2018) is detailed in Appendix 3.B.

3.3.3 Variational and Bayesian Codes

Another strategy for encoding weights with a limited precision is to represent these weights by random variables: the uncertainty on θ represents the precision with which θ is transmitted. The *variational code* turns this into an explicit encoding scheme, thanks to the *bits-back* argument (Honkela and Valpola, 2004). Initially a way to compute codelength bounds with

neural networks (Hinton and Van Camp, 1993), this is now often seen as a regularization technique (Blundell et al., 2015). This method yields the following codelength.

Definition 3.3 (Variational code)**.** Assume that Alice and Bob have agreed on a model class $(p_\theta)_{\theta\in\Theta}$ and a prior α over Θ. Then for any distribution β over Θ, there exists an encoding with codelength

$$L_\beta^{\text{var}}(y_{1:n}|x_{1:n}) = \text{KL}\left(\beta\|\alpha\right) + \mathbb{E}_{\theta\sim\beta}\left[L_{p_\theta}(y_{1:n}|x_{1:n})\right] = \text{KL}\left(\beta\|\alpha\right) - \mathbb{E}_{\theta\sim\beta}\left[\sum_{i=1}^{n}\log_2 p_\theta(y_i|x_i)\right] \tag{3.3.2}$$

This can be minimized over β, by choosing a parametric model class $(\beta_\varphi)_{\varphi\in\Phi}$, and minimizing (3.3.2) over φ. A common model class for β is the set of multivariate Gaussian distributions $\{\mathcal{N}(\mu,\Sigma), \mu\in\mathbb{R}^d, \Sigma \text{ diagonal}\}$, and μ and Σ can be optimized with a stochastic gradient descent algorithm (Graves, 2011; Kucukelbir et al., 2017). Σ can be interpreted as the precision with which the parameters are encoded.

The variational bound L_β^{var} is an upper bound for the Bayesian description length bound of the Bayesian model p_θ with parameter θ and prior α. Considering the Bayesian distribution of y,

$$p_{\text{Bayes}}(y_{1:n}|x_{1:n}) = \int_{\theta\in\Theta} p_\theta(y_{1:n}|x_{1:n})\alpha(\theta)d\theta, \tag{3.3.3}$$

then Proposition 3.1 provides an associated code via (3.2.1) with model p_{Bayes}: $L^{\text{Bayes}}(y_{1:n}|x_{1:n}) = -\log_2 p_{\text{Bayes}}(y_{1:n}|x_{1:n})$ Then, for any β we have (Graves, 2011)

$$L_\beta^{\text{var}}(y_{1:n}|x_{1:n}) \geqslant L^{\text{Bayes}}(y_{1:n}|x_{1:n}) \tag{3.3.4}$$

with equality if and only if β is equal to the Bayesian posterior $p_{\text{Bayes}}(\theta|x_{1:n}, y_{1:n})$. Variational methods can be used as approximate Bayesian inference for intractable Bayesian posteriors.

We computed practical compression bounds with variational methods on MNIST and CIFAR10. Neural networks that give the best variational compression bounds appear to be smaller than networks trained the usual way. We tested various fully connected networks and convolutional networks (Appendix 3.C): the models that gave the best variational compression bounds were small LeNet-like networks. To test the link between compression and test accuracy, in Table 3.1 we report the best model based on compression, not test accuracy. This results in a drop of test accuracy with respect to other settings.

On MNIST, this provides a codelength of the labels (knowing the inputs) of 24.1 kbits, i.e., a compression ratio of 0.12. The corresponding model achieved 95.5% accuracy on the test set.

On CIFAR, we obtained a codelength of 89.0 kbits, i.e., a compression ratio of 0.54. The corresponding model achieved 61.6% classification accuracy on the test set.

We can make two observations. First, choosing the model class which minimizes variational codelength selects smaller deep learning models than would cross-validation. Second, the model with best variational codelength has low classification accuracy on the test set on MNIST and CIFAR, compared to models trained in a non-variational way. This aligns with a common criticism of Bayesian methods as too conservative for model selection compared with cross-validation (Rissanen et al., 1992; Foster and George, 1994; Barron and Yang, 1999; Grünwald, 2007).

3.3.4 Prequential or Online Code

The next coding procedure shows that deep neural models which generalize well also compress well.

The prequential (or online) code is a way to encode both the model and the labels without *directly* encoding the weights, based on the *prequential approach to statistics* (Dawid, 1984), by using *prediction strategies*. Intuitively, a model with default values is used to encode the first few data; then the model is trained on these few encoded data; this partially trained model is used to encode the next data; then the model is retrained on all data encoded so far; and so on.

Precisely, we call p a *prediction strategy* for predicting the labels in $\mathcal{Y}$ knowing the inputs in $\mathcal{X}$ if for all k, $p(y_{k+1}|x_{1:k+1}, y_{1:k})$ is a conditional model; namely, any strategy for predicting the $k+1$- label after already having seen k input-output pairs. In particular, such a model may *learn* from the first k data samples. Any prediction strategy p defines a model on the whole dataset:

$$p^{\text{preq}}(y_{1:n}|x_{1:n}) = p(y_1|x_1) \cdot p(y_2|x_{1:2}, y_1) \cdot \ldots \cdot p(y_n|x_{1:n}, y_{1:n-1}) \tag{3.3.5}$$

Let $(p_\theta)_{\theta\in\Theta}$ be a deep learning model. We assume that we have a learning algorithm which computes, from any number of data samples $(x_{1:k}, y_{1:k})$, a trained parameter vector $\hat{\theta}(x_{1:k}, y_{1:k})$. Then the data is encoded in an incremental way: at each step k, $\hat{\theta}(x_{1:k}, y_{1:k})$ is used to predict y_{k+1}.

In practice, the learning procedure $\hat{\theta}$ may only reset and retrain the network at certain timesteps. We choose timesteps $1 = t_0 < t_1 < ... < t_S = n$, and we encode the data by blocks, always using the model learned from the already transmitted data (Algorithm 2 in Appendix 3.D). A uniform encoding is used for the first few points. (Even though the encoding procedure is called "online", it does not mean that only the most recent sample is used to update the parameter $\hat{\theta}$: the optimization procedure $\hat{\theta}$ can be any predefined technique using all the previous samples $(x_{1:k}, y_{1:k})$, only requiring that the algorithm has an explicit stopping criterion.) This yields the following description length:

Definition 3.4 (Prequential code)**.** Given a model p_θ, a learning algorithm $\hat{\theta}(x_{1:k}, y_{1:k})$, and retraining timesteps $1 = t_0 < t_1 < ... < t_S = n$, the *prequential* codelength is

$$L^{\text{preq}}(y_{1:n}|x_{1:n}) = t_1 \log_2 K + \sum_{s=0}^{S-1} -\log_2 p_{\hat{\theta}_{t_s}}(y_{t_s+1:t_{s+1}}|x_{t_s+1:t_{s+1}}) \tag{3.3.6}$$

where for each s, $\hat{\theta}_{t_s} = \hat{\theta}(x_{1:t_s}, y_{1:t_s})$ is the parameter learned on data samples 1 to t_s.

The model parameters are never encoded explicitly in this method. The difference between the prequential codelength $L^{\text{preq}}(y_{1:n}|x_{1:n})$ and the log-loss $\sum_{t=1}^{n} -\log_2 p_{\hat{\theta}_{t_K}}(y_t|x_t)$ of the final trained model, can be interpreted as the amount of information that the trained parameters contain about the data contained: the former is the data codelength if Bob does not know the parameters, while the latter is the codelength of the same data knowing the parameters.

Prequential codes depend on the performance of the underlying training algorithm, and take advantage of the model's generalization ability from the previous data to the next. In particular, the model training should yield good generalization performance from data $[1; t_s]$ to data $[t_s + 1; t_{s+1}]$.

In practice, optimization procedures for neural networks may be stochastic (initial values, dropout, data augmentation...), and Alice and Bob need to make all the same random actions in order to get the same final model. A possibility is to agree on a random seed ω (or pseudorandom numbers) beforehand, so that the random optimization procedure $\hat{\theta}(x_{1:t_s}, y_{1:t_s})$ is deterministic given ω, Hyperparameters may also be transmitted first (the cost of sending a few numbers is small).

Prequential coding with deep models provides excellent compression bounds. On MNIST, we computed the description length of the labels with different networks (Appendix 3.D). The best compression bound was given by a convolutional network of depth 8. It achieved a description length of 4.10 kbits, i.e., a compression ratio of 0.021, with 99.5% test set accuracy (Table 3.1). This codelength is 6 times smaller than the variational codelength.

On CIFAR, we tested a simple multilayer perceptron, a shallow network, a small convolutional network, and a VGG convolutional network (Simonyan and Zisserman, 2014) first without data augmentation or batch normalization (VGGa) (Ioffe and Szegedy, 2015), then with both of them (VGGb) (Appendix 3.D). The results are in Figure 3.2. The best compression bound was obtained with VGGb, achieving a codelength of 45.3 kbits, i.e., a compression ratio of 0.27, and 93% test set accuracy (Table 3.1). This codelength is twice smaller than the variational codelength. The difference between VGGa and VGGb also shows the impact of the training procedure on codelengths for a given architecture.

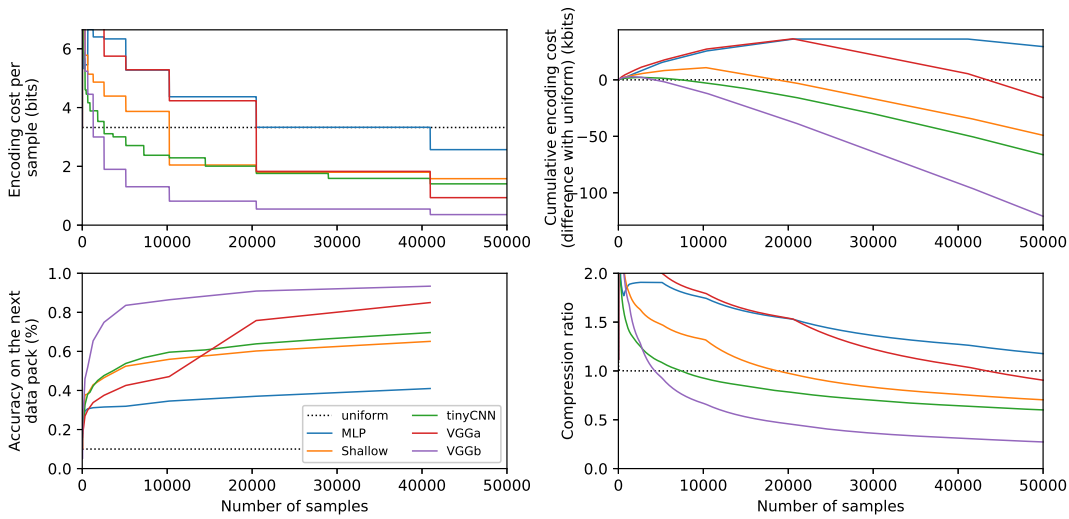


Figure 3.2: **Prequential code results on CIFAR.** Results of prequential encoding on CIFAR with 5 different models: a small Multilayer Perceptron (MLP), a shallow network, a small convolutional layer (tinyCNN), a VGG-like network without data augmentation and batch normalization (VGGa) and the same VGG-like architecture with data augmentation and batch normalization (VGGb) (see Appendix 3.D). Performance is reported during online training, as a function of the number of samples seen so far. Top left: codelength per sample (log loss) on a pack of data $[t_k; t_{k+1})$ given data $[1; t_k)$. Bottom left: test accuracy on a pack of data $[t_k; t_{k+1})$ given data $[1; t_k)$, as a function of t_k. Top right: difference between the prequential cumulated codelength on data $[1; t_k]$, and the uniform encoding. Bottom right: compression ratio of the prequential code on data $[1; t_k]$.

Model Switching. A weakness of prequential codes is the *catch-up phenomenon* (Van Erven et al., 2012). Large architectures might overfit during the first steps of the prequential encoding, when the model is trained with few data samples. Thus the encoding cost of the first packs of data might be worse than with the uniform code. Even after the encoding cost on current labels becomes lower, the cumulated codelength may need a lot of time to "catch up" on its initial lag. In Fig. 3.2, the VGGb model needs 5,000 samples on CIFAR to reach a cumulative compression ratio <1, even though the encoding cost per label becomes drops below uniform after just 1,000 samples. This is efficiently solved by *switching* (Van Erven et al., 2012) between models (see Appendix 3.E). Switching further improves the practical compression bounds (Fig. 3.3, Table 3.2).

3.4 Discussion

Too Many Parameters in Deep Learning Models? >From an information theory perspective, the goal of a model is to extract as much mutual information between the labels and inputs as possible—equivalently (Section 3.2.4), to compress the labels. This cannot be achieved with 2-part codes or practical network compression. With the variational code, the models do compress the data, but with a worse prediction performance: one could conclude that deep learning models that achieve the best prediction performance cannot compress the data.

Thanks to the prequential code, we have seen that deep learning models, even with a large number of parameters, compress the data well: from an information theory point of view, *the number of parameters is not an obstacle to compression.* This is consistent with Chaitin's hypothesis that "*comprehension is compression*", contrary to previous observations with the

variational code.

Prequential Code and Generalization. The prequential encoding shows that a model that generalizes well for every dataset size, will compress well. The efficiency of the prequential code is directly due to the generalization ability of the model at each time.

Theoretically, three of the codes (two-parts, Bayesian, and prequential based on a maximum likelihood or MAP estimator) are known to be asymptotically equivalent under strong assumptions (d-dimensional *identifiable* model, data coming from the model, suitable Bayesian prior, and technical assumptions ensuring the effective dimension of the trained model is not lower than d): in that case, these three methods yield a codelength $L(y_{1:n}|x_{1:n}) = nH(Y|X) + \frac{d}{2}\log_2 n + \mathcal{O}(1)$ (Grünwald, 2007). This corresponds to the BIC criterion for model selection. Hence there was no obvious reason for the prequential code to be an order of magnitude better than the others.

However, deep learning models do not usually satisfy *any* of these hypotheses. Moreover, our prequential codes are not based on the maximum likelihood estimator at each step, but on standard deep learning methods (so training is regularized at least by dropout and early stopping).

Inefficiency of Variational Models for Deep Networks. The objective of variational methods is equivalent to minimizing a description length. Thus, on our image classification tasks, variational methods do not have good results *even for their own objective*, compared to prequential codes. This makes their relatively poor results at test time less surprising.

Understanding this observed inefficiency of variational methods is an open problem. As stated in (3.3.4), the variational codelength is an upper bound for the Bayesian codelength. More precisely,

$$L^{\text{var}}_{\beta}(y_{1:n}|x_{1:n}) = L^{\text{Bayes}}(y_{1:n}|x_{1:n}) + \text{KL}\left(p_{\text{Bayes}}(\theta|x_{1:n}, y_{1:n})\|\beta\right) \tag{3.4.1}$$

with notation as above, and with $p_{\text{Bayes}}(\theta|x_{1:n}, y_{1:n})$ the Bayesian posterior on θ given the data. Empirically, on MNIST and CIFAR, we observe that $L^{\text{preq}}(y_{1:n}|x_{1:n}) \ll L^{\text{var}}_{\beta}(y_{1:n}|x_{1:n})$.

Several phenomena could contribute to this gap. First, the optimization of the parameters φ of the approximate Bayesian posterior might be imperfect. Second, even the optimal distribution β^* in the variational class might not approximate the posterior $p_{\text{Bayes}}(\theta|x_{1:n}, y_{1:n})$ well, leading to a large KL term in (3.4.1); this would be a problem with the choice of variational posterior class β. On the other hand we do not expect the choice of Bayesian prior to be a key factor: we tested Gaussian priors with various variances as well as a conjugate Gaussian prior, with similar results. Moreover, Gaussian initializations and L2 weight decay (acting like a Gaussian prior) are common in deep learning. Finally, the (untractable) Bayesian codelength based on the exact posterior might itself be larger than the prequential codelength. This would be a problem of underfitting with parametric Bayesian inference, perhaps related to the catch-up phenomenon or to the known conservatism of Bayesian model selection (end of Section 3.3.3).

3.5 Conclusion

Deep learning models can represent the data *together with the model* in fewer bits than a naive encoding, despite their many parameters. However, we were surprised to observe that variational inference, though explicitly designed to minimize such codelengths, provides very poor such values compared to a simple incremental coding scheme. Understanding this limitation of variational inference is a topic for future research.

Appendix

3.A Fake labels are not compressible

In the introduction, we stated that fake labels could not be compressed. This means that the optimal codelength for this labels is *almost* the uniform one. This can be formalized as follows. We define a *code* for $y_{1:n}$ as any program (in a reference Turing machine) that outputs $y_{1:n}$, and denote $L(y_{1:n})$ the length of this program, or $L(y_{1:n}|x_{1:n})$ for programs that may use $x_{1:n}$ as their input.

Proposition 3.5. *Assume that $x_1, ..., x_n$ are inputs, and that $Y_1, ..., Y_n$ are iid random labels uniformly sampled in $\{1, ..., K\}$. Then for any $\delta \in \mathbb{N}^*$, with probability $1 - 2^{-\delta}$ the values $Y_1, \dots, Y_n$ satisfy that for any possible coding procedure L (even depending on the values of $x_{1:n}$), the codelength of $Y_{1:n}$ is at least*

$$L(Y_{1:n}|x_{1:n}) \geqslant nH(Y) - \delta - 1 \tag{3.A.1}$$

$$= n \log_2 K - \delta - 1. \tag{3.A.2}$$

We insist that this does not require any assumptions on the coding procedure used, so this result holds for all possible models. Moreover, this is really a property of the sampled values $Y_1, \dots Y_n$: most values of $Y_{1:n}$ can just not be compressed by any algorithm.

Proof. This proposition is a standard counting argument, or an immediate consequence of Theorem 2.2.1 in (Li and Vitányi, 2008). Let $\mathcal{A} = \{1, ..., K\}^n$ be the set of all possible outcomes for the sequence of random labels. We have $|\mathcal{A}| = K^n$. Let δ be an integer, $\delta \in \mathbb{N}^*$, we want to know how many elements in $\mathcal{A}$ can be encoded in less than $\log_2 |\mathcal{A}| - \delta$ bits. We consider, on a given Turing machine, the number of programs of length less than $\lfloor \log_2 |\mathcal{A}| - \delta \rfloor$. This number is less than :

$$\sum_{i=0}^{\lfloor \log_2 |\mathcal{A}| \rfloor - \delta - 1} 2^i = 2^{\lfloor \log_2 |\mathcal{A}| \rfloor - \delta} - 1 \tag{3.A.3}$$

$$\leqslant 2^{-\delta}|\mathcal{A}| - 1 \tag{3.A.4}$$

Therefore, the number of elements in $\mathcal{A}$ which can be described in less than $\log_2 |\mathcal{A}| - \delta$ bits is less than $2^{-\delta}|\mathcal{A}| - 1$. We can deduce from this that the number of elements in $\mathcal{A}$ which cannot be described by *any* program in less than $2^{-\delta}|\mathcal{A}| - 1$ bits is at least $|\mathcal{A}|(1 - 2^{-\delta})$. Equivalently, there are at least $|\mathcal{A}|(1 - 2^{-\delta})$ elements $(y_1, ..., y_n)$ in $|\mathcal{A}|$ such that for any coding scheme, $L(y_{1:n}|x_{1:n}) \geqslant n \log_2 K - \delta - 1$. Since the random labels $Y_1, ..., Y_n$ are uniformly distributed, the result follows.

3.B Technical details on compression bounds with random affine subspaces

We describe in Algorithm 1 the detailed procedure which allows to compute compression bounds with the random affine subspace method (Li et al., 2018). To compute the numerical results in Table 3.1, we took the *intrinsic dimension* computed in the original paper, and considered that the precision of the parameter was 32 bits, following the authors' suggestion. Then, the description length of the model itself is 32× the intrinsic dimension. This does not take into account the description length of the labels given the model, which is non-negligible (to take this quantity into account, we would need to know the loss on the training set of the model, which was not specified in the original paper). Thus we only get a lower bound.

For MNIST, the model with the smaller intrinsic dimension is the LeNet, which has an intrinsic dimension of 290 for an accuracy of 90% (the threshold at which (Li et al., 2018) stop by definition, hence the performance in Table 3.1). This leads to a description length for the model of 9280 bits, which corresponds to a 0.05 compression ratio, without taking into account the description length of the labels given the model.

For CIFAR, again with the LeNet architecture, the intrinsic dimension is 2,900. This leads to a description length for the model of 92800 bits, which corresponds to a 0.05 compression ratio, without taking into account the description length of the labels given the model.

Algorithm 1 Encoding with random affine subspaces

Alice transmits a parametric model $(p_\theta)_{\theta\in\Theta}$.
Alice transmits the random seed ω (if using stochastic optimization), and a dimension k.
Alice and Bob both sample a random affine subspace $\tilde{\Theta} \subset \Theta$, with the seed ω. This means that they sample θ_0 and a matrix W of dimension $k \times d$ where d is the dimension of Θ. It defines a new parametric model $\tilde{p}_\varphi = p_{\theta_0 + W\cdot\varphi}$
Alice optimizes the parameter φ^* with a gradient descent algoritm in order to minimize $-\log_2 \tilde{p}_\varphi(y_{1:n}|x_{1:n})$.
Alice sends φ^* with a precision ε to Bob. It costs $k \times \log_2 \varepsilon$.
Alice sends the labels $y_{1:n}$ with the models $\tilde{p}_{\varphi^*}$. It costs $-\log_2 \tilde{p}_{\varphi^*}(y_{1:n}|x_{1:n})$

These bounds could be improved by optimizing the precision ε. Indeed, reducing the precision makes the model less accurate and increases the encoding cost of the labels with the model, but it decreases the encoding cost of the parameters. Therefore, we could find an optimal precision ε^* to improve the compression bound. This would be a topic for future work.

3.C Technical Details on Variational Learning for Section 3.3.3

Variational learning was performed using the library Pyvarinf (Tallec and Blier, 2018).

We used a prior $\alpha = \mathcal{N}(0, \sigma_0^2 I_d)$ with $\sigma_0 = 0.05$, chosen to optimize the compression bounds.

The chosen class of posterior was the class of multivariate gaussian distributions with diagonal covariance matrix $\{\mathcal{N}(\mu, \Sigma)\ ,\ \mu \in \mathbb{R}^d\ \Sigma \text{ diagonal}\}$. It was parametrized by $(\beta_{\mu,\rho})_{(\mu,\rho)\in\mathbb{R}^d\times\mathbb{R}^d}$, with $\sigma \in \mathbb{R}^d$ defined as $\sigma_i = \log(1 + \exp(\rho_i))$, and the covariance matrix Σ as the diagonal matrix with diagonal values $\sigma_1^2, ..., \sigma_d^2$.

We optimize the bound (3.3.2) as a function of (μ, ρ) with a gradient descent method, and estimate its values and gradient with a Monte-Carlo method. Since the prior and posteriors are gaussian, we have an explicit formula for the first part of the variational loss $\mathrm{KL}(\beta_{\mu,\rho}\|\alpha)$ (Hinton and Van Camp, 1993). Therefore, we can easily compute its values and gradients. For the second part

$$(\mu, \rho) \to \mathbb{E}_{\theta\sim\beta_{\mu,\rho}}\Big[\sum_{i=1}^{n} -\log_2 p_\theta(y_i|x_i)\Big], \tag{3.C.1}$$

we can use the following proposition (Graves, 2011). For any function $f\colon \Theta \to \mathbb{R}$, we have

$$\frac{\partial}{\partial\mu_i}\mathbb{E}_{\theta\sim\beta_{\mu,\rho}}[f(\theta)] = \mathbb{E}_{\theta\sim\beta_{\mu,\rho}}\Big[\frac{\partial f}{\partial\theta_i}(\theta)\Big] \tag{3.C.2}$$

$$\frac{\partial}{\partial\rho_i}\mathbb{E}_{\theta\sim\beta_{\mu,\rho}}[f(\theta)] = \frac{\partial\sigma_i}{\partial\rho_i}\cdot\mathbb{E}_{\theta\sim\beta_{\mu,\rho}}\Big[\frac{\partial f}{\partial\theta_i}\cdot\frac{\theta_i-\mu_i}{\sigma_i}\Big] \tag{3.C.3}$$

Therefore, we can estimate the values and gradients of (3.3.2) with a Monte-Carlo algorithm:

$$\frac{\partial}{\partial\mu_i}\mathbb{E}_{\theta\sim\beta_{\mu,\rho}}[f(\theta)] \approx \sum_{s=1}^{S}\frac{\partial f}{\partial\theta_i}(\theta^s) \tag{3.C.4}$$

$$\frac{\partial}{\partial\rho_i}\mathbb{E}_{\theta\sim\beta_{\mu,\rho}}[f(\theta)] \approx \frac{\partial\sigma_i}{\partial\rho_i}\cdot\sum_{s=1}^{S}\frac{\partial f}{\partial\theta_i}(\theta^s)\cdot\frac{\theta_i^s-\mu_i}{\sigma_i} \tag{3.C.5}$$

where $\theta^1, ..., \theta^S$ are sampled from $\beta_{\mu,\rho}$. In practice, we used $S = 1$ both for the computations of the variational loss and its gradients.

We used both convolutional and fully connected architectures, but in our experiments fully connected models were better for compression. For CIFAR and MNIST, we used fully connected networks with two hidden layers of width 256, trained with SGD, with a 0.005 learning rate and mini-batchs of size 128.

For CIFAR and MNIST, we used a LeNet-like network with 2 convolutional layers with 6 and 16 filters, both with kernels of size 5 and 3 fully connected layers. Each convolutional is followed by a ReLU activation and a max-pooling layer. The code will be publicly available. The first and the second fully connected layers are of dimension 120 and 84 and are followed by ReLU activations. The last one is followed by a softmax activation layer. The code for all models will be publicly available.

During the test phase, we sampled parameters $\hat{\theta}$ from the learned distribution β, and used the model $p_{\hat{\theta}}$ for prediction. This explains why our test accuracy on MNIST is lower than other numerical results (Blundell et al., 2015), since they use for prediction the averaged model with parameters $\hat{\theta} = \mathbb{E}_{\theta\sim\beta_{m,r}}[\theta] = \mu$. But our goal was not to get the best prediction score, but to evaluate the model which was used for compression on the test set.

Algorithm 2 Prequential encoding

Input: data $x_{1:n}, y_{1:n}$, timesteps $1 = t_0 < t_1 < ... < t_S = n$
Alice transmits the random seed ω (if using stochastic optimization).
Alice encodes $y_{1:t_1}$ with the uniform code. This costs $t_1 \log_2 K$ bits. Bob decodes $y_{1:t_1}$.
for $s = 1$ **to** $S - 1$ **do**
  Alice and Bob both compute $\hat{\theta}_s = \hat{\theta}(x_{1:t_s}, y_{1:t_s}, \omega)$.
  Alice encodes $y_{t_s+1:t_{s+1}}$ with model $p_{\hat{\theta}_s}$. This costs $-\log_2 p_{\hat{\theta}_s}(y_{t_s+1:t_{s+1}}|x_{t_s+1:t_{s+1}})$ bits
  Bob decodes $y_{t_s+1:t_{s+1}}$
end for

3.D Technical details on prequential learning

Prequential Learning on MNIST. On MNIST, we used three different models:

1. The uniform probability over the labels.
2. A fully connected network or Multilayer Perceptron (MLP) with two hidden layers of dimension 256.
3. A VGG-like convolutional network with 8 convolutional layers with 32, 32, 64, 64, 128, 128, 256 and 256 filters respectively and max pooling operators every two convolutional layers, followed by two fully connected layers of size 256.

For the two neural networks we used Dropout with probability 0.5 between the fully connected layers, and optimized the network with the Adam algorithm with learning rate 0.001.

The successive timestep for the prequential learning $t_1, t_2, ..., t_s$ are 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 and 32768.

For the prequential code results in Table 3.1, we selected the best model, which was the VGG-like network.

Prequential Learning on CIFAR. On CIFAR, we used five different models:

1. The uniform probability over the labels.
2. A fully connected network or Multilayer Perceptron (MLP) with two hidden layers of dimension 512.
3. A shallow network, with one hidden layer and width 5000.
4. A convolutional network (tinyCNN) with four convolutional layers with 32 filters, and a maxpooling operator after every two convolutional layers. Then, two fully connected layers of dimension 256. We used Dropout with probability 0.5 between the fully connected layers.
5. A VGG-like network with 13 convolutional layers from (Zagoruyko, 2015). We trained this architecture with two learning procedures. The first one (VGGa) without batch-normalization and data augmentation, and the second one (VGGb) with both of them, as introduced in (Zagoruyko, 2015). In both of them, we used dropout regularization with parameter 0.5.

We optimized the network with the Adam algorithm with learning rate 0.001.

For prequential learning, the timesteps $t_1, t_2, ..., t_s$ were: 10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480, 40960. The training results can be seen in Figure 3.2.

For the prequential code, all the results are in Figure 3.2. For the results in Table 3.1, we selected the best model for the prequential code, which was VGGb.

3.E Switching between models against the *catch-up phenomenon*

3.E.1 Switching between model classes

The solution introduced by (Van Erven et al., 2012) against the catch-up phenomenon described in Section 3.3.4, is to *switch* between models, to always encode a data block with the best model at that point. That way, the encoding adapts itself to the number of data samples seen. The switching pattern itself has to be encoded.

Assume that Alice and Bob have agreed on a set of prediction strategies $\mathcal{M} = \{p^k, k \in \mathcal{I}\}$. We define the set of switch sequences, $\mathbb{S} = \{((t_1, k_1), ..., (t_L, k_L)), 1 = t_1 < t_2 < ... < t_L \ , \ k \in \mathcal{I}\}$.

Let $s = ((t_1, k_1), ..., (t_L, k_L))$ be a switch sequence. The associated prediction strategy $p_s(y_{1:n}|x_{1:n})$ uses model p^{k_i} on the time interval $[t_i; t_{i+1})$, namely

$$p_s(y_{1:i+1}|x_{1:i+1}, y_{1:i}) = p^{K_i}(y_{i+1}|x_{1:i+1}, y_{1:i}) \tag{3.E.1}$$

where K_i is such that $K_i = k_l$ for $t_l \leqslant i < t_{l+1}$. Fix a prior distribution π over switching sequences (see (Van Erven et al., 2012) for typical examples).

Table 3.2: **Compression bounds by switching between models.** Compression bounds given by different codes on two datasets, MNIST and CIFAR10. The *Codelength* is the number of bits necessary to send the labels to someone who already has the inputs. This codelength *includes* the description length of the model. The *compression ratio* for a given code is the ratio between its codelength and the codelength of the uniform code. The *test accuracy* of a model is the accuracy of its predictions on the test set. For variational and prequential codes, we selected the model and hyperparameters providing the best compression bound.

Code	MNIST			CIFAR10		
	Codelength (kbits)	Comp. Ratio	Test Acc	Codelength (kbits)	Comp. Ratio	Test Acc
Uniform	199	1.	10%	166	1.	10%
Variational	24.1	0.12	95.5%	89.0	0.54	61,6%
Prequential	**4.10**	**0.02**	**99.5%**	45.3	0.27	**93.3%**
Switch	**4.05**	**0.02**	**99.5%**	**34.6**	**0.21**	**93.3%**
Self-Switch	**4.05**	**0.02**	**99.5%**	**34.9**	**0.21**	**93.3%**

Definition 3.6 (Switch code)**.** Assume that Alice and Bob have agreed on a set of prediction strategies $\mathcal{M}$ and a prior π over $\mathbb{S}$. The *switch code* first encodes a switch sequence s strategy, then uses the prequential code with this strategy:

$$L_s^{\mathrm{sw}}(y_{1:n}, x_{1:n}) = L_\pi(s) + L_{p_s}^{\mathrm{preq}}(y_{1:n}, x_{1:n}) = -\log_2 \pi(s) - \sum_{i=1}^{n} \log_2 p^{K_i}(y_i | x_{1:i}, y_{1:i-1}) \tag{3.E.2}$$

where K_i is the model used by switch sequence s at time i.

We then choose the switching strategy s^* wich minimizes $L_s^{\mathrm{sw}}(y_{1:n}, x_{1:n})$. We tested switching between the uniform model, a small convolutional network (tinyCNN), and a VGG-like network with two training methods (VGGa, VGGb) (Appendix 3.D). On MNIST, switching between models does not make much difference. On CIFAR10, switching by taking the best model on each interval $[t_k; t_{k+1})$ saves more than 11 kbits, reaching a codelength of 34.6 kbits, and a compression ratio of 0.21. The cost $L_\pi(s)$ of encoding the switch s is negligible (see Table 3.2).

3.E.2 Self-Switch: Switching between variants of a model or hyperparameters

With standard switch, it may be cumbersome to work with different models in parallel. Instead, for models learned by gradient descent, we may use the same architecture but with different parameter values corresponding obtained at different gradient descent stopping times. This is a form of regularization via early stopping.

Let $(p_\theta)_{\theta \in \Theta}$ be a model class. Let $\hat{\theta}_j(x_{1:k}, y_{1:k})$ be the parameter obtained by some optimization procedure after j epochs of training on data $[1; k]$. For instance, $j = 0$ would correspond to using an untrained model (usually close to the uniform model).

We call *self-switch code* the switch code obtained by switching among the family of models with different gradient descent stopping times j (based on the same parametric family $(p_\theta)_{\theta \in \Theta}$). In practice, this means that at each step of the prequential encoding, after having seen data $[1; t_k)$, we train the model on those data and record, at each epoch j, the loss obtained on data $[t_k; t_{k+1})$. We then switch optimally between those. We incur the small additional cost of encoding the best number of epochs to be used (which was limited to 10) at each step.

The catch-up phenomenon and the beneficial effect of the self-switch code can be seen in Figure 3.3.

The self-switch code achieves similar compression bounds to the switch code, while storing only one network. On MNIST, there is no observable difference. On CIFAR, self-switch is only 300 bits (0.006 bit/label) worse than full 4-architecture switch.

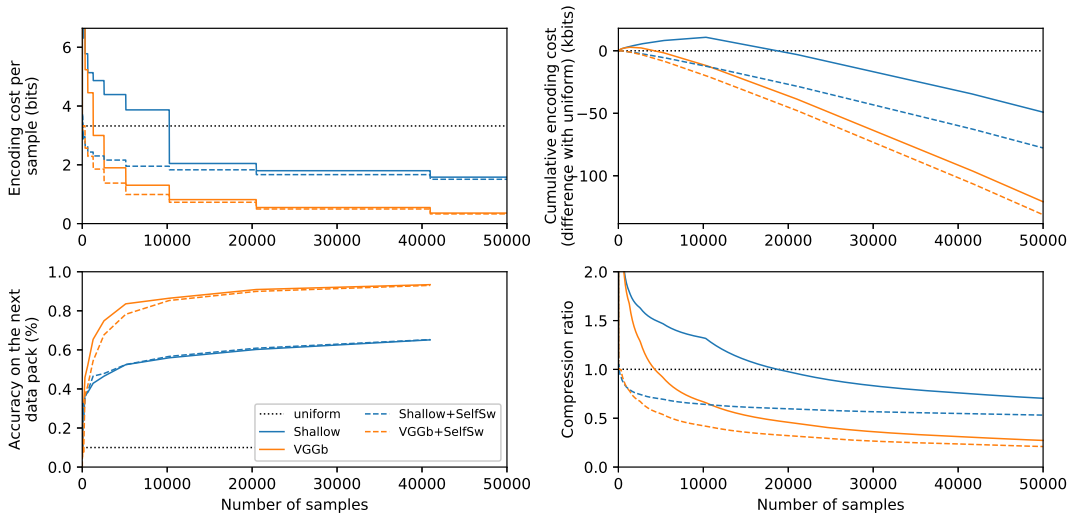


Figure 3.3: **Compression with the self-switch method:** Results of the self-switch code on CIFAR with 2 different models: the shallow network, and the VGG-like network trained with data augmentation and batch normalization (VGGb). Performance is reported during online training, as a function of the number of samples seen so far. Top: test accuracy on a pack of data $[t_k; t_{k+1})$ given data $[1; t_k)$, as a function of t_k. Second: codelength per sample (log loss) on a pack of data $[t_k; t_{k+1})$ given data $[1; t_k)$. Third: difference between the prequential cumulated codelength on data $[1; t_k]$, and the uniform encoding. Bottom: compression ratio of the prequential code on data $[1; t_k]$. The catch-up phenomenon is clearly visible for both models: even if models with and without the self-switch have similar performances after a training on the entire dataset, the standard model has lower performances than the uniform model (for the 1280 first labels for the VGGb network, and for the 10,000 first labels for the shallow network), and the code length for these first labels is large. The self-switch method solves this problem.

Part III

Mathematical approaches towards robustness in Deep Reinforcement Learning

Chapter 4

Learning with All Learning Rates At Once

In this chapter, we present the published paper:

> Blier, L., Wolinski, P., and Ollivier, Y. (2019). Learning with Random Learning Rates. In *ECML PKDD 2019 - European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*

This project started from discussions on how to design a learning system able both to adapt very quickly to new observed patterns, but also to learn complex patterns which might require a slow learning. I was starting to work in Reinforcement Learning at that time. In RL, the learning setting is *non-stationnary*: the distribution of observations can change while the agent is improving its policy, and the learning methods has to be able to adapt to all of these regimes. My co-author Pierre Wolinski was interested in topics in AutoML (Guyon et al., 2016). From this viewpoint, a learning system has to be able to adapt to multiple learning settings without any hyperparameter tuning, such as settings which would require small or large learning rates. Many methods were designed to directly set optimal per-parameter learning rates (Tieleman and Hinton, 2012; Kingma and Ba, 2015), such as the popular Adam optimizer, but they still require some hyperparameter tuning. This discussion lead to the idea of a learning system which would be a mixture of *slow learning units*, and *fast learning units*, by learning with *all learning rates at once*.

4.1 Introduction

Deep learning models require delicate hyperparameter tuning (Zoph and Le, 2016): when facing new data or new model architectures, finding a configuration that enables fast learning requires both expert knowledge and extensive testing. This prevents deep learning models from working out-of-the-box on new problems without human intervention (AutoML setup, Guyon et al. (2016)). One of the most critical hyperparameters is the learning rate of the gradient descent (Theodoridis, 2015, p. 892). With too large learning rates, the model does not learn; with too small learning rates, optimization is slow and can lead to local minima and poor generalization (Jastrzebski et al., 2017; Kurita, 2018; Mack, 2016; Surmenok, 2017).

Efficient methods with no learning rate tuning are a necessary step towards more robust learning algorithms, ideally working out of the box. Many methods were designed to directly set optimal per-parameter learning rates (Tieleman and Hinton, 2012; Duchi et al., 2011; Kingma and Ba, 2015; Schaul et al., 2013; LeCun et al., 1998b), such as the popular Adam optimizer. The latter comes with default hyperparameters which reach good performance on many problems and architectures; yet fine-tuning and scheduling of its learning rate is still frequently needed (Denkowski and Neubig, 2017), and the default setting is specific to current

problems and architecture sizes. Indeed Adam's default hyperparameters fail in some natural setups (Section 4.6.2). This makes it unfit in an out-of-the-box scenario.

We propose *All Learning Rates At Once* (Alrao), a gradient descent method for deep learning models that leverages redundancy in the network. Alrao uses multiple learning rates at the same time in the same network, spread across several orders of magnitude. This creates a mixture of slow and fast learning units. Alrao departs from the usual philosophy of trying to find the "right" learning rates; instead we take advantage of the overparameterization of network-based models to produce a diversity of behaviors from which good network outputs can be built. The width of the architecture may optionally be increased to get enough units within a suitable learning rate range, but surprisingly, performance was largely satisfying even without increasing width.

Our contributions are as follows:

- We introduce Alrao, a gradient descent method for deep learning models with no learning rate tuning, leveraging redundancy in deep learning models via a range of learning rates in the same network. Surprisingly, Alrao does manage to learn well over a range of problems from image classification, text prediction, and reinforcement learning.
- In our tests, Alrao's performance is always close to that of SGD with the optimal learning rate, without any tuning.
- Alrao combines performance with *robustness*: not a single run failed to learn with the default learning rate range we used. In contrast, our parameter-free baseline, Adam with default hyperparameters, is not reliable across the board.
- Alrao vindicates the role of redundancy in deep learning: having enough units with a suitable learning rate is sufficient for learning.

Acknowledgments. We would like to thank Corentin Tallec for his technical help and extensive remarks. We thank Olivier Teytaud for pointing useful references, Hervé Jégou for advice on the text, and Léon Bottou, Guillaume Charpiat, and Michèle Sebag for their remarks on our ideas.

4.2 Related Work

Redundancy in deep learning. Alrao specifically exploits the redundancy of units in network-like models. Several lines of work underline the importance of such redundancy in deep learning. For instance, dropout (Srivastava et al., 2014) relies on redundancy between units. Similarly, many units can be pruned after training without affecting accuracy (LeCun et al., 1990; Han et al., 2015a,b; See et al., 2016). Wider networks have been found to make training easier (Bengio et al., 2006; Hinton et al., 2015; Zhang et al., 2017a), even if not all units are useful a posteriori.

The *lottery ticket hypothesis* (Frankle and Carbin, 2018; Frankle et al., 2019) posits that "large networks that train successfully contain subnetworks that—when trained in isolation—converge in a comparable number of iterations to comparable accuracy'. This subnetwork is the *lottery ticket winner*: the one which had the best initial values. In this view, redundancy helps because a larger network has a larger probability to contain a suitable subnetwork. Alrao extends this principle to the learning rate.

Learning rate tuning. Automatically using the "right" learning rate for each parameter was one motivation behind "adaptive" methods such as RMSProp (Tieleman and Hinton, 2012), AdaGrad (Duchi et al., 2011) or Adam (Kingma and Ba, 2015). Adam with its default setting is currently considered the default method in many works (Wilson et al., 2017). However, further global adjustment of the Adam learning rate is common (Liu et al., 2018a). Other heuristics for setting the learning rate have been proposed (Schaul et al., 2013); these heuristics often start

with the idea of approximating a second-order Newton step to define an optimal learning rate (LeCun et al., 1998b). Indeed, asymptotically, an arguably optimal preconditioner is either the Hessian of the loss (Newton method) or the Fisher information matrix (Amari, 1998). Another approach is to perform gradient descent on the learning rate itself through the whole training procedure (Jacobs, 1988; Schraudolph, 1999; Mahmood et al., 2012; Maclaurin et al., 2015; Massé and Ollivier, 2015; Baydin et al., 2018). Despite being around since the 80's (Jacobs, 1988), this has not been widely adopted, because of sensitivity to hyperparameters such as the meta-learning rate or the initial learning rate (Erraqabi and Le Roux, 2018). Of all these methods, Adam is probably the most widespread at present (Wilson et al., 2017), and we use it as a baseline.

The learning rate can also be optimized within the framework of architecture or hyperparameter search, using methods from from reinforcement learning (Zoph and Le, 2016; Baker et al., 2016; Li et al., 2017), evolutionary algorithms (Stanley and Miikkulainen, 2002; Jozefowicz et al., 2015; Real et al., 2017), Bayesian optimization (Bergstra et al., 2013), or differentiable architecture search (Liu et al., 2018b). Such methods are resource-intensive and do not allow for finding a good learning rate in a single run.

4.3 Motivation and Outline

We first introduce the general ideas behind Alrao. The detailed algorithm is explained in Section 4.4 and in Algorithm 3. We also release a Pytorch (Paszke et al., 2017) implementation, including tutorials: http://github.com/leonardblier/alrao.

Different learning rates for different units. Instead of using a single learning rate for the model, Alrao samples once and for all a learning rate for each *unit* in the network. These rates are taken from a log-uniform distribution in an interval $[\eta_{\min}; \eta_{\max}]$. The log-uniform distribution produces learning rates spread over several order of magnitudes, mimicking the log-uniform grids used in standard grid searches on the learning rate.

A *unit* corresponds for example to a feature or neuron for fully connected networks, or to a channel for convolutional networks. Thus we build "slow-learning" and "fast-learning" units. In contrast, with per-parameter learning rates, every unit would have a few incoming weights with very large learning rates, and possibly diverge.

Intuition. Alrao is inspired by the fact that not all units in a neural network end up being useful. Our idea is that in a large enough network with learning rates sampled randomly per unit, a sub-network made of units with a good learning rate will learn well, while the units with a wrong learning rate will produce useless values and just be ignored by the rest of the network. Units with too small learning rates will not learn anything and stay close to their initial values; this does not hurt training (indeed, even leaving some weights at their initial values, corresponding to a learning rate 0, does not hurt training [Appendix 4.G]). Units with a too large learning rate may produce large activation values, but those will be mitigated by subsequent normalizing mechanisms in the computational graph, such as sigmoid/tanh activations or BatchNorm.

Alrao can be interpreted within the *lottery ticket hypothesis* (Frankle and Carbin, 2018): viewing the per-unit learning rates of Alrao as part of the initialization, this hypothesis suggests that in a wide enough network, there will be a sub-network whose initialization (both values and learning rate) leads to good convergence.

Slow and fast learning units for the output layer. Sampling a learning rate per unit at random in the last layer would not make sense. For classification, each unit in the last layer represents a single category: using different learning rates for these units would favor some categories during learning. Moreover for scalar regression tasks there is only one output unit, thus we would be back to selecting a single learning rate.

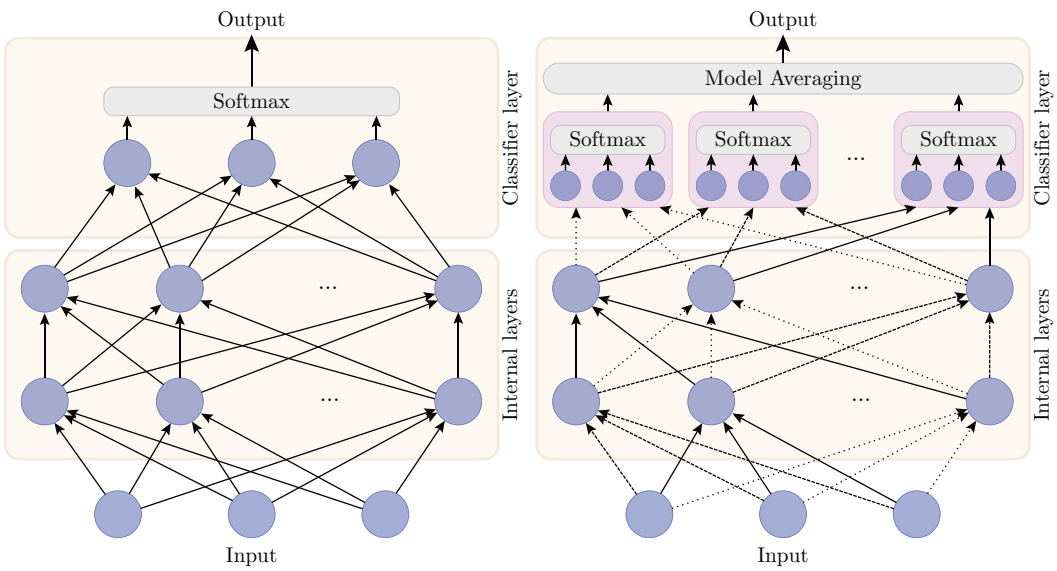


Figure 4.1: Left: a standard fully connected neural network for a classification task with three classes, made of several internal layers and an output layer. Right: Alrao version of the same network. The single classifier layer is replaced with a set of parallel copies of the original classifier, averaged with a model averaging method. Each unit uses its own learning rate for its incoming weights (represented by different styles of arrows).

The simplest way to obtain the best of several learning rates for the last layer, without relying on heuristics to guess an optimal value, is to use *model averaging* over several copies of the output layer (Fig. 4.1), each copy trained with its own learning rate from the interval Appendix 4.B contains a proof (under convexity assumptions) that this mechanism works. All these untied copies of the output layer share the same Alrao internal layers (Fig. 4.1). This can be seen as a smooth form of model selection or grid-search over the output layer learning rate; actually, this part of the architecture can even be dropped after a few epochs (Appendix 4.C), as the model averaging quickly concentrates on one model.

Increasing network width. With Alrao, neurons with unsuitable learning rates will not learn: those with too large learning rates might learn no useful signal, while those with too small learning rates will learn too slowly. Thus, Alrao may reduce the *effective width* of the network to only a fraction of the actual architecture width, depending on $[\eta_{\min}; \eta_{\max}]$. This may be compensated by multiplying the width of the network by a factor γ. Our first intuition was that $\gamma > 1$ would be necessary; still Alrao turns out to work well even without width augmentation.

4.4 All Learning Rates At Once: Description

4.4.1 Notation

We now describe Alrao more precisely for deep learning models with softmax output, on classification tasks; the case of regression is similar.

Let $\mathcal{D} = \{(x_1, y_1), ..., (x_N, y_N)\}$, with $y_i \in \{1, ..., K\}$, be a classification dataset. The goal is to predict the y_i given the x_i, using a deep learning model Φ_θ. For each input x, $\Phi_\theta(x)$ is a probability distribution over $\{1, ..., K\}$, and we want to minimize the categorical cross-entropy loss ℓ over the dataset: $\frac{1}{N}\sum_i \ell(\Phi_\theta(x_i), y_i)$.

We denote $\log\text{-}\mathcal{U}(\cdot;\eta_{\min},\eta_{\max})$ the *log-uniform* probability distribution on an interval $[\eta_{\min};\eta_{\max}]$. Namely, if $\eta \sim \log\text{-}\mathcal{U}(\cdot;\eta_{\min},\eta_{\max})$, then $\log\eta$ is uniformly distributed between $\log\eta_{\min}$ and $\log\eta_{\max}$. Its density function is $\log\text{-}\mathcal{U}(\eta;\eta_{\min},\eta_{\max}) = \frac{1}{\eta}\,\frac{\mathbb{1}_{\eta_{\min}\leqslant\eta\leqslant\eta_{\max}}}{\log(\eta_{\max})-\log(\eta_{\min})}$.

Algorithm 3 Alrao-SGD for model $\Phi_\theta = C_{\theta^{\text{out}}} \circ \varphi_{\theta^{\text{r}}}$ with N_{out} classifiers and learning rates in $[\eta_{\min};\eta_{\max}]$

$a_j \leftarrow 1/N_{\text{out}}$ for each $1 \leqslant j \leqslant N_{\text{out}}$ {Initialize the N_{out} model averaging weights a_j}
$\Phi_\theta^{\text{Alrao}}(x) := \sum_{j=1}^{N_{\text{out}}} a_j\, C_{\theta_j^{\text{out}}}(\varphi_{\theta^{\text{int}}}(x))$ {Define the Alrao architecture}
for layers l, **for all** unit i in layer l **do**
  Sample $\eta_{l,i} \sim \log\text{-}\mathcal{U}(.;\eta_{\min},\eta_{\max})$. {Sample a learning rate for each unit}
end for
for Classifiers j, $1 \leqslant j \leqslant N_{\text{out}}$ **do**
  Define $\log\eta_j = \log\eta_{\min} + \frac{j-1}{N_{\text{out}}-1}\log\frac{\eta_{\max}}{\eta_{\min}}$. {Set a learning rate for each classifier}
end for
while Stopping criterion is False **do**
  $z_t \leftarrow \varphi_{\theta^{\text{int}}}(x_t)$ {Store the output of the last internal layer}
  for layers l, **for all** unit i in layer l **do**
    $\theta_{l,i} \leftarrow \theta_{l,i} - \eta_{l,i}\cdot\nabla_{\theta_{l,i}}\ell(\Phi_\theta^{\text{Alrao}}(x_t), y_t)$ {Update the repr. netw. weights}
  end for
  for Classifier j **do**
    $\theta_j^{\text{out}} \leftarrow \theta_j^{\text{out}} - \eta_j\cdot\nabla_{\theta_j^{\text{out}}}\ell(C_{\theta_j^{\text{out}}}(z_t), y_t)$ {Update the classifiers' weights}
  end for
  $a \leftarrow \texttt{ModelAveraging}(a, (C_{\theta_i^{\text{out}}}(z_t))_i, y_t)$ {Update the model averaging weights.}
  $t \leftarrow t+1 \bmod N$
end while

4.4.2 Alrao Architecture

Multiple Alrao output layers. A deep learning model Φ_θ for classification can be decomposed into two parts: first, *internal layers* compute some function $z = \varphi_{\theta^{\text{int}}}(x)$ of the inputs x, fed to a final *output (classifier) layer* $C_{\theta^{\text{out}}}$, so that the overall network output is $\Phi_\theta(x) := C_{\theta^{\text{out}}}(\varphi_{\theta^{\text{int}}}(x))$. For a classification task with K categories, the output layer $C_{\theta^{\text{out}}}$ is defined by $C_{\theta^{\text{out}}}(z) := \text{softmax}\circ\left(W^T z + b\right)$ with $\theta^{\text{out}} := (W, b)$, and $\text{softmax}(u_1,...,u_K)_k := e^{u_k}/\left(\sum_i e^{u_i}\right)$.

In Alrao, we build multiple copies of the original output layer, with different learning rates for each, and then use a model averaging method among them. The averaged classifier and the overall Alrao model are:

$$C_{\theta^{\text{out}}}^{\text{Alrao}}(z) := \sum_{j=1}^{N_{\text{out}}} a_j\, C_{\theta_j^{\text{out}}}(z), \qquad \Phi_\theta^{\text{Alrao}}(x) := C_{\theta^{\text{out}}}^{\text{Alrao}}(\varphi_{\theta^{\text{int}}}(x)) \tag{4.4.1}$$

where the $C_{\theta_j^{\text{out}}}$ are copies of the original classifier layer, with non-tied parameters, and $\theta^{\text{out}} := (\theta_1^{\text{out}},...,\theta_{N_{\text{out}}}^{\text{out}})$. The a_j are the parameters of the model averaging, with $0 \leqslant a_j \leqslant 1$ and $\sum_j a_j = 1$. The a_j are not updated by gradient descent, but via a model averaging method from the literature (see below).

Increasing the width of internal layers. As explained in Section 4.3, we may compensate the effective width reduction in Alrao by multiplying the width of the network by a factor γ. This means multiplying the number of units (or filters for a convolutional layer) of all internal layers by γ.

4.4.3 Alrao Update for the Internal Layers: A Random Learning Rate for Each Unit

In the internal layers, for each unit i in each layer l, a learning rate $\eta_{l,i}$ is sampled from the probability distribution log-$\mathcal{U}(.; \eta_{\min}, \eta_{\max})$, once and for all at the beginning of training. [1]

The incoming parameters of each unit in the internal layers are updated in the usual SGD way, only with per-unit learning rates (Eq. 4.4.2): for each unit i in each layer l, its incoming parameters are updated as:

$$\theta_{l,i} \leftarrow \theta_{l,i} - \eta_{l,i} \cdot \nabla_{\theta_{l,i}} \ell(\Phi_\theta^{\text{Alrao}}(x), y) \tag{4.4.2}$$

where $\Phi_\theta^{\text{Alrao}}$ is the Alrao loss (4.4.1) defined above.

What constitutes a *unit* depends on the type of layers in the model. In a fully connected layer, each component of a layer is considered as a unit for Alrao: all incoming weights of the same unit share the same Alrao learning rate. On the other hand, in a convolutional layer we consider each convolution filter as constituting a unit: there is one learning rate per filter (or channel), thus preserving translation-invariance over the input image. In LSTMs, we apply the same learning rate to all components in each LSTM cell (thus the vector of learning rates is the same for input gates, for forget gates, etc.).

We set a learning rate *per unit*, rather than per parameter. Otherwise, every unit would have some parameters with large learning rates, and we would expect even a few large incoming weights to be able to derail a unit. Having diverging parameters within every unit is hurtful, while having diverging units in a layer is not necessarily hurtful since the next layer can learn to disregard them.

4.4.4 Alrao Update for the Output Layer: Model Averaging from Output Layers Trained with Different Learning Rates

Learning the output layers. The j-th copy $C_{\theta_j^{\text{out}}}$ of the classifier layer is attributed a learning rate η_j defined by $\log \eta_j := \log \eta_{\min} + \frac{j-1}{N_{\text{out}}-1} \log\left(\frac{\eta_{\max}}{\eta_{\min}}\right)$, so that the classifiers' learning rates are log-uniformly spread on the interval $[\eta_{\min}; \eta_{\max}]$. Then the parameters θ_j^{out} of each classifier j are updated as if this classifier alone was the only output of the model:

$$\theta_j^{\text{out}} \leftarrow \theta_j^{\text{out}} - \eta_j \cdot \nabla_{\theta_j^{\text{out}}} \ell(C_{\theta_j^{\text{out}}}(\varphi_{\theta^{\text{int}}}(x)), y), \tag{4.4.3}$$

(still sharing the same internal layers $\varphi_{\theta^{\text{int}}}$). This ensures that classifiers with low weights a_j still learn, and is consistent with model averaging philosophy. Algorithmically this requires differentiating the loss N_{out} times with respect to the last layer, but no additional backpropagations through the internal layers.

Model averaging. To set the weights a_j, several model averaging techniques are available, such as Bayesian Model Averaging (Wasserman, 2000). We use the *Switch* model averaging (Van Erven et al., 2012), a Bayesian method which is both simple, principled, and very responsive to changes in performance of the various models. After each mini-batch, the switch computes a modified posterior distribution (a_j) over the classifiers. This computation is directly taken from (Van Erven et al., 2012) and explained in Appendix 4.A.

Additional experiments in Appendix 4.C show that the model averaging method acts like a smooth model selection procedure: after only a few hundreds gradient steps, a single output layer is selected, with its parameter a_j very close to 1. Actually, Alrao's performance is unchanged if the extraneous output layer copies are thrown away when the posterior weight a_j of one of the copies gets close to 1.

[1] With learning rates resampled at each time, each step would be, in expectation, an ordinary SGD step with learning rate $\mathbb{E}\eta_{l,i}$, thus just yielding an ordinary SGD trajectory with more variance.

Table 4.1: Performance of Alrao, SGD with tuned learning rate, and Adam with its default setting. Three convolutional models are reported for image classification on CIFAR10, three others for ImageNet, one recurrent model for character prediction (Penn Treebank), and two experiments on RL problems. Four of the image classification architectures are further tested with a width multiplication factor $\gamma = 3$. Alrao learning rates are taken in a wide, a priori reasonable interval $[\eta_{\min}; \eta_{\max}] = [10^{-5}; 10]$, and the optimal learning rate for SGD is chosen in the set $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1., 10.\}$. Each experiment is run 10 times (CIFAR10 and RL), 5 times (PTB) or 1 time (ImageNet); the confidence intervals report the standard deviation over these runs. For RL tasks, the return has to be maximized, not minimized.

Model	SGD with optimal LR			Adam - Default		Alrao	
	LR	Loss	Top1 (%)	Loss	Top1 (%)	Loss	Top1 (%)
CIFAR10							
MobileNet	0.1	$0.37 \pm .01$	$90.2 \pm .3$	$1.01 \pm .95$	78 ± 11	$0.42 \pm .02$	$88.1 \pm .6$
MobileNet, $\gamma = 3$	0.1	$0.33 \pm .01$	$90.3 \pm .5$	$0.32 \pm .02$	$90.8 \pm .4$	$0.35 \pm .01$	$89.0 \pm .6$
GoogLeNet	0.01	$0.45 \pm .05$	$89.6 \pm 1.$	$0.47 \pm .04$	$89.8 \pm .4$	$0.47 \pm .03$	$88.9 \pm .8$
GoogLeNet, $\gamma = 3$	0.1	$0.34 \pm .02$	$90.5 \pm .8$	$0.41 \pm .02$	$88.6 \pm .6$	$0.37 \pm .01$	$89.8 \pm .8$
VGG19	0.1	$0.42 \pm .02$	$89.5 \pm .2$	$0.43 \pm .02$	$88.9 \pm .4$	$0.45 \pm .03$	$87.5 \pm .4$
VGG19, $\gamma = 3$	0.1	$0.35 \pm .01$	$90.0 \pm .6$	$0.37 \pm .01$	$89.5 \pm .8$	$0.381 \pm .004$	$88.4 \pm .7$
ImageNet							
AlexNet	0.01	2.15	53.2	6.91	0.10	2.56	43.2
Densenet121	1	1.35	69.7	1.39	67.9	1.41	67.3
ResNet50	1	1.49	67.4	1.39	67.1	1.42	67.5
ResNet50, $\gamma = 3$	-	-	-	1.99	60.8	1.33	70.9
Penn Treebank							
LSTM	1	$1.566 \pm .003$	$66.1 \pm .1$	$1.587 \pm .005$	$65.6 \pm .1$	$1.706 \pm .004$	$63.4 \pm .1$
RL		Return		Return		Return	
Pendulum	0.0001	-372 ± 24		-414 ± 64		-371 ± 36	
LunarLander	0.1	188 ± 23		155 ± 23		186 ± 45	

4.5 Experimental Setup

We tested Alrao on various convolutional networks for image classification (Imagenet and CIFAR10), on LSTMs for text prediction, and on reinforcement learning problems. We always use the same learning rate interval $[10^{-5}; 10]$, corresponding to the values we would have tested in a grid search, and 10 Alrao output layer copies, for every task.

We compare Alrao to SGD with an optimal learning rate selected in the set $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1., 10.\}$, and, as a tuning-free baseline, to Adam with its default setting ($\eta = 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999$), arguably the current default method (Wilson et al., 2017).

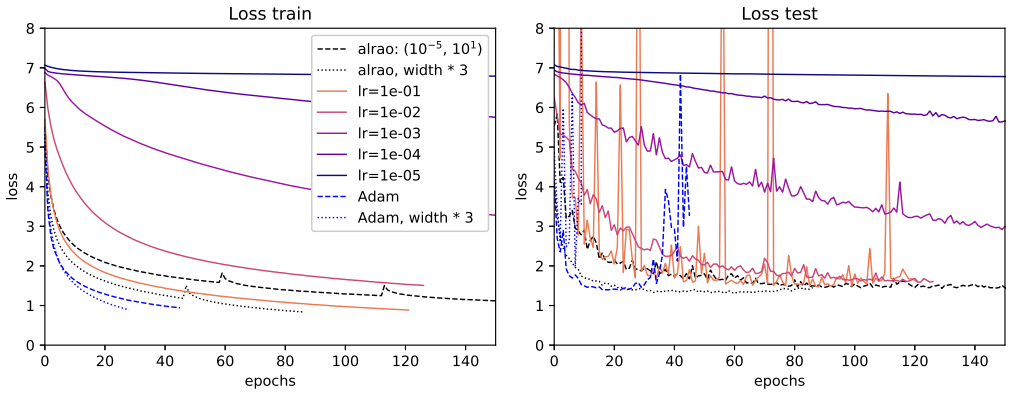
The results are presented in Table 4.1. Fig. 4.2 presents learning curves for AlexNet and Resnet50 on ImageNet, with additional curves in Appendix 4.D.

4.5.1 Image Classification on ImageNet and CIFAR10

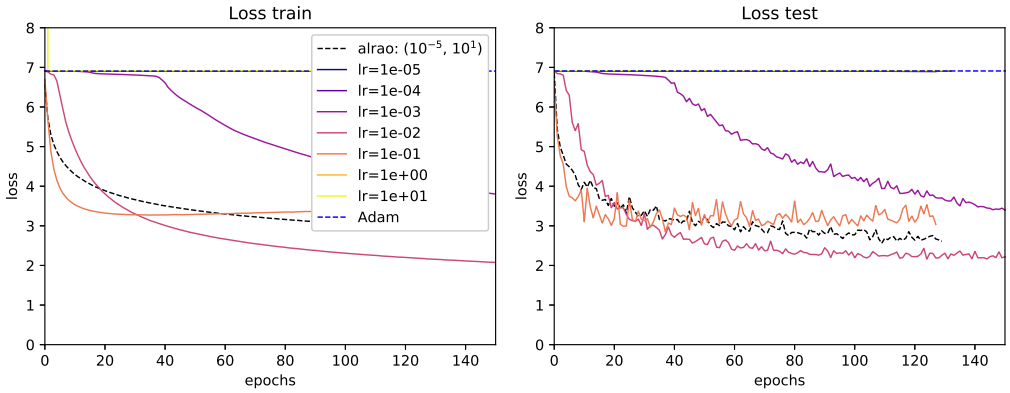
For image classification, we used the ImageNet (Deng et al., 2009) and CIFAR10 (Krizhevsky, 2009) datasets. The ImageNet dataset is made of 1,283,166 training and 60,000 testing data; we split the training set into a smaller training set and a validation set with 60,000 samples. We do the same on CIFAR10: the 50,000 training samples are split into 40,000 training samples and 10,000 validation samples.

For each architecture, training was stopped when the validation loss had not improved for 20 epochs. The epoch with best validation loss was selected and the corresponding model tested on the test set. The inputs are normalized, and training used data augmentation: random cropping and random horizontal flipping (see Appendix 4.D for details). For CIFAR10, each setting was run 10 times: the confidence intervals presented are the standard deviation over these runs. For ImageNet, because of high computation time, we performed only a single run per experiment.

We tested Alrao on several standard architectures. On ImageNet, we tested Resnet50 (He et al., 2016), Densenet121 (Huang et al., 2017), and Alexnet (Krizhevsky, 2014), using the



(a) Resnet50 trained on ImageNet



(b) AlexNet trained on ImageNet

Figure 4.2: Learning curves for Alrao, SGD with various learning rates, and Adam with its default setting, on ImageNet. Left: training loss; right: test loss. Curves are interrupted by the early stopping criterion. Alrao's performance is comparable to the optimal SGD learning rate.

default Pytorch implementation. On CIFAR10, we tested GoogLeNet (Szegedy et al., 2015), VGG19 (Simonyan and Zisserman, 2014), and MobileNet (Howard et al., 2017), as implemented in (Kianglu, 2018). We also tested wider architectures, with a width multiplication factor $\gamma = 3$. On the largest model, Resnet50 on ImageNet with triple width, systematic SGD learning rate grid search was not performed due to the excessive computational burden, hence the omitted value in Tab. 4.1.

4.5.2 Other Tasks: Text Prediction, Reinforcement Learning

Text prediction on Penn TreeBank. To test Alrao on other kinds of tasks, we first used a recurrent neural network for text prediction on the Penn Treebank (PTB) (Marcus et al., 1993) dataset. The Alrao experimental procedure is the same as above.

The loss in Table 4.1 is given in bits per character and the accuracy is the proportion of correct character predictions. The model is a two-layer LSTM (Hochreiter and Schmidhuber, 1997) with an embedding size of 100, and 100 hidden units. A dropout layer with rate 0.2 is included before the decoder. The training set is divided into 20 minibatchs. Gradients are computed via truncated backprop through time (Werbos, 1990) with truncation every 70 characters.

The model was trained for *character* prediction rather than word prediction. This is technically easier for Alrao implementation: since Alrao uses copies of the output layer, memory

issues arise for models with most parameters on the output layer. Word prediction (10,000 classes on PTB) requires many more output parameters than character prediction; see Section 4.7 and Appendix 4.F.

Reinforcement learning tasks. Next, we tested Alrao on two standard reinforcement learning problems: the Pendulum and Lunar Lander environments from OpenAI Gym (Brockman et al., 2016). We use standard deep Q-learning (Mnih et al., 2015). The Q-network is a standard MLP with 2 hidden layers. The experimental setting is the same as above, with regressors instead of classifiers on the output layer. More details on the Q-learning implementation are given in Appendix 4.D. For each environment, we select the best epoch on validation runs, and then report the return of the selected model on new test runs in that environment.

4.6 Performance and Robustness of Alrao

4.6.1 Alrao Compared to SGD with Optimal Learning Rate

First, Alrao does manage to learn; this was not obvious a priori.

Second, SGD with an optimally tuned learning rate usually performs better than Alrao. This can be expected when comparing a tuning-free method with a method that tunes the hyperparameter in hindsight.

Still, the difference between Alrao and optimally-tuned SGD is reasonably small across every setup, even with wide intervals $[\eta_{\min}; \eta_{\max}]$, with a somewhat larger gap in one case (AlexNet on ImageNet). Notably, this occurs even though SGD achieves good performance only for a few learning rates within the interval $[\eta_{\min}; \eta_{\max}]$. With $\eta_{\min} = 10^{-5}$ and $\eta_{\max} = 10$, among the 7 SGD learning rates tested ($10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1$, and 10), only three are able to learn with AlexNet, and only one is better than Alrao (Fig. 4.2b); with ResNet50, only three are able to learn well, and only two of them achieve performance similar to Alrao (Fig. 4.2a); on the Pendulum environment, only two are able to learn well, only one of which converges as fast as Alrao (Fig. 4.7d in Appendix 4.D). Appendix 4.D contains more examples.

Thus, surprisingly, Alrao manages to learn at a nearly optimal rate, even though most units in the network have learning rates unsuited for SGD.

4.6.2 Robustness of Alrao, and Comparison to Default Adam

Overall, Alrao learns reliably in every setup in Table 4.1. Moreover, this is quite stable over the course of learning: Alrao curves shadow optimal SGD curves over time (Fig. 4.2).

Often, Adam with its default parameters almost matches optimal SGD, but this is not always the case. Over the 13 setups in Table 4.1, default Adam gives a significantly poor performance in three cases. One of those is a pure optimization issue: with AlexNet on ImageNet, optimization does not start with the default parameters (Fig. 4.2b). The other two cases are due to strong overfit despite good train performance: MobileNet on CIFAR (Fig. 4.7c in Appendix 4.D) and ResNet with increased width on ImageNet (Fig. 4.2a).

In two further cases, Adam achieves good validation performance in Table 4.1, but actually overfits shortly after its peak score: ResNet (Fig. 4.2a) and DenseNet (Fig. 4.7b in Appendix 4.D), both on ImageNet. On the whole, this confirms a known risk of overfit with Adam (Wilson et al., 2017; Keskar and Socher, 2017).

Overall, default Adam tends to give slightly better results than Alrao when it works, but does not learn reliably with its default hyperparameters. It can exhibit two kinds of lack of robustness: optimization failure, and overfit or non-robustness over the course of learning. On the other hand, every single run of Alrao reached reasonably close-to-optimal performance. Alrao also performs steadily over the course of learning (Fig. 4.2, and Fig. 4.7b in Appendix 4.D).

4.6.3 Sensitivity Study to test $[\eta_{\min}; \eta_{\max}]$

We claim to remove a hyperparameter, the learning rate, but replace it with two hyperparameters $\eta_{\min}$ and $\eta_{\max}$. Formally, this is true. But a systematic study of the impact of these two hyperparameters (Fig. 4.3) shows that the sensitivity to $\eta_{\min}$ and $\eta_{\max}$ is much lower than the original sensitivity to the learning rate.

To assess this, we tested every combination of $\eta_{\min}$ and $\eta_{\max}$ in a grid from 10^{-9} to 10^{7} on GoogLeNet for CIFAR10 (left plot in Fig. 4.3, with SGD on the diagonal). The largest satisfactory learning rate for SGD is 1 (diagonal on Fig. 4.3). Unsurprisingly, if all the learning rates in Alrao are too large, or all too small, then Alrao fails (rightmost and leftmost zones in Fig. 4.3). Extremely large learning rates diverge numerically, both for SGD and Alrao.

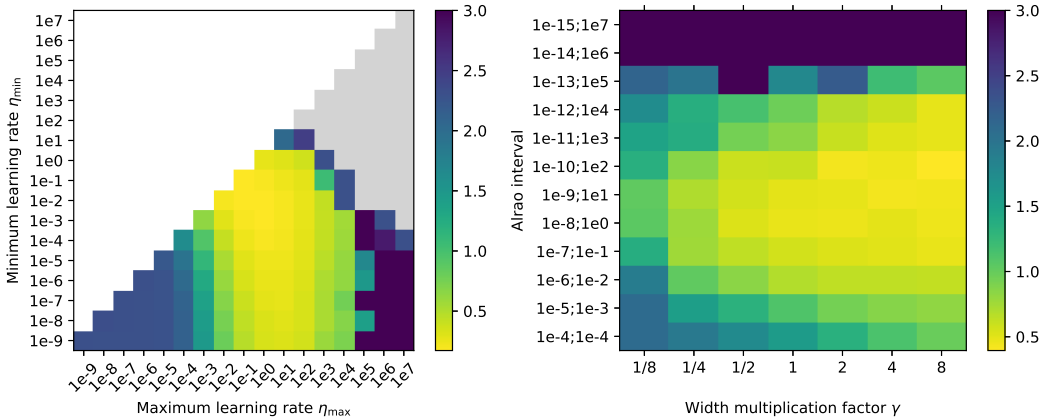


Figure 4.3: Influence of $[\eta_{\min}; \eta_{\max}]$ and of network width on Alrao performance, with GoogLeNet on CIFAR10. Results are reported after 15 epochs, and averaged on three runs. Left plot: each point with coordinates $[\eta_{\min}; \eta_{\max}]$ below the diagonal represents the loss for Alrao with this interval. Points (η, η) on the diagonal represent standard SGD with learning rate η. Grey squares represent numerical divergence (NaN). Alrao works as soon as $[\eta_{\min}; \eta_{\max}]$ contains at least one suitable learning rate. Right plot: varying network width.

On the other hand, Alrao converges as soon as $[\eta_{\min}; \eta_{\max}]$ contains a reasonable learning rate (central zone Fig. 4.3), even with values of $\eta_{\max}$ for which SGD fails. A wide range of choices for $[\eta_{\min}; \eta_{\max}]$ will contain one good learning rate and achieve close-to-optimal performance. Thus, as a general rule, we recommend to just use an interval containing all the learning rates that would have been tested in a grid search, e.g., 10^{-5} to 10.

For a fixed network size, one might expect Alrao to perform worse with large intervals $[\eta_{\min}; \eta_{\max}]$, as most units would become useless. On the other hand, in a larger network, many units would have extreme learning rates, which might disturb learning. We tested how increasing or decreasing network width changes Alrao's sensitivity to $[\eta_{\min}; \eta_{\max}]$ (right plot of Fig. 4.3 for Alrao, and Fig. 4.5 in Appendix 4.D for SGD). The sensitivity of Alrao to $[\eta_{\min}; \eta_{\max}]$ decreases markedly with network width. For instance, a wide interval $[\eta_{\min}; \eta_{\max}] = [10^{-12}; 10^{4}]$ works reasonably well with an 8-fold network, even though most units receive unsuitable learning rates.

So, even if the choice of $\eta_{\min}$ and $\eta_{\max}$ is important, the results are much more stable to varying these two hyperparameters than to the original learning rate, especially with large networks.

4.7 Discussion, Limitations, and Perspectives

Alrao specifically exploits redundancy between units in deep learning models, relying on the overall network approach of combining a large number of units built for diversity of behavior.

Alrao would not make sense in a classical convex optimization setting. That Alrao works at all is already informative about some phenomena at play in deep neural networks.

Alrao can make lengthy SGD learning rate sweeps unnecessary on large models, such as the triple-width ResNet50 for ImageNet above. Incidentally, in our experiments, wider networks provided increased performance both for SGD and Alrao (Table 4.1, Fig. 4.3 and Fig. 4.5): network size is still a limiting factor for the models used, independently of the algorithm.

Increased number of parameters for the classification layer. Since Alrao modifies the output layer of the optimized model, the number of parameters in the classification layer is multiplied by the number of classifier copies. (The number of parameters in the internal layers is unchanged.) This is a limitation for models with most parameters in the classifier layer.

On CIFAR10 (10 classes), the number of parameters increases by less than 5% for the models used. On ImageNet (1000 classes), it increases by 50–100% depending on the architecture. On Penn Treebank, the number of parameters increased by 26% in our setup (at character level); working at word level it would have increased fivefold (Appendix 4.F).

This can be mitigated by handling the copies of the classifiers on distinct computing units: in Alrao these copies work in parallel given the internal layers. Moreover, the additional output layer copies may be thrown away early in training (Appendix 4.C). Finally, models with a large number of output classes usually rely on other parameterizations than a direct softmax, such as a hierarchical softmax (see references in (Jozefowicz et al., 2016)); Alrao can be used in conjunction with such methods.

Multiple output layer copies and expressiveness. Using several copies of the output layer in Alrao formally provides more expressiveness to the model, as it creates a larger architecture with more parameters. We performed two control experiments to check that Alrao's performance does not just stem from this. First, we performed ablation of the output layer copies in Alrao after one epoch, only keeping the copy with the highest model averaging weight a_i: the learning curves are identical. Second, we trained default Adam using copies of the output layer (all with the same Adam default learning rate): the learning curves are identical to Adam on the unmodified architecture. The details are in Appendix 4.C. Thus, the copies of the output layer do not bring any useful added expressiveness.

Learning rate schedules, other optimizers, other hyperparameters... Learning rate schedules are often effective (Bengio, 2012). We did not use them here: this may partially explain why the results in Table 4.1 are worse than the state-of-the-art. One might have hoped that the diversity of learning rates in Alrao would effortlessly bring it to par with step size schedules, but the results above do not support this. Still, nothing prevents using a scheduler together with Alrao, e.g., by dividing all Alrao learning rates by a time-dependent constant.

The Alrao idea can also be used with other optimizers than SGD, such as Adam. We tested combining Alrao and Adam, and found the combination less reliable than standard Alrao (Appendix 4.E, Fig. 4.8): curves on the training set mostly look good, but the method quickly overfits (Fig. 4.8).

The Alrao idea could be used on other hyperparameters as well, such as momentum. However, with more hyperparameters initialized randomly for each unit, the fraction of units having suitable values for all their hyperparameters simultaneously will quickly decrease.

4.8 Conclusion

Applying stochastic gradient descent with multiple learning rates for different units is surprisingly resilient in our experiments, and provides performance close to SGD with an optimal learning rate, as soon as the range of random learning rates is not excessive. Alrao could save time when testing deep learning models, opening the door to more out-of-the-box uses of deep learning.

Appendix

4.A Model Averaging with the Switch

As explained is Section 4.4, we use a model averaging method on the classifiers of the output layer. We could have used the Bayesian Model Averaging method (Wasserman, 2000). But one of its main weaknesses is the *catch-up phenomenon* (Van Erven et al., 2012): plain Bayesian posteriors are slow to react when the relative performance of models changes over time. Typically, for instance, some larger-dimensional models need more training data to reach good performance: at the time they become better than lower-dimensional models for predicting current data, their Bayesian posterior is so bad that they are not used right away (their posterior needs to "catch up" on their bad initial performance). This leads to very conservative model averaging methods.

The solution from (Van Erven et al., 2012) against the catch-up phenomenon is to *switch* between models. It is based on previous methods for prediction with expert advice (see for instance (Herbster and Warmuth, 1998; Volf and Willems, 1998) and the references in (Koolen and De Rooij, 2008; Van Erven et al., 2012)), and is well rooted in information theory. The switch method maintains a Bayesian posterior distribution, not over the set of models, but over the set of *switching strategies* between models. Intuitively, the model selected can be adapted online to the number of samples seen.

We now give a quick overview of the switch method from (Van Erven et al., 2012): this is how the model averaging weights a_j are chosen in Alrao.

Assume that we have a set of prediction strategies $\mathcal{M} = \{p^j, j \in \mathcal{I}\}$. We define the set of *switch sequences*, $\mathbb{S} = \{((t_1, j_1), ..., (t_L, j_L)), 1 = t_1 < t_2 < ... < t_L \ , \ j \in \mathcal{I}\}$. Let $s = ((t_1, j_1), ..., (t_L, j_L))$ be a switch sequence. The associated prediction strategy $p_s(y_{1:n}|x_{1:n})$ uses model p^{j_i} on the time interval $[t_i; t_{i+1})$, namely

$$p_s(y_{1:i+1}|x_{1:i+1}, y_{1:i}) = p^{K_i}(y_{i+1}|x_{1:i+1}, y_{1:i}) \tag{4.A.1}$$

where K_i is such that $K_i = j_l$ for $t_l \leqslant i < t_{l+1}$. We fix a prior distribution π over switching sequences. In this work, $\mathcal{I} = \{1, ..., N_C\}$ the prior is, for a switch sequence $s = ((t_1, j_1), ..., (t_L, j_L))$:

$$\pi(s) = \pi_L(L)\pi_K(j_1) \prod_{i=2}^{L} \pi_T(t_i|t_i > t_{i-1})\pi_K(j_i) \tag{4.A.2}$$

with $\pi_L(L) = \frac{\theta^L}{1-\theta}$ a geometric distribution over the switch sequences lengths, $\pi_K(j) = \frac{1}{N_C}$ the uniform distribution over the models (here the classifiers) and $\pi_T(t) = \frac{1}{t(t+1)}$.

This defines a Bayesian mixture distribution:

$$p_{sw}(y_{1:T}|x_{1:T}) = \sum_{s \in \mathbb{S}} \pi(s) p_s(y_{1:T}|x_{1:T}) \tag{4.A.3}$$

Then, the model averaging weight a_j for the classifier j after seeing T samples is the posterior of the switch distribution: $\pi(K_{T+1} = j|y_{1:T}, x_{1:T})$.

$$\begin{aligned} a_j &= p_{sw}(K_{T+1} = j|y_{1:T}, x_{1:T}) \\ &= \frac{p_{sw}(y_{1:T}, K_{T+1} = j|x_{1:T})}{p_{sw}(y_{1:T}|x_{1:T})} \end{aligned}$$

These weights can be computed online exactly in a quick and simple way (Van Erven et al., 2012), thanks to dynamic programming methods from hidden Markov models.

The implementation of the switch used in Alrao exactly follows the pseudo-code from (Van Erven et al., 2008), with hyperparameter $\theta = 0.999$ (allowing for many switches a priori). It can be found in the accompanying online code.

4.B Convergence result in a simple case

We prove a convergence result on Alrao in a simplified case: we assume that the loss is convex, that the internal layers are fixed, that we work with full batch gradients rather than stochastic gradient descent, and that

the Alrao model averaging method is standard Bayesian model averaging. The convexity and fixed classifier assumptions cover, for instance, standard logistic regression: in that case the Alrao output layer contains copies of a logistic classifier with various learning rates, and Alrao's internal layers are the identity, (or any fixed linear map).

For each Alrao classifier j, for simplicity we denote its parameters by θ_j instead of θ_j^{cl} (there is no more ambiguity since the internal layers are fixed).

The loss of some classifier C on a dataset with features (x_i) and labels (y_i) is $L(C) := \frac{1}{N}\sum_i \ell(C(x_i), y_i)$, where for each input x_i, $(C(x_i)_y)_{y\in\mathcal{Y}}$ is a probability distribution over the possible labels $y \in \mathcal{Y}$, and we use the log-loss $\ell(C(x_i), y_i) := -\log C(x_i)_{y_i}$.

For a classifier C_θ with parameter θ, let us abbreviate $L(\theta) := L(C_\theta)$. We assume that $L(\theta)$ is a non-negative convex function, with $\nabla^2 L(\theta) \preceq \lambda I$ for all θ. Let L^* be its global infimum; we assume L^* is a minimum, reached at some point θ^*, namely $L(\theta^*) = L^*$. Moreover we assume that L is locally strongly convex at its minimum θ^*: $\nabla^2 L(\theta^*) \succ 0$.

The Alrao architecture for such a classifier C_θ uses N_{cl} copies of the same classifier, with different parameter values:

$$\Phi^{\text{Alrao}}_{\theta_{\text{Alrao}}}(x) = \sum_{j=1}^{N_{\text{cl}}} a_j C_{\theta_j} \tag{4.B.1}$$

where $\theta_{\text{Alrao}} := (\theta_1, ..., \theta_{N_{\text{cl}}})$, and where the $(a_j)_j$ are the weights given by the model averaging method. We abbreviate $L(\theta_{\text{Alrao}}) := L(\Phi^{\text{Alrao}}_{\theta_{\text{Alrao}}})$.

The Alrao classification layer uses a set of learning rates $(\eta_j)_{j\in J}$, and starting points $(\theta_j^{(0)})_{j\in J}$. Using full-batch (non stochastic) Alrao updates we have

$$\begin{aligned}
\theta_j^{(t+1)} &= \theta_j^{(t)} - \eta_j \nabla L(\theta_j^{(t)}) \\
a^{(t+1)} &= \texttt{ModelAveraging}(a^{(t)}, (C_{\theta_i}(x_{1:N}))_i, y_{1:N})
\end{aligned}$$

We assume that the model averaging method is Bayesian Model Averaging.

We have assumed that the Hessian of the loss of the model satisfies $\nabla^2 L(\theta) \preceq \lambda I$. Under this condition, the standard theory of gradient descent for convex functions requires that the learning rate be less than $1/\lambda$, otherwise the gradient descent might diverge. Therefore, for Alrao we assume that at least *one* of the learning rates considered by Alrao is below this threshold.

Theorem 4.1. *Assume that at least one of the Alrao learning rates η_j satisfies $\eta_j < 1/\lambda$, with λ as above. Then, under the assumptions above, the Alrao loss is at most the optimal loss when $t \to \infty$:*

$$\limsup_t L(\theta^{(t)}_{\text{Alrao}}) \leqslant L^* \tag{4.B.2}$$

Proof. Let us analyze the dynamics of the different models in the model averaging method. Let us split the set of Alrao classifiers in two categories according to whether their sum of errors is finite or infinite, namely,

$$A := \left\{ j \in J \text{ such that } \sum_{t\geqslant 0} \left(L(\theta_j^{(t)}) - L^* \right) < \infty \right\},$$

$$B := \left\{ j \in J \text{ such that } \sum_{t\geqslant 0} \left(L(\theta_j^{(t)}) - L^* \right) = \infty \right\}$$

and in particular, for any $j \in A$, $\lim_t L(\theta_j^{(t)}) = L^*$.

The proof is organized as follows: We first show that A is not empty. Then, we show that $\lim_{t\to\infty} a_j^{(t)} = 0$ for all $j \in B$: these models are eliminated by the model averaging method. Then we will be able to conclude.

First, we show that A is not empty: namely, that there is least one j such that $\sum_{t\geqslant 0}(L(\theta_j^{(t)}) - L^*) < \infty$. We know that there is j such that $\eta_j < \frac{2}{\lambda}$. Hence, the standard theory of gradient descent for convex functions shows that this particular classifier converges (e.g., (Tibshirani and Marchetti-Bowick, 2013)), namely, the loss $(L(\theta_j^{(t)}))_t$ converges to L^*. Moreover, since L is locally strictly convex around θ^*, this implies that $\lim_t \theta_j^{(t)} = \theta^*$.

We now show that the sum of errors for this specific j converges. We assumed that $L(\theta)$ is locally strongly convex in θ^*. Let $\mu > 0$ such that $\nabla^2 L(\theta^*) \succeq \mu I$. Since L is C^2, there is ε' such that for any θ such that $\|\theta - \theta^*\| \leqslant \varepsilon'$, then $\nabla^2 L(\theta) \succeq \frac{\mu}{2} I$. Let $\tau \in \mathbb{N}$ such that $\|\theta_j^{(\tau)} - \theta^*\| < \varepsilon'$. Then, from the theory of gradient descent for strongly convex functions (Tibshirani and Marchetti-Bowick, 2013), we know there is some $\gamma < 1$

such that for $t > \tau$, $L(\theta_j^{(t)}) - L^* \leqslant C\|\theta_j^{(\tau)} - \theta^*\|\gamma^t$. We have:

$$\begin{aligned}
&\sum_{s=1}^{t} \Big(L(\theta_j^{(s)}) - L^* \Big) = \\
&\quad = \sum_{s=1}^{\tau} \Big(L(\theta_j^{(s)}) - L^* \Big) + \sum_{s=\tau}^{t} \Big(L(\theta_j^{(s)}) - L^* \Big) \\
&\quad \leqslant \sum_{s=1}^{\tau} \Big(L(\theta_j^{(s)}) - L^* \Big) + C\|\theta_j^{(\tau)} - \theta^*\|\gamma^\tau \frac{1}{1-\gamma}
\end{aligned}$$

Thus $\sum_{t \geqslant 0} \Big(L(\theta_j^{(t)}) - L^* \Big) < \infty$. Therefore, A is not empty.

We now show that the weights $a_j^{(t)}$ tend to 0 for any $j \in B$, namely, $\lim_{t\to\infty} a_j^{(t)} = 0$. Let $j \in B$ and take some $i \in A$. In Bayesian model averaging, the weights are

$$\begin{aligned}
a_j^{(t)} &= \frac{\prod_{s=1}^{t} p_{\theta_j^{(s)}}(y_{1:N}|x_{1:N})}{\sum_k \prod_{s=1}^{t} p_{\theta_k^{(s)}}(y_{1:N}|x_{1:N})} \\
&\leqslant \prod_{s=1}^{t} \frac{p_{\theta_j^{(s)}}(y_{1:N}|x_{1:N})}{p_{\theta_i^{(s)}}(y_{1:N}|x_{1:N})} \\
&= \prod_{s=1}^{t} \frac{\exp(-NL(\theta_j^{(s)}))}{\exp(-NL(\theta_i^{(s)}))} \\
&= \exp\Big(- N \sum_{s=1}^{t} \Big(L(\theta_j^{(s)}) - L^* \Big) + \\
&N \sum_{s=1}^{t} \Big(L(\theta_i^{(s)}) - L^* \Big) \Big)
\end{aligned}$$

Since $i \in A$ and $j \in B$, by definition of A and B this tends to 0. Therefore, $\lim_t a_j^{(t)} = 0$ for all $j \in B$.

We now prove the statement of the theorem. We have:

$$\begin{aligned}
L(\theta_{\text{Alrao}}^{(t)}) &= \frac{1}{N} \sum_i - \log \Big(\sum_{j \in A} a_j e^{-\ell(C_{\theta_j^{(t)}}(x_i), y_i)} + \\
&\sum_{j \in B} a_j e^{-\ell(C_{\theta_j^{(t)}}(x_i), y_i)} \Big) \\
&\leqslant \frac{1}{N} \sum_i - \log \left(\sum_{j \in A} a_j e^{-\ell(C_{\theta_j^{(t)}}(x_i), y_i)} \right)
\end{aligned}$$

For all $i \in A$, set $\tilde{a}_i^{(t)} := \frac{a_i^{(t)}}{\sum_{j \in A} a_j^{(t)}} = \frac{a_i^{(t)}}{1 - \sum_{j \in B} a_j^{(t)}}$. Then

$$\begin{aligned}
L(\theta_{\text{Alrao}}^{(t)}) \leqslant & - \log \left(1 - \sum_{j \in B} a_j^{(t)} \right) \\
&+ \frac{1}{N} \sum_i - \log \left(\sum_{j \in A} \tilde{a}_j e^{-\ell(C_{\theta_j^{(t)}}(x_i), y_i)} \right) \\
&\leqslant \frac{1}{N} \sum_i \sum_{j \in A} \tilde{a}_j \ell(C_{\theta_j^{(t)}}(x_i), y_i) + o(1) \\
&= \sum_{j \in A} \tilde{a}_j L(\theta_j^{(t)}) + o(1) \\
&= L^* + o(1)
\end{aligned}$$

thanks to Jensen's inequality for $-\log$, then because $\lim_t a_j^{(t)} = 0$ for $j \in B$, and finally because $\lim_t L(\theta_j^{(t)}) = L^*$ for $j \in A$. Taking the lim sup, we have:

$$\limsup_t L(\theta_{\text{Alrao}}^{(t)}) \leqslant L^* \tag{4.B.3}$$

which ends the proof.

4.C Influence of Model Averaging in Alrao

We investigated the importance of the model averaging method in Alrao.

Evolution of the model averaging weights. In Figure 4.6a, we represent the evolution of the model averaging weights a_j during training of GoogLeNet with Alrao on CIFAR10. We can make several observations. First, after only a few gradient descent steps, the model averaging weights corresponding to the three classifiers with the largest learning rates go to practically zero. This means that their parameters are moving too fast, and their loss is getting very large. Next, for a short time, a classifier with a moderately large learning rate gets the largest posterior weight, presumably because it is the first to learn a useful model. Finally, after the model has seen approximately 4,000 samples, a classifier with a slightly smaller learning rate is assigned a posterior weight a_j close to 1, while all the others go to 0. Thus, after a number of gradient steps, the model averaging method acts like a model selection method.

Model selection instead of model averaging. This evolution of the model averaging weights suggests that the averaging in the last layer is acting as a model selection procedure. Once the weight a_j of some classifier j is close to 1, the output of the full Alrao architecture with model averaging is close to the output of the original architecture with a single classifier with weights θ_j^{out} from this classifier. We compared Alrao with a modified version of Alrao in which after 1 epoch, the classifier with largest model averaging weight is selected, and the other classifiers are dropped (Fig. 4.6b). The behaviors of these two variants are exactly the same.

Adam with the Alrao output layer. In order to control the effect of the increased expressiveness induced by the expanded output layer in the Alrao architecture, we ran Adam (with its default parameters) on GoogLeNet modified as in Alrao (namely, with model averaging over 10 classifiers) (Fig. 4.6c). The learning behavior is exactly the same as Adam on the original architecture. Thus, changing the architecture by replacing the single classifier layer with an average of classifiers does not by itself improve training performance.

4.D Additional Experimental Details and Results

In the case of CIFAR-10 and ImageNet, we normalize each input channel x_i ($1 \leqslant i \leqslant 3$), using its mean and its standard deviation over the training set. Let μ_i and σ_i be respectively the mean and the standard deviation of the i-th channel. Then each input (x_1, x_2, x_3) is transformed into $(\frac{x_1-\mu_1}{\sigma_1}, \frac{x_2-\mu_2}{\sigma_2}, \frac{x_3-\mu_3}{\sigma_3})$. This operation is done over all the data (training, validation and test).

Moreover, we use data augmentation: every time an image of the training set is sent as input of the NN, this image is randomly cropped and randomly flipped horizontally. Cropping consists in filling with black a band at the top, bottom, left and right of the image. The size of this band is randomly chosen between 0 and 4 in our experiments.

The batch size is: 32 on CIFAR10 for every architecture, 20 on PTB, and 256 on ImageNet for Alexnet and ResNet50, and 128 for Densenet121.

On Reinforcement Learning environments, we use vanilla Q-learning (Mnih et al., 2015) with a soft target update as in (Lillicrap et al., 2015) $\tau = 0.9$, and a memory buffer of size 1,000,000. The architecture for the Q network is a MLP with 2 hidden layers. The learning curves are in Fig. 4.7d. For the optimization, the switch is used with 10 output layers. An output layer is a linear layer. Since the switch is a probability model averaging method, we consider each output layer as a probabilistic model, defined as a Normal distribution with variance 1 and mean the predicted value by the output layer. The loss for the Alrao model is the negative log-likelihood of the model mixture.

4.E Alrao with Adam

In Figure 4.8, we report our experiments with Alrao-Adam on CIFAR10. As explained in Section 4.7, Alrao is much less reliable with Adam than with SGD.

This is especially true for the test performance, which can even diverge while training performance remains either good or acceptable (Fig. 4.8). Thus Alrao-Adam seems to send the model into atypical regions of the search space.

We have no definitive explanation for this at present. It might be that changing Adam's learning rate requires changing its momentum parameters accordingly. It might be that Alrao does not work on Adam because Adam is more sensitive to its hyperparameters.

4.F Number of Parameters

As explained in Section 4.7, Alrao increases the number of parameters of a model, due to output layer copies. The additional number of parameters is approximately equal to $(N_{\text{out}} - 1) \times K \times d$ where N_{out} is the number of classifier copies used in Alrao, d is the dimension of the input to the output layer, and K is the number of classes in the classification task (assuming a standard softmax output; classification with many classes often uses other kinds of output parameterization instead).

Table 4.2: Comparison between the number of parameters in models used without and with Alrao. LSTM (C) is a simple LSTM cell used for character prediction while LSTM (W) is the same cell used for word prediction.

Model	Number of parameters	
	Without Alrao	With Alrao
GoogLeNet	6.166M	6.258M
VGG	20.041M	20.087M
MobileNet	2.297M	2.412M
LSTM (C)	0.172M	0.217M
LSTM (W)	2.171M	11.261M

The number of parameters for the models used, with and without Alrao, are in Table 4.2. Using Alrao for classification tasks with many classes, such as word prediction (10,000 classes on PTB), increases the number of parameters noticeably.

For those model with significant parameter increase, the various classifier copies may be done on parallel GPUs.

4.G Frozen Features Do Not Hurt Training

As explained in the introduction, several works support the idea that not all units are useful when learning a deep learning model. Additional results supporting this hypothesis are presented in Figure 4.4. We trained a GoogLeNet architecture on CIFAR10 with standard SGD with learning rate η_0, but learned only a random fraction p of the features (chosen at startup), and kept the others at their initial value. This is equivalent to sampling each learning rate η from the probability distribution $P(\eta = \eta_0) = p$ and $P(\eta = 0) = 1 - p$.

We observe that even with a fraction of the weights not being learned, the model's performance is close to its performance when fully trained.

When training a model with Alrao, many features might not learn at all, due to too small learning rates. But Alrao is still able to reach good results. This could be explained by the resilience of neural networks to partial training.

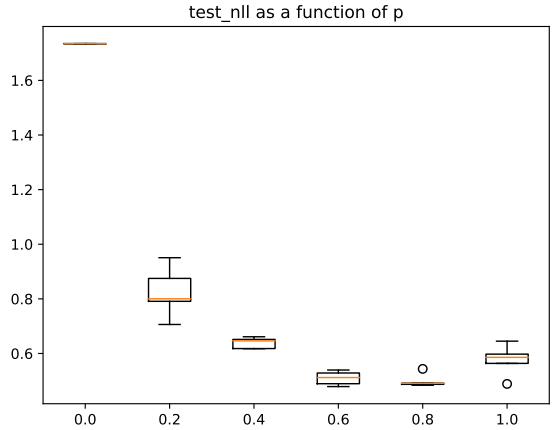


Figure 4.4: Loss of a model where only a random fraction p of the features are trained, and the others left at their initial value, as a function of p. The architecture is GoogLeNet, trained on CIFAR10.

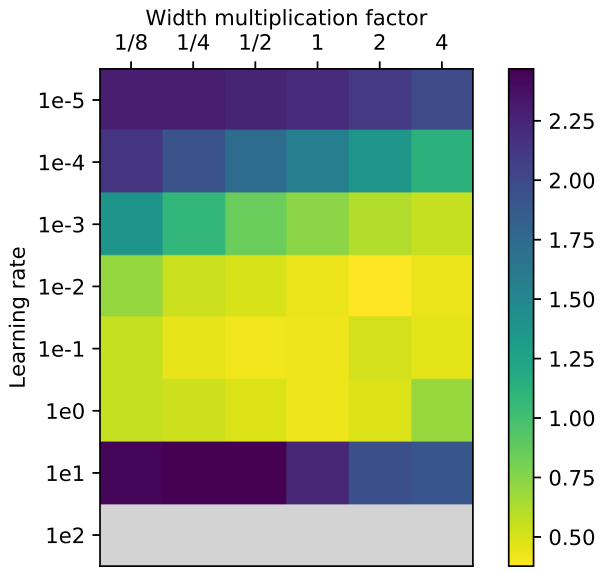
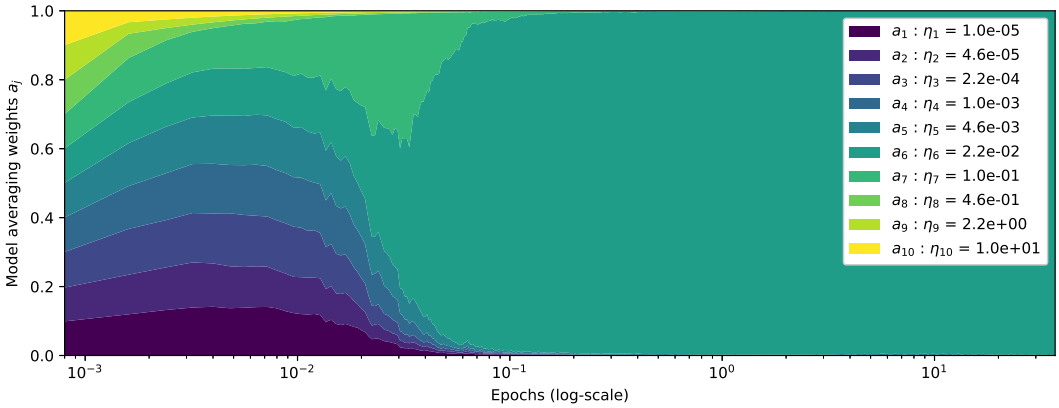
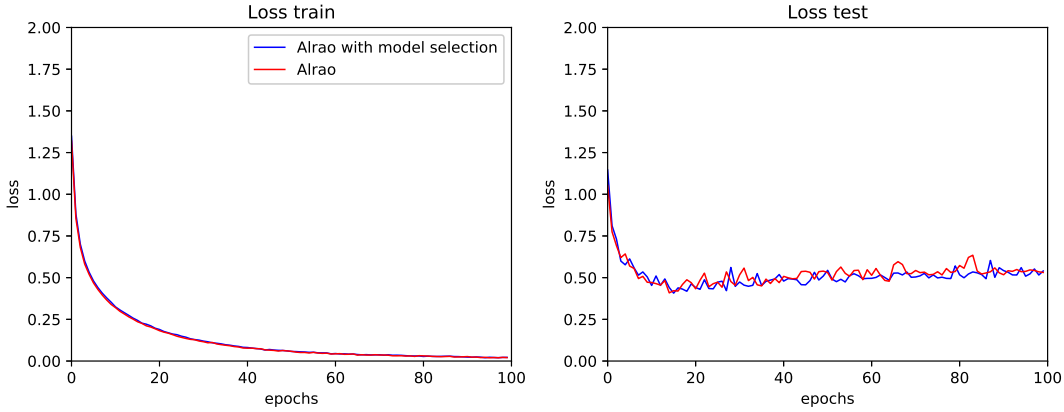


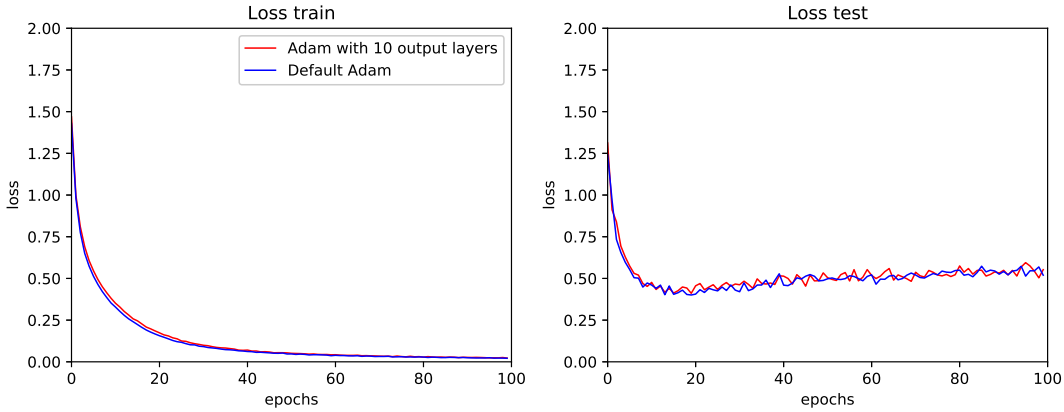
Figure 4.5: Effect of width with SGD on GoogLeNet trained on CIFAR10, after 15 epochs (average over three runs). Grey means numerical divergence (NaN).



(a) Model averaging weights during training of GoogLeNet with Alrao on CIFAR10 with 10 classifiers. We represent the weights a_j, depending on the corresponding classifier's learning rate.

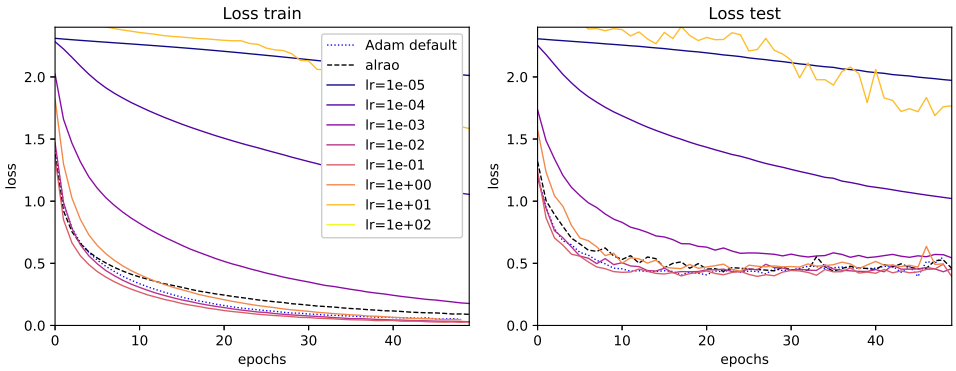


(b) Training with standard Alrao and with Alrao with model selection. With model selection, after one epoch the best classifier is selected according to the model averaging weights, so that the architecture reverts to the original architecture without model averaging. The results are averaged on three runs.

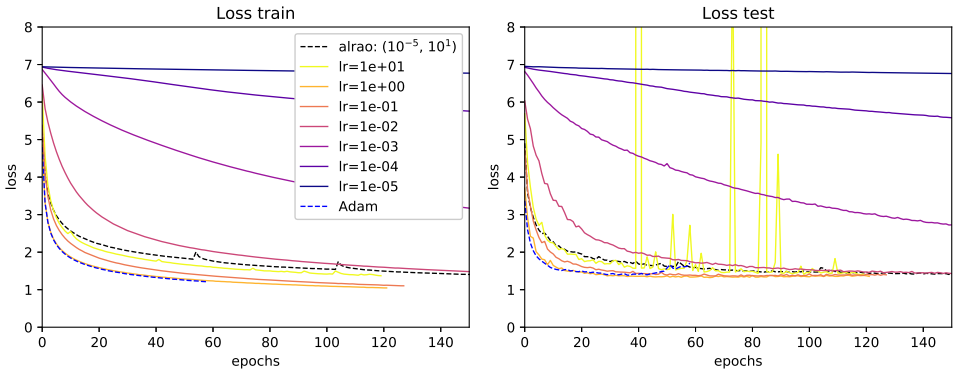


(c) Training with Adam on GoogLeNet, and on the same architecture modified as in Alrao (with 10 output layer copies). Every classifier copy uses the same default Adam learning rate. The overall output of the model is, as in Alrao, a switch model averaging over the classifier. The results are averaged on three runs.

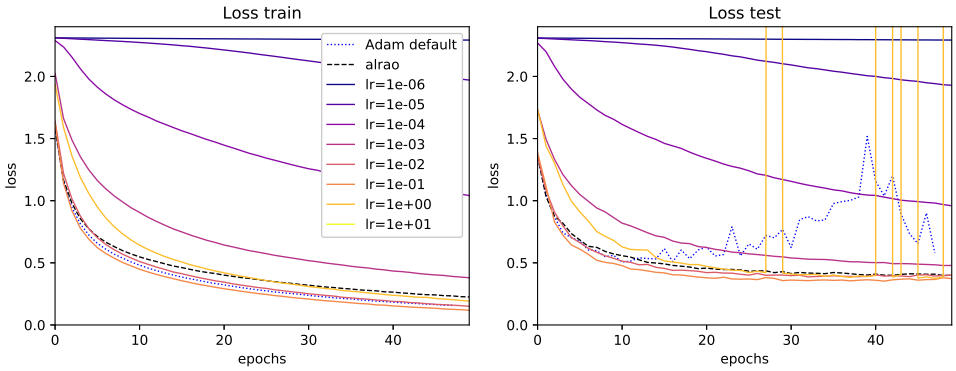
Figure 4.6: Experiments on the effect of the model averaging layer in Alrao. In all experiments, GoogLeNet is trained on CIFAR10.



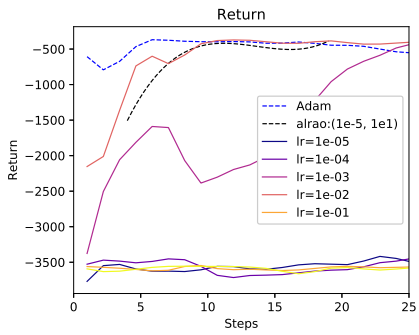
(a) GoogLeNet on CIFAR10 (Average on three runs)



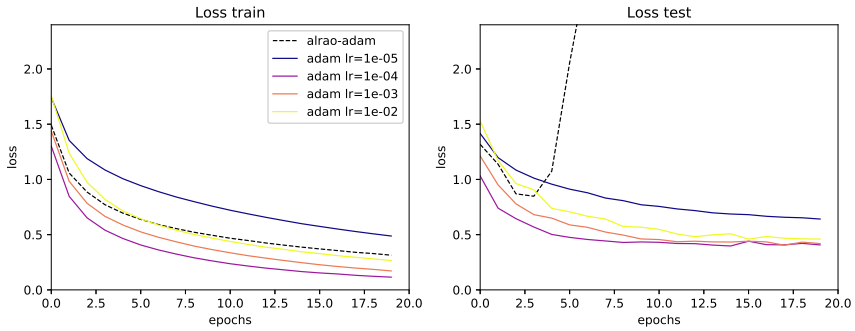
(b) Densenet121 trained on ImageNet



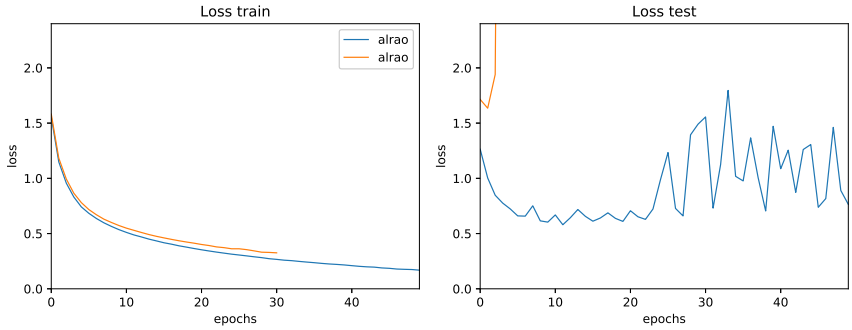
(c) MobileNetV2 on Cifar10 (average over 3 runs)



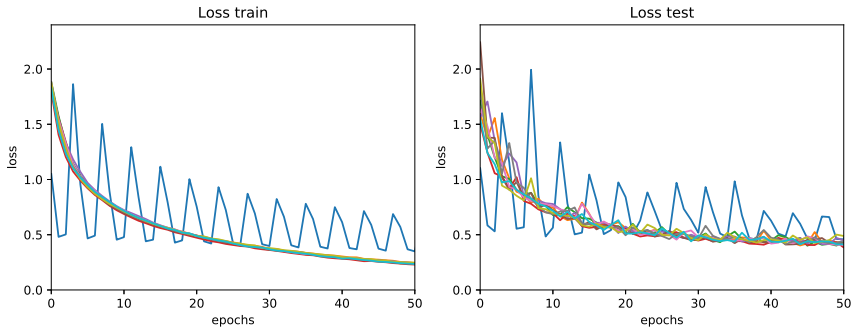
(d) Reinforcement Learning in the pendulum environment



(a) Alrao-Adam with GoogLeNet on CIFAR10: Alrao-Adam compared with standard Adam with various learning rates. Alrao uses 10 classifiers and learning rates in the interval $(10^{-6}; 1)$. Each plot is averaged on 10 experiments. We observe that optimization with Alrao-Adam is efficient, since train loss is comparable to the usual Adam methods. But the model starkly overfits, as the test loss diverges.



(b) Alrao-Adam with MobileNet on CIFAR10: Alrao-Adam with two different learning rate intervals, $(10^{-6}; 10^{-2})$ for the first one, $(10^{-6}; 10^{-1})$ for the second one, with 10 classifiers each. The first one is with $\eta_{\min} = 10^{-6}$. Each plot is averaged on 10 experiments. Exactly as with GoogLeNet model, optimization itself is efficient (for both intervals). For the interval with the smallest $\eta_{\max}$, the test loss does not converge and is very unstable. For the interval with the largest $\eta_{\max}$, the test loss diverges.



(c) Alrao-Adam with VGG19 on CIFAR10: Alrao-Adam on the interval $(10^{-6}, 1)$, with 10 classifiers. The 10 plots are 10 runs of the same experiments. While 9 of them do converge and generalize, the last one exhibits wide oscillations, both in train and test.

Figure 4.8: Alrao-Adam: Experiments with the VGG19, GoogLeNet and MobileNet networks on CIFAR10.

Chapter 5

Making Deep Q-learning methods robust to time discretization

In this chapter, we present the published paper:

> Tallec, C., Blier, L., and Ollivier, Y. (2019). Making Deep Q-learning methods robust to time discretization. In *ICML 2019 - Thirty-sixth International Conference on Machine Learning*

5.1 Introduction

In recent years, *Deep Reinforcement Learning* (DRL) approaches have provided impressive results in a variety of domains, achieving superhuman performance with no expert knowledge in perfect information zero-sum games (Silver et al., 2017), reaching top player level in video games (OpenAI 2018b, Mnih et al. 2015), or learning dexterous manipulation from scratch without demonstrations (OpenAI, 2018a). In spite of those successes, DRL approaches are sensitive to a number of factors, including hyperparameterization, implementation details or small changes in the environment parameters (Henderson et al. 2017, Zhang et al. 2018). This sensitivity, along with sample inefficiency, largely prevents DRL from being applied in real world settings. Notably, high sensitivity to environment parameters prevents transfer from imperfect simulators to real world scenarios.

In this paper we focus on the sensitivity to time discretization of DRL approaches, such as what happens when an agent receives 50 observations and is expected to take 50 actions per second instead of 10. In principle, decreasing time discretization, or equivalently shortening reaction time, should only improve agent performance. Robustness to time discretization is especially relevant in *near-continuous* environments, which includes most continuous control environments, robotics, and many video games.

Standard approaches based on estimation of state-action value functions, such as *Deep Q-learning* (DQN, Mnih et al. 2015) and *Deep deterministic policy gradient* (DDPG, Lillicrap et al. 2015) are not at all robust to changes in time discretization. This is shown experimentally in Sec. 5.5. Intuitively, as the discretization timestep decreases, the effect of individual actions on the total return decreases too: $Q^*(s, a)$ is the value of playing action a then playing optimally, and if a is only maintained for a very short time its advantage over other actions will be accordingly small. (This occurs even with a suitably adjusted decay factor γ.) If the discretization timestep becomes infinitesimal, the effect of every individual action vanishes: there is no continuous-time Q-function (Thm. 5.2), hence the poor performance of Q-learning with small time steps. These statements can be fully formalized in the framework of continuous-time reinforcement learning (Sec. 5.3) (Doya, 2000; Baird, 1994).

We focus on continuous time because this leads to a clear theoretical framework, but our observations make sense in any setting in which the value results from taking a large number

of small individual actions. Our results suggest standard Q-learning will fail in such settings without a delicate balance of hyperparameter scalings and reparameterizations.

We are looking for an algorithm that would be as invariant as possible to changing the discretization timestep. Such an algorithm should remain viable when this timestep is small, and in particular admit a continuous-time limit when the discretization timestep goes to 0. This leads to precise design choices in term of agent architecture, exploration policy, and learning rates scalings. The resulting algorithm is shown to provide better invariance to time discretization than vanilla DQN or DDPG, on many environments (Sec. 5.5). On a new environment, as soon as the order of magnitude of the time discretization is known, our analysis readily provides relevant scalings for a number of hyperparameters.

Our contribution is threefold:

- Building on (Baird, 1994), we formally show that the Q-function collapses to the V-function in near-continuous time, and thus that standard Q-learning is ill-behaved in this setting.
- Our analysis of properties in the continuous-time limit leads to a robust off-policy algorithm. In particular, it provides insights on architecture design, and constrains exploration schemes and learning rates scalings.
- We empirically show that standard Q-learning methods are not robust to changes in time discretization, exhibiting degraded performance, while our algorithm demonstrates substantial robustness.

5.2 Related Work

Our approach builds on (Baird, 1994), who identified the collapse of Q-learning for small time steps and, as a solution, suggested the Advantage Updating algorithm, with proper scalings for the V and advantage parts depending on timescale δt; testing was only done on a quadratic-linear problem.

We expand on (Baird, 1994) in several directions. First, we modify the algorithm by using a different normalization step for A, which forgoes the need to learn the normalization itself, thanks to the parameterization (5.4.8). Second, we test Advantage Updating for the first time on a variety on RL environments using deep networks, establishing Deep Advantage Updating as a viable algorithm in this setting. Third, we provide formal proofs in a general setting for the collapse of Q-learning when the timescale δt tends to 0, and for the non-collapse of Advantage Updating with the proper scalings. Fourth, we also discuss how to obtain δt-invariant exploration. Fifth, we provide stringent experimental tests of the actual robustness to changing δt.

Our study focuses on off-policy algorithms. Some on-policy algorithms, such as A3C (Mnih et al., 2016), PPO (Schulman et al., 2017) or TRPO (Schulman et al., 2015) may be time discretization invariant with specific setups. This is out of the scope of our work and would require a separate study.

(Wang et al., 2015) also use a parameterization separating the value and advantage components of the Q-function. But contrary to (Baird, 1994)'s Advantage Updating, learning is still done in a standard way on the Q-function obtained from adding these two components. Thus this approach reparameterizes Q but does not change scalings and does not result in an invariant algorithm for small δt.

The problem studied here is a continuity effect quite distinct from multiscale RL approaches: indeed the issue arises even if there is only one timescale in the environment. Arguably, a small δt can be seen as a mismatch between the algorithm's timescale and the physical system's timescale, but the collapse of the Q function to the V function is an intrinsic mathematical phenomenon arising from time continuity.

Reinforcement learning has been studied from a mathematical perspective when time and space are both continuous, in connection with optimal control and the Hamilton–Jacobi–Bellman

(HJB) equation (a PDE which characterizes the value function for continuous space-time). Explicit algorithms for continuous space-time can be found in (Doya, 2000; Munos and Bourgine, 1998) (see also the references therein). (Munos and Bourgine, 1998) use a grid approach to provably solve the HJB equation when discretization tends to 0 (assuming every state in the grid is visited a large number of times). However, the resulting algorithms are impractical (Doya, 2000) for larger-dimensional problems. (Doya, 2000) focusses on algorithms specific to the continuous space-time case, including advantage updating and modelling the time derivative of the environment.

Here on the other hand we focus on generic deep RL algorithms that can handle both discrete and continuous time and space, without collapsing in continuous time, thus being robust to arbitrary timesteps.

5.3 Near Continuous-Time Reinforcement Learning

Many reinforcement learning environments are not intrinsically time-discrete, but discretizations of an underlying continuous-time environment. For instance, many simulated control environments, such as the Mujoco environments (Lillicrap et al., 2015) or OpenAI Gym classic control environments (Brockman et al., 2016), are discretizations of continuous-time control problems. In simulated environments, the time discretization is fixed by the simulator, and is often used to approximate an underlying differential equation. In this case, the timestep may correspond to the number of frames generated by second. In real world environments, sensors and actuators have a fixed time precision: cameras can only capture a fixed amount of frames per second, and physical limitations prevent actuators from responding instantaneously. The quality of these components thus imposes a lower bound on the discretization timestep. As the timestep δt is largely a constraint imposed by computational ressources, we would expect that decreasing δt would only improve the performance of RL agents (though it might make optimization harder). RL algorithms should, at least, be resilient to a change of δt, and should remain viable when $\delta t \to 0$. Besides, designing a time discretization invariant algorithm could alleviate tedious hyperparameterization by providing better defaults for time-horizon-related parameters.

We are thus interested in the behavior of RL algorithms in discretized environments, when the discretization timestep is small. We will refer to such environments as *near-continuous environments*. A formalized view of near-continuous environments is given below, along with δt-dependent definitions of return, discount factor and value functions, that converge to well defined continous-time limits. The state-action value function is shown to collapse to the value function as δt goes to 0. Consequently there is no Q-learning in continuous time, foreshadowing problematic behavior of Q-learning with small timesteps.

5.3.1 Framework

Let $\mathcal{S} = \mathbb{R}^d$ be a set of states, and $\mathcal{A}$ be a set of actions. Consider the continuous-time *Markov Decision Process* (MDP) defined by the differential equation

$$ds_t/dt = F(s_t, a_t) \tag{5.3.1}$$

where $F\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describes the dynamics of the environment. The agent interacts with the environment through a deterministic policy function $\pi\colon \mathcal{S} \to \mathcal{A}$, so that $a_t = \pi(s_t)$. Actions can be discrete or continuous. For simplicity we assume here that both the dynamics and exploitation policy are deterministic; [1] the exploration policy will be random, but care must

[1] We believe the results presented here hold more generally, assuming states follow a *stochastic* differential equation

$$ds = F(s, a)dt + \Sigma(s, a)dB_t \tag{5.3.2}$$

with B_t a multidimensional Brownian motion and Σ a covariance matrix. A formal treatment of SDEs is beyond the scope of this paper.

be taken to define proper random policies in continuous time, especially with discrete actions (Sec. 5.4.2).

For any timestep $\delta t > 0$, we can define an MDP $\mathcal{M}_{\delta t} = \langle \mathcal{S}, \mathcal{A}, T_{\delta t}, r_{\delta t}, \gamma_{\delta t} \rangle$ as a discretization of the continuous-time MDP with *time discretization* δt. The transition function of a state s is the state obtained when starting at $s_0 = s$ and maintaining $a_t = a$ constant for a time δt. This corresponds to an agent evolving in the continuous environment (5.3.1), but only making observations and choosing actions every δt. The rewards and decay factor are specified below. We call such an MDP $\mathcal{M}_{\delta t}$ *near-continuous*.

A necessary condition for robustness of an algorithm for near-continuous time MDPs is to remain viable when $\delta t \to 0$. Thus we will try to make sure the various quantities involved converge to meaningful limits when $\delta t \to 0$.

We give semi-formal statements below; the full statements, proofs, and technical assumptions (typically, differentiability assumptions) can be found in the supplementary material.

Return and discount factor. Suitable δt-dependent scalings of the discount factor $\gamma_{\delta t}$ and reward $r_{\delta t}$ are as follows. These definitions fit the discrete case when $\delta t = 1$, and provide well-defined, non-trivial returns and value functions when δt goes to 0.

For a continuous MDP and a continuous trajectory $\tau = (s_t, a_t)_t$, the return is defined as (Doya, 2000)

$$R(\tau) := \int_{t=0}^{\infty} \gamma^t \, r(s_t, a_t) \, dt. \tag{5.3.3}$$

A natural time discretization is obtained by defining the discretized return $R_{\delta t}$ of the MDP $\mathcal{M}_{\delta t}$ as

$$R_{\delta t}(\tau) := \sum_{k=0}^{\infty} \gamma^{k\delta t} \, r(s_{k\delta t}, a_{k\delta t}) \, \delta t \tag{5.3.4}$$

and the discretized return will correspond to the continuous-time return if we set the decay factor $\gamma_{\delta t}$ and rewards $r_{\delta t}$ of the discretized MDP $\mathcal{M}_{\delta t}$ to

$$\gamma_{\delta t} := \gamma^{\delta t}, \qquad r_{\delta t} := \delta t . r. \tag{5.3.5}$$

Physical time vs algorithmic time, time horizon. In near-continuous environments, there are two notions of time: the algorithmic time k (number of steps or actions taken), and the physical time t (time spent in the underlying continuous time environment), related via $t = k.\delta t$.

The time horizon is, informally, the time range over which the agent optimizes its return. As a rule of thumb, the time horizon of an agent with discount factor γ is of order $\frac{1}{1-\gamma}$ steps; beyond that, the decay factor kicks in and the influence of further rewards becomes small.

The definition (5.3.5) of the decay factor $\gamma_{\delta t}$ in near-continuous environments keeps the time horizon constant in *physical* time, by making $\gamma_{\delta t}$ close to 1 in algorithmic time. Indeed, physical time horizon is δt times the algorithmic time horizon, namely

$$\frac{\delta t}{1 - \gamma^{\delta t}} = -\frac{1}{\log \gamma} + O(\delta t) \approx \frac{1}{1-\gamma}, \tag{5.3.6}$$

which is thus stable when $\delta t \to 0$. On the other hand, if $\gamma_{\delta t}$ was left constant as δt goes to 0, the corresponding time horizon in physical time would be $\approx \frac{\delta t}{1-\gamma}$ which goes to 0 when δt goes to 0: such an agent would be increasingly short-sighted as $\delta t \to 0$.

In the following, we use the suitably-scaled decay factor (5.3.5) both for Deep Advantage Updating and for the classical deep Q-learning baselines.

Value function. The return (5.3.3) leads to the following continuous-time value function

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s\right] \tag{5.3.7}$$

$$= \mathbb{E}_{\tau\sim\pi}\left[\int_0^\infty \gamma^t\, r(s_t, a_t)\, dt \mid s_0 = s\right]. \tag{5.3.8}$$

Meanwhile, the value function (in the ordinary sense) of the discrete MDP $\mathcal{M}_{\delta t}$ is

$$V^\pi_{\delta t}(s) = \mathbb{E}_{\tau\sim\pi}\left[R_{\delta t}(\tau) \mid s_0 = s\right] \tag{5.3.9}$$

$$= \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=0}^\infty \gamma^{k\delta t}\, r(s_{k\delta t}, a_{k\delta t})\, \delta t \mid s_0 = s\right] \tag{5.3.10}$$

which obeys the Bellman equation [2]

$$V^\pi_{\delta t}(s) = r(s, \pi(s))\, \delta t + \gamma^{\delta t}\, \mathbb{E}_{s_{(k+1)\delta t}|s_{k\delta t}=s} V^\pi_{\delta t}(s_{(k+1)\delta t}) \tag{5.3.11}$$

When the timestep tends to 0, one converges to the other.

Theorem 5.1. *Under suitable smoothness assumptions, $V^\pi_{\delta t}(s)$ converges to $V^\pi(s)$ when $\delta t \to 0$.*

5.3.2 There is No Q-Function in Continuous Time

Contrary to the value function, the action-value function Q is ill-defined for continuous-time MDPs. More precisely, the Q-function collapses to the V-function when $\delta t \to 0$. In near continuous time, the effect of individual actions on the Q-function is of order $O(\delta t)$. This will make ranking of actions difficult, especially with an approximate Q-function. This argument appears informally in (Baird, 1994). Formally:

Theorem 5.2. *Under suitable smoothness assumptions, The action-value function of a near-continuous MDP is related to its value function via*

$$Q^\pi_{\delta t}(s, a) = V^\pi_{\delta t}(s) + O\left(\delta t\right) \tag{5.3.12}$$

when $\delta t \to 0$, for every (s, a).

As a consequence, in exact continuous time, Q^π is equal to V^π: it does not bear *any* information on the ranking of actions, and thus cannot be used to select actions with higher returns. There is no continuous-time Q-learning.

Proof. The discrete-time Q-function of the MDP $\mathcal{M}_{\delta t}$ satisfies the Bellman equation

$$Q^\pi_{\delta t}(s, a) = r(s, a)\, \delta t + \gamma^{\delta t} \mathbb{E}_{s'|s,a}\left[V^\pi_{\delta t}(s')\right]. \tag{5.3.13}$$

The dynamics (5.3.1) of the environment yields

$$s' = s + F(s, a)\, \delta t + o(\delta t). \tag{5.3.14}$$

Assuming that $V^\pi_{\delta t}$ is continuously differentiable with respect to the state, and that its derivatives are uniformly bounded, we obtain,

$$V^\pi_{\delta t}(s') = V^\pi_{\delta t}(s) + \nabla_s V^\pi_{\delta t}(s) \cdot F(s, a)\, \delta t + o(\delta t) \tag{5.3.15}$$

$$= V^\pi_{\delta t}(s) + O(\delta t) \tag{5.3.16}$$

Expanding $V^\pi_{\delta t}(s')$ into $Q^\pi_{\delta t}$ yields

$$Q^\pi_{\delta t}(s, a) = r(s, a)\, \delta t + \gamma^{\delta t}(V^\pi_{\delta t}(s) + O(\delta t)) \tag{5.3.17}$$

$$= O(\delta t) + (1 + O(\delta t))(V^\pi_{\delta t}(s) + O(\delta t)) \tag{5.3.18}$$

$$= V^\pi_{\delta t}(s) + O(\delta t). \tag{5.3.19}$$

which ends the proof.

[2] If the continuous MDP follows the dynamics (5.3.1), the limit of the Bellman equation (5.3.11) for $V^\pi_{\delta t}$ when $\delta t \to 0$ is the *Hamilton–Jacobi–Bellman* equation on V^π (Doya, 2000), namely, $r + \nabla_s V^\pi \cdot F = -\log(\gamma) V^\pi$.

5.4 Reinforcement Learning with a Continuous-Time Limit

We now define a discrete algorithm with a well-defined continuous-time limit. It relies on three elements: defining and learning a quantity that still contains information on action rankings in the limit, using exploration methods with a meaningful limit, and scaling learning rates to induce well-behaved parameter trajectories when δt goes to 0.

5.4.1 Advantage Updating

As seen above, there is no continuous time limit to Q-learning, because Q^π becomes independent of actions and thus cannot be used to select actions. With small but nonzero δt, $Q^\pi_{\delta t}$ still depends on actions, and could still be used to choose actions. However, when approximating $Q^\pi_{\delta t}$, if the approximation error is much larger than $O(\delta t)$, this error dominates, the ranking of actions given by the approximated $Q^\pi_{\delta t}$ is likely to be erroneous.

To define an object which contains the same information on actions as $Q^\pi_{\delta t}$, but admits a learnable action-dependent limit, it is natural to define (Baird, 1994)

$$A^\pi_{\delta t}(s,a) := \frac{Q^\pi_{\delta t}(s,a) - V^\pi_{\delta t}(s)}{\delta t}, \tag{5.4.1}$$

a rescaled version of the advantage function, as the difference between between $Q^\pi_{\delta t}(s,a)$ and $V^\pi_{\delta t}(s)$ is of order $O(\delta t)$. This amounts to splitting Q into value and advantage, and observing that these scale very differently when $\delta t \to 0$.

Contrary to the Q-function, this rescaled advantage function converges when $\delta t \to 0$ to a non-degenerate action-dependent quantity.

Theorem 5.3. *Under suitable smoothness assumptions, $A^\pi_{\delta t}(s,a)$ has a limit $A^\pi(s,a)$ when $\delta t \to 0$. The limit keeps information about actions: namely, if a policy π' strictly dominates π, then $A^\pi(s,\pi'(s)) > 0$ for some state s.*

Learning A^π. The discretized Q-function rewrites as

$$Q^\pi_{\delta t}(s,a) = V^\pi_{\delta t}(s) + \delta t A^\pi_{\delta t}(s,a). \tag{5.4.2}$$

A natural way to approximate $V^\pi_{\delta t}$ and $A^\pi_{\delta t}$ is to apply *Sarsa* or Q-learning to a reparameterized Q-function approximator

$$Q_\Theta(s,a) := V_\theta(s) + \delta t A_\psi(s,a). \tag{5.4.3}$$

with $\Theta := (\theta, \psi)$. At initialization, if both V_θ and A_ψ are initialized independently of δt, this parameterization provides reasonable scaling of the contribution of actions versus states in Q. Our goal is for V_θ to approximate $V^\pi_{\delta t}$ and for A_ψ to approximate $A^\pi_{\delta t}$.

Still, this reparameterization does not, on its own, guarantee that A correctly approximates $A^\pi_{\delta t}$ if Q_Θ approximates $Q^\pi_{\delta t}$. Indeed, for any given pair $(V_\theta,\ A_\psi)$, the pair $(V_\theta(s) - f(s),\ A_\psi(s,a) + f(s)/\delta t)$ (for an arbitrary f) yields the exact same function Q_Θ. This new A_ψ still defines the same ranking of actions, yet this phenomenon might cause numerical problems or instability of A_ψ when $\delta t \to 0$, and prevents direct interpretation of the learned A_ψ. To enforce identifiability of A_ψ, one must enforce the consistency equation

$$V^\pi_{\delta t}(s) = Q^\pi_{\delta t}(s, \pi(s)) \tag{5.4.4}$$

on the approximate A_ψ and V_θ. This translates to

$$A_\psi(s, \pi(s)) = 0. \tag{5.4.5}$$

With this additional constraint, if $Q_\Theta = Q^\pi_{\delta t}$, then $A_\psi = A^\pi_{\delta t}$ and $V_\theta = V^\pi_{\delta t}$: indeed

$$A^\pi_{\delta t}(s,a) = \frac{Q^\pi_{\delta t}(s,a) - V^\pi_{\delta t}(s)}{\delta t} \tag{5.4.6}$$

$$= \frac{Q_\Theta(s,a) - Q_\Theta(s,\pi(s))}{\delta t} = A_\psi(s,a). \tag{5.4.7}$$

In the spirit of (Wang et al., 2015), instead of directly parameterizing A_ψ, we define a parametric function $\bar{A}_\psi$ (typically a neural network), and use $\bar{A}_\psi$ to define A_ψ as

$$A_\psi(s,a) := \bar{A}_\psi(s,a) - \bar{A}_\psi(s,\pi(s)) \tag{5.4.8}$$

so that A_ψ directly verifies the consistency condition.

This approach will lead to δt-robust algorithms for approximating $A^\pi_{\delta t}$, from which a ranking of actions can be derived.

5.4.2 Timestep-Invariant Exploration

To obtain a timestep-invariant RL algorithm, a timestep-invariant exploration scheme is required. For *continuous* actions, (Lillicrap et al., 2015) already introduced such a scheme, by adding an *Ornstein–Uhlenbeck* (Uhlenbeck and Ornstein, 1930) (OU) random process to the actions. Formally, this is defined as

$$\pi^{\text{explore}}(s_{k\delta t}, z_{k\delta t}) := \pi(s_{k\delta t}) + z_{k\delta t} \tag{5.4.9}$$

with $z_{k\delta t}$ the discretization of a continuous-time OU process,

$$dz_t = -z_t\,\kappa\,dt + \sigma\,dB_t. \tag{5.4.10}$$

where B_t is a brownian motion, κ a stiffness parameter and σ a noise scaling parameter. The discretized trajectories of z converge to nontrivial continuous-time trajectories, exhibiting Brownian behavior with a recall force towards 0.

This exploration can be extended to schemes of the form

$$a_{k\delta t} = \pi^{\text{explore}}_{\delta t}(s_{k\delta t}, z_{k\delta t}) \tag{5.4.11}$$

with $(z_{k\delta t})_{k\geqslant 0}$ a sequence of random variables independent from the a's and s's. A sufficient condition for this policy to admit a continuous-time limit is for the sequence $z_{k\delta t}$ to converge in law to a well-defined continuous stochastic process z_t as δt goes to 0.

Thus, for *discrete* actions we can obtain a consistent exploration scheme by taking $z_{\delta t}$ to be a discretization of an $(\#\mathcal{A})$-dimensional continuous OU process, and setting

$$\pi^{\text{explore}}(s_{k\delta t}, z_{k\delta t}) := \arg\max_a \left(A_\psi(s_{k\delta t}, a) + z_{k\delta t}[a]\right) \tag{5.4.12}$$

where $z_{k\delta t}[a]$ denotes the a-th component of $z_{k\delta t}$. Namely, we perturb the *advantage values* by a random process before selecting an action. The resulting scheme converges in continuous time to a nontrivial exploration scheme.

On the other hand, ε-greedy exploration is likely *not* to explore, i.e., to collapse to a deterministic policy, when δt goes to 0. Intuitively, with very small δt, changing the action at random every δt time step just averages out the randomness due to the law of large numbers. More precisely:

Theorem 5.4. *Consider a near-continuous MDP in which an agent selects an action according to an ε-greedy policy that mixes a deterministic exploitation policy π with an action taken from a noise policy $\pi^{noise}(a|s)$ with probability ε at each step. Then the agent's trajectories converge when $\delta t \to 0$ to the solutions of the* deterministic *equation*

$$ds_t/dt = (1-\varepsilon)F(s_t, \pi(s_t)) + \varepsilon \mathbb{E}_{a\sim\pi^{noise}(a|s)} F(s_t, a) \tag{5.4.13}$$

Algorithm 4 Deep Advantage Updating (Discrete actions)

Inputs:
θ and ψ, parameters of V_θ and $\bar{A}_\psi$.
π^{explore} and $\nu_{\delta t}$ defining an exploration policy.
$\mathbf{opt}_V$, $\mathbf{opt}_A$, $\alpha^V \delta t$ and $\alpha^A \delta t$ optimizers and learning rates.
$\mathcal{D}$, buffer of transitions (s, a, r, d, s'), with d the episode termination signal.
δt and γ, time discretization and discount factor.
nb_epochs number of epochs.
nb_steps, number of steps per epoch.

Observe initial state s^0
$t \leftarrow 0$
for $e = 0, \mathbf{nb\_epochs}$ **do**
  for $j = 1, \mathbf{nb\_steps}$ **do**
    $a^k \leftarrow \pi^{\text{explore}}(s^k, \nu^k_{\delta t})$.
    Perform a^k and observe $(r^{k+1}, d^{k+1}, s^{k+1})$.
    Store $(s^k, a^k, r^{k+1}, d^{k+1}, s^{k+1})$ in $\mathcal{D}$.
    $k \leftarrow k + 1$
  end for
  for $k = 0, \text{nb\_learn}$ **do**
    Sample a batch of N random transitions from $\mathcal{D}$
    $Q^i \leftarrow V_\theta(s^i) + \delta t\Big(\bar{A}_\psi(s^i, a^i) - \max_{a'} \bar{A}_\psi(s^i, a')\Big)$
    $\tilde{Q}^i \leftarrow r^i \delta t + (1 - d^i)\gamma^{\delta t} V_\theta(s'^i)$
    $\Delta\theta \leftarrow \frac{1}{N} \sum_{i=1}^{N} \frac{(Q^i - \tilde{Q}^i)\partial_\theta V_\theta(s^i)}{\delta t}$
    $\Delta\psi \leftarrow \frac{1}{N} \sum_{i=1}^{N} \frac{(Q^i - \tilde{Q}^i)\partial_\psi \left(\bar{A}_\psi(s^i, a^i) - \max_{a'} \bar{A}_\psi(s^i, a')\right)}{\delta t}$
    Update θ with $\mathbf{opt}_1$, $\Delta\theta$ and learning rate $\alpha^V \delta t$.
    Update ψ with $\mathbf{opt}_2$, $\Delta\psi$ and learning rate $\alpha^A \delta t$.
  end for
end for

5.4.3 Algorithms for Deep Advantage Updating

We learn V_θ and A_ψ via suitable variants of Q-learning for continuous and discrete action spaces. Namely, the true $A^\pi_{\delta t}$ and $V^\pi_{\delta t}$ of a near-continuous MDP with greedy exploitation policy $\pi(s) := \text{argmax}_{a'} A^\pi_{\delta t}(s, a')$ are the unique solution to the Bellman and consistency equations

$$V^\pi_{\delta t}(s) + \delta t\, A^\pi_{\delta t}(s, a) = r\, \delta t + \gamma^{\delta t}\, \mathbb{E}_{s'} V^\pi_{\delta t}(s') \tag{5.4.14}$$

$$A^\pi_{\delta t}(s, \pi(s)) = 0. \tag{5.4.15}$$

as seen in 5.4.1. Thus V_θ and A_ψ are trained to approximately solve these equations.

Maximization over actions for π is implemented exactly for discrete actions, and, for continuous actions, approximated by a policy neural network $\pi_\varphi(s)$ trained to maximize $A_\psi(s, \pi_\varphi(s))$, similarly to (Lillicrap et al., 2015).

Eq. (5.4.15) is directly verified by A_ψ, owing to the reparametrization $A_\psi(s, a) = \bar{A}_\psi(s, a) - \bar{A}_\psi(s, \pi(s))$, described in 5.4.1. To approximately verify (5.4.14), the corresponding squared Bellman residual is minimized by an approximate gradient descent. The update equations when learning from a transition (s, a, r, s'), either from an exploratory trajectory or from a replay

buffer (Mnih et al., 2015), are

$$\delta Q_{\delta t} \leftarrow A_\psi(s,a)\,\delta t - \big(r\,\delta t + \gamma^{\delta t}\,V_\theta(s') - V_\theta(s)\big) \tag{5.4.16}$$

$$\theta_{\delta t} \leftarrow \theta_{\delta t} + \eta^V_{\delta t}\,\partial_\theta V_\theta(s)\,\frac{\delta Q_{\delta t}}{\delta t} \tag{5.4.17}$$

$$\psi_{\delta t} \leftarrow \psi_{\delta t} + \eta^A_{\delta t}\,\partial_\psi A_\psi(s,a)\,\frac{\delta Q_{\delta t}}{\delta t}. \tag{5.4.18}$$

where the η's are learning rates. Appropriate scalings for the learning rates $\eta^V_{\delta t}$ and $\eta^A_{\delta t}$ in terms of δt to obtain a well defined continuous limit are derived next.

5.4.4 Scaling the Learning Rates

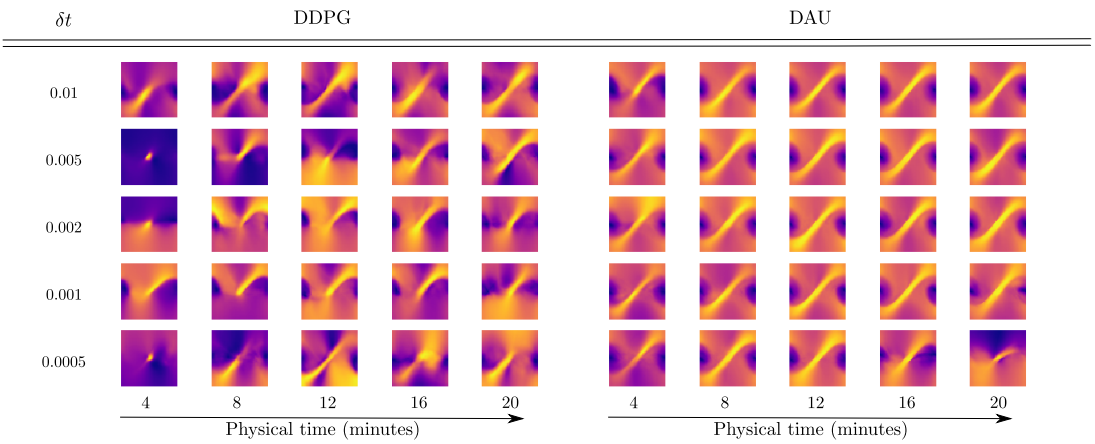


Figure 5.1: Value functions obtained by DDPG (unscaled version) and DAU at different instants in physical time of training on the pendulum swing-up environment. Each image represents the learnt value function (the x-axis is the angle, and the y-axis the angular velocity). The lighter the pixel, the higher the value.

For the algorithm to admit a continuous-time limit, the discrete-time trajectories of parameters must converge to well-defined trajectories as δt goes to 0. This in turn imposes precise conditions on the scalings of the learning rates.

Informally, in the parameter updates (5.4.16)–(5.4.18), the quantity $\delta Q_{\delta t}$ is of order $O(\delta t)$, because $s' = s + O(\delta t)$ in a near-continuous system. Therefore, $\delta Q_{\delta t}/\delta t$ is $O(1)$, so that the gradients used to update θ and ψ are $O(1)$. Therefore, if the learning rates are of order δt, one would expect the parameters θ and ψ to change by $O(\delta t)$ in each time interval δt, thus hopefully converging to smooth continuous-time trajectories. The next theorem formally confirms that learning rates of order δt are the only possibility.

Theorem 5.5. *Let (s_t, a_t) be some exploration trajectory in a near-continuous MDP. Set the learning rates to $\eta^V_{\delta t} = \alpha^V \delta t^\beta$ and $\eta^A = \alpha^A \delta t^\beta$ for some $\beta \geqslant 0$, and learn the parameters θ and ψ by iterating (5.4.16)–(5.4.18) along the trajectory (s_t, a_t). Then, when $\delta t \to 0$:*

- *If $\beta = 1$ the discrete parameter trajectories converge to continuous parameter trajectories.*
- *If $\beta > 1$ the parameters stay at their initial values.*
- *If $\beta < 1$, the parameters can reach infinity in arbitrarily small physical time.*

The resulting algorithm with suitable scalings, *Deep Advantage Updating* (DAU), is specified in Alg. 4 for discrete actions, and in the Supplementary for continuous actions.

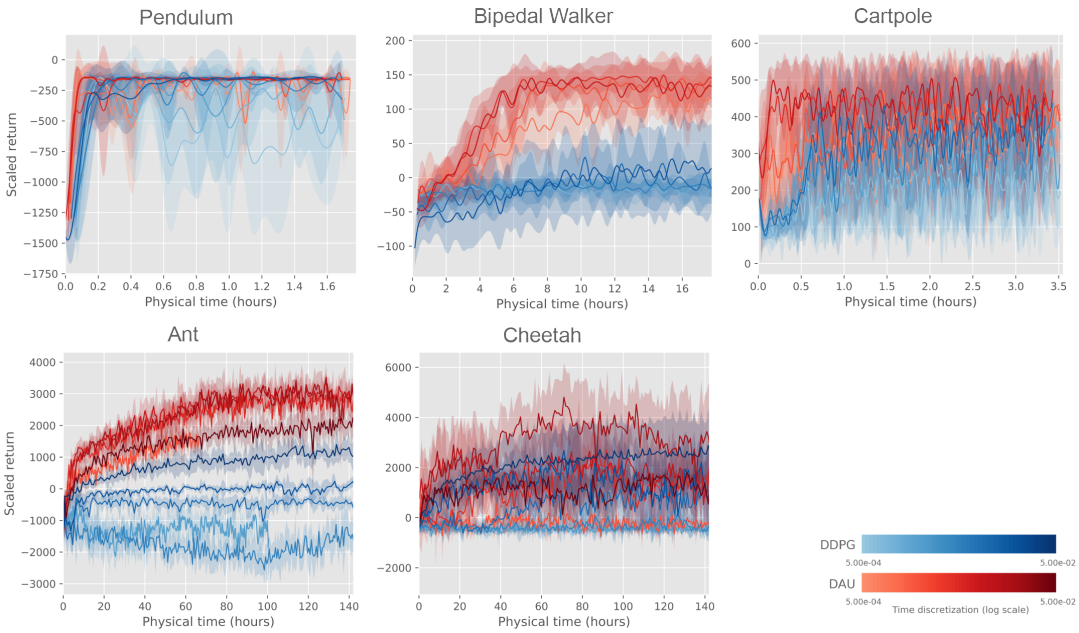


Figure 5.2: Learning curves for DAU and DDPG on classic control benchmarks for various time discretization δt: Scaled return as a function of the physical time spent in the environment.

5.5 Experiments

The experiments provided here are specifically aimed at showing that the proposed method, DAU, works efficiently over a wide range of time discretizations, without specific tuning, while standard deep Q-learning approaches do not. DAU is compared to DDPG for continuous actions and to DQN for discrete actions. As mentionned earlier, we do not study the time discretization invariance of on-policy methods (A3C, PPO, TRPO...).

In all setups, we use the algorithms described in Alg. 4 and Supplementary Alg. 1. The variants of DDPG and DQN used are described in the Supplementary, as well as all hyperparameters. We tested two different setups for DDPG and DQN. In one, we scaled the discount factor (to avoid shortsightedness with small δt), but left all other hyperparameters constant across time discretizations. In the other, we used the properly rescaled discount factor and reward from Eq. (5.3.5), as well as $O(\delta t)$ learning rates for RMSProp. The first variant yields slightly better results, and is presented here, with the second variant in the Supplementary. For all setups, quantitative results are averaged over five runs.

Let us stress that the quantities plotted are rescaled to make comparison possible across different timesteps. For example, returns are given in terms of the discretized return $R_{\delta t}$ as defined in (5.3.4),[3] and, most notably, time elapsed is always given in *physical* time, i.e., the amount of time that the agent spent interacting with the environment (this is not the number of steps).

Qualitative experiments: Visualizing policies and values. To provide qualitative results, and check robustness to time discretization both in terms of returns and in terms of convergence of the approximate value function and policies, we first provide results on the simple pendulum environment from the OpenAI Gym classic control suite. The state space is of dimension 2. We visualize both the learnt value and policy functions by plotting, for each point of the phase diagram $(\theta, \dot{\theta})$, its value and policy. The results are presented in Fig. 5.1 (value function) and Figs. 1, 2, 3 in Supplementary.

[3]This mostly amounts to scaling rewards by a factor δt when this scaling is not naturally done in the environment. Environment-specific details are given in the Supplementary.

We plot the learnt policy at several instants in physical time during training, for various time discretizations δt, for both DAU and DDPG. With DAU, the agent's policy and value function quickly converge for every time discretization without specific tuning. On the contrary, with DDPG, learning of both value function and policy vary greatly from one discretization to another.

Quantitative experiments. We benchmark DAU against DDPG on classic control benchmarks: Pendulum, Cartpole, BipedalWalker, Ant, and Half-Cheetah environments from OpenAI Gym. On Pendulum, Bipedal Walker and Ant, DAU is quite robust to variations of δt and displays reasonable performance in all cases. On the other hand, DDPG's performance varies with δt; performance either degrades as δt decreases (Ant, Cheetah), or becomes more variable as learning progresses (Pendulum) for small δt. On Cartpole, noise dominates, making interpretation difficult. On Half-Cheetah, DAU is not clearly invariant to time discretization. This could be explained by the multiple suboptimal regimes that coexist in the Half-Cheetah environment (walking on the head, walking on the back), which create discontinuities in the value function (see Discussion).

5.6 Discussion

The method derived in this work is theoretically invariant to time discretization, and indeed seems to yield improved timestep robustness on various environments, e.g., simple locomotion tasks. However, on some environments there is still room for improvment. We discuss some of the issues that could explain this theoretical/practical discrepancy.

Note that Alg. 4 requires knowledge of the timestep δt. In most environments, this is readily available, or even directly set by the practitioner: depending on the environment it is given by the frame rate, the maximum frequency of actuators or observation acquisition, the timestep of a physics simulator, etc.

Smoothness of the value function. In our proofs, V^π is assumed to be continuously differentiable. This hypothesis is not always satisfied in practice. For instance, in the pendulum swing-up environment, depending on initial position and momentum, the optimal policy may need to oscillate before reaching the target state. The optimal value function is discontinuous at the boundary between states where oscillations are needed and those where they are not. This results in non-infinitesimal advantages for actions on the boundary. In such environments where a given policy has different regimes depending on the initial state, the continuous-time analysis only holds almost-everywhere.

Memory buffer size. Thm. 5.5 is stated for transitions sampled sequentially from a fixed trajectory. In practice, transitions are sampled from a memory replay buffer, to prevent excessive correlations. We used a fixed-size circular buffer, filled with samples from a single growing exploratory trajectory. In our experiments, the same buffer size was used for all time discretizations. Thus the physical-time freshness of samples in the buffer varies with the time discretization, and in the strictest sense using a fixed-size buffer breaks timestep invariance. A memory-intensive option would be to use a buffer of size $\frac{1}{\delta t}$ (fixed memory in physical time).

Near-continuous reinforcement learning and RMSProp. RMSProp (Tieleman and Hinton, 2012) divides gradient steps by the square root of a moving average estimate of the second moment of gradients. This may interact with the learning rate scaling discussed above. In deterministic environments, gradients typically scale as $O(1)$ in terms of δt, as seen in (5.4.18). In that case, RMSProp preconditioning has no effect on the suitable order of magnitude for learning rates. However, in near continuous *stochastic* environments (Eq. 5.3.2), variance of $\delta Q_{\delta t}/\delta t$ and of the gradients typically scales as $O\left(1/\delta t\right)$. With a fixed batch size, RMSProp

will multiply gradients by a factor $O(\sqrt{\delta t})$. In that case, learning rates need only be scaled as $\sqrt{\delta t}$ instead of δt.

More generally, the continuous-time analysis should in principle be repeated for every component of a system. For instance, if a recurrent model is used to handle state memory or partial observability, care should be taken that the model is able to maintain memory for a non-infinitesimal physical time when $\delta t \to 0$ (see e.g. Tallec and Ollivier 2018).

5.7 Conclusion

Q-learning methods have been found to fail to learn with small time steps, both theoretically and empirically. A theoretical analysis help in building a practical off-policy deep RL algorithm with better robustness to time discretization. This robustness is confirmed empirically.

Appendix

5.A Proofs

We now give proofs for all the results presented in the paper. Most proofs follow standard patterns from calculus and numerical schemes for differential equations, except for Theorem 5.9, which uses an argument specific to reinforcement learning to prove that the continuous-time advantage function contains all the necessary information for policy improvement.

The first result presented is a proof of convergence for discretized trajectories.

Lemma 5.6. *Let $F\colon \mathcal{S}\times\mathcal{A}\to\mathbb{R}^n$ and $\pi\colon \mathcal{S}\to\mathcal{A}$ be the dynamic and policy functions. Assume that, for any a, $s\to F(s,a)$ and $s\to F(s,\pi(s))$ are $\mathcal{C}^1$, bounded and K-lipschitz. For a given s_0, define the trajectory $(s_t)_{t\geqslant 0}$ as the unique solution of the differential equation*

$$\frac{ds_t}{dt} = F(s_t,\pi(s_t)). \tag{5.A.1}$$

For any $\delta t>0$, define the discretized trajectory $(s^k_{\delta t})_k$ which amounts to maintaining each action for a time interval δt; it is defined by induction as $s^0_{\delta t}=s_0$, $s^{k+1}_{\delta t}$ is the value at time δt of the unique solution of

$$\frac{d\tilde{s}_t}{dt} = F(\tilde{s}_t,\pi(s^k_{\delta t})) \tag{5.A.2}$$

with initial point $s^k_{\delta t}$. Then, there exists $C>0$ such that, for every $t\geqslant 0$

$$\|s_t - s^{\lfloor \frac{t}{\delta t}\rfloor}_{\delta t}\| \leqslant \delta t\frac{C}{K}e^{Kt}. \tag{5.A.3}$$

Therefore, discretized trajectories converge pointwise to continuous trajectories.

Proof. The proof mostly follos the classical argument for convergence of the Euler scheme for differential equations. For any k, define

$$e^k_{\delta t} = \|s^k_{\delta t} - s_{\delta t k}\|. \tag{5.A.4}$$

Let $\tilde{s}_t$ be the solution of Eq. (5.A.35) with initial state s^k_δ. This $\tilde{s}_t$ is $\mathcal{C}^2$ on $[0,\delta t]$. Consequently, the Taylor integral formula gives

$$\begin{aligned} s^{k+1}_{\delta t} &= s^k_{\delta t} + F(s^k_{\delta t},\pi(s^k_{\delta t}))\delta t + \int_0^{\delta t}(\delta t - t)\frac{d^2\tilde{s}_t}{dt^2}dt \\ s_{\delta t(k+1)} &= s_{\delta t k} + F(s_{\delta t k},\pi(s_{\delta t k}))\delta t + \int_0^{\delta t}(\delta t - t)\frac{d^2 s_{t+\delta t k}}{dt^2}dt. \end{aligned}$$

Now, both $d^2 s_t/dt^2$ and $d^2\tilde{s}_t/dt^2$ are uniformly bounded, by boundedness and Lipschitzness of $s\to F(s,\pi(s))$ and $s\to F(s,\pi(s^k_{\delta t}))$. Consequently, there exists C such that

$$\begin{aligned} e^{k+1}_{\delta t} &\leqslant \|s^k_{\delta t} - s_{\delta t k}\| + \|F(s^k_{\delta t},\pi(s^k_{\delta t})) - F(s_{\delta t k},\pi(s_{\delta t k}))\|\delta t + C\delta t^2 \\ &\leqslant (1+K\delta t)e^k_{\delta t} + C\delta t^2. \end{aligned}$$

Now, it is easy to prove by induction that

$$e^k_{\delta t} \leqslant (1+K\delta t)^k(e^0_{\delta t} + \frac{C}{K}\delta t) - \frac{C}{K}\delta t. \tag{5.A.5}$$

As $e^0_{\delta t}=0$, this translates to

$$\begin{aligned} e^k_{\delta t} &\leqslant ((1+K\delta t)^k - 1)\delta t\frac{C}{K} \\ &\leqslant (e^{K\delta t k} - 1)\delta t\frac{C}{K}. \end{aligned}$$

Consequently,

$$e_{\delta t}^{\lfloor t/\delta t \rfloor} \leqslant (e^{K(t+\delta t)} - 1)\delta t \frac{C}{K}. \tag{5.A.6}$$

Finally, by boundedness, of $s \to F(s, \pi(s))$, there exists C' such that

$$\|s_{\delta t \lfloor t/\delta t \rfloor} - s_t\| \leqslant \delta t C'. \tag{5.A.7}$$

Combining Eq. (5.A.7) with Eq. (5.A.6), one can find C'' such that

$$\|s_t - s_{\delta t}^{\lfloor t/\delta t \rfloor}\| \leqslant \delta t \frac{C''}{K} e^{Kt}. \tag{5.A.8}$$

In what follows, we assume that the continuous-time reward function $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is bounded, to ensure existence of V^π and $V_{\delta t}^\pi$ for all δt .

Theorem 5.7. *Assume that $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is bounded, and that $s \to r(s, \pi(s))$ is L_r-Lipschitz continuous, then for all $s \in \mathcal{S}$, one has $V_{\delta t}^\pi(s) = V^\pi(s) + o(1)$ when $\delta t \to 0$.*

Proof.
We use the notation $\tilde{r}(s) = r(s, \pi(s))$. Let $\tilde{s}_{\delta t}^t := s_{\delta t}^{\lfloor t/\delta t \rfloor}$. We have:

$$V_{\delta t}^\pi(s) = \int_t \gamma^t \tilde{r}(\tilde{s}_{\delta t}^t) dt + O(\delta t)$$

Indeed:

$$\begin{aligned}
V_{\delta t}^\pi(s) &= \sum_{k=0}^{\infty} \gamma^{k\delta t} \tilde{r}(s_{\delta t}^k)\delta t \\
&= \sum_{k=0}^{\infty} \gamma^{k\delta t} \int_{u=k}^{k+1} \tilde{r}(\tilde{s}_{\delta t}^{u\delta t}) du \\
&= \sum_{k=0}^{\infty} \frac{\delta t \log \gamma}{\gamma^{\delta t} - 1} \int_{u=k}^{k+1} \gamma^{u\delta t} \tilde{r}(\tilde{s}_{\delta t}^{u\delta t}) du \\
&= \frac{\delta t \log \gamma}{\gamma^{\delta t} - 1} \int_{t=0}^{\infty} \gamma^t \tilde{r}(\tilde{s}_{\delta t}^t) dt
\end{aligned}$$

But:

$$\begin{aligned}
\frac{\delta t \log \gamma}{\gamma^{\delta t} - 1} &= \frac{\delta t \log \gamma}{\delta t \log \gamma + O(\delta t^2)} \\
&= 1 + O(\delta t)
\end{aligned}$$

Therefore:

$$V_{\delta t}^\pi(s) = \int_t \gamma^t \tilde{r}(\tilde{s}_{\delta t}^t) dt + O(\delta t)$$

We now have, for any $T > 0$,

$$\begin{aligned}
|V_{\delta t}^\pi(s) - V^\pi(s)| &= |\int_{t=0}^{\infty} \gamma^t \left(\tilde{r}(\tilde{s}_{\delta t}^t) - \tilde{r}(s_t)\right) dt| + O(\delta t) \\
&= |\int_{t=0}^{T} \gamma^t \left(\tilde{r}(\tilde{s}_{\delta t}^t) - r(s_t)\right) dt| \\
&+ |\int_{t=T}^{\infty} \gamma^t \left(\tilde{r}(\tilde{s}_{\delta t}^t) - \tilde{r}(s_t)\right) dt| + O(\delta t)
\end{aligned}$$

The second term can be bounded by the supremum of the reward:

$$|\int_{t=T}^{\infty} \gamma^t \left(\tilde{r}(\tilde{s}_{\delta t}^t) - \tilde{r}(\tilde{s}_t)\right) dt| \leqslant 2 \frac{\|r\|_\infty}{\log(\frac{1}{\gamma})} \gamma^T$$

The first term can be bounded by using Lemma. 1:

$$|\int_{t=0}^{T} \gamma^t \left(\tilde{r}(\tilde{s}^t_{\delta t}) - \tilde{r}(s_t)\right) dt| \leqslant \int_{t=0}^{T} \gamma^t L_r \|s_t - \tilde{s}^t_{\delta t}\| dt \tag{5.A.9}$$

$$\leqslant \int_{t=0}^{T} L_r \frac{C\delta t}{K} \exp((K + \log\gamma)t) dt \tag{5.A.10}$$

$$\leqslant \frac{L_r C}{K(K + \log\gamma)} \exp((K + \log\gamma)T)\delta t \tag{5.A.11}$$

Let us set $T := -\frac{1}{K}\log(\delta t)$. By plugging into Eq. (5.A.9), we have:

$$|\int_{t=T}^{\infty} \gamma^t \left(\tilde{r}(\tilde{s}^t_{\delta t}) - \tilde{r}(s_t)\right) dt| = O(\delta t^{-\log\gamma}) = o(1).$$

By plugging T into equation (5.A.11), we have:

$$|\int_{t=0}^{T} \gamma^t \left(\tilde{r}(\tilde{s}^t_{\delta t}) - \tilde{r}(s_t)\right) dt| = O(\delta t^{-\frac{\log\gamma}{K}}) = o(1),$$

yielding our result.

For the following proof, we further assume that both V^π and $V^\pi_{\delta t}$ are continuously differentiable, and that the gradient and Hessian of $V^\pi_{\delta t}$ w.r.t. s are uniformly bounded in both s and δt. We also assume convergence of $\partial_s V^\pi_{\delta t}(s)$ to $\partial_s V^\pi(s)$ for all s.

Theorem 5.8. *Under the hypothesis above, there exists $A^\pi : \mathcal{S} \to \mathbb{R}$ such that $A^\pi_{\delta t}$ converges pointwise to A^π as δt goes to 0. Besides,*

$$A^\pi(s,a) = r(s,a) + \partial_s V^\pi(s) F(s,a) + \log\gamma V^\pi(s). \tag{5.A.12}$$

Proof. Denote $\tilde{s}^t_{\delta t}(s_0)$ the evaluation at instant t of the solution of $d\tilde{s}_t/dt = F(\tilde{s}_t, \pi(s_0))$ with starting point s_0.

The Bellman equation on $Q^\pi_{\delta t}$ yields

$$Q^\pi_{\delta t}(s,a) = r(s,a)\delta t + \gamma^{\delta t} V^\pi_{\delta t}(\tilde{s}^{\delta t}_{\delta t}(s)).$$

For all s, a first-order Taylor expansion yields

$$\tilde{s}^{\delta t}_{\delta t}(s) = s + F(s,a)\delta t + O(\delta t^2) \tag{5.A.13}$$

where the constant in $O()$ is uniformly bounded thanks to the assumptions on the Hessian. Thus, by uniform boundedness of the Hessian of $V^\pi_{\delta t}$,[a]

$$Q^\pi_{\delta t}(s,a) = r(s,a)\delta t +$$
$$\left(1 + \ln(\gamma)\delta t + O(\delta t^2)\right)\left(V^\pi_{\delta t}(s) + \delta t \partial_s V^\pi_{\delta t}(s) F(s,a) + O(\delta t^2)\right).$$

Now, this yields

$$A^\pi_{\delta t}(s,a) = r(s,a) + \ln(\gamma) V^\pi_{\delta t}(s) + \partial_s V^\pi_{\delta t}(s) F(s,a) + O(\delta t), \tag{5.A.14}$$

and using the convergence of $V^\pi_{\delta t}(s)$ to $V^\pi(s)$ (Thm. 5.1) and $\partial_s V^\pi_{\delta t}(s)$ to $\partial_s V^\pi(s)$ (hypothesis) yields the result with

$$A^\pi(s,a) = r(s,a) + \ln(\gamma) V^\pi(s) + \partial_s V^\pi(s) F(s,a). \tag{5.A.15}$$

[a] Without boundedness of the Hessian, we cannot write the second order Taylor expansion of $V^\pi_{\delta t}(\tilde{s}^{\delta t}_{\delta t}(s))$ in term of δt.

We now show that policy improvement works with the continous time advantage function, i.e.

Theorem 5.9. *Let π and π' be two policies such that both $s \to r(s, \pi(s))$ and $s \to r(s, \pi'(s))$ are continuous. Assume that both V^π and $V^{\pi'}$ are continuously differentiable. Define the advantage function for policies π and π' as in Eq.* (5.A.15).

If for all s, $A^\pi(s, \pi'(s)) \geqslant 0$, then for all s, $V^\pi(s) \leqslant V^{\pi'}(s)$. Moreover, if for all s, $V^{\pi'}(s) > V^\pi(s)$, then there exists s' such that $A^\pi(s', \pi'(s')) > 0$.

Proof. Let $(s_t)_{t \geqslant 0}$ be a trajectory sampled from π' i.e. solution of the equation

$$ds_t/dt = F(s_t, \pi'(s_t)) \tag{5.A.16}$$

with initial condition $s_0 = s$.

Define

$$B(T) = \int_{t=0}^{T} \gamma^t r(s_t, \pi'(s_t)) dt + \gamma^T V^\pi(s_T). \tag{5.A.17}$$

This function if continuously differentiable, and its derivative is

$$\dot{B}(T) = \gamma^T r(s_T, \pi'(s_T)) \tag{5.A.18}$$
$$+ \gamma^T \partial_s V^\pi(s) F(s, \pi'(s)) + \gamma^T \ln(\gamma) V^\pi(s_T) \tag{5.A.19}$$
$$= \gamma^T A^\pi(s_T, \pi'(s_T)) \tag{5.A.20}$$
$$\geqslant 0. \tag{5.A.21}$$

Thus B is increasing, and $B(0) = V^\pi(s)$, $\lim_{T\to\infty} B(t) = V^{\pi'}(s)$. Consequently, $V^\pi(s) \leqslant V^{\pi'}(s)$. Furthermore, if $V^\pi(s) < V^{\pi'}(s)$, then there exists T_0 such that $\dot{B}(T_0) > 0$ (otherwise B is constant), and $A^\pi(s_{T_0}, \pi'(s_{T_0})) > 0$.

Theorem 5.10. *Let $\mathcal{A} = \mathbb{R}^{\mathcal{A}}$ be the action space, and let $\mathcal{P}_1 = \mathbb{R}^{p_1}$ and $\mathcal{P}_2 = \mathbb{R}^{p_2}$ be parameter spaces. Let $A\colon \mathcal{P}_1 \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $V\colon \mathcal{P}_2 \times \mathcal{S} \to \mathbb{R}$ be $\mathcal{C}^2$ function approximators with bounded gradients and Hessians. Let $(a_t)_{t\geqslant 0}$ be a $\mathcal{C}^1$ exploratory action trajectory and $(s_t)_{t\geqslant 0}$ the resulting state trajectory, when starting from s_0 and following $ds_t/dt = F(s_t, a_t)$. Let $\theta^k_{\delta t}$ and $\psi^k_{\delta t}$ be the discrete parameter trajectories resulting from the gradient descent steps in the main text, with learning rates $\eta^V = \alpha^V \delta t^\beta$ and $\eta^A = \alpha^A \delta t^\beta$ for some $\beta \geqslant 0$. Then,*

- *If $\beta = 1$ the discrete parameter trajectories converge to continuous parameter trajectories as δt goes to 0.*
- *If $\beta > 1$, parameter trajectories become stationary as δt goes to 0.*
- *If $\beta < 1$, parameters can grow arbitrarily large after an arbitrarily small physical time when δt goes to 0.*

Proof. Let $(s_t, a_t)_{t\geqslant 0}$ be the trajectory on which parameters are learnt. To simplify notations, define

$$A_\psi(s, a) = \bar{A}_\psi(s, a) - \bar{A}_\psi(s, \pi(s)). \tag{5.A.22}$$

Define F as

$$\begin{aligned} F^\theta(\theta, \psi, s, a) &= \alpha^V (r(s, a) + \ln(\gamma) V_\theta(s) \\ &\quad + \partial_s V_\theta(s) F(s, a) - A_\psi(s, a)) \partial_\theta V_\theta(s) \\ F^\psi(\theta, \psi, s, a) &= \alpha^A (r(s, a) + \ln(\gamma) V_\theta(s) \\ &\quad + \partial_s V_\theta(s) F(s, a) - A_\psi(s, a)) \partial_\psi A_\psi(s, a). \end{aligned}$$

From the bounded Hessians and Gradients hypothesis, V, A, $\partial_s V$, $\partial_\theta V$ and $\partial_\psi A$ are uniformly Lipschitz continuous in θ and ψ, thus F is Lipschitz continuous.

The discrete equations for parameters updates with learning rates $\alpha^V \delta t^\beta$ and $\alpha^A \delta t^\beta$ are

$$\begin{aligned} \delta Q &= r(s_{k\delta t}, a_{k\delta t}) \delta t + \gamma^{\delta t} V_{\theta^k_{\delta t}}(s_{(k+1)\delta t}) \\ &\quad - V_{\theta^k_{\delta t}}(s_{k\delta t}) - A_\psi(s_{k\delta t}, a_{k\delta t}) \\ \theta^{k+1}_{\delta t} &= \theta^k_{\delta t} + \alpha^V \delta t^\beta \frac{\delta Q}{\delta t} \partial_\theta V_{\theta^k_{\delta t}}(s_{k\delta t}) \\ \psi^{k+1}_{\delta t} &= \psi^k_{\delta t} + \alpha^A \delta t^\beta \frac{\delta Q}{\delta t} \partial_\psi A_{\theta^k_{\delta t}}(s_{k\delta t}, a_{k\delta t}) \end{aligned}$$

Under uniform boundedness of the Hessian of $s \mapsto V_\theta(s)$, one can show

$$\begin{pmatrix} \theta^{k+1}_{\delta t} \\ \psi^{k+1}_{\delta t} \end{pmatrix} = \begin{pmatrix} \theta^k_{\delta t} \\ \psi^k_{\delta t} \end{pmatrix} + \delta t^\beta F(\theta^k_{\delta t}, \psi^k_{\delta t}, s_{k\delta t}, a_{k\delta t}) + O(\delta t^\beta \delta t), \tag{5.A.23}$$

with a O independent of k. With the additional hypothesis that the gradient of $(s, a) \to \bar{A}_\psi(s, a)$ is uniformly bounded, we have

- For $\beta = 1$, a proof scheme identical to that of Thm. 5.6 shows that discrete trajectories converge pointwise to continuous trajectories defined by the differential equation

$$\frac{d}{dt} \begin{pmatrix} \theta_t \\ \psi_t \end{pmatrix} = F(\theta_t, \psi_t, s_t, a_t), \tag{5.A.24}$$

  which admits unique solutions for all initial parameters, since F is uniformly lipschitz continuous.

- Similarly, for $\beta > 1$, the proof scheme of Thm. 5.6 shows that discrete trajectories converge pointwise to continuous trajectories defined by the differential equation

$$\frac{d}{dt} \begin{pmatrix} \theta_t \\ \psi_t \end{pmatrix} = 0 \tag{5.A.25}$$

  and thus that trajectories shrink to a single point as δt goes to 0.

We now turn to proving that when $\beta < 1$, trajectories can diverge instantly in physical time. Consider the following continuous MDP,

$$s_t = \sin(t) \tag{5.A.26}$$

whatever the actions, with reward 0 everywhere and $0 < \gamma < 1$. The resulting value function is $V(s) = 0$ (since there are no actions, V is independent of a policy), and the advantage function is 0. We consider the function approximator $V_\theta(s) = \theta s$ (which can represent the true value function). The update rule for θ is

$$\delta Q^k_{\delta t} = \gamma^{\delta t}\theta^k_{\delta t}\sin((k+1)\delta t) - \theta^k_{\delta t}\sin(k\delta t) \tag{5.A.27}$$

$$\theta^{k+1}_{\delta t} = \theta^k_{\delta t} + \alpha\delta t^\beta \frac{\gamma^{\delta t}\theta^k_{\delta t}\sin((k+1)\delta t) - \theta^k_{\delta t}\sin(k\delta t)}{\delta t}\sin(k\delta t) \tag{5.A.28}$$

Set $K_{\delta t} := \lfloor \delta t^{-\frac{\beta+3}{4}} \rfloor$, then for all $k \leqslant K_{\delta t}$, $o(k\delta t) = o(1)$ and

$$\theta^{k+1}_{\delta t} = \theta^k_{\delta t}(1 + \alpha\delta t^\beta(1 + o(1))\sin(k\delta t)) \tag{5.A.29}$$

$$\tag{5.A.30}$$

Let $\rho^k_{\delta t} := \log\theta^k_{\delta t}$. Then

$$\rho^k_{\delta t} = \rho^k_{\delta t} + \alpha k\delta t^{\beta+1} + o(k\delta t^{\beta+1}). \tag{5.A.31}$$

Finally,

$$\rho^{K_{\delta t}}_{\delta t} = \rho^0_{\delta t} + \alpha\frac{K_{\delta t}(K_{\delta t}+1)}{2}\delta t^\beta + o(K^2_{\delta t}\delta t^{\beta+1}) \tag{5.A.32}$$

$$= \rho^0_\delta + \alpha\delta t^{\frac{\beta-1}{3}} + o(\delta t^{\frac{\beta-1}{3}}) \tag{5.A.33}$$

$$\xrightarrow[\delta t\to 0]{} +\infty. \tag{5.A.34}$$

Thus parameters diverge in an infinitesimal physical time when δt goes to 0.

Theorem 5.11. *Let $F\colon \mathcal{S}\times\mathcal{A}\to\mathbb{R}^n$ be the dynamic, and $\pi\colon \mathcal{S}\times\mathcal{A}\to[0,1]$ be the policy, such that $\pi(s,\cdot)$ is a probability distribution over $\mathcal{A}$. Assume that F is C^1 with bounded derivatives, and that π is C^1 and bounded. For any $\delta t > 0$, define the discretized trajectory $(s^k_{\delta t})_k$ which amounts to sample an action from $\pi(s,\cdot)$ and maintaining each action for a time interval δt; it is defined by induction as $s^0_{\delta t} = s_0$, $s^{k+1}_{\delta t}$ is the value at time δt of the unique solution of*

$$\frac{d\tilde{s}_t}{dt} = F(\tilde{s}_t, a_k) \tag{5.A.35}$$

with $a_k \sim \pi(s^k_{\delta t},\cdot)$ and initial point $s^k_{\delta t}$.

Then the agent's trajectories converge when $\delta t\to 0$ to the solutions of the deterministic *equation:*

$$\frac{ds_t}{dt} = \mathbb{E}_{a\sim\pi(s_t,\cdot)}F(s_t,a). \tag{5.A.36}$$

Notably, if π is an epsilon greedy strategy that mixes a deterministic exploitation policy $\pi^{deterministic}$ with an action taken from a noise policy π^{noise} with probability ε at, the trajectory converge to the solutions of the equation:

$$ds_t/dt = (1-\varepsilon)F(s_t, \pi^{deterministic}(s_t)) + \varepsilon\mathbb{E}_{a\sim\pi^{noise}(a|s)}F(s_t,a) \tag{5.A.37}$$

Proof.

Consider $(s_{\delta t^2})$ the random trajectory of a near-continuous MDP with time-discretization δt^2 obtained by taking at each step k an action a_k along $a_k \sim \pi(a|s^k_{\delta t^2})$ independantly. We have:

$$\begin{aligned} s^{\lfloor 1/\delta t\rfloor}_{\delta t^2} &= s^0_{\delta t^2} + \sum_{k=1}^{\lfloor 1/\delta t\rfloor} s^k_{\delta t^2} - s^{k-1}_{\delta t^2} + O(\delta t^2) \\ &= s^0_{\delta t^2} + \sum_{k=1}^{\lfloor 1/\delta t\rfloor} F(s^{k-1}_{\delta t^2}, a_{k-1})\delta t^2 + O(\delta t^2) \end{aligned}$$

We define $f(s) := \mathbb{E}_{a\sim\pi(s)}[F(s,a)] = \int_{a\in\mathcal{A}} F(s,a)\pi(s,a)$. Since π and F are bounded and C^1, we know that f is C^1. We have:

$$\begin{aligned} s^{\lfloor 1/\delta t\rfloor}_{\delta t^2} &= s^0_{\delta t^2} + \sum_{k=1}^{\lfloor 1/\delta t\rfloor} f(s^{k-1}_{\delta t^2})\delta t^2 \\ &\quad + \sum_{k=1}^{\lfloor 1/\delta t\rfloor} (F(s^{k-1}_{\delta t^2}, a_{k-1}) - f(s^{k-1}_{\delta t^2}))\delta t^2 + O(\delta t^2) \\ s^{\lfloor 1/\delta t\rfloor}_{\delta t^2} &= s^0_{\delta t^2} + \sum_{k=1}^{\lfloor 1/\delta t\rfloor} f(s^{k-1}_{\delta t^2})\delta t^2 + \xi + O(\delta t^2) \end{aligned}$$

with $\xi := \delta t^2 \sum_{k=1}^{\lfloor 1/\delta t \rfloor} \left(F(s_{\delta t^2}^{k-1}, a_{k-1}) - f(s_{\delta t^2}^{k-1}) \right)$. By definition, we have $\mathbb{E}[\xi] = 0$. Moreover, by using the independance of actions and the boundness of F, there is $\sigma > 0$ such that:

$$\mathbb{E}[\|\xi\|^2] \leqslant \sigma^2 \delta t^3$$

We know that f is C^1 on a compact space. Therefore, there is L_f such that f is L_f Lipschitz, and we have:

$$\| \left(\sum_{k=1}^{\lfloor 1/\delta t \rfloor} f(s_{\delta t^2}^{k-1}) \delta t \right) - f(s_{\delta t^2}^0) \| \leqslant \delta t L_f \sum_{k=1}^{\lfloor 1/\delta t \rfloor} \| s_{\delta t^2}^{k-1} - s_{\delta t^2}^0 \|$$

Since F is bounded, we know that $\|s_{\delta t^2}^k - s_{\delta t^2}^{k-1}\| \leqslant C\delta t$. Therefore:

$$\begin{aligned} \| \left(\sum_{k=1}^{\lfloor 1/\delta t \rfloor} f(s_{\delta t^2}^{k-1}) \delta t \right) - f(s_{\delta t^2}^0) \| &\leqslant \delta t L_f C \sum_{k=1}^{\lfloor 1/\delta t \rfloor} k \delta t \\ &= O(\delta t^2) \end{aligned}$$

Therefore:

$$s_{\delta t^2}^{\lfloor 1/\delta t \rfloor} = s_{\delta t^2}^0 + f(s_{\delta t^2}^0)\delta t + \xi + O(\delta t^2)$$

Therefore, we have $a > 0$ such that $\|s_{\delta t^2}^{\lfloor 1/\delta t \rfloor} - s_{\delta t^2}^0 - f(s_{\delta t^2}^0)\delta t\| \leqslant \|\xi\| + a\delta t^2$

We define $(\tilde{s}_{\delta t})$ the deterministic near-continuous process with time discretization δt defined by $\tilde{s}_{\delta t}^{k+1} := s_{\delta t}^k + f(s_{\delta t}^k)\delta t$. We have:

$$\begin{aligned} &\|s_{\delta t^2}^{(k+1)\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^{k+1}\| \\ &\qquad \leqslant \|s_{\delta t^2}^{(k+1)\lfloor 1/\delta t \rfloor} - s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - f(s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor})\delta t\| \\ &\qquad + \|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} + f(s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor})\delta t - \tilde{s}_{\delta t}^{k+1}\| \end{aligned}$$

We know that

$$\|s_{\delta t^2}^{(k+1)\lfloor 1/\delta t \rfloor} - s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - f(s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor})\delta t\| \leqslant \|\xi_k\| + a\delta t^2 \tag{5.A.38}$$

Moreover:

$$\begin{aligned} &\|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} + f(s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor})\delta t - \tilde{s}_{\delta t}^{k+1}\| \\ &\qquad \leqslant \|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^k\| + \delta t \|f(s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor}) - f(\tilde{s}_{\delta t}^k)\| \\ &\qquad \leqslant (1 + L_f \delta t)\|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^k\| \end{aligned}$$

Therefore, we have:

$$\begin{aligned} &\|s_{\delta t^2}^{(k+1)\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^{k+1}\| \\ &\qquad \leqslant \|\xi_k\| + a\delta t^2 + (1 + L_f \delta t)\|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^k\| \end{aligned}$$

By induction, and by taking $k = \lfloor t/\delta t \rfloor$:

$$\|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^k\| \leqslant \frac{a\delta t}{L_f} \exp(L_f t) + \sum_{j=0}^{\lfloor t/\delta t \rfloor} (1 + \delta t L_f)^j \|\xi_j\|$$

Therefore, if $\varepsilon > 0$, we have :

$$\begin{aligned} &\mathbb{P}\left(\|s_{\delta t^2}^{k\lfloor 1/\delta t \rfloor} - \tilde{s}_{\delta t}^k\| > \varepsilon \right) \\ &\leqslant \mathbb{P}\left(\sum_{j=0}^{\lfloor t/\delta t \rfloor} (1 + \delta t L_f)^j \|\xi_j\| > \varepsilon - \frac{a\delta t}{L_f} \exp(L_f t) \right) \\ &\leqslant \frac{\mathbb{E}\left[\sum_{j=0}^{\lfloor t/\delta t \rfloor} (1 + \delta t L_f)^j \|\xi_j\| \right]}{\varepsilon - \frac{a\delta t}{L_f} \exp(L_f t)} \\ &\leqslant \frac{\mathbb{E}\left[\|\xi\|\right]}{\varepsilon - \frac{a\delta t}{L_f} \exp(L_f t)} \frac{\exp(L_f t)}{L_f \delta t} \end{aligned}$$

But $\mathbb{E}\left[\|\xi\|\right] \leqslant \sqrt{\mathbb{E}\left[\|\xi\|^2\right]} \leqslant \sigma\delta t^{3/2}$. Therefore, we have:

$$\mathbb{P}\left(\|s_{\delta t^2}^{\lfloor t/\delta t\rfloor\lfloor 1/\delta t\rfloor} - \tilde{s}_{\delta t}^k\| > \varepsilon\right) = O(\sqrt{\delta t})$$

Therefore, the process $t \mapsto s_{\delta t^2}^{\lfloor t/\delta t\rfloor\lfloor 1/\delta t\rfloor}$ converges in probability to $\tilde{s}$. Furthermore, by a similar argument than in Lemma 1, we know that the discretized process $\tilde{s}$ converge to the continuous process defined by $\frac{ds}{dt} = f(s_t)$. We can conclude ou result.

5.B Implementation details

All the details specifying our implementation are given in this section. We first give precise pseudo code descriptions for both *Continuous Deep Advantage Updating* (Alg. 5), as well as the variants of DDPG (Alg. 6) and DQN (Alg. 7) used.

Algorithm 5 Continuous DAU

Inputs:
θ, ψ and φ, parameters of V_θ, $\bar{A}_\psi$ and π_φ.
π^{explore} and $\nu_{\delta t}$ defining an exploration policy.
$\mathbf{opt}_V$, $\mathbf{opt}_A$, $\mathbf{opt}_\pi$, $\alpha^V\delta t$, $\alpha^A\delta t$ and $\alpha^\pi\delta t$, optimizers and learning rates.
$\mathcal{D}$, buffer of transitions (s, a, r, d, s'), with d the episode termination signal.
δt and γ, time discretization and discount factor.
nb_epochs number of epochs.
nb_steps, number of steps per epoch.

Observe initial state s^0
$t \leftarrow 0$
for $e = 0, \mathbf{nb\_epochs}$ **do**
  for $j = 1, \mathbf{nb\_steps}$ **do**
    $a^k \leftarrow \pi^{\text{explore}}(s^k, \nu_{\delta t}^k)$.
    Perform a^k and observe $(r^{k+1}, d^{k+1}, s^{k+1})$.
    Store $(s^k, a^k, r^{k+1}, d^{k+1}, s^{k+1})$ in $\mathcal{D}$.
    $k \leftarrow k + 1$
  end for
  for $k = 0, \text{nb\_learn}$ **do**
    Sample a batch of N random transitions from $\mathcal{D}$
    $Q^i \leftarrow V_\theta(s^i) + \delta t\big(\bar{A}_\psi(s^i, a^i) - \bar{A}_\psi(s^i, \pi_\varphi(s^i))\big)$
    $\tilde{Q}^i \leftarrow r^i\delta t + (1 - d^i)\gamma^{\delta t}V_\theta(s'^i)$
    $\Delta\theta \leftarrow \frac{1}{N}\sum_{i=1}^{N} \frac{(Q^i - \tilde{Q}^i)\partial_\theta V_\theta(s^i)}{\delta t}$
    $\Delta\psi \leftarrow \frac{1}{N}\sum_{i=1}^{N} \frac{(Q^i - \tilde{Q}^i)\partial_\psi\big(\bar{A}_\psi(s^i, a^i) - \bar{A}_\psi(s^i, \pi_\varphi(s^i))\big)}{\delta t}$
    $\Delta\varphi \leftarrow \frac{1}{N}\sum_{i=1}^{N} \partial_a\bar{A}_\psi(s^i, \pi_\varphi(s^i))\partial_\varphi\pi_\varphi(s^i)$
    Update θ with $\mathbf{opt}_V$, $\Delta\theta$ and learning rate $\alpha^V\delta t$.
    Update ψ with $\mathbf{opt}_A$, $\Delta\psi$ and learning rate $\alpha^A\delta t$.
    Update φ with $\mathbf{opt}_\pi$, $\Delta\varphi$ and learning rate $\alpha^\pi\delta t$.
  end for
end for

For DDPG and DQN, two different settings were experimented with:

- One with time discretization scalings, to keep the comparison fair. In this setting, the discount factor is still scaled as $\gamma^{\delta t}$, rewards are scaled as $r\delta t$, and learning rates are scaled to obtain parameter updates of order δt. As RMSprop is used for all experiments, this amounts to using a learning rate scaling as $\alpha^Q = \tilde{\alpha}^Q\delta t$, $\alpha^\pi = \tilde{\alpha}^\pi\delta t$.
- One without discretization scalings. In that case, only the discount factor is scaled as $\gamma^{\delta t}$, to prevent unfair shortsightedness. All other parameters are set with a reference $\delta t_0 = 1e - 2$. For instance, for all

Algorithm 6 DDPG

Inputs:
ψ and φ, parameters of Q_ψ and π_φ.
ψ' and φ', parameters of target networks $Q_{\psi'}$ and $\pi_{\varphi'}$.
π^{explore} and ν defining an exploration policy.
$\mathbf{opt}_Q$, $\mathbf{opt}_\pi$, α^Q and α^π, optimizers and learning rates.
$\mathcal{D}$, buffer of transitions (s, a, r, d, s'), with d the episode termination signal.
γ discount factor.
τ target network update factor.
nb_epochs number of epochs.
nb_steps, number of steps per epoch.

Observe initial state s^0
$t \leftarrow 0$
for $e = 0, \mathbf{nb\_epochs}$ **do**
  for $j = 1, \mathbf{nb\_steps}$ **do**
    $a^k \leftarrow \pi^{\text{explore}}(s^k, \nu^k)$.
    Perform a^k and observe $(r^{k+1}, d^{k+1}, s^{k+1})$.
    Store $(s^k, a^k, r^{k+1}, d^{k+1}, s^{k+1})$ in $\mathcal{D}$.
    $k \leftarrow k + 1$
  end for
  for $k = 0, \text{nb\_learn}$ **do**
    Sample a batch of N random transitions from $\mathcal{D}$
    $\tilde{Q}^i \leftarrow r^i + (1 - d^i)\gamma Q_{\psi'}(s'^i, \pi_{\varphi'}(s'^i))$
    $\Delta\psi \leftarrow \frac{1}{N} \sum_{i=1}^{N} \left(Q^i - \tilde{Q}^i\right) \partial_\psi Q(s^i, a^i)$
    $\Delta\varphi \leftarrow \frac{1}{N} \sum_{i=1}^{N} \partial_a Q_\psi(s^i, \pi_\varphi(s^i)) \partial_\varphi \pi_\varphi(s^i)$
    Update ψ with $\mathbf{opt}_Q$, $\Delta\psi$ and learning rate α^Q.
    Update φ with $\mathbf{opt}_\pi$, $\Delta\varphi$ and learning rate α^π.
    $\psi' \leftarrow \tau\psi' + (1 - \tau)\psi$
    $\varphi' \leftarrow \tau\varphi' + (1 - \tau)\varphi$
  end for
end for

Algorithm 7 DQN

Inputs:
ψ parameter of Q_ψ.
ψ', parameters of target networks $Q_{\psi'}$.
π^{explore} and ν defining an exploration policy.
$\mathbf{opt}_Q$, α^Q optimizer and learning rate.
$\mathcal{D}$, buffer of transitions (s, a, r, d, s'), with d the episode termination signal.
γ discount factor.
τ target network update factor.
nb_epochs number of epochs.
nb_steps, number of steps per epoch.

Observe initial state s^0
$t \leftarrow 0$
for $e = 0, \mathbf{nb\_epochs}$ **do**
  for $j = 1, \mathbf{nb\_steps}$ **do**
    $a^k \leftarrow \pi^{\text{explore}}(s^k, \nu^k)$.
    Perform a^k and observe $(r^{k+1}, d^{k+1}, s^{k+1})$.
    Store $(s^k, a^k, r^{k+1}, d^{k+1}, s^{k+1})$ in $\mathcal{D}$.
    $k \leftarrow k + 1$
  end for
  for $k = 0, \text{nb\_learn}$ **do**
    Sample a batch of N random transitions from $\mathcal{D}$
    $\tilde{Q}^i \leftarrow r^i + (1 - d^i)\gamma \max_{a'} Q_{\psi'}(s'^i, a')$
    $\Delta\psi \leftarrow \frac{1}{N} \sum_{i=1}^{N} \left(Q^i - \tilde{Q}^i\right) \partial_\psi Q(s^i, a^i)$
    Update ψ with $\mathbf{opt}_Q$, $\Delta\psi$ and learning rate α^Q.
    $\psi' \leftarrow \tau\psi' + (1 - \tau)\psi$
  end for
end for

δt's, the reward perceived is $r * \delta t_0$, and similarily for learning rates, $\alpha^Q = \tilde{\alpha}^Q \delta t_0$, $\alpha^\pi = \tilde{\alpha}^Q \delta t_0$. These scalings don't depend on the discretization, but perform decently at least for the highest discretization.

5.B.1 Global hyperparameters

The following hyperparameters are maintained constant throughout all our experiments,

- All networks used are of the form

```
Sequential(
    Linear(nb_inputs, 256),
    LayerNorm(256),
    ReLU(),
    Linear(256, 256),
    LayerNorm(256),
    ReLU(),
    Linear(256, nb_outputs)
).
```

  Policy networks have an additional tanh layer to constraint action range. On certain environments, network inputs are normalized by applying a mean-std normalization, with mean and standard deviations computed on each individual input features, on all previously encountered samples.
- $\mathcal{D}$ is a cyclic buffer of size 1000000.
- **nb_steps** is set to 10, and 256 environments are run in parallel to accelerate the training procedure, totalling 2560 environment interactions between learning steps.
- **nb_learn** is set to 50.
- The physical γ is set to 0.8. It is always scaled as $\gamma^{\delta t}$ (even for unscaled DQN and DDPG).
- N, the batch size is set to 256.
- RMSprop is used as an optimizer without momentum, and with $\alpha = 1 - \delta t$ (or $1 - \delta t_0$ for unscaled DDPG and DQN).
- Exploration is always performed as described in the main text. The OU process used as parameters $\kappa = 7.5$, $\sigma = 1.5$.
- Unless otherwise stated, $\alpha_1 := \tilde{\alpha}^Q = \alpha^V = \alpha^A = 0.1$, $\alpha_2 := \tilde{\alpha}^\pi = \alpha^\pi = 0.03$.
- $\tau = 0.9$

5.B.2 Environment dependent hyperparameters

We hereby list the hyperparameters used for each environment. Continuous actions environments are marked with a (C), discrete actions environments with a (D).

- **Ant (C)**: State normalization is used. Discretization range: $[0.05, 0.02, 0.01, 0.005, 0.002]$.
- **Cheetah (C)**: State normalization is used. Discretization range: $[0.05, 0.02, 0.01, 0.005, 0.002]$
- **Bipedal Walker (C)**[4]: State normalization is used, $\alpha_2 = 0.02$. Discretization range: $[0.01, 0.005, 0.002, 0.001]$.
- **Cartpole (D)**: $\alpha_2 = 0.02$, $\tau = 0$. Discretization range: $[0.01, 0.005, 0.002, 0.001, 0.0005]$.
- **Pendulum (C)**: $\alpha_2 = 0.02$, $\tau = 0$. Discretization range: $[0.01, 0.005, 0.002, 0.001, 0.0005]$.

5.C Additional results

Additional results mentionned in the text are presented in this section.

[4]The reward for Bipedal Walker is modified not to scale with δt. This does not introduce any change for the default setup.

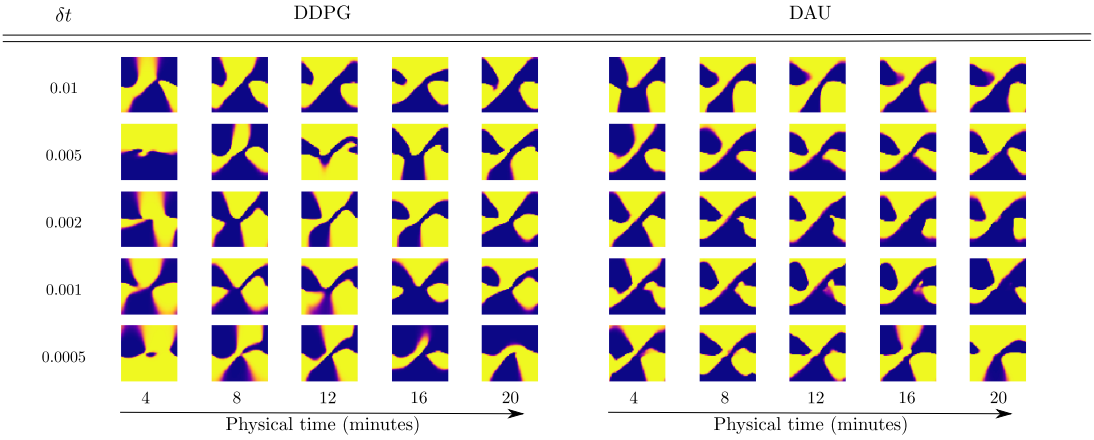


Figure 5.3: Policies obtained by DDPG (unscaled version) and AU at different instants in physical time of training on the pendulum swing-up environment. Each image represents the policy learnt by the policy network, with x-axis representing angle, and y-axis angular velocity. The lighter the pixel, the closer to 1 the action, the darker, the closer to -1.

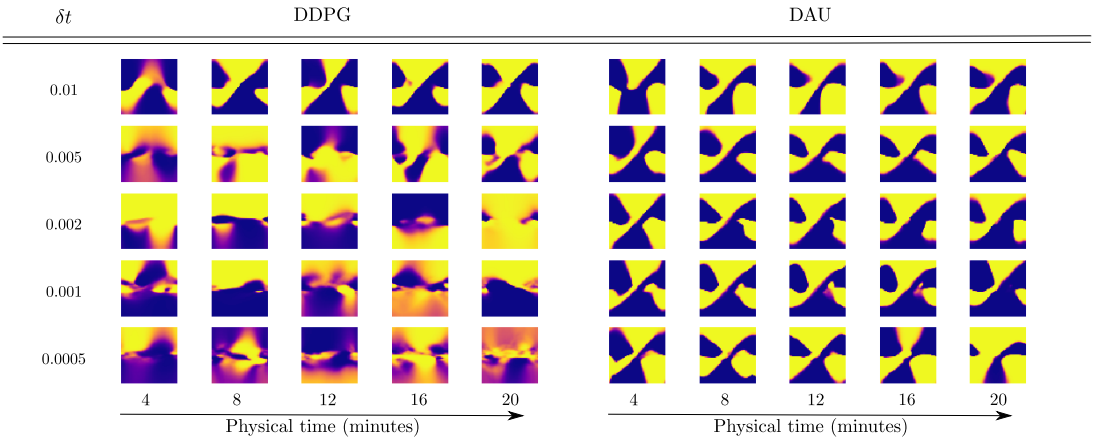


Figure 5.4: Policies obtained by DDPG (scaled version) and AU at different instants in physical time of training on the pendulum swing-up environment. Each image represents the policy learnt by the policy network, with x-axis representing angle, and y-axis angular velocity. The lighter the pixel, the closer to 1 the action, the darker, the closer to -1.

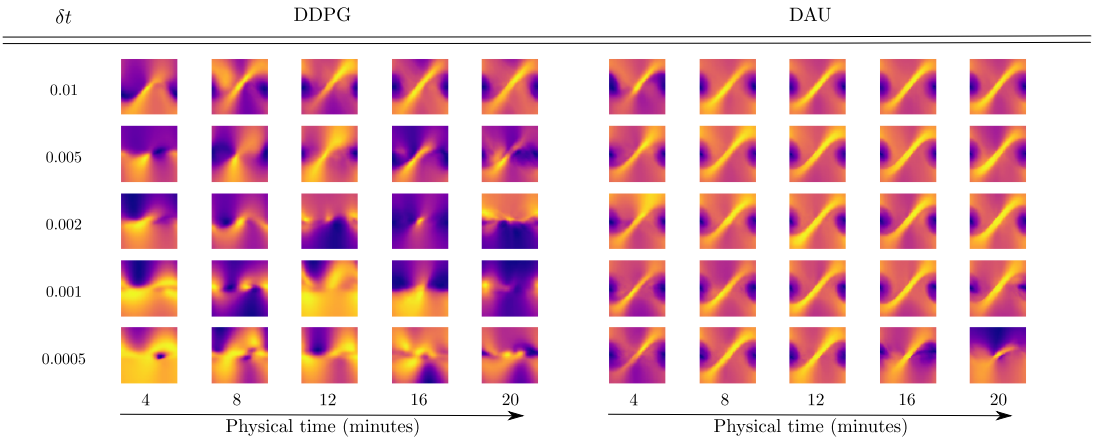


Figure 5.5: Value functions obtained by DDPG (scaled version) and AU at different instants in physical time of training on the pendulum swing-up environment. Each image represents the value function learnt, with x-axis representing angle, and y-axis angular velocity. The lighter the pixel, the higher the value.

Part IV

Policy Evaluation via the Successor States Operator

Chapter 6

The Successor States Operator

We now introduce the successor states operator, and its main properties. We start in Section 6.1 with the case of finite state spaces, for which all the objects can be seen as vectors or matrices. This is the case treated in (Dayan, 1993). We then formally define the successor states operator in general state spaces (Section 6.2).

6.1 The Successor State Matrix in a Finite State Space

We first consider a Markov Process $\mathcal{M}$ with a finite state space $\mathcal{S} = \{1, ..., S\}$, a transition matrix P_{s_1,a,s_2}, and a policy $\pi(a|s)$. We consider the transition matrix $P^{\pi}_{s_1,s_2} = \sum_{a\in\mathcal{A}} \pi(a|s_1)P_{s_1,a,s_2}$ defined when using policy π in the Markov Decision process. In the following, unless explicitly specified, the policy π is always fixed, and we only consider the Markov Reward Process obtained when using policy π in $\mathcal{M}$. For simplicity, we write P for P^{π}, V for V^{π}, and M for M^{π}.

Informally, for finite state spaces (Dayan, 1993), given two states s_1 and s_2 in a Markov process, the successor state matrix M is a matrix whose entry $M_{s_1 s_2}$ can be interpreted as:

- $M_{s_1 s_2}$ is the *expected discounted time* spent at s_2 if starting the process at s_1. Thus, lines of M contain the expected future occupancy measure for all starting points.
- $M_{s_1 s_2}$ is also the value function at s_1 if the reward is located at s_2. Thus, columns of M contain the value functions of all single-target rewards.

We can now define the Successor State Operator M for finite state spaces.

Definition-Theorem 6.1. *Let $\mathcal{M}$ be a finite Markov Reward Process, and P its transition matrix.*

We define the Laplace operator as $\Delta := \mathrm{Id} - \gamma P$. Then, Δ is invertible, and we define the successor state matrix M as:

$$M := \Delta^{-1} = (\mathrm{Id} - \gamma P)^{-1} \tag{6.1.1}$$

Moreover, we have:

$$M = \sum_{t\geqslant 0} (\gamma P)^t \tag{6.1.2}$$

The formal proof will be given in the general case, with Theorem 6.4. Still, we will give here formal statements and proofs in the tabular case, as the tabular case is more simple than the general continuous state space case, and makes it easier to understand.

The following proposition shows the connection between M and the expected future occupancy of a trajectory:

Proposition 6.2. *Let $\mathcal{M}$ be a finite Markov Reward Process.*

We consider a stochastic trajectory $(S_t)_{t\geqslant 0}$ starting at $S_0 = s_0$ and following policy π. We define an independent random variable $T \sim \texttt{Geom}(1-\gamma)$ ($\mathbb{P}(T = t) = (1-\gamma)\gamma^t$), and consider S_T, the agent's position at random time step T. Then, we have:

$$\mathbb{P}(S_T = s) = (1-\gamma)M_{s_0,s} \tag{6.1.3}$$

Hence, $M_{s_0,s}$ measures the expected time a trajectory starting in s_0 will spend in s, if the time is discounted by γ. We say that $M_{s_0,\cdot}$ is the discounted future occupancy measure starting from s_0.

Proof.

$$\mathbb{P}(S_T = s) = \sum_{t\geqslant 0}(1-\gamma)\gamma^t\mathbb{P}(S_t = s) = (1-\gamma)\sum_{t\geqslant 0}\gamma^t P^t_{s_0 s} = (1-\gamma)M_{s_0,s} \tag{6.1.4}$$

The following proposition makes the connection between the successor state matrix and the value function. This proposition will allow us to derive policy evaluation algorithms via the successor state operator.

Proposition 6.3. *Let $\mathcal{M}$ be a finite Markov Reward Process. Then M is the matrix that transforms a reward function into the corresponding value function: for any reward function $R \in \mathbb{R}^S$, the associated value function in $\mathcal{M}$ with policy π is:*

$$V = MR. \tag{6.1.5}$$

Proof. Let $(S_t, R_t)_{t\geqslant 0}$ be a stochastic trajectory starting at $S_0 = s_0$ and following policy π. We have:

$$V(s_0) = \mathbb{E}\left[\sum_{t\geqslant 0}\gamma^t R_t\right] \quad = \sum_{t\geqslant 0}\gamma^t\sum_s (P^t)_{s_0,s}R(s) = \sum_s\left(\sum_{t\geqslant 0}\gamma^t(P^t)_{s_0,s}\right)R(s) \quad = (MR)_{s_0} \tag{6.1.6}$$

In the next section, we generalize these results to general (continuous) state spaces.

6.2 The Successor State Operator in a General State Space

M is also well-defined in general state spaces, using the Markov process formalism of Section 3.2.1, as follows. This extends (Dayan, 1993) to arbitrary $\mathcal{S}$.

Interpreting P and the successor state as operators on functions over $\mathcal{S}$ clarifies the statements of the results below. We follow the standard theory of Markov kernels (Hairer, 2010, 2006). We denote by $B(\mathcal{S})$ the set of bounded measurable functions on $\mathcal{S}$. P acts on such functions as follows. If f is a function in $B(\mathcal{S})$, Pf is defined as

$$(Pf)(s) := \mathbb{E}_{s'\sim P(s,\mathrm{d}s')}\left[f(s')\right]. \tag{6.2.1}$$

This is compatible with the matrix notation Pf in the finite case, viewing f as a vector. In the text, we freely identify Markov kernels with the corresponding operators.

If P_1 and P_2 are two such Markov kernel operators, their composition P_1P_2 is again a Markov kernel operator, and coincides with matrix multiplication in the finite case. In particular, P^n represents n steps of P. The identity operator Id corresponds to always staying in the same state, namely, a transition operator $P(s, \mathrm{d}s') = \delta_s(\mathrm{d}s')$ with δ_s the Dirac measure at s.

We denote $\Delta := \mathrm{Id} - \gamma P$, the discrete Laplace operator of the Markov process. Finally, if A is an operator acting on functions over $\mathcal{S}$, we denote its inverse by A^{-1}, if it exists.

Theorem 6.4. *The* successor state operator *M of a Markov reward process is defined as*

$$M := \sum_{n\geqslant 0}\gamma^n P^n, \qquad M(s_1, \mathrm{d}s_2) = \sum_{n\geqslant 0}\gamma^n P^n(s_1, \mathrm{d}s_2). \tag{6.2.2}$$

where $P^0 := \mathrm{Id}$ as on operator on $B(\mathcal{S})$, and equivalently $P^0(s_1, \mathrm{d}s_2) = \delta_{s_1}(\mathrm{d}s_2)$ as a measure.

- *For each s_1, $M(s_1, \mathrm{d}s_2)$ is a measure on s_2, with total mass $M(s_1, \mathcal{S}) = \frac{1}{1-\gamma}$. If $(S_t)_{t\geqslant 0}$ is a stochastic trajectory starting at $S_0 = s_0$, and if we define an independent random variable $T \sim \texttt{Geom}(1-\gamma)$ ($\mathbb{P}(T = t) = (1-\gamma)\gamma^t$), and consider S_T, the agent's position at random time step T. Then, we have:*

$$s_t \sim (1-\gamma)M(s_0, \mathrm{d}s) \tag{6.2.3}$$

- *M is a well-defined operator over the set $B(\mathcal{S})$ of bounded measurable functions on $\mathcal{S}$. Moreover,*

$$M = (\mathrm{Id} - \gamma P)^{-1} \tag{6.2.4}$$

  as operators over $B(\mathcal{S})$

- *For any reward function R, we have:*

$$V = MR \tag{6.2.5}$$

Similarly to the successor state operator $M(s_1, \mathrm{d}s_2)$, we can define a state-action successor operator $Q(s, a, \mathrm{d}s_2)$, the expected discounted time spent in $\mathrm{d}s_2$ if starting from s and doing action a. While $M(s, \mathrm{d}s_2)$ is comparable to the value function $V(s)$, the state-action successor operator $Q(s, a, \mathrm{d}s_2)$ is comparable to the Q function. This will be discussed in Section 7.9.

Proof. By the definition of M in (6.2.2), for any measurable set $A \subset \mathcal{S}$, for any $s \in \mathcal{S}$, $M(s, A)$ is defined as

$$M(s, A) = \sum_{n\geqslant 0} \gamma^n P^n(s, A). \tag{6.2.6}$$

Since each $P^n(s, \cdot)$ is a probability distribution, $P^n(s, A) \leqslant 1$ so that this sum of non-negative terms is bounded by $\frac{1}{1-\gamma}$, and therefore the sum converges. $M(s, \cdot)$ is a positive measure as a convergent sum of positive measures (σ-additivity for $M(s, \cdot)$ follows from the dominated convergence theorem). Its total mass is $M(s, \mathcal{S}) = \sum_{n\geqslant 0} \gamma^n P(s, \mathcal{S}) = \sum_{n\geqslant 0} \gamma^n = \frac{1}{1-\gamma}$.

As a positive measure with finite mass, $M(s, \cdot)$ acts on bounded measurable functions, just like P, via $(Mf)(s) = \int f(s')M(s, \mathrm{d}s')$. Since M has mass $\frac{1}{1-\gamma}$ for any s, this integral is bounded by $\frac{1}{1-\gamma} \sup f$, so that $\sup Mf \leqslant \frac{1}{1-\gamma} \sup f$ for any function $f \in B(\mathcal{S})$. Thus, M is well-defined as an operator from $B(\mathcal{S})$ to $B(\mathcal{S})$.

As an operator, one has $\gamma PM = \gamma P \sum_{n\geqslant 0} \gamma^n P^n = \sum_{n\geqslant 1} \gamma^n P^n$. Therefore, $(\mathrm{Id} - \gamma P)M = M - \gamma PM = \sum_{n\geqslant 0} \gamma^n P^n - \sum_{n\geqslant 1} \gamma^n P^n = \gamma^0 P^0 = \mathrm{Id}$ (the sums converge absolutely by the same boundedness argument as before, thus justifying the infinite sum manipulations). This proves that M is a right inverse of $\mathrm{Id} - \gamma P$ as operators. The computation is identical for the left inverse; therefore, M and $\mathrm{Id} - \gamma P$ are inverses as operators on $B(\mathcal{S})$.

Finally, let R be any (bounded, measurable) reward function. Since $(\mathrm{Id} - \gamma P)M = \mathrm{Id}$, one has $(\mathrm{Id} - \gamma P)MR = R$ namely $MR = R + \gamma PMR$. This proves that $V = MR$ satisfies the Bellman equation $V = R + \gamma PV$, and so MR is the value function of the Markov reward process.

If $\rho_0(\mathrm{d}s_0)$ is an initial state distribution, we can define:

$$\nu(\mathrm{d}s) = \int_{s_0} \rho(\mathrm{d}s_0) M(s_0, \mathrm{d}s) \tag{6.2.7}$$

The measure $\nu(\mathrm{d}s)$ is the expected discounted occupancy measure for trajectories starting from states sampled from $\rho_0(\mathrm{d}s_0)$. It typically appears in the policy gradient formula (see Section 1.5). Ensuring that this measure is well spread in the state space, by adding an entropy regularizer on $\nu(\mathrm{d}s)$ can help exploration and regularize policy optimization (Islam et al., 2019).

Successor state operator and paths in the Markov process M can be interpreted as *paths* in the Markov process: $M(s_1, \mathrm{d}s_2)$ represents the number of paths from s_1 to s_2, weighted by their probability and discounted by their length. This will be relevant to compare the algorithms in Chapters 7, 8 and 9. Indeed, in the finite-state case and using matrix notation, $P^n_{ss'}$ is the probability to go from s to s' in n steps; therefore

$$M_{ss'} = \sum_{n\geqslant 0} \gamma^n (P^n)_{ss'} = \sum_{n\geqslant 0} \gamma^n \sum_{s=s_0, s_1, \ldots, s_{n-1}, s_n = s'} P_{s_0 s_1} \cdots P_{s_{n-1} s_n} \tag{6.2.8}$$

$$= \sum_{p \text{ path from } s \text{ to } s'} \gamma^{|p|} \, \mathbb{P}(p) \tag{6.2.9}$$

where, if $p = (s_0, ..., s_n)$ is a path, $\mathbb{P}(p) = P_{s_0 s_1} \cdots P_{s_{n-1} s_n}$ is its probability and $|p| = n$ its length. The same holds with integrals instead of sums in continuous spaces.

6.3 Representing and learning the successor state operator.

With continuous states, M cannot be represented as a matrix. Instead, we will learn a function of a pair of states. Namely, we will learn a parametric model of M via its density with respect to a data distribution ρ over states:

$$M(s_1, \mathrm{d}s_2) \approx m_\theta(s_1, s_2)\rho(\mathrm{d}s_2) \tag{6.3.1}$$

where m_θ is a function over pairs of states, depending smoothly on some parameter θ. The data distribution ρ is unknown, but all algorithms below only require the ability to sample states from ρ. Typically, ρ can be a large buffer of states observed from past trajectories. From this parametrization, we have:

$$V(s_1) = \mathbb{E}_{s_2 \sim \rho}[m_\theta(s_1, s_2)R(s_2)] \tag{6.3.2}$$

Hence, if we learned a model m_θ, we can estimate the value function with the equation above by estimating this expectation. This is described in more details in Chapter 12.

An other possibility for parametrizing M is via $M(s_1, \mathrm{d}s_2) \approx \delta_{s_1}(\mathrm{d}s_2) + \tilde{m}_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ where δ_{s_1} is the Dirac measure at s_1, and where $\tilde{m}_\theta$ is a learned density. The motivation for the second version is as follows. In continuous spaces, M has a singular part, corresponding to the immediate reward in V, and to the term Id in the series for M: for each s_1, the measure $M(s_1, \cdot)$ comprises a Dirac mass at s_1. In continuous spaces, this singular part cannot be represented as $\tilde{m}(s_1, s_2)\rho(\mathrm{d}s_2)$ for a smooth function $\tilde{m}$. But since this singular part δ_{s_1} is known, we can just parametrize and learn the absolutely continuous part $m(s_1, s_2)$. Thus, the second version may represent M exactly (at least if P is smooth), while in general the first version cannot. Still, the first version may provide useful approximations.

In the following, we will only study algorithms for the first parametrization $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ because this parametrization is simpler for some use cases such as policy gradient methods. More details on the second parametrization with $\tilde{m}$ are explained in our preprint (Blier et al., 2021). We will derive well-principled algorithms to learn $m(s_1, s_2)$ from observations of the Markov process.

In this text, we define several algorithms for learning m_θ: the extension of temporal difference (TD) to successor states (Chapter 7); backward TD for successor states (Chapter 8); and second-order-type methods (Chapter 9). The matrix-factorized forward-backward parametrization $m_\theta(s_1, s_2) = F_\theta(s_1)^\top B_\theta(s_2)$ has many additional properties and is treated in Chapter 11. Chapter 12 gives more details about the ways to use M to learn value functions and policies.

6.4 Norms on successor or transition operators

Both $P(s, \mathrm{d}s')$ and the successor state operator $M(s, \mathrm{d}s')$ are measures on s' that depend on s. We will use the following norms on such objects: if $\rho(\mathrm{d}s)$ is some reference probability measure on $\mathcal{S}$, and $M_1(s, \mathrm{d}s')$ and $M_2(s, \mathrm{d}s')$ are two such objects, we define

$$\|M_1 - M_2\|_\rho^2 := \mathbb{E}_{s \sim \rho,\, s' \sim \rho}\,(m_1(s, s') - m_2(s, s'))^2 \tag{6.4.1}$$

where $m_1(s, s') := M_1(s, \mathrm{d}s')/\rho(\mathrm{d}s')$ is the density of M_1 with respect to ρ (if it exists; if not, the norm is infinite), and likewise for M_2.

In finite environments, we can see that this norm corresponds to the expected norm between value functions if the reward is a sparse reward located in a random target state sampled from ρ. If s_{tar} is a target state, we consider the sparse reward $R^{s_{\text{tar}}}(s) = \frac{1}{\rho(s_{\text{tar}})}\mathbb{1}_{s=s_{\text{tar}}}$, which is the

sparse reward located in s_{tar} and scaled such that $\mathbb{E}_{s\sim\rho}\left[R^{s_{\text{tar}}}(s)\right] = 1$. We know that the value function corresponding to the reward $R^{s_{\text{tar}}}$ is $V^{\pi}_{s_{\text{tar}}}(s) = (M^{\pi} \cdot R^{s_{\text{tar}}})(s)$. Hence, if M_1 and M_2 are two estimates of M^{π}, we can use $V_1^{s_{\text{tar}}} := M_1 \cdot R^{s_{\text{tar}}}$ and $V_2^{s_{\text{tar}}} := M_2 \cdot R^{s_{\text{tar}}}$ as estimates of the value function if the reward is $R^{s_{\text{tar}}}$. In that case, we have:

$$\begin{aligned}
\|M_1 - M_2\|_{\rho}^2 &= \mathbb{E}_{s_1\sim\rho,\, s_2\sim\rho}\left(\frac{M_1(s_1,s_2)}{\rho(s_2)} - \frac{M_2(s_1,s_2)}{\rho(s_2)}\right)^2 \\
&= \mathbb{E}_{s_1\sim\rho,\, s_2\sim\rho}\left((M_1 \cdot R^{s_2})(s_1) - (M_2 \cdot R^{s_2})(s_1)\right)^2 \\
&= \mathbb{E}_{s_{\text{tar}}\sim\rho}\left[\|V_1^{s_{\text{tar}}} - V_2^{s_{\text{tar}}}\|_{\rho}^2\right]
\end{aligned}$$

Hence the norm $\|.\|_{\rho}$ on successor operators defined in equation (6.4.1) corresponds to a standard quantity for value functions. We will use other similar L_2 norms, which will always be specified.

We will also use the *total variation* norm

$$\|M_1 - M_2\|_{\rho,\text{TV}} := \mathbb{E}_{s\sim\rho}\,\|M_1(s,\cdot) - M_2(s,\cdot)\|_{\text{TV}} \tag{6.4.2}$$

with $\|p_1 - p_2\|_{\text{TV}} := \sup_{A\subset\mathcal{S}} |p_1(A) - p_2(A)|$ the usual total variation distance between two measures.

Additionally, in Chapter 17, we will updates based on the generalized KL-divergence between measures, and more generally on adaptive norms. While this is discussed in the context of multi-goal RL, it can be applied to most of updates in the following chapters for the successor states operator.

6.5 Observation model.

We assume access to observations from the Markov reward process, such as a fixed dataset of stored transitions, or some sampled trajectories. Each observation is a triplet (s, r, s') with $s' \sim P(s, \mathrm{d}s')$ and $r \sim \mathcal{R}(.|s)$ the associated reward. Consecutive observations need not be independent. We denote by $\rho(\mathrm{d}s)$ be the distribution of states s coming from the observations. We cannot choose the states s: ρ is unknown and we do not make any assumptions on it. For instance, if we have access to trajectories from the process, obtained by some exploration policy, then ρ would be the law of states visited under that policy. If we just have a finite dataset of transitions, ρ would be the (unknown) law from which this dataset was sampled.

6.6 Some related work on successor states.

The successor state operator is linked to various existing objects under various names. It has even been identified in the neurosciences (Stachenfeld et al., 2017) as an interesting model for some functions of the hippocampus. In this section we briefly introduce some previous works related to the successor states operator. Others will be mentionned throughout this thesis.

The fundamental matrix of a Markov Process For discount factor $\gamma = 1$, the successor matrix M is known as the *fundamental matrix* (Kemeny and Snell, 1960; Brémaud, 1999; Grinstead and Snell, 1997) of a Markov process. If the process is *absorbing*, the matrix $(\mathrm{Id} - P)$ is invertible and the fundamental matrix is defined as $Z = (\mathrm{Id} - P)^{-1}$ (Grinstead and Snell, 1997, Definition 11.3). In that case, the expected absorbing time t_s starting from state s is related to Z via $t_s = (Z \cdot \mathbb{1})_s = \sum_{s'} Z_{ss'}$ (Grinstead and Snell, 1997, Theorem 11.6). If the process is non-absorbing but an ergodic measure ρ, then the matrix $(\mathrm{Id} - P + \mathbb{1} \cdot \rho^{\top})$ is invertible ($\mathbb{1} \cdot \rho^{\top}$ is the matrix with every rows equal to ρ), the *fundamental matrix* is defined as $Z := (\mathrm{Id} - P + \mathbb{1} \cdot \rho^{\top})^{-1}$. The fundamental matrix encodes many properties of the Markov chain, such as the *mean first passage* A_{ij} (which is the expected time to reach j from i) which can be computed as: $A_{ij} = \frac{1}{\rho_j}(Z_{jj} - Z_{ij})$ (Grinstead and Snell, 1997, Theorem 11.16).

The successor states operator defined with $\gamma < 1$ as above and the fundamental matrix of a markov process can be related as follows: If M_γ is the successor states operator defined with $\gamma < 1$, $M_\gamma = (\mathrm{Id} - \gamma P)^{-1}$, and Z is the fundamental matrix of the process $(\mathrm{Id} - P + \mathbb{1} \cdot \rho^\top)^{-1}$ which is independent of γ, then we have:

$$M_\gamma = Z + \frac{1}{1-\gamma} \mathbb{1} \cdot \rho^\top + O_{\gamma \to 1}(1-\gamma) \tag{6.6.1}$$

Hence, the dependency of the successor states operator with respect to *gamma*, when $\gamma \to 1$, is only via the rank one term $\frac{1}{1-\gamma} \mathbb{1} \cdot \rho^\top$. If we consider the value function for $\gamma < 1$, we hence have:

$$V_\gamma(s) = (M_\gamma \cdot R)(s) = (ZR)(s) + \frac{C}{1-\gamma} + O_{\gamma \to 1}(1-\gamma) \tag{6.6.2}$$

where $C = \mathbb{E}_{s' \sim \rho}[R(s')]$. Therefore, the value of state s, when $\gamma \to$ is approximately equal to $(ZR)(s)$ (independent of s) up to a constant independent of s (thus useless for policy improvement). While many algorithms described in this thesis could be applied to learn directly Z and not M_γ for $\gamma < 1$, we decided to work on M_γ to stay closer to RL practice.

Learning successor states by (forward) temporal difference is mentioned in (Dayan, 1993) for the tabular case and with linear approximations; the parametric case has never been derived as far as we know. Our second-order algorithms in Chapter 9 are based on an implicit process estimation approach, which corresponds to the very specific tabular case of LSTD (Bradtke and Barto, 1996).

Successor representations In a deep learning context, several recent works have used the related *successor representations* (Kulkarni et al., 2016), e.g., for transfer (Barreto et al., 2017; Borsa et al., 2018; Zhang et al., 2017b; Lehnert et al., 2017; Ma et al., 2018; Barreto et al., 2020), hierarchical RL (Machado et al., 2018), exploration (Machado et al., 2019), or for off-policy evaluation via Marginalized Importance Sampling (Fujimoto et al., 2021).

In Deep Successor Representation (Kulkarni et al., 2016), we consider a representation function $\varphi : \mathcal{S} \mapsto \mathbb{R}^k$. The objective is then to learn $m(s)$ the expected discounted representation of future states, starting from a state s:

$$m(s) = \mathbb{E}\left[\sum_{t \geqslant 0} \gamma^t \varphi(s_t) | s_0 = s\right] \tag{6.6.3}$$

The successor representation is linked to the successor states operator as follows: we have:

$$m(s) = (M \cdot \varphi)(s) = \int_{s_2} M(s, \mathrm{d}s_2) \varphi(s_2) \tag{6.6.4}$$

The main differences between the two objects are as follows: First the successor states operator M does not depend on any representation function. Then, the successor states operator is a *measure*, while the successor representation m is an *expected value*. The former represents the distribution of future states, while the latter represents the expected value of the representation φ. Learning distributions of future outcomes in an environment, rather than an expected value, is similar the the *distributional RL* principle (Bellemare et al., 2022).

The successor representation m satisfies a Bellman equation, as:

$$m(s) = \varphi(s) + \gamma \mathbb{E}_{s' \sim P(.|s)}[m(s')] \tag{6.6.5}$$

and can be learned via temporal difference. Similarly, we will see in Chapter 7 that the successor states operator satisfies a (forward) Bellman equation, and can be learned via temporal difference, both in the tabular and parametric case. Additionally, the successor states operator M is a richer object than the successor states representation, and satisfies more Bellman equation:

we will derive the *Backward* Bellman equation in Chapter 8, as well as the Bellman–Newton equation in Chapter 9. These equations has no equivalent for the value function, or the successor representation.

In Deep Successor Representation (Kulkarni et al., 2016), the representation φ is learned together with its successor value $m(s)$. In that case, $\varphi = m = 0$ is a fixed point of the method. Hence, a reconstruction loss (such as pixel reconstruction) must be used to prevent collapse during learning. On the other side, our learning methods for the successor states operator M only depend of the observed transitions, and not of a reconstruction loss.

Multi-task and unsupervised RL. Successor states provide the value function for every goal state: this is related to learning multiple RL tasks (Sutton et al., 2011; Schaul et al., 2015; Jin et al., 2020; Pinto and Gupta, 2017) which performs joint V- or Q-learning for a set of goals. In Part V of this thesis, we apply the mathematical tools derived for the successor operator to the case of multi-goal RL. Some of the objects developed in this thesis were further extended to a broader multi-task setting by (Touati and Ollivier, 2021).

More generally, successor state learning comes in the context of *unsupervised RL*, in which relevant features of the environment are learned without the supervision of a reward signal. Many works have suggested that unsupervised RL improves sample efficiency (Sun et al., 2019). Notably, this includes model-based methods (François-Lavet et al., 2018). Contrary to the latter, successor state learning does not require synthesizing accurate future states.

Chapter 7

TD Algorithms for Deep Successor State Learning

In this chapter, we will define a Temporal Difference algorithm for the successor states operator. As described in the previous chapter, we consider a *model* $M_\theta(s_1, \mathrm{d}s_2) := m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ parameterized by its density $m_\theta(s_1, s_2)$ with respect to a measure ρ. Our approach will be similar to standard temporal difference for the value function, as described in Section 1.4.2:

First, we derive a Bellman equation for the successor states operator. This equation is called the Forward Bellman equation. We use the name *forward* in order to highlight the difference with the Backward Bellman equation introduced in Chapter 8. This is similar to the Bellman equation on the value function V introduced in Proposition 1.1, but for the successor states operator.

Then, we derive a temporal difference algorithm with function approximators for the density $m_\theta(s_1, s_2)$ using the Forward Bellman Equation. This is similar to temporal difference with function approximation described in Proposition 1.2.

We then discuss the convergence properties of this algorithm, how to extend it to standard variants of TD (TD with a target network, TD(n), TD for the action-value function $Q(s, a)$).

7.1 The Forward Bellman Equation

Theorem 7.1 (Bellman equation for successor states)**.** *The successor state operator M is the only operator which satisfies the Bellman equation*

$$M = \mathrm{Id} + \gamma P M \tag{7.1.1}$$

Proof. An operator M' satisfies the left Bellman equation $M' = \mathrm{Id} + \gamma P M'$ if and only if $M' - \gamma P M' = \mathrm{Id}$, or $(\mathrm{Id} - \gamma P)M' = \mathrm{Id}$, namely, M' is a right inverse of $\mathrm{Id} - \gamma P$. By Theorem 6.4, $\mathrm{Id} - \gamma P$ is invertible and its inverse is M. Therefore, the only right inverse of $\mathrm{Id} - \gamma P$ is M.

This Bellman equation makes sense, as operators, on any state space, discrete or continuous. In finite spaces, each column of the matrix M contains the value function for a reward located at a specific target state, and the Bellman equation for M is just the collection of the standard Bellman equations for every target state; the Id term is the reward for reaching state s when the target is s.

This Bellman operator on M has the same contractivity properties as the usual Bellman operator.

Proposition 7.2 (Contractivity of the Bellman operator on M)**.** *Equip the space of functions $B(\mathcal{S})$ with the sup norm $\|f\|_\infty := \sup_{s \in S} |f(s)|$. Equip the space of bounded linear operators from $B(\mathcal{S})$ to $B(\mathcal{S})$ with the operator norm $\|M\|_{\mathrm{op}} := \sup_{f \in B(\mathcal{S}),\, f \neq 0} \|Mf\|_\infty / \|f\|_\infty$.*

Then the Bellman operator $M \mapsto \mathrm{Id} + \gamma P M$ is γ-contracting for this norm.

Consequently, for any learning rate $\eta \leqslant 1$, iterated application of the Bellman operator $M \leftarrow (1-\eta)M + \eta(\mathrm{Id} + \gamma PM)$ converges to the successor state operator.

Proof. By definition of the operator P, for any function f we have $\|Pf\|_\infty = \sup_s \int f(s')P(s, \mathrm{d}s') \leqslant \sup_{s'} f(s') = \|f\|_\infty$, so that P is 1-contracting. Therefore, for any bounded operator M and function f, one has $\|PMf\|_\infty \leqslant \|Mf\|_\infty \leqslant \|M\|_{\mathrm{op}} \|f\|_\infty$, so that $\|PM\|_{\mathrm{op}} \leqslant \|M\|_{\mathrm{op}}$ for any M. Therefore, given two operators M and M', one has $\|(\mathrm{Id} + \gamma PM) - (\mathrm{Id} + \gamma PM')\|_{\mathrm{op}} = \gamma \|P(M - M')\|_{\mathrm{op}} \leqslant \gamma \|M - M'\|_{\mathrm{op}}$.

Consider the sequence defined as $M_{t+1} := (1-\eta)M_t + \eta(\mathrm{Id} + \gamma PM_t)$. We have:

$$\begin{aligned}
\|M_{t+1} - M\|_{\mathrm{op}} &\leqslant (1-\eta)\|M_t - M\|_{\mathrm{op}} + \eta\|(\mathrm{Id} + \gamma PM_t) - M\|_{\mathrm{op}} \\
&= (1-\eta)\|M_t - M\|_{\mathrm{op}} + \eta\|(\mathrm{Id} + \gamma PM_t)(M_t - M)\|_{\mathrm{op}} \\
&\leqslant (1 - \eta + \eta\gamma)\|M_t - M\|_{\mathrm{op}}
\end{aligned}$$

By induction, M_t converges to M.

7.2 Forward TD for Successor States: Tabular Case

Given that the Bellman equation on M is a collection of ordinary Bellman equations for every target state, an obvious algorithm to learn M in finite state spaces is to perform ordinary TD in parallel for all these single-state rewards, as in (Dayan, 1993). Let s_{tar} be some target state and consider the reward $\mathbb{1}_{s_{\mathrm{tar}}}$. Upon observing a transition $s \to s'$, ordinary TD for this reward updates V by $V_s \leftarrow V_s + \eta\,\delta V_s$, where η is some learning rate and $\delta V_s = \mathbb{1}_{s=s_{\mathrm{tar}}} + \gamma V(s') - V(s)$. Performing TD in parallel for every column of M with target state s_{tar} is equivalent to the following (Dayan, 1993).

Definition 7.3 (Tabular temporal difference for successor states)**.** The TD algorithm for M, in a finite state space, maintains M as a matrix. Upon observing a transition $s \to s'$ in the Markov process, M is updated by $M \leftarrow M + \eta\,\delta M$ where η is a learning rate and δM has entries

$$\delta M_{ss_2} := \mathbb{1}_{s=s_2} + \gamma M_{s's_2} - M_{ss_2} \qquad \forall s_2 \tag{7.2.1}$$

Equivalently, the update on M when observing a transition $s \to s'$ is updating all values of M of line s, as:

$$\forall s_2 \quad : \quad M_{ss_2} \leftarrow M_{ss_2} + \eta\,(\gamma M_{s's_2} - M_{ss_2}) \tag{7.2.2}$$

$$M_{ss} \leftarrow M_{ss} + \eta \tag{7.2.3}$$

Equivalence between TD on V and policy evaluation via TD on M. In the tabular case, if the reward is deterministic, learning V via ordinary TD is equivalent to learning V via the matrix product $V = MR$ with M learned via tabular TD, as follows. This is formalized in the following proposition:

Proposition 7.4. *Consider a Markov reward process with deterministic reward R. Initialize an estimate $\hat{V}$ of V to 0 and an estimate $\hat{M}$ of M to 0. Each time a transition $s \to s'$ with reward $r_s = R_s$ is observed, update $\hat{V}$ via ordinary TD and $\hat{M}$ via TD for successor states, with learning rate η, namely*

$$\hat{V}_s \leftarrow \hat{V}_s + \eta\left(r_s + \gamma\hat{V}_{s'} - \hat{V}_s\right), \tag{7.2.4}$$

$$\hat{M}_{ss_2} \leftarrow \hat{M}_{ss_2} + \eta\left(\mathbb{1}_{s=s_2} + \gamma\hat{M}_{s's_2} - \hat{M}_{ss_2}\right) \qquad \forall s_2. \tag{7.2.5}$$

Then at every time step, $\hat{V} = \hat{M}R$.

Proof. By induction on the time step. This is true at time 0 thanks to the initialization. If $\hat{V} = \hat{M}R$ at one

time step, then the update of $\hat{M}R$ at the next time step is

$$\begin{aligned}(\hat{M}R)_s &= \sum_{s_2} \hat{M}_{ss_2} R_{s_2} \\ &\leftarrow \sum_{s_2} \left(\hat{M}_{ss_2} R_{s_2} + \eta \left(\mathbb{1}_{s=s_2} + \gamma \hat{M}_{s's_2} - \hat{M}_{ss_2}\right) R_{s_2}\right) \\ &= (\hat{M}R)_s + \eta \left(R_s + \gamma (\hat{M}R)_{s'} - (\hat{M}R)_s\right)\end{aligned}$$

which is the same update as $\hat{V}_s$. The values at the other states are not updated. Therefore, if $\hat{V} = \hat{M}R$ before the update, this still holds after the update.

However, this equivalence does not hold with function approximation, which introduces generalization between states. In continuous state spaces, if we consider a reward in a single state s^{tar}, the probability to reach exactly that state is 0. Hence, applying parametric TD naively in parallel for every target state would always provide reward 0 in continuous environments. The parametric TD updates we present below are not equivalent to this naive TD: they have the same expectation but avoid the zero-reward problem.

7.3 Forward TD for Successor States: Function Approximation

In continuous environments, it is not possible to store M as a matrix. But we can maintain a model m_θ of the density of M, as explained in Section 6.3. As in usual parametric TD, we learn θ by defining an "ideal" update given by the Bellman equation, and update θ so that M gets closer to it. As introduced in Section 6.5, we assume we are observing transitions (s, s') such that $s \sim \rho(\mathrm{d}s)$ and $s' \sim P(s'|s)$, where ρ is a reference measure. It can be a fixed dataset, or the distributions of observed states along trajectories sampled via policy π in the environment. We do not assume we know ρ, but only we have access to samples from the distribution ρ.

Theorem 7.5 (TD for successor states with function approximation)**.** *Let $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $M(s_1, \mathrm{d}s_2)$. Consider $M^{\text{tar}} = \mathrm{Id} + \gamma P M_\theta$, a target estimate for M defined via the Forward Bellman equation.*

Let (s, s') be a sample of the environment such that $s' \sim P(s'|s)$ and $s_2 \sim \rho$ is sampled independently, we define $\widehat{\delta\theta}_{F\text{-}TD}(s, s', s_2)$ as:

$$\widehat{\delta\theta}_{F\text{-}TD}(s, s', s_2) := \partial_\theta m_\theta(s, s) + \partial_\theta m_\theta(s, s_2)\left(\gamma m_\theta(s', s_2) - m_\theta(s, s_2)\right) \tag{7.3.1}$$

Then $\widehat{\delta\theta}_{F\text{-}TD}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), s_2\sim\rho}\left[\widehat{\delta\theta}_{F\text{-}TD}(s, s', s_2)\right] = -\frac{1}{2}\partial_\theta \|M_\theta - M^{\text{tar}}\|_\rho^2 \tag{7.3.2}$$

where the norm $\|.\|_\rho$ is introduced in (6.4.1)*. In particular, the true successor state operator M is a fixed point of this update: if $M_\theta = M$, then $\mathbb{E}\left[\widehat{\delta\theta}_{F\text{-}TD}\right] = 0$.*

The update $\widehat{\delta\theta}_{\text{F-TD}}$ defined in Equation (7.3.1) allow us to define a algorithm learning the successor state operator, similar to standard temporal difference algorithm for the value function with function approximators: see Algorithm 8.

This algorithm uses a transition $s \to s'$ and one additional random state s_2, independent from s and s'. The Bellman–Newton update (Section 9.5) will use two additional random states s_1 and s_2 (but no additional transition). The law of s_2 is ρ, which means s_2 is just another state sampled from the distribution ρ. For instance, if the distribution ρ is sampling from a finite dataset of sampled trajectories $(s_t)_{t\geqslant 0}$, when observing a transition $s_t \to s_{t+1}$, additional independent state samples can be obtained by using states $s_{t'}$ at times t' independent from t (such as a random $t' \leqslant t$). This requires maintaining a replay buffer of observed states.

Algorithm 8 Forward TD for successor states with function approximation.

Input: Policy $\pi(a|s)$, randomly initialized model $m_\theta(s_1, s_2)$; `TransitionMemory`, maximum number of time steps T
repeat
  for K trajectories **do**
    Get an initial state s_0 from the environment.
    for $0 \leqslant t \leqslant T$ steps do **do**
      Sample $a_t \sim \pi(.|s_t)$, execute a_t and observe s_{t+1}
      Store in the transition memory the transition `TransitionMemory` $\leftarrow (s_t, s_{t+1})$
    end for
    for L gradient steps **do**
      Sample $(s, s') \sim$ `TransitionMemory` and $(s_2, \_) \sim\sim$ `TransitionMemory`
      $\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2) := \partial_\theta m_\theta(s, s) + \partial_\theta m_\theta(s, s_2)\,(\gamma m_\theta(s', s_2) - m_\theta(s, s_2))$
      Stochastic gradient step: $\theta \leftarrow \theta + \eta\widehat{\delta\theta}_{\text{F-TD}}$.
    end for
  end for
until end of learning

Proof. In this proof, we freely go back and forth between M or M^{tar} as measure-valued functions, and M or M^{tar} as operators on bounded functions. Notably, the operator Id corresponds to the measure $\delta_{s_1}(\mathrm{d}s_2)$.

Here, there is a hidden mathematical subtlety with continuous states. Indeed, in that case, M_θ is absolutely continuous with respect to ρ, while M^{tar} is not, due to the Id term, as discussed in Section 6.3. This makes the norm $J(\theta) = \frac{1}{2}\left\|M_\theta - M^{\text{tar}}\right\|_\rho^2$ infinite (see its definition in (6.4.1)). However, the *gradient* of this norm is actually still well-defined. To handle this rigorously, observe that the loss $J(\theta)$ is equal to $\frac{1}{2}\|M_\theta\|_\rho^2 - \langle M_\theta, M^{\text{tar}}\rangle_\rho + \frac{1}{2}\left\|M^{\text{tar}}\right\|_\rho^2$ and has the same minima and the same gradients as the loss $J'(\theta) = \frac{1}{2}\|M_\theta\|_\rho^2 - \langle M_\theta, M^{\text{tar}}\rangle_\rho$ for a given M^{tar}. Namely, J and J' differ by a constant in the finite case, and by an "infinite constant" in the continuous case. We will work with the loss J', which is finite even in the continuous case.

Here $\langle M_1, M_2\rangle_\rho = \int_{s,s_2} \frac{M_1(s,\mathrm{d}s_2)}{\rho(\mathrm{d}s_2)} \frac{M_2(s,\mathrm{d}s_2)}{\rho(\mathrm{d}s_2)}\,\rho(\mathrm{d}s)\rho(\mathrm{d}s_2)$ is the dot product associated with the norm (6.4.1). Since the integrand can be rewritten as $\frac{M_1(s,\mathrm{d}s_2)}{\rho(\mathrm{d}s_2)}\rho(\mathrm{d}s)M_2(s,\mathrm{d}s_2)$, it is well-defined as soon as at least one of M_1 or M_2 is absolutely continuous with respect to ρ. Namely,

$$\langle M_1, M_2\rangle_\rho = \int_{s,s_2} \frac{M_1(s,\mathrm{d}s_2)}{\rho(\mathrm{d}s_2)}\rho(\mathrm{d}s)\, M_2(s,\mathrm{d}s_2). \tag{7.3.3}$$

Let us compute $J'(\theta) = \frac{1}{2}\|M_\theta\|_\rho^2 - \langle M_\theta, M^{\text{tar}}\rangle_\rho$. We define $\bar\theta$ such that $\bar\theta$ has the same value than θ ($\bar\theta = \theta$) but is not differentiated through θ. By definition of $M^{\text{tar}} = \mathrm{Id} + \gamma P M_{\bar\theta}$, and by definition of the action of the operator P, we have

$$M^{\text{tar}}(s,\mathrm{d}s_2) = \delta_s(\mathrm{d}s_2) + \gamma\int_{s'} P(s,\mathrm{d}s')M_{\bar\theta}(s',\mathrm{d}s_2) \tag{7.3.4}$$

$$= \delta_s(\mathrm{d}s_2) + \gamma\,\mathbb{E}_{s'\sim P(s,\mathrm{d}s')}[m_{\bar\theta}(s',s_2)\rho(\mathrm{d}s_2)] \tag{7.3.5}$$

by definition of the model $M_\theta(s_1,\mathrm{d}s_2) = m_\theta(s_1,s_2)\rho(\mathrm{d}s_2)$. Therefore, by (7.3.3),

$$\begin{aligned}\langle M_\theta, M^{\text{tar}}\rangle_\rho &= \int_{s,s_2} m_\theta(s,s_2)\,\rho(\mathrm{d}s)\,M^{\text{tar}}(s,\mathrm{d}s_2)\\ &= \int_s m_\theta(s,s)\,\rho(\mathrm{d}s) + \gamma\int_{s,\,s',\,s_2} m_\theta(s,s_2)\,m_{\bar\theta}(s',s_2)\,\rho(\mathrm{d}s)\,P(s,\mathrm{d}s')\,\rho(\mathrm{d}s_2)\end{aligned}$$

thanks to (7.3.5). Next, since $M_\theta(s,\mathrm{d}s_2) = m_\theta(s,s_2)\rho(\mathrm{d}s_2)$, the definition of the norm (6.4.1) yields

$$\frac{1}{2}\|M_\theta\|_\rho^2 = \frac{1}{2}\int_{s,s_2} m_\theta(s,s_2)^2\,\rho(\mathrm{d}s)\,\rho(\mathrm{d}s_2). \tag{7.3.6}$$

Collecting, and rewriting the integrals as expectations, we find

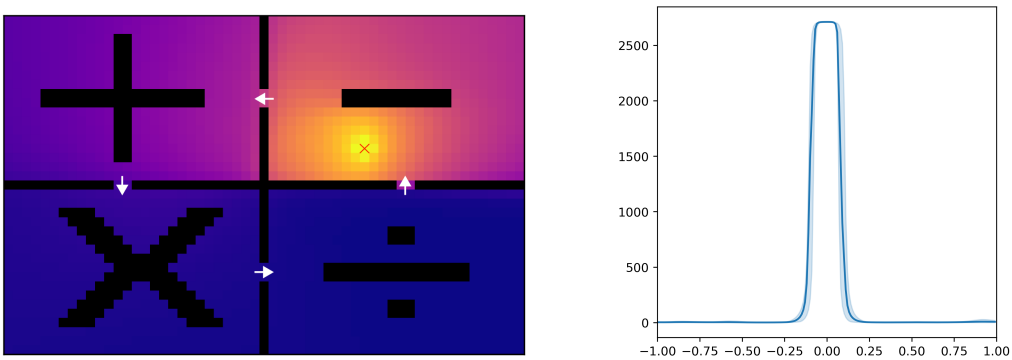
$$J'(\theta) \;=\; \frac{1}{2}\mathbb{E}_{s\sim\rho,\,s_2\sim\rho}\left[m_\theta(s,s_2)^2 - m_\theta(s,s)\right] \;-\; \gamma\mathbb{E}_{s\sim\rho,\,s'\sim P(s,\mathrm{d}s'),\,s_2\sim\rho}[m_\theta(s,s_2)\,m_{\bar\theta}(s',s_2)] \tag{7.3.7}$$

hence

$$\begin{aligned}
&\partial_\theta J'(\theta) = \\
&= \mathbb{E}_{s\sim\rho,\, s_2\sim\rho}\left[\partial m_\theta(s,s_2)\, m_\theta(s,s_2) - \partial m_\theta(s,s)\right] - \gamma \mathbb{E}_{s\sim\rho,\, s'\sim P(s,\mathrm{d}s'),\, s_2\sim\rho}[\partial m_\theta(s,s_2)\, m_{\bar\theta}(s',s_2)] \\
&= \mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), s_2\sim\rho}\left[\widehat{\delta}\theta_{\text{F-TD}}(s,s',s_2)\right]
\end{aligned} \tag{7.3.8}$$

7.4 Toy experiments in continuous environments

In this section, we check in simple environments that the parametric TD algorithm (Algorithm 8) introduced for M manage to learn M, parameterized by the two-state function $m_\theta(s_1, s_2)$. The experiments details are in Appendix 7.A. First, we use a discrete maze (Figure 7.1a), treated in a parametric way: each state is represented by its continuous 2D coordinates (x, y) encoding its position in the maze. These coordinates are fed to a multilayer perceptron that is trained to learn the successor state operator m_θ. We sample transitions $s \to s'$ from 64 simultaneous trajectories, together with 256 random states s_2 at each step.



(a) The learned successor states in a parametric maze, for a random policy. The model $m_\theta(s_1, s_2)$ is a MLP taking as input the (x, y) 2D coordinates.

(b) High dimensional torus: $s \mapsto m_\theta(s, s_{\text{target}})$ evaluated on 1 dimensional lines going through the target state $s_{\text{target}} = 0$.

Figure 7.1: Qualitative results, showing the learned successor states estimates $m_\theta(s_1, s_2)$ in toy continuous environments.

Then, we consider a scenario where the reward is located at a known state, but too sparse to ever be visited. This is obtained simply by increasing dimension, so that the probability to randomly visit a state is small. More precisely, we consider the 8D torus $[0; 1]^8$ (Figure 7.1b). The policy is a Gaussian random walk with $\sigma = .05$. , and we let the reward be nonzero whenever the agent is at distance less than $\varepsilon = .1$ from some target state s_{tar}.

In practice, an agent will almost never observe the reward. Indeed, the probability to hit the reward with the random policy is $\approx 1/25,000,000$: no reward is ever collected, thus a TD method has no signal and learns nothing. With successor states, the value function can be estimated as $m(s, s_{\text{tar}})$ (assuming, in this scenario, that the target state s_{tar} is known, namely, its representation, as used for the input to the MLP, is known). The training trajectories never get very close to the target state, yet successor state learning correctly infers the value function for this target state, by generalization from how other states have been reached. We can observe this by selecting a target state s_{target}, and plotting $m_\theta(s, s_{\text{tar}})$ for s on random lines going through s_{tar} (Figure 7.1b). Qualitatively, after $100,000$ steps in the environment we learned the right shape of the successor state.

7.5 Convergence properties for TD on successor states

In the tabular case, we proved that the Forward TD update on M is equivalent to the forward TD update on the value function, simultaneously for every reward $R_{s_{\text{tar}}}(s) = \mathbb{1}_{s_{\text{tar}}=s}$ located on

a target state. Hence, theoretical guarantees on the convergence of TD for the value function can be transferred to a similar guarantee on M. We briefly present two such results: convergence of tabular TD and convergence of TD on-policy for any parametrization if the process is reversible. In each case, we refer to the original works for additional technical conditions:

Tabular case In the tabular case, forward TD on M (Definition 7.3) converges, with pairs (s, s_2) sampled at each step from essentially any selection scheme (stochastic or deterministic) that ensures every pair is selected infinitely often, and with suitable learning rates. Indeed, TD on M is equivalent to learning simultaneously the value function for every reward $R_{s_{\text{tar}}}(s) = \mathbb{1}_{s_{\text{tar}}=s}$, and every of these value function estimation via temporal difference converge in the tabular case, if every pair is selected infinitely often, and with suitable learning rates (Tsitsiklis, 1994).

Reversible processes For TD on the value function with arbitrary parametric families V_θ, convergence is known assuming that the Markov operator P is *reversible*, namely, that ρ is its steady-state distribution and further satisfies the *detailed balance* condition $\rho(\mathrm{d}s)P(\mathrm{d}s'|s) = \rho(\mathrm{d}s')P(\mathrm{d}s|s')$, in other words, steady-state flows from state s to s' and s' to s are equal. Then, parametric TD is a stochastic gradient descent of a global loss between the approximate and true value function (Ollivier, 2018). This result extends to MDPs which are "reversible enough" (Brandfonbrener and Bruna, 2019).

We can now extend this result to the successor states operator. Define the loss function:

$$\ell(\theta) := (1-\gamma)\left\|M_\theta - M^\pi\right\|_\rho^2 + \gamma\left\|M_\theta - M^\pi\right\|_{\text{Dir}}^2 \tag{7.5.1}$$

where M^π is the true successor states operator, and $\|.\|_{\text{Dir}}^2$ is the *Dirichlet norm*, defined as follows: if $M(s_1, \mathrm{d}s_2)$ is absolutely continuous with respect to ρ (with $M(s_1, \mathrm{d}s_2) = m(s_1, s_2)\rho(\mathrm{d}s_2)$) is

$$\|M(s_1, \mathrm{d}s_2)\|_{\text{Dir}}^2 := \int_{s,s',s_2} \rho(\mathrm{d}s)P(\mathrm{d}s'|s)\rho(\mathrm{d}s_2)(m(s', s_2) - m(s, s_2))^2. \tag{7.5.2}$$

and is infinite if M is not absolutely continuous with respect to ρ.

In continuous environments, the loss function $\ell(\theta)$ is infinite, because M^π is not absolutely continuous with respect to ρ. Still, as already explained in the proof of Theorem 7.5, its gradients are well-defined, and this issue can be handled formally. We will not give details here and refer to the proof of Theorem 7.5.

Applying the result of (Ollivier, 2018) to successor states via the state-goal process yields the following: The parametric TD step for M (Theorem 7.5) is equal to the gradient of this loss, $-\frac{1}{2}\partial_\theta \ell(\theta)$. This is a global loss between the parametric model and the true value M^π, contrary to the loss in Theorem 7.5 which uses a loss with respect to the right-hand-side of the Bellman equation, which depends on the current estimate. Thus, in the reversible case with ρ the stationary distribution, parametric TD for M converges to a local minimum of the global loss (7.5.1), under the general conditions for convergence of stochastic gradient descent.

7.6 TD with a target network

A standard technique in practice for policy evaluation is to maintain two models for the value function: the current network V_θ and a *target* network $V_{\bar\theta}$. Typically, $\bar\theta$ can be obtained from previous values of θ via Polyak averaging (Lee and He, 2019). The same technique can be used for learning the successor state operator. Theorem 7.5 can be adapted as follow for the use of target networks:

Theorem 7.6 (TD with target network)**.** *Let $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $M(s_1, \mathrm{d}s_2)$, and $\bar\theta$ be a* target *parameter. Consider $M^{\text{tar}} = \mathrm{Id} + \gamma P M_{\bar\theta}$, a target*

estimate for M defined via the Forward Bellman equation, with value $\bar{\theta}$. We define for every (s, s', s_2):

$$\widehat{\delta\theta}_{F\text{-}TD}(s, s', s_2) := \partial_\theta m_\theta(s, s) + \partial_\theta m_\theta(s, s_2)\left(\gamma m_{\bar{\theta}}(s', s_2) - m_\theta(s, s_2)\right) \tag{7.6.1}$$

Then $\widehat{\delta\theta}_{F\text{-}TD}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), s_2\sim\rho}\left[\widehat{\delta\theta}_{F\text{-}TD}(s, s', s_2)\right] = -\frac{1}{2}\partial_\theta \|M_\theta - M^{\mathrm{tar}}\|_\rho^2. \tag{7.6.2}$$

The proof is identical to that of Theorem 7.5, but with $\bar{\theta}$ instead of θ for M^{tar}.

7.7 TD(n) on the Successor States

A multistep, horizon-h version of TD on M can be defined by iterating the Bellman equation, which yields $M = \mathrm{Id} + \gamma P + \cdots + \gamma^{h-1}P^{h-1} + \gamma^h P^h M$. This requires being able to observe h consecutive transitions from the process. The corresponding parametric update is as follows, and is equivalent to standard TD for $h = 1$:

Theorem 7.7 (Multi-step TD for successor states with function approximation)**.** *Let $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $M(s_1, \mathrm{d}s_2)$. For $h \geqslant 1$, define a target update of M via the horizon-h Bellman equation,*

$$M^{\mathrm{tar}} := \mathrm{Id} + \gamma P + \cdots + \gamma^{h-1}P^{h-1} + \gamma^h P^h M_\theta. \tag{7.7.1}$$

Let $(s_0, s_1, ..., s_h, s_{\mathrm{tar}})$ be a sample of the environment such that $s_{t+1} \sim P(\mathrm{d}s|s_t)$ and $s_{\mathrm{tar}} \sim \rho$ is sampled independently, we define $\widehat{\delta\theta}_{F\text{-}TD(n)}(s_0, s_1, ..., s_h, s_{\mathrm{tar}})$ as:

$$\begin{aligned}\widehat{\delta\theta}_{F\text{-}TD(n)}(s_0, s_1, ..., s_h, s_{\mathrm{tar}}) := \sum_{t=0}^{h-1} \gamma^t \partial m_\theta(s_0, s_t) + \\ + \partial_\theta m_\theta(s_0, s_{\mathrm{tar}})\left(\gamma^h m_\theta(s_h, s_{\mathrm{tar}}) - m_\theta(s_0, s_{\mathrm{tar}})\right)\end{aligned} \tag{7.7.2}$$

Then $\widehat{\delta\theta}_{F\text{-}TD(n)}$ is an unbiased estimate of the n-step Bellman error:

$$\mathbb{E}_{(s_0, s_1, ..., s_h, s_{\mathrm{tar}})}\left[\widehat{\delta\theta}_{F\text{-}TD(n)}(s_0, s_1, ..., s_h, s_{\mathrm{tar}})\right] = -\partial_\theta \|M_\theta - M^{\mathrm{tar}}\|_\rho^2$$

Similarly to standard TD, we can define the corresponding tabular update, to get more intuition. The corresponding F-TD(n) update for a tabular model $M_{s_1 s_2}$ when observing a sub-trajectory $(s_0, s_1, ..., s_h)$ is:

$$\begin{aligned}&\forall s_{\mathrm{tar}} \quad : \quad M_{s_0 s_{\mathrm{tar}}} \leftarrow M_{s_0 s_{\mathrm{tar}}} + \eta\left(\gamma^h M_{s_h s_{\mathrm{tar}}} - M_{s_0 s_{\mathrm{tar}}}\right) \\ &M_{s_0 s_0} \leftarrow M_{s_0 s_0} + \eta \\ &M_{s_0 s_1} \leftarrow M_{s_0 s_1} + \gamma\eta \\ &... \\ &M_{s_0 s_{h-1}} \leftarrow M_{s_0 s_{h-1}} + \gamma^{h-1}\eta\end{aligned} \tag{7.7.3}$$

The intuition is the following: from a sub-trajectory $(s_0, s_1, ..., s_h)$, we can increase the value functions estimates corresponding to sparse rewards in target states $s_0, ..., s_{h-1}$, and propagate the reward for any additional target state s_{tar} from s_h to s_{tar}.

Proof. The proof is very similar to the proof of Theorem 7.5. We define $J'(\theta) = \frac{1}{2}\|M_\theta\|_\rho^2 - \langle M_\theta, M^{\mathrm{tar}}\rangle$. We have:

$$J'(\theta) = \frac{1}{2}\int_{s_0, s_{\mathrm{tar}}} \rho(\mathrm{d}s_0)\rho(\mathrm{d}s_{\mathrm{tar}}) m_\theta(s_0, s_{\mathrm{tar}})^2 - \int_{s_0, s_{\mathrm{tar}}} \rho(\mathrm{d}s_0) m_\theta(s_0, s_{\mathrm{tar}}) M^{\mathrm{tar}}(s_0, s_{\mathrm{tar}}) \tag{7.7.4}$$

The second part can be written as:

$$\int_{s_0,s_{\text{tar}}} \rho(\mathrm{d}s_0) m_\theta(s_0,s_{\text{tar}}) M^{\text{tar}}(s_0,s_{\text{tar}}) =$$

$$= \int_{s_0,s_{\text{tar}}} \rho(\mathrm{d}s_0) \left(\sum_{t=0}^{h-1} \gamma^t P^t(s_0,\mathrm{d}s_{\text{tar}}) + \gamma^h P^h M_{\bar{\theta}}(s_0,\mathrm{d}s_{\text{tar}}) \right) m_\theta(s_0,s_{\text{tar}})$$

$$= \int_{s_0,s_1,\dots,s_h,s_{\text{tar}}} \rho(\mathrm{d}s_0) P(\mathrm{d}s_1|s_0)...P(\mathrm{d}s_h|s_{h-1}) \rho(\mathrm{d}s_{\text{tar}}) \left(\sum_{t=0}^{h-1} \gamma^t m_\theta(s_0,s_t) + \gamma^h m_{\bar{\theta}}(s_h,s_{\text{tar}}) m_\theta(s_0,s_{\text{tar}}) \right)$$

where $\bar{\theta} = \theta$ but $\bar{\theta}$ is not differentiated via ∂_θ. Therefore:

$$-\partial_\theta J'(\theta) = \int_{s_0,s_1,\dots,s_h,s_{\text{tar}}} \rho(\mathrm{d}s_0) P(\mathrm{d}s_1|s_0)...P(\mathrm{d}s_h|s_{h-1}) \rho(\mathrm{d}s_{\text{tar}}) \times$$
$$\times \left(\sum_{t=0}^{h-1} \gamma^t m_\theta(s_0,s_t) + \partial m_\theta(s_0,s_{\text{tar}}) \left(\gamma^h m_\theta(s_h,s_{\text{tar}}) - m_\theta(s_0,s_{\text{tar}}) \right) \right)$$

7.8 Having Targets on Features of the State

Learning M is particularly suitable when the reward is located at a single known goal state g. If the reward is exactly the Dirac reward δ_g located in g, the value function $V(s)$ is exactly to $m(s,g)$ (up to a multiplicative constant independent of the state). If the reward is located in a neighborhood of the target state g ($R(s) = \mathbb{1}_{\|s-g\| \leqslant \varepsilon}$ with $\varepsilon > 0$), we can prove that $m(s,g)$ approximates the value function when $\varepsilon \to 0$ (see Chapter 13 for more details, in the multi-goal setting).

Another scenario is to have a target value for *some* features of the state, not necessarily the whole state itself: namely, the reward is nonzero when some known feature $\varphi(s)$ of state s is equal to some known goal g. Consider for example the FetchPush environment in multi-goal RL (OpenAI, 2018a): a robot learns to push a cube towards a target goal. The state space contains not only the cube position, but also every angles and velocity for the robot, but the reward only depends on a part of the state $\varphi(s)$ which contains the position of the cube. Then, the reward is non-zero only if $\varphi(s) \approx g$.

In that case, it is convenient to learn a smaller object than M, from which the value function can be read directly. This is also useful if the reward is known to depend only on $\varphi(s)$.

Definition 7.8. Let $\varphi \colon S \to \mathbb{R}^k$ be any measurable map. The *successor feature operator* M^φ is defined as follows: for each state s_1, $M^\varphi(s_1,\mathrm{d}g)$ is a measure on $\mathbb{R}^k$ equal to the pushforward of $M(s_1,\mathrm{d}s_2)$ by the map $s_2 \mapsto g = \varphi(s_2)$.

The successor feature operator M^φ in two cases in this thesis will allow us to derive some results in two important cases: in Section 7.9 to derive an TD algorithm for the *state-action successor operator*, and in Chapter 16 to derive a policy evaluation algorithm in the multi-goal setting.

This operator is different from successor representations: here we keep track of the whole future *distribution* of values of $\varphi(s')$ for every starting point s. On the contrary, successor representations learns the expected future value of $\varphi(s')$ for every starting point s.

M^φ can be used to compute the value function of any reward that depends only on $\varphi(s)$.

Proposition 7.9. *Assume that the reward function at state s is equal to $R(\varphi(s))$, namely, it depends only on φ. Let τ be any probability distribution on features in $\mathbb{R}^k$. Assume that M^φ is parameterized as $M^\varphi(s,\mathrm{d}g) = m^\varphi(s,g)\tau(\mathrm{d}g)$. Then the value function of a state s for this reward is*

$$V(s) = \mathbb{E}_{g\sim\tau}[m^\varphi(s,g)R(g)]. \tag{7.8.1}$$

In particular, if the reward is nonzero exactly when the feature $\varphi(s)$ is equal to some target value g, then the value function is proportional to $m^\varphi(s,g)$.

This is useful only if an algorithm to learn m^φ is available. Forward TD can be defined on M^φ, based on the following Bellman equation.

Proof. Assume that the reward function at a state s is equal to $R(\varphi(s))$. By definition of the successor state operator M, the corresponding value function satisfies $V(s) = \int_{s'} R(\varphi(s'))M(s, \mathrm{d}s')$. By definition of the pushforward measure, the latter is equal to $\int_g R(g)M^\varphi(s, \mathrm{d}g)$. If $M^\varphi(s, \mathrm{d}g)$ is equal to $m^\varphi(s, g)\tau(\mathrm{d}g)$ for some probability distribution τ, this rewrites as $\mathbb{E}_{g\sim\tau} m^\varphi(s, g)R(g)$. This proves Proposition 7.9.

Proposition 7.10. *M^φ satisfies the Bellman equation*

$$M^\varphi(s, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma \mathbb{E}_{s'\sim P(s,\mathrm{d}s')} M^\varphi(s', \mathrm{d}g). \tag{7.8.2}$$

Proof. We have the Bellman equation on M:

$$M(s, \mathrm{d}s_2) = \delta_s(\mathrm{d}s_2) + \gamma \mathbb{E}_{s'\sim P(\mathrm{d}s'|s)} M(s', \mathrm{d}s_2)$$

Then take the pushforward by φ on both sides. Because the pushforward operator is linear, we have:

$$\begin{aligned}\varphi_* \mathbb{E}_{s'\sim P(\mathrm{d}s'|s)} M(s', \mathrm{d}s_2) &= \mathbb{E}_{s'\sim P(\mathrm{d}s'|s)} \varphi_* M(s', \mathrm{d}s_2) \\ &= \mathbb{E}_{s'\sim P(\mathrm{d}s'|s)} M^\varphi(s', \mathrm{d}s_2)\end{aligned}$$

Moreover, the pushforward of the Dirac mass at s is the Dirac mass at $\varphi(s)$. This provides the Bellman equation for M^φ.

We can now derive the forward TD algorithm with features of the state. We define the norm $\|.\|_{\rho\otimes\tau}$ similarly to the norm $\|.\|_\rho$ measures. For two feature measures $M_1^\varphi(s, \mathrm{d}g)$ and $M_2^\varphi(s, \mathrm{d}g)$, we define

$$\|M_1^\varphi - M_2^\varphi\|^2_{\rho\otimes\tau} := \mathbb{E}_{s\sim\rho,\, g\sim\tau} \left(m_1^\varphi(s, g) - m_2^\varphi(s, g)\right)^2 \tag{7.8.3}$$

where $m_1^\varphi(s, s') := M_1^\varphi(s, \mathrm{d}s')/\rho(\mathrm{d}s')$ is the density of M_1^φ with respect to τ (if it exists; if not, the norm is infinite), and likewise for M_2^φ.

Theorem 7.11 (TD for successor states on feature space)**.** *Let $\tau := \varphi_*\rho$, the push-forward measure of ρ via φ on the feature space $\mathbb{R}^k$. Let $M_\theta^\varphi(s_1, \mathrm{d}g) = m_\theta^\varphi(s_1, g)\tau(\mathrm{d}g)$ be a current estimate of $M^\varphi(s_1, \mathrm{d}g)$. Consider $(M^\varphi)^{\mathrm{tar}} = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma\mathbb{E}_{s'\sim P(s,\mathrm{d}s')} M_\theta^\varphi(s', \mathrm{d}g)$, a target estimate for M defined via the Forward Bellman equation on feature space.*

Let (s, s') be a sample of the environment such that $s' \sim P(s'|s, a)$ and $g \sim \tau$ is sampled independently, we define $\widehat{\delta\theta}_{F\text{-}TD}(s, s', g)$ as:

$$\widehat{\delta\theta}_{F\text{-}TD\text{-}\varphi}(s, s', g) := \partial_\theta m_\theta^\varphi(s, \varphi(s)) + \partial_\theta m_\theta^\varphi(s, g)\left(\gamma m_\theta^\varphi(s', g) - m_\theta^\varphi(s, g)\right) \tag{7.8.4}$$

Then $\widehat{\delta\theta}_{F\text{-}TD\text{-}\varphi}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), g\sim\tau}\left[\widehat{\delta\theta}_{F\text{-}TD\text{-}\varphi}(s, s', g)\right] = -\frac{1}{2}\partial_\theta \|M_\theta^\varphi - (M^\varphi)^{\mathrm{tar}}\|^2_{\rho\otimes\tau} \tag{7.8.5}$$

In particular, the true successor state operator on feature space M^φ is a fixed point of this update: if $M_\theta^\varphi = M^\varphi$, then $\mathbb{E}\left[\widehat{\delta\theta}_{F\text{-}TD}\right] = 0$.

Proof. We define the model of the successor state (on the state space, not on feature space) as

$$M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2) \tag{7.8.6}$$

with

$$m_\theta(s_1, s_2) = m_\theta^\varphi(s_1, \varphi(s_2)) \tag{7.8.7}$$

For this proof, we first show that the update (7.8.4) is exactly the Forward-TD update (7.3.1) defined in Theorem 7.5, with model (7.8.7). Indeed, update (7.3.1) with model (7.8.7) when observing (s, s', s_2) is:

$$\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2) = \tag{7.8.8}$$

$$= \partial_\theta m_\theta(s, s) + \partial_\theta m_\theta(s, s_2)\left(\gamma m_\theta(s', s_2) - m_\theta(s, s_2)\right) \tag{7.8.9}$$

$$= \partial_\theta m_\theta^\varphi(s, \varphi(s)) + \partial_\theta m_\theta^\varphi(s, \varphi(s_2))\left(\gamma m_\theta^\varphi(s', \varphi(s_2)) - m_\theta^\varphi(s, \varphi(s_2))\right) \tag{7.8.10}$$

$$= \widehat{\delta\theta}_{\text{F-TD-}\varphi}(s, s', \varphi(s_2)) \tag{7.8.11}$$

Since $\tau = \varphi_*\rho$, we have:

$$\begin{aligned}\mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), g\sim\tau}\left[\widehat{\delta\theta}_{\text{F-TD-}\varphi}(s, s', g)\right] &= \mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), s_2\sim\rho}\left[\widehat{\delta\theta}_{\text{F-TD}}(s, s', \varphi(s_2))\right] \\ &= \partial_\theta \|M_\theta - M^{\mathrm{tar}}\|^2_\rho\end{aligned}$$

where $M^{\text{tar}} = \text{Id} + \gamma P M_\theta$.

We now show that $\|M_\theta - M^{\text{tar}}\|_\rho^2 = \|M_\theta^\varphi - (M^\varphi)^{\text{tar}}\|_{\rho\otimes\tau}^2$. More generally, we show that if $M_\theta(s_1, \mathrm{d}s_2) = m^\varphi(s_1, \varphi(s_2))\rho(\mathrm{d}s_2)$ and $M'(s_1, \mathrm{d}s_2)$ is any operator on $B(\mathcal{S})$, we have:

$$\begin{aligned}\langle M_\theta, M'\rangle_\rho &= \int_{s_1,s_2} \rho(\mathrm{d}s_1) m_\theta^\varphi(s_1, \varphi(s_2)) M'(s_1, \mathrm{d}s_2)\\ &= \int_{s_1,g} \rho(\mathrm{d}s_1) m_\theta^\varphi(s_1, g)\varphi_* M'(s_1, \mathrm{d}g)\\ &= \langle M_\theta^\varphi, \varphi_* M'\rangle_{\rho\otimes\tau}\end{aligned}$$

Therefore

$$\|M_\theta - M^{\text{tar}}\|_\rho^2 = \|M_\theta^\varphi - \varphi_* M^{\text{tar}}\|_{\rho\otimes\tau}^2 \tag{7.8.12}$$

$$= \|M_\theta^\varphi - (M^\varphi)^{\text{tar}}\|_{\rho\otimes\tau}^2 \tag{7.8.13}$$

This concludes the proof

Once more, the term $\partial_\theta m_\theta^\varphi(s, \varphi(s))$ makes every transition informative: when visiting state s, we increase the probability to reach the goal $\varphi(s)$.

7.9 Learning the state-action successor operator $Q(s, a, \mathrm{d}s_2)$ via Forward TD

Our results on estimating the $M(s_1, \mathrm{d}s_2)$ can be transferred to $Q(s_1, a, \mathrm{d}s_2)$, the expected discounted time spent in $\mathrm{d}s_2$ if starting from s and doing action a. In this section we formally define the state-action successor operator $Q(s, a, \mathrm{d}s_2)$ and extend the Forward TD algorithm developed for the successor states operator to this object. Our approach is to define the augmented state-action MRP, in which the state space is $\tilde{\mathcal{S}} := \mathcal{S} \times \mathcal{A}$ and the transition operator $\tilde{P}(\mathrm{d}s', \mathrm{d}a'|s, a) = P(\mathrm{d}s'|s, a)\pi(\mathrm{d}a'|s)$, then directly apply Theorems 7.10 and 7.11 to that extended environment to define $Q(s_1, a, \mathrm{d}s_2)$ and its TD algorithm.

7.9.1 The augmented state-action MRP

Let $\mathcal{M}$ be a MDP with state space $\mathcal{S}$, action state $\mathcal{A}$, transition operator $P(\mathrm{d}s'|s, a)$, and $\pi(\mathrm{d}a|s)$ a policy. We consider the augmented MRP with state space $\tilde{\mathcal{S}} := \mathcal{S} \times \mathcal{A}$, and transition operator $\tilde{P}(\mathrm{d}s', \mathrm{d}a'|s, a) = P(\mathrm{d}s'|s, a)\pi(\mathrm{d}a'|s)$, and $\tilde{R}(s, a) = R(s)$. The following proposition shows that the augmented MRP viewpoint is useful to go from results on the value function to results on the action-value function:

Proposition 7.12. *Let $Q(s, a)$ be the action-value function of the MDP $\mathcal{M}$ with policy π. Let $\tilde{V}$ be the value function of the augmented MRP $\tilde{\mathcal{M}}$. Then we have:*

$$Q(s, a) = \tilde{V}((s, a)) \tag{7.9.1}$$

Proof. Here, we denote $P^\pi(\mathrm{d}s'|s) = \int_a \pi(\mathrm{d}a|s)P(\mathrm{d}s'|s, a)$ (usually noted P in this document). We have, for $t \geqslant 1$:

$$\begin{aligned}\tilde{P}^t(\mathrm{d}s_t, \mathrm{d}a_t|s_0, a_0) &= \int_{s_1,a_1,\ldots,s_{t-1},a_{t-1}} \tilde{P}(\mathrm{d}s_1, \mathrm{d}a_1|s_0, a_0)\ldots\tilde{P}(\mathrm{d}s_t, \mathrm{d}a_t|s_{t-1}, a_{t-1})\\ &= \int_{s_1,a_1,\ldots,s_{t-1},a_{t-1}} P(\mathrm{d}s_1|s_0, a_0)\pi(\mathrm{d}a_1|s_1)\ldots P(\mathrm{d}s_t|s_{t-1}, a_{t-1})\pi(\mathrm{d}a_t|s_t)\\ &= \int_{s_1} P(\mathrm{d}s_1|s_0, a_0)(P^\pi)^{t-1}(\mathrm{d}s_t|s_1)\pi(\mathrm{d}a_t|s_t)\end{aligned}$$

Therefore:

$$\begin{aligned}\tilde{V}((s_0,a_0)) &= R(s_0) + \sum_{t\geqslant 1}\gamma^t \int_{s_t,a_t} \tilde{P}^t(\mathrm{d}s_t,\mathrm{d}a_t|s_0,a_0)R(s_t)\\ &= R(s_0) + \int_{s_1} P(\mathrm{d}s_1|s_0,a_0)\sum_{t\geqslant 1}\gamma^t \int_{s_t,a_t}(P^\pi)^{t-1}(\mathrm{d}s_t|s_1)R(s_t)\\ &= R(s_0) + \gamma\int_{s_1} P(\mathrm{d}s_1|s_0,a_0)V(s_1)\\ &= Q(s_0,a_0)\end{aligned}$$

7.9.2 The state-action successor operator

Definition 7.13. Let $\mathcal{M}$ be a MDP, and π a policy. We consider $\tilde{M}(s_1,a_1,\mathrm{d}s_2,\mathrm{d}a_2)$ the successor state operator of the augmented MRP for policy π. Then, we define the state-action successor state operator $Q(s_1,a,\mathrm{d}s_2)$ as:

$$Q(s_1,a,\mathrm{d}s_2) := \int_{a_2}\tilde{M}(s_1,a_1,\mathrm{d}s_2,\mathrm{d}a_2) \tag{7.9.2}$$

Equivalently, Q is the push-forward measure $Q(s_1,a,.) = \psi_*\tilde{M}(s_1,a,.,.)$ where $\psi_*(s,a) = s$.

Similarly to the results on the successor state operator, we have a forward Bellman equation on $Q(s_1,a,\mathrm{d}s_2)$:

Theorem 7.14. *The state-action successor operator $Q(s,a,\mathrm{d}s_2)$ satisfies the forward Bellman equation:*

$$Q(s,a,\mathrm{d}s_2) = \delta_s(\mathrm{d}s_2) + \gamma\mathbb{E}_{s'\sim P(\mathrm{d}s'|s,a),a'\sim\pi(\mathrm{d}a'|s')}Q(s',a',\mathrm{d}s_2) \tag{7.9.3}$$

Proof. This statement is a direct consequence of Theorem 7.10 applied to the augmented MRP, with $\varphi = \psi$.

7.9.3 Learning the state-action successor operator

Similarly to our approach for learning $M(s_1,\mathrm{d}s_2)$, we can define a model of Q as a density with respect to ρ: $Q_\theta(s_1,a,\mathrm{d}s_2) = q_\theta(s_1,a,s_2)\rho(\mathrm{d}s_2)$. The following theorem defines the temporal difference update for the state-action operator with function approximators. We consider the norm $\|.\|_{\rho\otimes\pi}$, similarly to the norm $\|.\|_\rho$ defined is Section 6.4 if $Q_1(s,a,\mathrm{d}s') = q_1(s,a,s')\rho(\mathrm{d}s')$ and $Q_2(s,a,\mathrm{d}s') = q_2(s,a,s')\rho(\mathrm{d}s')$, we define:

$$\|Q_1 - Q_2\|^2_{\rho\otimes\pi} := \int_{s,a,s'}\rho(\mathrm{d}s)\pi(\mathrm{d}a|s)\rho(\mathrm{d}s')\left(q_1(s,a,s') - q_2(s,a,s')\right)^2 \tag{7.9.4}$$

and the norm is infinite if Q_1 or Q_2 has no density with respect to ρ.

Theorem 7.15 (TD for the state-action successor operator)**.** *Let $Q_\theta(s_1,a,\mathrm{d}s_2) = q_\theta(s_1,a,s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $Q(s_1,a,\mathrm{d}s_2)$. Consider*

$$Q^{\mathrm{tar}}(s,a,\mathrm{d}s_2) = \delta_s(\mathrm{d}s_2) + \gamma\mathbb{E}_{s'\sim P(\mathrm{d}s'|s,a),a'\sim\pi(\mathrm{d}a'|s')}Q(s',a',\mathrm{d}s_2), \tag{7.9.5}$$

a target estimate for Q defined via the Forward Bellman equation.

Let (s,a,s',a',s_2) be a sample of the environment such that $a\sim\pi(\mathrm{d}a|s)$, $s'\sim P(s'|s,a)$, $a'\sim\pi(\mathrm{d}a'|s')$ and $s_2\sim\rho$ is sampled independently, we define $\widehat{\delta\theta}_{TD\text{-}Q}(s,a,s',a',s_2)$ as:

$$\widehat{\delta\theta}_{TD\text{-}Q}(s,a,s',s_2) := \partial_\theta q_\theta(s,a,s) + \partial_\theta q_\theta(s,a,s_2)\left(\gamma q_\theta(s',a',s_2) - q_\theta(s,a,s_2)\right) \tag{7.9.6}$$

Then $\widehat{\delta\theta}_{TD\text{-}Q}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}_{s\sim\rho,a\sim\pi(.|s),s'\sim P(\mathrm{d}s'|s,a),a'\sim\pi(.|s'),s_2\sim\rho}\left[\widehat{\delta\theta}_{TD\text{-}Q}(s,a,s',a',s_2)\right] = -\frac{1}{2}\partial_\theta\|Q_\theta - Q^{\mathrm{tar}}\|^2_{\rho\otimes\pi} \tag{7.9.7}$$

Proof. This statement is a direct consequence of Theorem 7.11 applied to the augmented MRP, with $\varphi = \psi$.

In the next Chapter, we will define the Backward Bellman equation and the Backward TD algorithm. This algorithm is similar to the Forward TD algorithm, but there is no equivalent update for the value function.

Appendix

7.A Experimental details

Here are some details on the experiments presented in Figure 7.1.

Continuous mazes. First, we considered a random walk in the non-reversible maze presented in Figure 7.1a. In every state, the agent uniformly choose a direction and goes in that direction. If the movement is impossible (wall, ...) it does not move. The four doors are non-reversible: they can be crossed only in a single way, shown by the white arrow. This choice is because of the specific convergence properties of temporal difference algorithms in reversible environments (Ollivier, 2018), as explained in Section 7.5.

This environment is considered as a continuous environment. We considered a continuous parameterization of our discrete mazes. Each state is represented by its 2D coordinates (x, y) ranging in $(-1, 1)$. The network used for learning M is a MLP with 3 hidden layers of width 1024; its input is a set of four coordinates, corresponding to two input states s_1 and s_2. Each layer is made of a linear layer followed by a layer normalization (Ba et al., 2016) and a ReLU activation. The network used for learning V directly via TD has the same architecture, but with a 2-dimensional input instead of a 4-dimensional input. For forward and backward TD on M and TD on V, we use the Adam optimizer (Kingma and Ba, 2015) with its default hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$), with a learning rate schedule $\eta_t = \frac{0.001}{1+\sqrt{t/10^5}}$. We sample the transitions $s \to s'$ from 64 simultaneous trajectories, together with 256 states (s_1, s_2) at each step, sampled from the stationary distribution.

Torus environments. Next, we considered a simple d-dimensional torus environment. A state in the torus is a d-tuple $(\theta_1, ..., \theta_d)$ with $0 \leqslant \theta_i < 1$. We then define a random walk on the torus: the next state s_{t+1} is sampled from a starting point s_t as $s_{t+1} = s_t + \sigma\varepsilon \bmod 1$, where ε is a random normally distributed vector of dimension d. In practice, we used $\sigma = 0.05$ and defined $\gamma = 0.95$.

We parameterize the state space as $(\cos(2\pi\theta_1), \sin(2\pi\theta_1), ..., \cos(2\pi\theta_d), \sin(2\pi\theta_d))$, so that a state is represented by $2d$ numbers. We learn M with a MLP with 2 hidden layers (defined as for the continuous maze environment) of width 512.

We consider the torus of dimension $d = 8$; this is high enough so that the probability to encounter a reward is extremely low (see Section 9.3).

In this environment, we cannot compute an approximation of the true successor state operator with a discretization. Indeed, even if we use a discretization with 8-dimensional cubes of side 0.1, computing M would require inverting a matrix with 10^{16} entries. This is not possible in practice. Instead, we look at qualitative results, and check whether we are able to learn that $s \to M(s, s_{\text{target}})$ is a spike around the target, and 0 elsewhere.

Here we trained m_θ with forward TD. We used the Adam optimizer with its default hyperparameters, with learning rate 10^{-3}. We sample the transitions $s \to s'$ from 64 simultaneous trajectories, together with 256 states s_1 at each step. We stop the training when a total number of $100,000$ transitions was seen (approximately 1500 transition per trajectory).

We select a target state s_{target} (here $s_{\text{target}} = 0$ as all states are equivalent). Then, to plot the value function, we sample a random direction in the torus by sampling a vector v from the normal distribution in dimension $d = 8$, and define a vector u as $u = \frac{|v|}{\|v\|}$ (where $|.|$ is the elementwise absolute value). Hence, u is a random vector sampled uniformly from the part of the sphere in which every coordinate is positive. Then we consider the line $L_u = \{\varepsilon.u, \varepsilon \in [-1, 1]\}$, and plot the value $m_\theta(s, s_{\text{target}})$ for s in L_u. In practice, we consider a discretization of L_u with 200 states. In Figure 7.1b, we sampled 5 such random directions $(u_k)_{1 \leqslant k \leqslant 5}$, and plotted for every ε the mean, minimum and maximum value of $(m_\theta(\varepsilon.u_k, s_{\text{target}}))_k$.

Qualitatively, after $100,000$ steps in the environment we learned the right shape of the successor state.

Chapter 8

Backward Temporal Differences Algorithms

In this chapter, we introduce the backward Bellman equation for the successor states operator (Section 8.1). While the standard Bellman equation (for the value function or the successor states operator) relates the value of different states for a fixed reward (using the transition matrix), the backward Bellman equation relates the value of a fixed state for different rewards, also using the transition matrix. This equation has no equivalent on the value function. In Section 8.4, we define the backward temporal difference algorithm, both in the tabular case and with function approximators. In Section 8.3, we discuss an interpretation of the differences between forward and backward TD as two ways to combine *paths* in an environment. Finally, in Section 8.4 we show that the backward temporal algorithm is equivalent in expectation to the forward temporal difference algorithm in the reversed time backward process.

8.1 The Backward Bellman Equation

The following theorem states the backward Bellman equation:

Theorem 8.1. *The successor state operator M is the only operator which satisfies the backward Bellman equation,*

$$M = \mathrm{Id} + \gamma M P \tag{8.1.1}$$

Proof. The proof is identical to the proof for the forward Bellman equation, with right inverses instead of left inverses.

This equation has no analogue on V. The resulting operator has the same contractivity properties as the usual (forward) Bellman operator.

Proposition 8.2 (Contractivity of the backward Bellman operator on M)**.** *Equip the space of functions $B(\mathcal{S})$ with the sup norm $\|f\|_\infty := \sup_{s\in S} |f(s)|$. Equip the space of bounded linear operators from $B(\mathcal{S})$ to $B(\mathcal{S})$ with the operator norm $\|M\|_{\mathrm{op}} := \sup_{f\in B(\mathcal{S}),\, f\neq 0} \|Mf\|_\infty / \|f\|_\infty$.*

Then the backward Bellman operator $M \mapsto \mathrm{Id} + \gamma M P$ is γ-contracting for this norm.

Proof. For the backward Bellman operator, $M \mapsto \mathrm{Id} + \gamma M P$, the proof is similar, using that for any bounded operator M and function f, one has $\|MPf\|_\infty \leqslant \|M\|_{\mathrm{op}} \|Pf\|_\infty \leqslant \|M\|_{\mathrm{op}} \|f\|_\infty$, so that $\|MP\|_{\mathrm{op}} \leqslant \|M\|_{\mathrm{op}}$ for any M.

8.2 Backward Temporal Difference

Similarly to forward TD, we can define a backward temporal difference algorithm, leveraging the backward Bellman equation. In the tabular case, the update on M when observing a transition

$s \to s'$ is updating all values of M of column s', as:

$$\forall s_1 \quad : \quad M_{s_1 s} \leftarrow M_{s_1 s} + \eta \left(\gamma M_{s_1 s'} - M_{s_1 s}\right) \tag{8.2.1}$$

$$M_{ss} \leftarrow M_{ss} + \eta \tag{8.2.2}$$

Informally, the *forward* TD algorithm is using that, for every target state s_{tar}, if we observe a transition $s \to s'$ and our current estimate of the value function for reaching s_{tar} from s' $M(s', s_{\text{tar}})$ is high, then the value function for reaching s_{tar} from s $M(s, s_{\text{tar}})$ must be high as well. Hence, for every target state s_{tar}, forward TD is generalizing across starting states. On the contrary, the *backward* TD algorithm is generalizing across target states s for every starting state s_{start}. For every starting state s_{start}, it is using that when observing a transition $s \to s'$, if our current estimate of the value function for reaching s from s_{start}, $M(s_{\text{start}}, s)$ is high, then the value function for reaching s' from s_{start} $M(s_{\text{start}}, s')$ must be high as well.

The following theorem defines the backward TD algorithm with function approximators. It is the backward equivalent of Theorem 7.5:

Theorem 8.3. *Let $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $M(s_1, \mathrm{d}s_2)$. Consider $M^{\text{tar}} = \mathrm{Id} + \gamma M_\theta P$, a target estimate for M defined via the Backward Bellman equation.*

Let (s, s') be a sample of the environment such that $s' \sim P(s'|s, a)$ and $s_1 \sim \rho$ is sampled independently, we define $\widehat{\delta\theta}_{B\text{-}TD}(s_1, s, s')$ as:

$$\widehat{\delta\theta}_{B\text{-}TD}(s_1, s, s') := \partial_\theta m_\theta(s, s) + m_\theta(s_1, s)\left(\gamma\, \partial_\theta m_\theta(s_1, s') - \partial_\theta m_\theta(s_1, s)\right) \tag{8.2.3}$$

Then $\widehat{\delta\theta}_{F\text{-}TD}$ is an unbiased estimate of the backward Bellman error:

$$\mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), s_1\sim\rho}\left[\widehat{\delta\theta}_{B\text{-}TD}(s_1, s, s')\right] = -\frac{1}{2}\partial_\theta \|M_\theta - M^{\text{tar}}\|_\rho^2 \tag{8.2.4}$$

Proof. As in the proof of Theorems 7.5 and 7.7, we define $J'(\theta) = \frac{1}{2}\|M_\theta\|_\rho^2 - \langle M_\theta, M^{\text{tar}}\rangle$. We define $\bar{\theta} = \theta$ but $\bar{\theta}$ is not differentiated via ∂_θ. We have:

$$\begin{aligned}
J'(\theta) &= \frac{1}{2}\int_{s_1,s_2} m_\theta(s_1, s_2)^2 \rho(\mathrm{d}s_1)\rho(\mathrm{d}s_2) - \int_{s_1,s_2} m_\theta(s_1, s_2) M^{\text{tar}}(s_1, \mathrm{d}s_2)\rho(\mathrm{d}s_1) \\
&= \frac{1}{2}\int_{s_1,s} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s) m_\theta(s_1, s)^2 - \int_s \rho(\mathrm{d}s) m_\theta(s, s) - \gamma \int_{s_1,s,s'} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s) P(s, \mathrm{d}s') m_\theta(s_1, s') m_{\bar{\theta}}(s_1, s)
\end{aligned}$$

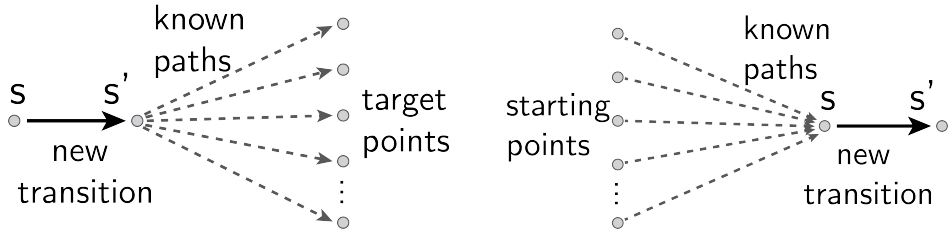
where in the first part s_2 is renamed as s, in the second term s_1 is renamed as s, and in the last one s_2 is renamed as s. Therefore:

$$\begin{aligned}
&-\partial_\theta J'(\theta) \\
&\quad = \int_{s_1,s} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s)\partial_\theta m_\theta(s_1, s) m_\theta(s_1, s) - \int_s \rho(\mathrm{d}s)\partial_\theta m_\theta(s, s) \\
&\qquad - \gamma \int_{s_1,s,s'} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s) P(s, \mathrm{d}s')\partial_\theta m_\theta(s_1, s') m_\theta(s_1, s) \\
&\qquad = \int_{s_1,s,s'} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s) P(s, \mathrm{d}s')\left(\partial_\theta m_\theta(s, s) + m_\theta(s_1, s)\left(\gamma \partial_\theta m_\theta(s_1, s') - \partial_\theta m_\theta(s_1, s)\right)\right)
\end{aligned}$$

Contrary to forward TD, learning M by backward TD then setting $V = MR$ is *not* equivalent to learning V via TD in the tabular case. Still, backward TD for M is not structurally different from forward TD: it corresponds to forward TD for the "time-reversed" Markov process, as shown in Section 8.4. But since states are typically observed in a time-ordered sequence, this might produce a difference in practice.

8.3 Path Combinatorics Interpretation of Forward and Backward TD

The difference between forward and backward TD for M is best understood in the path viewpoint on M (Eq. 6.2.8). Indeed, the current estimate of $M_{s_1 s_2}$ contains a current estimation



(a) Forward TD (b) Backward TD

Figure 8.1: Combining paths with forward TD and backward TD.

on the number of paths from s_1 to s_2, weighted by their estimated probabilities in the Markov process. TD replaces M with PM, and adds Id, which represents the trivial paths from s to s. Backward TD uses MP instead. In both cases, the operator P is sampled via an observed transition $s \to s'$. Thus, PM builds new known paths by taking all paths contained in M and adding the transition $s \to s'$ at the front of each path, while MP adds the transition $s \to s'$ at the back of each path in M (Fig. 8.1). Forward TD reasons at fixed *target states* (rewards) (Greydanus and Olah, 2019), while backward TD reasons at fixed *starting points*.

Thus, TD and backward TD on M differ in how they learn new paths from known paths when each new transition is observed. Arguably, both are reasonable ways to update a mental model of paths in an environment when discovering new transitions (e.g., if a new street $s \to s'$ opens in a city).

There is a third way to build new paths when observing a new transition $s \to s'$: take all known paths to s, all known paths from s', and insert $s \to s'$ in the middle. This exploits path concatenation, roughly doubling the length of known paths. This operation is involved in the way that M actually changes when the process is changed by increasing $P(s, \mathrm{d}s')$ (the way possible paths actually change when a new street opens). This is the basis of the "second-order" algorithms we present for M in Chapter 9.

8.4 Backward TD and the Backward Process

In this section, we prove that backward TD is forward TD on the time-reversed process (Theorem 8.5). Contrary to the rest of this text, we assume that the Markov process is ergodic and that the data are coming from a stationary random trajectory of the process.

8.4.1 The backward process

Define the *backward process* $P_{\text{back}}(s', \mathrm{d}s)$ by reversing time: it is the law of s given s' in a transition $s \to s'$. In this section, we assume that the Markov Process is ergodic, and that ρ is its invariant measure. We now define more precisely the backward process. Let (s, s') be a random pair of states distributed according to $\rho(\mathrm{d}s)P(s, \mathrm{d}s')$, and define $P_{\text{back}}(s', \mathrm{d}s)$ to be the conditional distribution of s given s' under this distribution. (This exists by the general theory of conditional distributions (Parthasarathy, 2005, Thm. 8.1), and is well-defined up to a set of ρ-measure 0.) Since ρ is the invariant measure of the process, the law of both s and s' is ρ, and one has

$$\rho(\mathrm{d}s)P(s, \mathrm{d}s') = \rho(\mathrm{d}s')P_{\text{back}}(s', \mathrm{d}s) \tag{8.4.1}$$

by definition of conditional probabilities.

Then, given s_t, the distribution of s_{t-n} follows the backward process from s_t. Namely, the law of any sequence of observations $(s_{t-n}, \ldots, s_t)$ from the stationary distribution of the process satisfies

$$\rho(\mathrm{d}s_{t-n})P(s_{t-n}, \mathrm{d}s_{t-n+1}) \cdots P(s_{t-1}, \mathrm{d}s_t) = \rho(\mathrm{d}s_t)P_{\text{back}}(s_t, \mathrm{d}s_{t-1}) \cdots P_{\text{back}}(s_{t-n+1}, \mathrm{d}s_{t-n}).$$

Lemma 8.4. *Let $M^{\text{back}} := (\text{Id} - \gamma P_{\text{back}})^{-1}$ be the successor state operator of the backward process. Then M^{back} is related to the successor states operator $M(s, \mathrm{d}s')$ as:*

$$\rho(\mathrm{d}s) M(s, \mathrm{d}s') = \rho(\mathrm{d}s') M^{\text{back}}(s', \mathrm{d}s) \tag{8.4.2}$$

Proof. By induction from the definition of the backward process, we have $\rho(\mathrm{d}s') P^n_{\text{back}}(s', \mathrm{d}s) = \rho(\mathrm{d}s) P^n(s, \mathrm{d}s')$. Then by definition of M^{back},

$$\rho(\mathrm{d}s') M^{\text{backward}}(s', \mathrm{d}s) = \rho(\mathrm{d}s') \sum_{n \geqslant 0} \gamma^n P^n_{\text{backward}}(s', \mathrm{d}s) = \sum_{n \geqslant 0} \gamma^n \rho(\mathrm{d}s) P^n(s, \mathrm{d}s') = \rho(\mathrm{d}s) M(s, \mathrm{d}s')$$

8.4.2 Backward TD and the backward process

We can now show that that backward TD is forward TD on the backward process. Remember that the forward and backward successor state operators are linked by $\rho(\mathrm{d}s_1) M(s_1, \mathrm{d}s_2) = \rho(\mathrm{d}s_2) M^{\text{back}}(s_2, \mathrm{d}s_1)$.

Theorem 8.5 (Backward TD is forward TD on the backward process)**.** *Let M and M^{back} be measure-valued functions such that M^{back} is the time-reverse of M, namely $\rho(\mathrm{d}s_1) M(s_1, \mathrm{d}s_2) = \rho(\mathrm{d}s_2) M^{\text{back}}(s_2, \mathrm{d}s_1)$. Then the backward TD update*

$$M \leftarrow \text{Id} + \gamma M P \tag{8.4.3}$$

is equivalent (ρ-almost everywhere) to

$$M^{\text{back}} \leftarrow \text{Id} + \gamma P_{\text{back}} M^{\text{back}}. \tag{8.4.4}$$

Hence, this theorems show that the forward and backward TD algorithms are very close, and leverage the information acquired from observed trajectories similarly. From this viewpoint, there is no theoretical reason to expect a gain in sample efficiency when using backward TD compared to forward TD. Still, as the updates are different, there could be practical differences.

Proof. Let $D_\rho(\mathrm{d}s_1, \mathrm{d}s_2)$ be the diagonal measure with marginal ρ, namely, $D_\rho(\mathrm{d}s_1, \mathrm{d}s_2) = \rho(\mathrm{d}s_1)\delta_{s_1}(\mathrm{d}s_2) = \rho(\mathrm{d}s_2)\delta_{s_2}(\mathrm{d}s_1)$. Remember that the operator Id corresponds to the process $\delta_{s_1}(\mathrm{d}s_2)$. By multiplying the backward TD update by $\rho(\mathrm{d}s_1)$ one gets

$$\begin{aligned}
\rho(\mathrm{d}s_1) M(s_1, \mathrm{d}s_2) &\leftarrow D_\rho(\mathrm{d}s_1, \mathrm{d}s_2) + \gamma \rho(\mathrm{d}s_1)(MP)(s_1, \mathrm{d}s_2) && (8.4.5)\\
&= D_\rho(\mathrm{d}s_1, \mathrm{d}s_2) + \gamma \int_{s'} \rho(\mathrm{d}s_1) M(s_1, \mathrm{d}s') P(s', \mathrm{d}s_2) && (8.4.6)\\
&= D_\rho(\mathrm{d}s_1, \mathrm{d}s_2) + \gamma \int_{s'} M^{\text{back}}(s', \mathrm{d}s_1) \rho(\mathrm{d}s') P(s', \mathrm{d}s_2) && (8.4.7)\\
&= D_\rho(\mathrm{d}s_1, \mathrm{d}s_2) + \gamma \int_{s'} M^{\text{back}}(s', \mathrm{d}s_1) \rho(\mathrm{d}s_2) P_{\text{back}}(s_2, \mathrm{d}s') && (8.4.8)\\
&= D_\rho(\mathrm{d}s_1, \mathrm{d}s_2) + \gamma \rho(\mathrm{d}s_2)(P_{\text{back}} M^{\text{back}})(s_2, \mathrm{d}s_1) && (8.4.9)
\end{aligned}$$

and since $\rho(\mathrm{d}s_1) M(s_1, \mathrm{d}s_2) = \rho(\mathrm{d}s_2) M^{\text{back}}(s_2, \mathrm{d}s_1)$, this rewrites as

$$\rho(\mathrm{d}s_2) M^{\text{back}}(s_2, \mathrm{d}s_1) \leftarrow \rho(\mathrm{d}s_2)\delta_{s_2}(\mathrm{d}s_1) + \gamma \rho(\mathrm{d}s_2)(P_{\text{back}} M^{\text{back}})(s_2, \mathrm{d}s_1) \tag{8.4.10}$$

namely (ρ-almost everywhere),

$$M^{\text{back}}(s_2, \mathrm{d}s_1) \leftarrow \delta_{s_2}(\mathrm{d}s_1) + \gamma (P_{\text{back}} M^{\text{back}})(s_2, \mathrm{d}s_1) \tag{8.4.11}$$

which is forward TD on M^{back} for the time-reversed process.

Chapter 9

Second-Order Methods for Successor States: Implicit Process Estimation and Bellman–Newton

We now turn to more complex, "second-order" algorithms for estimating successor states and value functions. These approaches were our first motivation to study and learn the successor state operator. Indeed, in Chapter 7, we saw that we are able to derive a Forward TD algorithm for the successor states operator. In the tabular case, we saw that learning the value function via a successor state estimate learned with TD is equivalent to learning directly the value function via TD directly. While this approach can still be useful, for a very sparse reward, or thanks to the generalization obtained from a parametric model $m_\theta(s_1, s_2)$, the Forward Bellman equation on M and the Bellman equation on V are arguably *leveraging the observed transitions in the same way.* In Chapter 8, we introduced the Backward TD, which has no equivalent on the value function. Still, we saw in Theorem 8.5 that the Backward TD update was equivalent (in expectation) to the forward TD update on the backward process. While these two approaches might obtain different results in practice, the two Bellman equations are *leveraging the observed transitions similarly.* Our goal with the successor state operator was to leverage all the information contained in this object, and derive richer update equations.

In this chapter, we study second-order updates for the successor states operator. First, we study in Section 9.1 the online estimate of M and V in the tabular case, obtained by directly estimating the transition matrix and reward function, and exactly solving the Bellman equation in this estimated process. This corresponds to the LSTD algorithm in the specific tabular case (Sutton and Barto, 2018, Section 9.8). In Section 9.2, we show that this provides an explicit online evolution equation for M and V from observed transitions, in which the transition matrix does not appear, hence called *successor states via implicit process estimation* (SSIPE). In Section 9.3, we compare experimentally the sample efficiency of policy evaluation with SSIPE, TD and TD(λ) in the tabular setting, and show that SSIPE is largely superior to these methods (a non-asymptotic convergence bound for SSIPE is given in Chapter 10). Therefore, there is a potential sample efficiency improvement in using the successor state operator for policy evaluation.

Then, in Section 9.4, we show that the SSIPE update of M, taken in expectation, defines a *Bellman–Newton operator*, so called because it corresponds exactly to the Newton method for inverting the matrix M. In Section 9.5 we derive a Bellman–Newton parametric update, extending it beyond full-matrix tabular updates to sampling in arbitrary state spaces.. This new Bellman–Newton operator has no equivalent on the Value function. Intuitively, it proceeds by concatenating known paths of the MDP, thus doubling the length of known paths, while TD and backward TD just add one transition to the set of known paths, as formalized in Section 9.6. Online estimation of M and the Bellman–Newton operator can be seen as "second-order" TD algorithms. Accordingly, they are also numerically trickier. Strengths and weaknesses are

discussed in Section 9.7.

9.1 Estimating a Markov Process Online

We now introduce estimates of M and V by online estimation of the Markov process, first in the tabular case, then via function approximation. In a (small) finite state space, an obvious approach to learn M is to first learn an estimate $(\hat{P}, \hat{R})$ of the transition matrix P and reward vector R of the Markov reward process, by direct empirical averages; then set M and V to their true values in the estimated Markov process, namely, $\hat{M} = \sum_{n \geqslant 0} \gamma^n \hat{P}^n = (\mathrm{Id} - \gamma \hat{P})^{-1}$ and $\hat{V} = \hat{M}\hat{R}$.

The empirical averages $\hat{P}$ and $\hat{R}$ are updated for each new transition $s \to s'$ with reward r_s, by updating the row s of the transition matrix $\hat{P}$, and the value $\hat{R}_s$ at s:

$$\hat{P}_{ss_2} \leftarrow (1 - 1/n_s)\hat{P}_{ss_2} + (1/n_s)\mathbb{1}_{s_2 = s'} \forall s_2, \tag{9.1.1}$$

$$\hat{R}_s \leftarrow (1 - 1/n_s)\hat{R}_s + (1/n_s) r_s \tag{9.1.2}$$

with n_s the number of visits to state s up to time t. The initialization of $\hat{P}$ and $\hat{R}$ is forgotten after the first observation at each state ($n_s = 1$), but to fix ideas we initialize to $\hat{P} = \hat{R} = 0$. The corresponding estimates $\hat{M} = (\mathrm{Id} - \gamma \hat{P})^{-1}$ and $\hat{V} = \hat{M}\hat{R}$ converge to their true values. This method corresponds to the very specific tabular case of *Least Squares Temporal Difference* (Bradtke and Barto, 1996) or to the plug-in estimate (Pananjady and Wainwright, 2019).

The update (9.1.1) is *model based* (see Section 1.3): it learns a *model* $\hat{P}$ of the transition operator and use it for policy evaluation. The inverse operation can only be performed in the tabular case. Still, we will see in next Section that the *process estimation* can be done *implicitly*: we can directly estimate the matrix $\hat{M}$, without storing any explicit model $\hat{P}$ or computing explicitly an inverse $(\mathrm{Id} - \gamma \hat{P})^{-1}$. Hence, this makes this approach *model-free*. This is done by online inversion via the Sherman-Morrison formula, and we call the method Successor States via Implicit Process Estimation (SSIPE). Then, we can approximate SSIPE with function approximators, via the Bellman–Newton update, as introduced in Section 9.5.

9.2 Successor States via Implicit Process Estimation

Direct matrix inversion is inconvenient. But since (9.1.1) is a rank-one update of the matrix $\hat{P}$, one can compute the update of $\hat{M}$ resulting from (9.1.1); this update does not explicitly involve $\hat{P}$ anymore. This will form the basis for the parametric version.

We call the resulting algorithm *successor states via implicit process estimation* (SSIPE).

Theorem 9.1 (SSIPE: Tabular online update of M)**.** *When a transition $s \to s'$ is added to an empirical estimate of a Markov reward process via* (9.1.1)*, the successor state matrix $\hat{M}$ of the estimated process becomes $\hat{M} \leftarrow \hat{M} + \delta M$ with*

$$\delta M_{s_1 s_2} = \frac{1}{n_s} \hat{M}_{s_1 s} \frac{\mathbb{1}_{s_2 = s} + \gamma \hat{M}_{s' s_2} - \hat{M}_{s s_2}}{1 - \frac{1}{n_s}(\gamma \hat{M}_{s' s} - \hat{M}_{ss} + 1)} \qquad \forall s_1, s_2 \tag{9.2.1}$$

with n_s the number of times state s has been sampled.

This update is similar to the LSTD update with the Sherman-Morrison formula (Geramifard et al., 2006) in the very specific tabular case, explained in (Sutton and Barto, 2018, Section 9.8).

Proof. Define $\hat{M} := (\mathrm{Id} - \gamma \hat{P})^{-1}$ where $\hat{P}$ is updated by (9.1.1). The update (9.1.1) can be rewritten as $\hat{P} \leftarrow \hat{P} + (1/n_s)\mathbb{1}_s(\mathbb{1}_{s'}^\top - \mathbb{1}_s^\top \hat{P})$. This is a rank-one update of $\hat{P}$. The update of $\mathrm{Id} - \gamma \hat{P}$ is $-\gamma$ times the update of $\hat{P}$, and is still rank-one: it is equal to $uv^\top$ with $u := -(\gamma/n_s)\mathbb{1}_s$ and $v^\top := (\mathbb{1}_{s'}^\top - \mathbb{1}_s^\top \hat{P})$. The Sherman–Morrison formula gives the update of the inverse of a matrix after a rank-one update. By this

formula, the update of $\hat{M} = (\mathrm{Id} - \gamma \hat{P})^{-1}$ is

$$\hat{M} \leftarrow \hat{M} - \frac{\hat{M} u v^\top \hat{M}}{1 + v^\top \hat{M} u} = \hat{M} + \frac{1}{n_s} \frac{\hat{M} \mathbb{1}_s (\gamma \mathbb{1}_{s'}^\top - \gamma \mathbb{1}_s^\top \hat{P}) \hat{M}}{1 - \frac{1}{n_s} (\gamma \mathbb{1}_{s'}^\top - \gamma \mathbb{1}_s^\top \hat{P}) \hat{M} \mathbb{1}_s}$$

Now, since $\hat{M} = (\mathrm{Id} - \gamma \hat{P})^{-1}$, we have $\gamma \hat{P} \hat{M} = \hat{M} - \mathrm{Id}$. Therefore, the terms $(\gamma \mathbb{1}_{s'}^\top - \gamma \mathbb{1}_s^\top \hat{P}) \hat{M}$ are equal to $\gamma \mathbb{1}_{s'}^\top \hat{M} - \mathbb{1}_s^\top \hat{M} + \mathbb{1}_s^\top$, and the update is

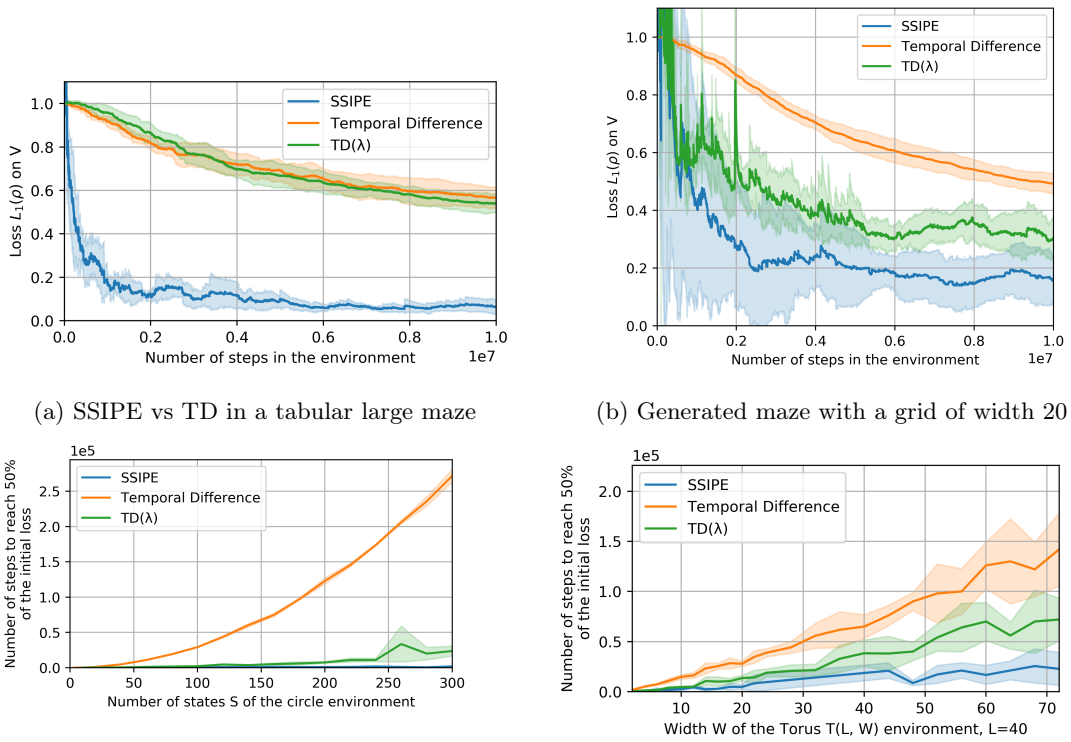
$$\begin{aligned} \hat{M} \leftarrow \hat{M} &+ \frac{1}{n_s} \frac{\hat{M} \mathbb{1}_s (\gamma \mathbb{1}_{s'}^\top \hat{M} - \mathbb{1}_s^\top \hat{M} + \mathbb{1}_s^\top)}{1 - \frac{1}{n_s} (\gamma \mathbb{1}_{s'}^\top \hat{M} - \mathbb{1}_s^\top \hat{M} + \mathbb{1}_s^\top) \mathbb{1}_s} \\ &= \hat{M} + \frac{1}{n_s} \frac{\hat{M} \mathbb{1}_s (\gamma \mathbb{1}_{s'}^\top \hat{M} - \mathbb{1}_s^\top \hat{M} + \mathbb{1}_s^\top)}{1 - \frac{1}{n_s} (\gamma \hat{M}_{s's} - \hat{M}_{ss} + 1)} \end{aligned}$$

which is the exact update of $\hat{M}$. This provides the update (9.2.1).

This describes the "true" change of M when the Markov process changes by increasing $P_{ss'}$. This update contains a two-sided term $M_{s_1 s} M_{s' s_2}$: in terms of paths, this term combines all known paths from s_1 to s, the transition $s \to s'$, then all known paths from s' to s_2 (Section 9.6).

In the following section, we will see that this SSIPE algorithms leads to a new fixed point equation on the successor states operator, which can be used to derive an update even in continuous environments with function approximators.

9.3 Tabular experiments with SSIPE, TD and TD(λ)



(a) SSIPE vs TD in a tabular large maze

(b) Generated maze with a grid of width 20

(c) Influence of the length L in the circle environment $C(L)$.

(d) Influence of environment size

Figure 9.1: Tabular experiments with SSIPE, TD and TD(λ)

In this section, we compare empirically the sample efficiency of SSIPE, TD and TD(λ) in the tabular setting. All the experimental details are in Appendix 9.A.

To test learning of M in tabular situations, we first generated a number of random mazes with up to 2,700 states (see figures in Appendix 9.A). We learn M via SSIPE, namely, via the update (9.2.1), with the reward estimated via the empirical average (9.1.1). The value function can then be estimated by the matrix product MR. For comparison, we also learn V via ordinary TD and TD(λ). The policy is a random walk in the maze. SSIPE is largely superior to TD and TD(λ) in these mazes (Fig 9.1a).

Next, we turn to a more systematic study of the efficiency of learning V via SSIPE on M, versus ordinary TD(λ). For this, we define a simple variable-size environment and study the effects of increased size. The environment has two directions: a pure "noise" component and a deterministic move component. Formally, the state space is $\{1, ..., L\} \times \{1, ..., W\}$ where the length L and width W are integers; the transition matrix is $P_{(l_1,w_1),(l_2,w_2)} = \frac{1}{W}\delta_{l_2=l_1+1 \bmod L}$, and the reward is sparse, located at state $(1, 1)$. For TD(λ), the parameter λ is tuned separately for each environment size. The results are in Figure 9.1d. With small W, TD(λ) is as efficient as SSIPE, but with large W, eligibility traces incur a higher variance (and the optimal λ gets close to 0), and the gap between TD and SSIPE increases. Compared to SSIPE, this example shows that TD(λ) does not always efficiently exploit the information from observed transitions.

These experiments show that TD or even TD(λ) do not exploit all the observed information in the environment, and that it is possible to improve policy evaluation sample efficiency. In Chapter 10, we provide non-asymptotic convergence bounds for policy evaluation with SSIPE in the tabular setting.

Still, the SSIPE method can only be used in a tabular setting. We now introduce the Bellman–Newton algorithm, a second-order policy evaluation method, which can be used in continuous environments.

9.4 The Bellman–Newton Operator

9.4.1 The expected SSIPE update

In Theorem 9.1, we saw that when a transition $s \to s'$ is observed and we update our model of P via (9.1.1), the successor state matrix $\hat{M}$ of the estimated process becomes $\hat{M} \leftarrow \hat{M} + \widehat{\delta M}$ with $\widehat{\delta M_{s_1 s_2}} = \frac{1}{n_s}\hat{M}_{s_1 s}\frac{\mathbb{1}_{s_2=s}+\gamma\hat{M}_{s's_2}-\hat{M}_{ss_2}}{1-\frac{1}{n_s}(\gamma\hat{M}_{s's}-\hat{M}_{ss}+1)}$.

We consider the expected update over observed transitions (s, s'): $\mathbb{E}_{s\sim\rho,s'\sim P(.|s)}[\delta M]$

Theorem 9.2. *Estimate the successor matrix of a finite MRP by $M_t = (\mathrm{Id} - \gamma P_t)^{-1}$ where P_t is estimated directly by the empirical averages* (9.1.1)*. Then, in expectation over the transition (s_t, s'_t) observed at step t, conditionally to the current estimate M_t, we have:*

$$\mathbb{E}_{s_t\sim\rho,s'_t\sim P(.|s_t)}[M_{t+1}|M_t] = M_t + \frac{1}{t}\delta M + o(1/t) \tag{9.4.1}$$

with

$$\delta M = M_t - M_t(\mathrm{Id} - \gamma P)M_t \tag{9.4.2}$$

Proof. First, note that the expectation in the statement is averaged over the next step, but conditional to $\hat{M}_t$ computed in the previous steps. In this proof, we will just write $\mathbb{E}$ for short.

We have:

$$\mathbb{E}_{s_t\sim\rho,s'_t\sim P(.|s_t)}[M_{t+1}|M_t] = M_t + \mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\widehat{\delta M}\right]$$

For every $s_1, s_2 \in \mathcal{S}$, we have:

$$\begin{aligned}\left(\mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\widehat{\delta M}\right]\right)_{s_1 s_2} &= \mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\frac{1}{n_s}(M_t)_{s_1 s}\frac{\mathbb{1}_{s_2=s}+\gamma(M_t)_{s's_2}-(M_t)_{ss_2}}{1-\frac{1}{n_s}(\gamma(M_t)_{s's}-(M_t)_{ss}+1)}\right]\\ &= \mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\frac{1}{t\rho_s+o(t)}(M_t)_{s_1 s}\frac{\mathbb{1}_{s_2=s}+\gamma(M_t)_{s's_2}-(M_t)_{ss_2}}{1-O(\frac{1}{t})}\right]\\ &= \frac{1}{t}\mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\frac{1}{\rho_s}(M_t)_{s_1 s}\left(\mathbb{1}_{s_2=s}+\gamma(M_t)_{s's_2}-(M_t)_{ss_2}\right)\right]+o(1/t)\end{aligned}$$

where the second line holds because $n_s = t\rho_s + o(t)$ by the law of large numbers (since s is sampled from ρ), and the last one is obtained by removing terms in $o(1/t)$. This leads to:

$$\begin{aligned}\left(\mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\widehat{\delta M}\right]\right)_{s_1 s_2} &= \frac{1}{t}\sum_{s,s'}\rho_s P_{ss'}\left(\frac{1}{\rho_s}(M_t)_{s_1 s}\left(\mathbb{1}_{s_2=s}+\gamma(M_t)_{s's_2}-(M_t)_{ss_2}\right)\right)+o(1/t)\\ &= \frac{1}{t}\sum_{s}\left((M_t)_{s_1 s}\left(\mathbb{1}_{s_2=s}+\gamma(PM_t)_{ss_2}-(M_t)_{ss_2}\right)\right)+o(1/t)\end{aligned}$$

Therefore, we have:

$$\mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[\widehat{\delta M}\right] = M_t - M_t(\mathrm{Id}-\gamma P)M_t + o(1/t) \tag{9.4.3}$$

and we can conclude:

$$\mathbb{E}_{s\sim\rho,s'\sim P(.|s)}\left[M_{t+1}|M_t\right] = M_t + \frac{1}{t}\delta M + o(1/t) \tag{9.4.4}$$

with

$$\delta M = M_t - M_t(\mathrm{Id}-\gamma P)M_t \tag{9.4.5}$$

In the following section, we will see that this expected SSIPE update corresponds to a new fixed-point equation on the successor states operator, which can be used for learning. We call this equation the Bellman–Newton equation, because of its relation with Newton's method for matrix inversion.

9.4.2 Definition of Bellman–Newton operator

We now translate the expected SSIPE update derived in Theorem 9.2 into an operator, similarly to the Bellman operator and the expected TD update: We know that the expected Temporal Difference update (either on V as in Section 1.4.2 or on M as in Section 7.3) can be written as

$$V_{t+1} := V_t + \eta_t \delta V,$$

with

$$\delta V = (R + \gamma P V_t - V_t) = T \cdot V_t - V_t,$$

with T the Bellman operator

$$T \cdot f = R + \gamma P f.$$

In Theorem 9.2, we proved that $M_{t+1} = M_t + \eta_t \delta M$ with

$$\delta M = M_t - M_t(\mathrm{Id}-\gamma P)M_t = T \cdot M_t) - M_t$$

with

$$T \cdot M := 2M - M(\mathrm{Id}-\gamma P)M.$$

This leads to the definition of the Bellman–Newton operator:

Definition 9.3 (Bellman–Newton operator)**.** We call *Bellman–Newton operator* the operator

$$M \mapsto 2M - M(\mathrm{Id}-\gamma P)M. \tag{9.4.6}$$

The Bellman–Newton operator defines a new fixed point for M, additionally to the forward and backward Bellman equation:

Proposition 9.4. *The successor state $M(s_1, \mathrm{d}s_2)$ is a fixed point of the Bellman–Newton operator*

Proof. We have:

$$2M - M(\mathrm{Id} - \gamma P)M = 2M - M(\mathrm{Id} - \gamma P)(\mathrm{Id} - \gamma P)^{-1} = 2M - M = M$$

9.4.3 Bellman–Newton operator and Newton method for matrix inversion

The reason for the name is the following: Inverting a matrix A by iterating $M \leftarrow 2M - MAM$ is the *Newton method* for matrix inversion, going as far back as 1933 (Pan and Schreiber, 1991). The Newton method has superexponential convergence, squaring the error (doubling precision) at each step.

Unfortunately, this method does not always converge. In particular, it is initialization-dependent. For instance, the initialization $M = 0$ is a fixed point. In general, the Bellman–Newton operator preserves the kernel and image of M, so there are many fixed points. Still, $M = (\mathrm{Id} - \gamma P)^{-1}$ is the only full-rank fixed point.

Convergence of the Newton method for matrix inversion is quite well understood (Pan and Schreiber, 1991) and works if the spectral radius of $\mathrm{Id} - AM$ is less than 1 at initialization. Otherwise, the method can diverge. For instance, $A = \mathrm{Id} - \gamma P$ for successor states, so initializing to $M = \mathrm{Id}$ converges.

9.5 Parametric Bellman–Newton Update

Similarly to TD algorithms for learning the successor state operator via the Bellman operators, we can derive a parametric update for successor state operator via the Bellman Newton operator, which leads to Algorithm 9. This is formalized in the following theorem:

Theorem 9.5 (Bellman-Newton update for successor states with function approximation)**.** *Let $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $M(s_1, \mathrm{d}s_2)$. Consider $M^{\mathrm{tar}} = 2M_\theta - M_\theta(\mathrm{Id} - \gamma P)M_\theta$, a target estimate for M defined via the Bellman-Newton equation.*

Let (s, s') be a sample of the environment such that $s \sim \rho, s' \sim P(s'|s, a)$ and $s_1, s_2 \sim \rho$ are sampled independently, we define $\widehat{\delta\theta}_{BN}(s_1, s, s', s_2)$ as:

$$\widehat{\delta\theta}_{BN}(s_1, s, s', s_2) := m_\theta(s_1, s)\, \partial_\theta m_\theta(s_1, s) + m_\theta(s_1, s)(\gamma m_\theta(s', s_2) - m_\theta(s, s_2))\, \partial_\theta m_\theta(s_1, s_2) \tag{9.5.1}$$

Then $\widehat{\delta\theta}_{BN}$ is an unbiased estimate of the Bellman-Newton error:

$$\mathbb{E}_{s, s_1, s_2 \sim \rho, s' \sim P(s, \mathrm{d}s')} \left[\widehat{\delta\theta}_{BN}(s_1, s, s', s_2) \right] = -\frac{1}{2} \partial_\theta \| M_\theta - M^{\mathrm{tar}} \|_\rho^2 \tag{9.5.2}$$

Proof. We have:

$$M^{\mathrm{tar}}(s_1, \mathrm{d}s_2) = 2M_\theta(s_1, \mathrm{d}s_2) - (M_\theta(\mathrm{Id} - \gamma P)M_\theta)(s_1, \mathrm{d}s_2) \tag{9.5.3}$$

and:

$$\begin{aligned}
&(M_\theta(\mathrm{Id} - \gamma P)M_\theta)(s_1, \mathrm{d}s_2) = \\
&= \int_{s, s'} \rho(\mathrm{d}s) m_\theta(s_1, s)(\mathrm{Id} - \gamma P)(s, \mathrm{d}s') m(s', s_2)\rho(\mathrm{d}s_2) \\
&= \int_s \rho(\mathrm{d}s) m_\theta(s_1, s) m_\theta(s, s_2)\rho(\mathrm{d}s_2) - \gamma \int_{s, s'} \rho(\mathrm{d}s) P(s, \mathrm{d}s') m_\theta(s_1, s) m(s', s_2)\rho(\mathrm{d}s_2) \\
&= \rho(\mathrm{d}s_2) \int_{s, s'} \rho(\mathrm{d}s) P(s, \mathrm{d}s') m_\theta(s_1, s)(m_\theta(s, s_2) - \gamma m(s', s_2))
\end{aligned}$$

Therefore, $M^{\mathrm{tar}}(s_1, \mathrm{d}s_2) = m^{\mathrm{tar}}(s_1, s_2)\rho(\mathrm{d}s_2)$ with

$$m^{\mathrm{tar}}(s_1, s_2) = 2m_\theta(s_1, s_2) + \int_{s, s'} \rho(\mathrm{d}s) P(s, \mathrm{d}s') m_\theta(s_1, s) \left(\gamma m(s', s_2) - m_\theta(s, s_2) \right) \tag{9.5.4}$$

Algorithm 9 Bellman–Newton for successor states with function approximation.

Input: Policy $\pi(a|s)$, randomly initialized model $m_\theta(s_1, s_2)$; `TransitionMemory`, maximum number of time steps T
repeat
  for K trajectories **do**
    Get an initial state s_0 from the environment.
    for $0 \leqslant t \leqslant T$ steps do **do**
      Sample $a_t \sim \pi(.|s_t)$, execute a_t and observe s_{t+1}
      Store in the transition memory the transition `TransitionMemory` $\leftarrow (s_t, s_{t+1})$
    end for
    for L gradient steps **do**
      Sample $(s, s') \sim$ `TransitionMemory`,
      Sample $(s_1, \_) \sim$ `TransitionMemory`, $(s_2, \_) \sim$ `TransitionMemory`
      $\widehat{\delta\theta}_{\mathrm{BN}}(s_1, s, s', s_2) := m_\theta(s_1, s)(\gamma m_\theta(s', s_2) - m_\theta(s, s_2))\, \partial_\theta m_\theta(s_1, s_2) + m_\theta(s_1, s)\partial_\theta m_\theta(s_1, s)$
      Stochastic gradient step: $\theta \leftarrow \theta + \eta\widehat{\delta\theta}_{\mathrm{BN}}$.
    end for
  end for
until end of learning

Therefore, we have:

$$\begin{aligned} J(\theta) &:= \frac{1}{2}\|M_\theta - M^{\mathrm{tar}}\|_\rho^2 \\ &= \int_{s_1 s_2} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s_2)\left(m_\theta(s_1, s_2) - m^{\mathrm{tar}}(s_1, s_2)\right)^2 \end{aligned}$$

and:

$$\begin{aligned} &-\partial_\theta J(\theta) = \frac{1}{2}\|M_\theta - M^{\mathrm{tar}}\|_\rho^2 = -\int_{s_1 s_2} \rho(\mathrm{d}s_1, \mathrm{d}s_2)\partial_\theta m_\theta(s_1, s_2)\left(m_\theta(s_1, s_2) - m^{\mathrm{tar}}(s_1, s_2)\right) \\ &= \int_{s_1, s_2 s, s'} \rho(\mathrm{d}s_1, \mathrm{d}s_2, \mathrm{d}s)P(s, \mathrm{d}s')\partial_\theta m_\theta(s_1, s_2)\left(m_\theta(s_1, s_2) + m_\theta(s_1, s)\left(\gamma m(s', s_2) - m_\theta(s, s_2)\right)\right) \\ &= \int_{s_1, s_2 s, s'} \rho(\mathrm{d}s_1, \mathrm{d}s_2, \mathrm{d}s)P(s, \mathrm{d}s')\left(\partial_\theta m_\theta(s_1, s)m_\theta(s_1, s) + \partial_\theta m_\theta(s_1, s_2)m_\theta(s_1, s)\left(\gamma m(s', s_2) - m_\theta(s, s_2)\right)\right) \end{aligned}$$

where for the last line we replace s_2 by s in the first term as they are both independent of s_1 and share the same law ρ.

Implementing this update requires sampling *two* additional states s_1 and s_2 from the dataset, in addition to the transition $s \to s'$.

9.6 The Bellman–Newton Operator and Path Composition

In Section 8.3, we explained the link between learning successor states and counting paths in a Markov process. Here, we formalize that link, by studying how updating M via the Bellman equation (or the backward Bellman equation), or the Bellman-Newton operator updates the paths represented in M. We will prove that after t steps, the estimate of M via Bellman–Newton exactly contains all paths up to length $2^t - 1$ with their correct probabilities in the Markov process, while forward and backward TD exactly contain all paths up to length t.

Thus for each algorithm (forward TD, backward TD, and Bellman–Newton), we consider the exact (deterministic, non-sampled) update: we set $M_0 = \mathrm{Id}$ and then define at step $t + 1$ the update M_{t+1} as the target update given by the corresponding fixed point equation. For forward TD, the operator update is defined as:

$$M_{t+1}^{\mathrm{TD}} = \mathrm{Id} + \gamma P M_t^{\mathrm{TD}}. \tag{9.6.1}$$

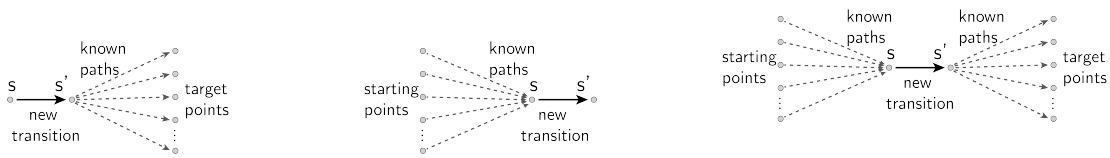


Figure 9.2: Combining paths: forward TD, backward TD, and path composition (Bellman–Newton).

For backward TD, the operator update is defined as:

$$M_{t+1}^{\mathrm{BTD}} = \mathrm{Id} + \gamma M_t^{\mathrm{BTD}} P. \tag{9.6.2}$$

The Bellman–Newton update (Definition 9.3) with learning rate 1 is

$$M_{t+1}^{\mathrm{BN}} = 2M_t^{\mathrm{BN}} - M_t^{\mathrm{BN}}(\mathrm{Id} - \gamma P) M_t^{\mathrm{BN}}. \tag{9.6.3}$$

We now relate the forward TD, backward TD, and Bellman–Newton updates to path composition. For each algorithm, we prove by induction that at step t, there exists an integer n_t such that $(M_t)_{ss'}$ is equal to the number of paths from s to s' with length at most n_t, weighted by their probability and discounted by their length, namely,

$$(M_t)_{ss'} = \sum_{p \text{ path from } s \text{ to } s', |p| \leqslant n_t} \gamma^{|p|} \mathbb{P}(p) = \sum_{k=0}^{n_t} \gamma^k \sum_{s = s_0, \ldots, s_{k-1}, s_k = s'} P_{s_0 s_1} \cdots P_{s_{n-1} s_{n_t}} \tag{9.6.4}$$

where $|p|$ denotes the length of a path p and $\mathbb{P}(p) = P_{s_0 s_1} \cdots P_{s_{n-1} s_n}$ its probability in the Markov process. Equivalently,

$$M_t = \sum_{0 \leqslant k \leqslant n_t} \gamma^k P^k. \tag{9.6.5}$$

The three algorithms will differ by the value of n_t. This is formalized in the following theorem:

Theorem 9.6. *For every t, we have:*

$$\begin{aligned}
M_t^{TD} &= \sum_{p \text{ path from } s \text{ to } s', |p| \leqslant t} \gamma^{|p|} \mathbb{P}(p) \\
M_t^{BTD} &= \sum_{p \text{ path from } s \text{ to } s', |p| \leqslant t} \gamma^{|p|} \mathbb{P}(p) \\
M_t^{BN} &= \sum_{p \text{ path from } s \text{ to } s', |p| \leqslant 2^t} \gamma^{|p|} \mathbb{P}(p)
\end{aligned}$$

Proof. For $t = 0$, $M_0 = \mathrm{Id}$, and the induction hypothesis is satisfied.

If the end point of a path p_1 corresponds to the starting point of a path p_2, we denote $p_1 \cdot p_2$ the concatenation of the two paths.

For forward TD, we have $M_{t+1}^{\mathrm{TD}} = \mathrm{Id} + \gamma P M_t^{\mathrm{TD}}$. By induction, if $M_t^{\mathrm{TD}} = \sum_{0 \leqslant k \leqslant n_t^{\mathrm{TD}}} \gamma^k P^k$, then we find $M_{t+1}^{\mathrm{TD}} = \mathrm{Id} + \gamma P \sum_{0 \leqslant k \leqslant n_t^{\mathrm{TD}}} \gamma^k P^k = \sum_{0 \leqslant k \leqslant n_t^{\mathrm{TD}}+1} \gamma^k P^k$. Equivalently, looking at paths we have

$$\begin{aligned}
(M_{t+1}^{\mathrm{TD}})_{ss'} &= \delta_{s=s'} + \gamma (P M_t^{\mathrm{TD}})_{ss'} \\
&= \delta_{s=s'} + \gamma \sum_{s''} P_{ss''} \sum_{p \text{ path from } s'' \text{ to } s',\, |p| \leqslant n_t^{\mathrm{TD}}} \gamma^{|p|} \mathbb{P}(p) \\
&= \delta_{s=s'} + \sum_{s''} \sum_{p \text{ path from } s'' \text{ to } s',\, |p| \leqslant n_t^{\mathrm{TD}}} \gamma^{|p|+1} \mathbb{P}((s, s'') \cdot p) \\
&= \delta_{s=s'} + \sum_{p \text{ path from } s \text{ to } s',\, 1 \leqslant |p| \leqslant n_t^{\mathrm{TD}}} \gamma^{|p|} \mathbb{P}(p) \\
&= \sum_{p \text{ path from } s \text{ to } s',\, |p| \leqslant n_t^{\mathrm{TD}}+1} \gamma^{|p|} \mathbb{P}(p)
\end{aligned}$$

Thus the induction hypothesis is satisfied with $n_{t+1}^{\mathrm{TD}} = n_t^{\mathrm{TD}} + 1$. By induction, $n_t^{\mathrm{TD}} = t$: at step t, M_t^{TD} is the weighted sum of paths of length at most t. M_{t+1}^{TD} is obtained from M_t^{TD} by adding a transition to the left to every known path (and re-adding the length-0 paths via the Id term).

Likewise, with backward TD we have

$$\begin{aligned}
(M_{t+1}^{\mathrm{BTD}})_{ss'} &= \delta_{s=s'} + \gamma (M_t^{\mathrm{BTD}} P)_{ss'} \\
&= \delta_{s=s'} + \sum_{s''} \sum_{p \text{ path from } s'' \text{ to } s',\, |p| \leqslant n_t^{\mathrm{BTD}}} \gamma^{|p|+1} \mathbb{P}(p \cdot (s'', s')) \\
&= \sum_{p \text{ path from } s \text{ to } s',\, |p| \leqslant n_t^{\mathrm{BTD}}+1} \gamma^{|p|} \mathbb{P}(p)
\end{aligned}$$

Contrary to forward TD, M_{t+1}^{BTD} is obtained from M_t^{BTD} by adding a transition to the right to every known path. This still leads to $n_t^{\mathrm{BTD}} = t$.

We now consider the Bellman–Newton operator update. We have

$$M_{t+1}^{\mathrm{BN}} = 2M_t^{\mathrm{BN}} - M_t^{\mathrm{BN}} (\mathrm{Id} - \gamma P) M_t^{\mathrm{BN}}.$$

Let us first compute $(\mathrm{Id} - \gamma P) M_t^{\mathrm{BN}}$. By the induction hypothesis and by the same reasoning as for forward TD, we have

$$\begin{aligned}
((\mathrm{Id} - \gamma P) M_t^{\mathrm{BN}})_{ss'} &= M_t^{\mathrm{BN}} - \gamma P M_t^{\mathrm{BN}} \\
&= \sum_{p \text{ path from } s \text{ to } s',\, |p| \leqslant n_t^{\mathrm{BN}}} \gamma^{|p|} \mathbb{P}(p) - \sum_{p \text{ path from } s \text{ to } s',\, 1 \leqslant |p| \leqslant n_t^{\mathrm{BN}}+1} \gamma^{|p|} \mathbb{P}(p) \\
&= \delta_{s=s'} - \gamma^{n_t^{\mathrm{BN}}+1} \left(P^{n_t^{\mathrm{BN}}+1} \right)_{ss'}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
M_{t+1}^{\mathrm{BN}} &= 2M_t^{\mathrm{BN}} - M_t^{\mathrm{BN}} (\mathrm{Id} - \gamma P) M_t^{\mathrm{BN}} \\
&= 2M_t^{\mathrm{BN}} - M_t^{\mathrm{BN}} (\mathrm{Id} - \gamma^{n_t^{\mathrm{BN}}+1} P^{n_t^{\mathrm{BN}}+1}) \\
&= M_t^{\mathrm{BN}} + \gamma^{n_t^{\mathrm{BN}}+1} M_t^{\mathrm{BN}} P^{n_t^{\mathrm{BN}}+1} \\
&= \sum_{p \text{ path from } s \text{ to } s',\, |p| \leqslant n_t^{\mathrm{BN}}} \gamma^{|p|} \mathbb{P}(p) + \sum_{p \text{ path from } s \text{ to } s',\, n_t^{\mathrm{BN}}+1 \leqslant |p| \leqslant 2n_t^{\mathrm{BN}}+1} \gamma^{|p|} \mathbb{P}(p) \\
&= \sum_{p \text{ path from } s \text{ to } s',\, |p| \leqslant 2n_t^{\mathrm{BN}}+1} \gamma^{|p|} \mathbb{P}(p)
\end{aligned}$$

Therefore, $n_{t+1}^{\mathrm{BN}} = 2n_t^{\mathrm{BN}} + 1$. At every step the Bellman–Newton operator update is doubling the maximal length of all known paths.

9.7 Discussion: strengths and weaknesses of second-order approaches

In a tabular setting, the SSIPE algorithm converges much faster than TD to compute the value function, empirically.

However, this results in an $O(|S|^2)$ cost per time step, so it is only interesting if sample efficiency is the main issue. The alternative is to sample a few states s_1 and s_2 and only update $M_{s_1 s_2}$ for those states. But in practice, we have found that this introduces many instabilities and requires reducing the learning rate so much (typically η smaller than $1/|S|^2$) that the benefit of second-order Newton convergence is lost. The same phenomenon is observed for the parametric version of Theorem 9.5.

In Chapter 11, we will introduce algorithms for learning low-rank representations of the successor states operator. These algorithms are related to Bellman–Newton methods (Section 11.5), but with lower variance (Section 11.3.2).

In the next Chapter, we will prove a non-asymptotic convergence bound for SSIPE in the tabular setting. Such an analysis is way to understand if there is room for improvement

Appendix

9.A Experimental details on the tabular experiments

Environments. In the tabular setting, we used three series of environments.

Environment in the first series are simple hand-designed mazes (see Figure 9.4b and Figure 9.4a for environment examples, and Figure 9.1 for the tabular results). Since it is known that TD has a specific behavior in reversible environments Ollivier (2018), we added *non-reversible states*: states from which it is possible to move in only one direction. These are represented by the arrows in the figure.

Second, we generated more complicated mazes, using a generator of non-reversible mazes of width W. [1] See Figure 9.4b for an example of such an environment, of width 8.

In these mazes, we use a uniform policy. The four possible actions are the four directions. If the chosen action is not admissible (either it goes through a wall, or the agent is at a non-reversible state), the agent does not move. If the action is possible, the agent moves in the chosen direction. In particular, in the non-reversible states, only one action is admissible. We set the discount factor γ as a function of the mixing time of the process. (The mixing time of a Markov chain with transition matrix P is defined as $\frac{1}{1-u}$ where u is the second largest eigenvalue of P.) We set γ such that $\frac{1}{1-\gamma}$ is equal to the mixing time: this way, the discount factor stays close to 1 for about as long as the initial state is not forgotten.

In each of these environments, we select once and for all a target state s at random from the invariant measure ρ, and set the reward to be nonzero at that state only. We set the reward value such that its expectation under ρ is 1. Thus, we set $R = \frac{1}{\rho(s)} \mathbb{1}_s$.

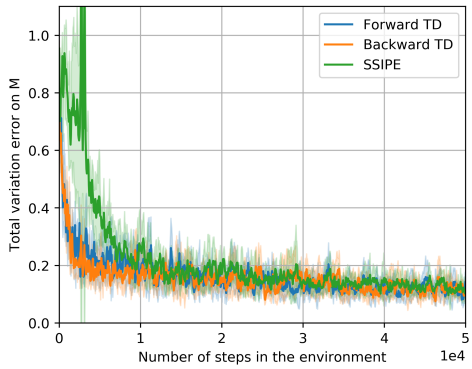
Finally, we considered the Torus and Circle environments. The torus environment was defined in section 9.3. In the torus environment, we set gamma such that $\frac{1}{1-\gamma} = \frac{L}{2}$. We also define an other simple variable-size environment in order to study the effects of increased size: the circle environment of length L: $C(L)$. The state space is $\{1, ..., L\}$; the transition matrix is $P_{l_1,l_2} = \frac{1}{2}\delta_{l_2=l_1+1 \bmod L} + \frac{1}{2}\delta_{l_2=l_1 \bmod L}$. In the circle $C(L)$ environment, we set gamma such that $\frac{1}{1-\gamma} = L$ (see figure 9.1c).

We sample a starting point according to the invariant distribution ρ, and a single random trajectory from the process. We learn the value function and successor states online from this trajectory, using the following algorithms.

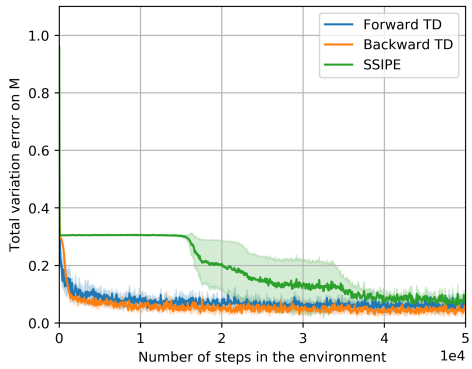
Algorithms. In the tabular setting, we compare SSIPE, TD and TD(λ). For TD, at every step, when a transition (s, s', r) is observed, the update is

$$V(s) \leftarrow V(s) + \eta_t(r + \gamma V(s') - V(s))$$

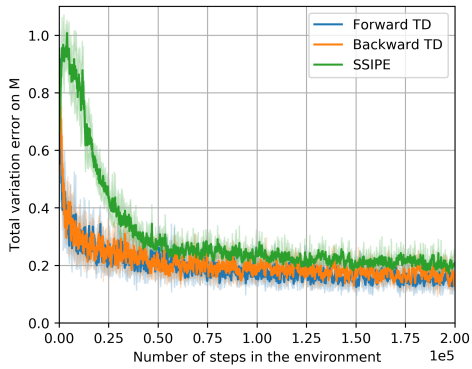
[1] These mazes are built as follows. We first consider a square grid of width W. The set of states $\mathcal{S}$ is the set of pairs $\mathcal{S} = \{(i, j),\ 1 \leqslant i, j \leqslant W\}$. We consider the grid as a torus: for every $1 \leqslant i \leqslant W$, the state $(1, i)$ is adjacent to (W, i) and the state $(i, 1)$ is adjacent to (i, W). We build the set of edges E of the maze as follows. For every pair of adjacent states (s_1, s_2) we sample a single directed edge from among $s_1 \to s_2$ or $s_2 \to s_1$ (with probability 1/2 each), and add it to the set of edges E. At this point, the graph $(\mathcal{S}, E)$ is not strongly connected. We then build the graph $(\tilde{S}, \tilde{E})$ of strongly connected components. $\tilde{S}$ is the set of strongly connected components of $(\mathcal{S}, E)$, and $(\tilde{s}_1, \tilde{s}_2) \in \tilde{E}$ if and only if there is a state $s_1 \in \tilde{s}_1$, and a state $s_2 \in \tilde{s}_2$ such that $(s_1, s_2) \in E$. We say that two strongly connected components s_1 and s_2 are adjacent if there is a state $s_1 \in \tilde{s}_1$, and a state $s_2 \in \tilde{s}_2$ such that s_1 and s_2 are adjacent. The goal is then to make the graph $(\tilde{S}, \tilde{E})$ strongly connected, by adding only edges between adjacent states, and by adding as few edges as possible. We consider the leaves of the graph $(\tilde{S}, \tilde{E})$ (the states $\tilde{s}$ such that no edge is starting from $\tilde{s}$) and its roots (the states $\tilde{s}$ such that no edge is arriving to $\tilde{s}$). For every leaf $\tilde{s}$, we consider the set of its adjacent states and add to $\tilde{E}$ the smallest number of edges such that $\tilde{s}$ is connected to all of its adjacent states. We do a similar operation for roots. Finally, for every new edge $(\tilde{s}_1, \tilde{s}_2)$ in $\tilde{E}$, we consider the set of $s_1 \in \tilde{s}_1$ and $s_2 \in \tilde{s}_2$ such that s_1 and s_2 are adjacent, randomly sample a transition (s_1, s_2) uniformly from this set and add it to E. Finally, we check that the graph is strongly connected. Hence, we get a random strongly connected graph. Finally, we consider the product of this graph with a small size 4 grid.



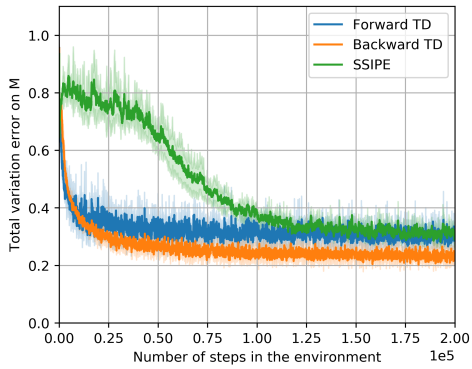
(a) Reversible torus of dimension $d = 1$



(b) Non Reversible torus of dimension $d = 1$



(c) Reversible torus of dimension $d = 2$



(d) Non Reversible torus of dimension $d = 2$

Figure 9.3: Parametric experiments: Learning the successor states operator M with Forward TD, Backward TD and SSIPE in the reversible and non reversible torus, for dimension $d = 1$ and $d = 2$.

with learning rate η_t. For TD(λ), we define the vector e_t of eligibility traces. The update is given by:

$$e \leftarrow e + \gamma\lambda \mathbb{1}_s$$
$$V \leftarrow V + \eta_t(r + \gamma V(s') - V(s))e.$$

For TD and TD(λ), V is initialized to 0. For SSIPE, P is initialized to 0 (which corresponds to $M = \mathrm{Id}$) and R is initialized to 0.

For SSIPE, there is no hyperparameter to tune.

For TD and TD(λ), we used learning rates schedules expressed as $\eta_t = \frac{\beta}{t+t_0}$. This results two hyperparameters for TD (β and t_0) and three for TD(λ) (β, t_0, and λ).

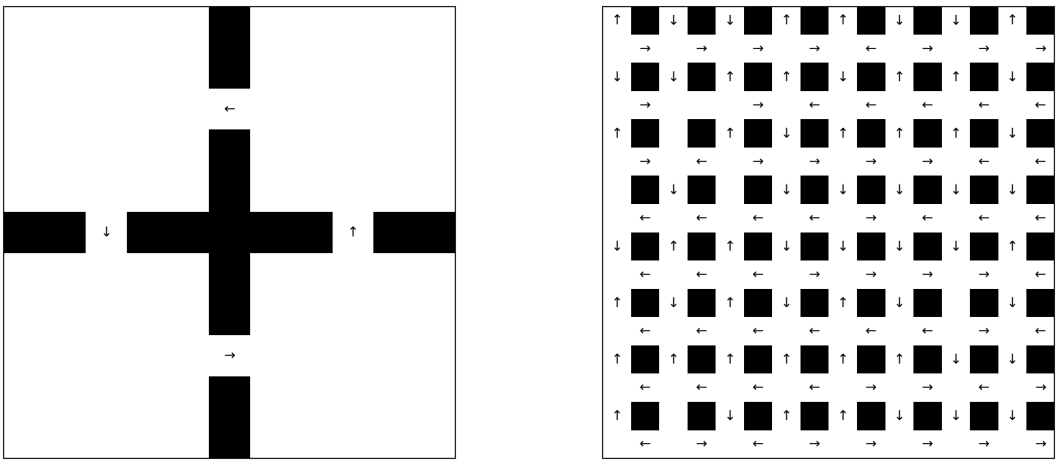
Since we tested TD and TD(λ) in a large number of environments, with different order of magnitudes of number of states or different γ, we had to find a reasonable scaling of the parameters t_0 and β with respect to the number of states S and γ.

For TD, we observed that choosing $\beta = t_0 = \frac{S}{1-\gamma}$ always performs very well. More precisely, for a large set of circle environments, torus environments, and mazes, we tested multiple t_0 and β, and no other choice of hyperparameter performed significantly better in these experiments than $\beta = t_0 = S$. Therefore, we used only these hyperparameters in the reported results.

For TD(λ), we searched for a similar scaling of the hyperparameters, as a function of γ, S, and λ. We observed that $\beta = t_0 = S\frac{1-\gamma\lambda}{1-\gamma}$ always performs very well, which means that in the set of environment we tested, for a fixed value of λ, no other choice of t_0 and β performed significantly better. [2] Then, for every environment, we tested multiple values of λ and selected the best one with cross validation. [3]

[2] The value $S\frac{1-\gamma\lambda}{1-\gamma}$ might not seem intuitive. It corresponds to the scaling of $\beta = t_0 = \frac{S}{1-\gamma}$ used for TD with the scaling of the vector e of eligibility traces. Indeed, the sum of the components of the eligibility trace is equal to $\frac{1}{1-\gamma\lambda}$. In particular if $\lambda = 0$, this schedule coincides with TD's schedule.

[3] In order to set λ, we consider the characteristic time of TD(λ): $\tau_\lambda = \frac{1}{1-\gamma\lambda}$. Intuitively, τ_λ is the number of



(a) Small non reversible environment

(b) Generated maze with a grid of width 8

Figure 9.4: Tabular environments, and the successor states for a given starting point.

For every environment, we run 5 experiments with TD with the schedule introduced above, and 5 experiments with SSIPE. For TD(λ), λ is selected by cross-validation: we run 10 experiments for every value of λ as explained above, then select the best λ with the first 5 runs, and report in the figure the performance on the remaining 5 runs. In the figure, we report the mean loss over the 5 loss, and a confidence interval corresponding to the standard deviation estimated on the 5 runs.

The loss is the rescaled $L_1(\rho)$ loss on the value function: $(1-\gamma)\sum_s \rho(s)|\hat{V}(s) - V(s)|$. With this rescaled loss, the loss at the initialization is exactly 1.

We refer to the main text for a discussion of the results.

9.B Additional experiments in a toy continuous environment: the low-dimensional torus.

Next, we considered a simple d-dimensional torus environment. A state in the torus is a d-tuple $(\theta_1, ..., \theta_d)$ with $0 \leqslant \theta_i < 1$. We then define two random walks on the torus: a reversible and a non-reversible random walk. In the reversible random walk, the next state s_{t+1} is sampled from a starting point s_t as $s_{t+1} = s_t + \sigma\varepsilon \bmod 1$, where ε is a random normally distributed vector of dimension d. In the non-reversible random walk, the next state is sampled as $s_{t+1} = s_t + \sigma|\varepsilon| \bmod 1$, where $|x|$ is the elementwise absolute value; this produces a directional drift. In practice, we used $\sigma = 0.05$ and defined $\gamma = 0.95$.

We parametrize the state space as $(\cos(2\pi\theta_1), \sin(2\pi\theta_1), ..., \cos(2\pi\theta_d), \sin(2\pi\theta_d))$, so that a state is represented by $2d$ numbers. We learn M with a MLP with 2 hidden layers (defined as for the continuous maze environment) of width 512.

This is a continuous environment, so it is not possible to compute the error on M by computing the true successor state operator. Still, in dimension $d = 1$ and $d = 2$, we can compute an approximation of the true successor state operator by choosing a discretization, computing the corresponding approximate transition operator P, and the approximate successor state operator M. In practice, in dimension $d = 1$ we used discretized the environment to 100 states, and in dimension $d = 2$ to 400 states. We learn M with Forward TD, Backward TD, and SSIPE. We use a single training trajectory (while we were using 64 parallel trajectories for the continuous maze). At every step, we sample 128 additional states (s_1, s_2) for Forward TD, Backward TD and SSIPE, from a uniform distribution.

In dimension 1 and 2, Forward and Backward TD on M are trained with the Adam optimizer with its default hyperparameters, including the constant learning rate 10^{-3}. In dimension 1, SSIPE is trained with Adam with its default hyperparameters. In dimension 2, it is trained with Adam with a learning rate 10^{-4}.

---

steps considered in the eligibility traces. Since $\lambda \in [0, 1]$, we know that $\tau_\lambda \in [1, \frac{1}{1-\gamma}]$. We parametrize τ_λ as a function of $\varepsilon \in [0, 1]$: $\tau_\lambda = 1 + \varepsilon(\frac{1}{1-\gamma} - 1)$. If $\varepsilon = 0$, then $\lambda = 0$. If $\varepsilon = 1$, then $\lambda = 1$. This parametrization corresponds to $\lambda = \frac{\varepsilon}{1-\gamma(1-\varepsilon)}$). With this parametrization, by choosing ε uniformly in $[0, 1]$ we obtain a range of values of λ well suited to the environment; otherwise, the grid search for λ would have been much larger to cover suitable values for every environment. For every environment, we tested values of ε in $\{1., .8, .6, .4, .2, .1, .01, .001\}$ and took the corresponding values of λ.

In Figure 9.3, we report the rescaled total variation error on M, estimated on the state space discretization: $\frac{1}{N^2}\sum_{s_1,s_2}|m_\theta(s_1,s_2) - m(s_1,s_2)|$, where $m(s_1,s_2)$ is the "true" value of m estimated on the discretization, as discussed above.

In these small environments, Forward TD, Backward TD and SSIPE are all able to learn, but SSIPE is always less sample efficient than Forward TD and Backward TD (Fig. 9.3).

Chapter 10

Non-asymptotic convergence bounds for policy evaluation via SSIPE in tabular setting

In this chapter, we provide non-asymptotic convergence bounds for the SSIPE algorithm defined in Chapter 9. We then compare these bounds with known convergence bounds for TD.

The convergence of policy evaluation methods is a well-studied topic. Such an analysis can be done for several observation models. The first one is via trajectories. In an ergodic process, we observe an infinite trajectory $\tau = (s_0, r_0, s_1, r_1, ...)$, with $r_t \sim \mathcal{R}(.|s_t)$ and $s_{t+1} \sim P(.|s_t)$ for every t. The second one is the i.i.d. observation model: we observe independent transitions (s_t, r_t, s'_t), with $r_t \sim \mathcal{R}(.|s_t)$ and $s'_t \sim P(.|s_t)$. Typically, we can assume that for every t, $s_t \sim \rho$ where ρ is the ergodic measure of the process. The i.i.d. model simplifies the theoretical analysis as it removes the dependence between observations. Finally, the last one is the *synchronous* observation model: at every step t, we observe a transition from every state $s \in \mathcal{S}$: formally at step t, we observe the vector of states $(s'_t)^{(s)}_{s\in\mathcal{S}}$, where for every $s \in \mathcal{S}$, $(s'_t)^s \sim P(\mathrm{d}s'|s)$. This last model again simplifies analysis, as it removes the issue of infrequently observed states.

The convergence of temporal difference was studied in these settings, in asymptotic (Tadić, 2004; Polyak and Juditsky, 1992; Jaakkola et al., 1994; Borkar and Meyn, 2000; Devraj and Meyn, 2017; Tsitsiklis and Van Roy, 1997; Ueno et al., 2008; Tagorti and Scherrer, 2015) or non-asymptotic regimes (Lakshminarayanan and Szepesvari, 2018; Bhandari et al., 2018; Srikant and Ying, 2019; Dalal et al., 2018; Khamaru et al., 2020). In this chapter, we will only consider the *tabular* setting, in which we estimate a vector $\hat{V} \in \mathbb{R}^S$. Additionally, Pananjady and Wainwright (2019) studied the sample efficiency of the process estimation estimate (9.1.1), (called the *plug-in estimate* in their setting) under the synchronous (or generative) observation model, in a tabular setting. In the tabular case, the SSIPE approach corresponds to a very specific case of the LSTD algorithm, whose sample efficiency was also studied in multiple settings (Lazaric et al., 2012; Tagorti and Scherrer, 2015).

In the next section, we derive non-asymptotic convergence bounds for the *SSIPE* method for policy evaluation, under the i.i.d. observation model, for the $L_1(\rho)$ norm, where ρ is the ergodic measure of the process. The most interesting feature this new convergence bound is that it does not depend on the number of states, or of the measure of infrequently visited states. Hence the result is non-vacuous even if some states are almost-never observed, or for a very large number of states, or even for a discrete infinite state space. Up to our knowledge, it is the first convergence bound for policy evaluation which shows that we can provably learn the value function in finite time, even with an arbitrarily large (or infinite) state space.

10.1 Statement of the convergence bounds

We now introduce non-asymptotic convergence bounds for the process estimation method in Theorem 10.1 and Theorem 10.2. The first ones introduced in Theorem 10.1 are slightly simpler to understand. Their main interesting property is that they do not depend on properties of ρ, especially on the measure of the most infrequently visited state $\inf_s \rho(s)$: if some states are only visited with probability ε and we take $\varepsilon \to 0$, the bound is still non-vacuous. As discussed afterwards in Section 10.2 with an example, it is not the case of known bounds for TD under the i.i.d. observation model. This is a desirable result: if a state s is *almost never observed*, our estimate $V(s)$ will be inaccurate. But since we consider the $L_1(\rho)$ or $L_2(\rho)$ norms which weights the error in state s with ρ_s, we could hope that this error would be controlled.

The second bounds introduced in Theorem 10.2 are stronger, and removes the dependence in the number of states S, without any additional hypothesis on the distributions ρ or on P. Hence, it is possible to learn the value function in finite time, with theoretical guarantees, in the tabular setting, even with an infinite number of states.

We assume ρ is the stationary distribution of the process P. We consider the i.i.d observation model: at every step $t \in \mathbb{N}$, we observe $s \sim \rho$, $s' \sim P_{ss'}$. We define the $L_1(\rho)$ norm as:

$$\|f\|_{L_1(\rho)} := \sum_s \rho(s)|f(s)| = \mathbb{E}_{s\sim\rho}\left[|f(s)|\right] \tag{10.1.1}$$

We consider a finite MRP with S and E edges ((s, s') is an edge if $P_{ss'} > 0$), and a bounded reward.

We consider the SSIPE estimates $\hat{V}_t$ and $\hat{M}_t$ defined in Theorem 9.1. These estimates corresponds to estimates $(\hat{P}, \hat{R})$ of the transition matrix P and reward vector R of the Markov reward process, by direct empirical averages; then define $\hat{M}_t = (\mathrm{Id} - \gamma\hat{P}_t)^{-1}$ and $\hat{V}_t = \hat{M}_t\hat{R}_t$. The empirical averages $\hat{P}$ and $\hat{R}$ are updated for each new transition $s \to s'$ with reward r_s, by updating the row s of the transition matrix $\hat{P}$, and the value $\hat{R}_s$ at s:

$$\hat{P}_{ss_2} \leftarrow (1 - 1/n_s)\hat{P}_{ss_2} + (1/n_s)\mathbb{1}_{s_2=s'}\forall s_2, \tag{10.1.2}$$

$$\hat{R}_s \leftarrow (1 - 1/n_s)\hat{R}_s + (1/n_s)r_s \tag{10.1.3}$$

with n_s the number of visits to state s up to time t. We initialize to $\hat{P}_0 = 0$ and $\hat{R}_0 = 0$. This algorithm corresponds to the Plug-in estimate in (Pananjady and Wainwright, 2019) and is similar to the LSTD algorithm (Bradtke and Barto, 1996; Geramifard et al., 2006), as explained in the previous chapter.

Then we have the following first theorem.

Theorem 10.1. *Consider a finite Markov reward process with S states and E edges , rewards almost surely bounded by $R_{\max}$, and stationary distribution ρ. We consider the SSIPE updates estimates $\hat{M}$ and $\hat{V}$.*

Then after t i.i.d. observations ($s \sim \rho$, $s' \sim P_{ss'}$), with probability $1 - \delta$, the estimates $\hat{M} = (\mathrm{Id} - \gamma\hat{P})^{-1}$ and $\hat{V} = \hat{M}\hat{R}$ satisfy

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{2\gamma}{(1-\gamma)^2}\sqrt{\frac{2E}{t}\log\frac{2}{\delta}} \tag{10.1.4}$$

and

$$\|\hat{V}(s) - V(s)\|_{L_1(\rho)} \leqslant \frac{3R_{\max}}{(1-\gamma)^2}\sqrt{\frac{2E}{t}\log\frac{4S}{\delta}}. \tag{10.1.5}$$

We observe that these convergence bounds are independent of ρ, and in particular of the measure of *almost never visited* states. As discussed in Section 10.2, this is not the case in standard bounds for policy evaluation under the i.i.d. observation model.

We now present a second convergence bound for $\hat{M}$ and $\hat{V}$ in the same setting (tabular, i.i.d. observation model, SSIPE algorithm). The most interesting feature this new convergence

bound on M (equation (10.1.8)) is that it does not depend anymore on the number of states. Hence, this bound is still non-vacuous for very large state spaces, or even infinite sets.

Our approach strongly relies on convergence bounds for learning discrete distributions with potentially infinite support from Cohen et al. (2020). In their work, they introduce the quantity $\Lambda_t(p)$, if p is a probability distribution over a discrete (but potentially infinite) set $\mathcal{S}$, as:

$$\Lambda_t(p) := \sum_{s|p_s<1/t} p_t + \frac{1}{\sqrt{t}} \sum_{s|p_s\geqslant 1/t} \sqrt{p_s} \tag{10.1.6}$$

The sequence $\Lambda_t(p)$ is decreasing when t increases, and we have $\Lambda_t(p) \geqslant \frac{1}{\sqrt{t}}$. If $\mathcal{S}$ is finite, we have $\Lambda_t(p) \leqslant \sqrt{\frac{S}{t}}$ (this bound is achieved for a uniform probability distribution). In a more general case, even for an infinite set $\mathcal{S}$ but if p has a finite $\|.\|_{1/2}$ norm ($\|p\|_{1/2} = \left(\sum_{s\in\mathcal{S}} \sqrt{p_s}\right)^2$, then $\Lambda_t \leqslant \sqrt{\frac{\|p\|_{1/2}}{t}}$ (the bound is achieved if $t \geqslant \frac{1}{\inf_s p_s}$). In the general case, the quantity $\Lambda_t(p)$ is well defined even if $\|p\|_{1/2}$ is infinite. Informally, it removes the issue of distributions with infinite $\|.\|_{1/2}$ norm by *almost ignoring* states with probability less than $\frac{1}{t}$, after t observations.

We will use the notation $\Lambda_t(\rho P)$, which is Λ_t applied to ρP seen as a probability distribution over pair of states (s, s'):

$$\Lambda_t(\rho P) = \sum_{(s,s')|\rho_s P_{ss'}<1/t} \rho_s P_{ss'} + \frac{1}{\sqrt{t}} \sum_{(s,s')|\rho_s P_{ss'}\geqslant 1/t} \sqrt{\rho_s P_{ss'}} \tag{10.1.7}$$

Theorem 10.2. *Consider a finite Markov reward process, rewards almost surely bounded by $R_{\max}$, and stationary distribution ρ. Update $\hat{P}$ and $\hat{R}$ online via* (10.1.2)*, initialized to $\hat{P} = \hat{R} = 0$.*

Then after t i.i.d. observations ($s \sim \rho$, $s' \sim P_{ss'}$), with probability $1-\delta$, the estimates $\hat{M} = (\mathrm{Id} - \gamma\hat{P})^{-1}$ and $\hat{V} = \hat{M}\hat{R}$ satisfy

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{2\gamma}{(1-\gamma)^2}\left(2\Lambda_t(\rho P) + \sqrt{\frac{11\log\frac{3}{\delta}}{t}}\right) \tag{10.1.8}$$

and

$$\|\hat{V}(s) - V(s)\|_{L_1(\rho)} \leqslant \frac{R_{\max}}{(1-\gamma)^2}\left(10\Lambda_t(\rho P) + 9\sqrt{\frac{\log\frac{4}{\delta}}{t}}\right) \tag{10.1.9}$$

We can briefly compare the two theorems 10.1 and 10.2. If we use $\Lambda_t(\rho P) \leqslant \sqrt{\frac{E}{t}}$ in bounds of Theorem 10.2, we recover results of the same order than the bounds in Theorem 10.1. Hence, the bounds in Theorem 10.2 are strictly improving the bounds of Theorem 10.1, especially in cases in which the probability distribution ρP is far from uniform, and contain many states with low-probability, or if the state space $\mathcal{S}$ is infinite[1]. In particular, it shows we can learn the value function in finite time, with theoretical guarantees, even in an arbitrarily large or infinite state space.

In the next section, we compare these results with other known convergence bounds for policy evaluation.

10.2 A discussion of Theorem 10.1 and 10.2 and comparison with related works

We now discuss the Theorems 10.1 and 10.2 compared to two results:

[1]The case of the tabular model in an infinite state space is irrelevant in practice, but it is mathematically interesting to observe that we can perform policy evaluation in a finite time, with theoretical guarantees, even with an infinite state space. Formally, at step t, the matrix $(\mathrm{Id} - \gamma\hat{P})$ is equal to Id, except on a finite sub-matrix. Hence, we can compute $(\mathrm{Id} - \gamma\hat{P})^{-1}$, as it only requires to inverse the finite sub-matrix not equal to Id

- The finite-time convergence bound for Temporal Difference in (Bhandari et al., 2018), which uses the same data sampling model as ours, but for Temporal Difference, and for the $L_2(\rho)$ norm instead of the $L_1(\rho)$ norm.
- The finite-time convergence result for the *Plug-in* estimate in (Pananjady and Wainwright, 2019). Their algorithm is strictly equivalent to SSIPE, but the data sampling strategy is different, and the result is for the L_∞ norm instead of the $L_1(\rho)$ norm.

10.2.1 A discussion of Theorem 10.1 compared to TD convergence bound

In their analysis, (Bhandari et al., 2018) consider Linear TD, which is more general than our approach. We consider the specific case of their Theorem 2.c, applied to a *tabular* model, which is a specific case of the *linear* model. They consider the following TD update: When observing a transition (s_t, r_t, s'_t) with $r_t \sim \mathcal{R}(.|s_t)$ and $s'_t \sim P(\mathrm{d}s'|s_t)$, we update V as:

$$V(s_t) \leftarrow (1-\alpha_t)V(s_t) + \alpha_t(r_t + \gamma V(s'_t)) \tag{10.2.1}$$

with $\alpha_t = \frac{\beta}{\lambda+t}$ with $\beta = \frac{2}{(1-\gamma)\omega}$, $\lambda = \frac{16}{(1-\gamma)^2\omega}$ and $\omega = \sqrt{\inf_s \rho(s)}$. In that case, we can apply their Theorem 2.c.[2] to the tabular TD update: if V_t is the estimate after t i.i.d. observation, we have:

$$\mathbb{E}\left[\|V^\pi - V_t\|^2_{L_2(\rho)}\right] \leqslant \frac{1}{(1-\gamma)^2}\frac{\nu}{\lambda+t} \tag{10.2.4}$$

where

$$\nu = \max\left\{\frac{8\gamma^2\|\sigma_V\|^2_{L_2(\rho)}}{\inf_s \rho(s)}, \frac{16\,\|V^\pi - V_0\|^2_{L_2(\rho)}}{\sqrt{\inf_s \rho(s)}}\right\} \tag{10.2.5}$$

and $\sigma_V \in \mathbb{R}^S$ is the vector of variances of the true value at the next step s' for every s:

$$\sigma^2_V(s) := \mathrm{Var}_{s'\sim P(\mathrm{d}s'|s)}(V^\pi(s')) \tag{10.2.6}$$

Our main remark is on the dependence in ρ. Indeed, in our analysis of the SSIPE estimates in Theorems 10.1 and 10.2, the result do not depend on properties of ρ, especially on the measure of the most infrequently visited state $\inf_s \rho(s)$. This is an interesting property. It means that adding an *almost-never-visited* state does not damage the policy evaluation in the environment. On the contrary, with the bound for TD with norm $L_2(\rho)$ (10.2.4), if a state has low measure $\rho(s)$, it can hurt the guarantee we have on our estimate, even though the norm $L_2(\rho)$ is weighting the loss with respect to ρ. For instance, consider the 3-states MDP defined in Figure 10.1 with deterministic reward $R(A) = 0$, $R(B) = 1$ and $R(B') = -1$. Then the value function can be computed exactly and is $V(A) = 0$, $V(B) = 1$ and $V(B') = -1$, and we have $\|\sigma_V\|^2_{L_2(\rho)} = \frac{(1-\varepsilon)}{2}$. For simplicity, we take $V_0 = 0$. Therefore, for fixed γ, when $\varepsilon \to 0$, $t \to \infty$, the bound (10.2.4) becomes asymptotically:

$$\mathbb{E}\left[\|V^\pi - V_t\|^2_{L_2(\rho)}\right]^{1/2} \leqslant O_{\varepsilon\to 0, t\to\infty}\left(\frac{1}{\sqrt{\varepsilon t}}\right) \tag{10.2.7}$$

[2]Here is how we derive the tabular case of the Theorem 2.c in (Bhandari et al., 2018). We keep their notation for simplicity. For every state s, the feature vector $\varphi(s)$ in the tabular model is $\alpha(s)\mathbb{1}_s$, where $\alpha(s) \in \mathbb{R}$ is a scaling factor. Because of their assumption on feature regularity, requiring $\|\varphi(s)\|_2 \leqslant 1$, we have:

$$\|\varphi(s)\|_2^2 = \alpha(s)^2\mathbb{1}_s^\top\Sigma\mathbb{1}_s \tag{10.2.2}$$

$$= \alpha(s)^2\sigma_{ss} = \alpha(s)^4\rho(s) \leqslant 1 \tag{10.2.3}$$

where $\pi(s)$ in the original paper corresponds to $\rho(s)$ in this text. Therefore, we take: $\alpha(s) = \rho(s)^{-1/2}$, and we have $\Sigma = \mathrm{Diag}(\rho^{1/2})$, with ω the minimum eigenvalue of Σ: $\omega = \sqrt{\inf_s \rho(s)}$. Then, we can apply the theorem

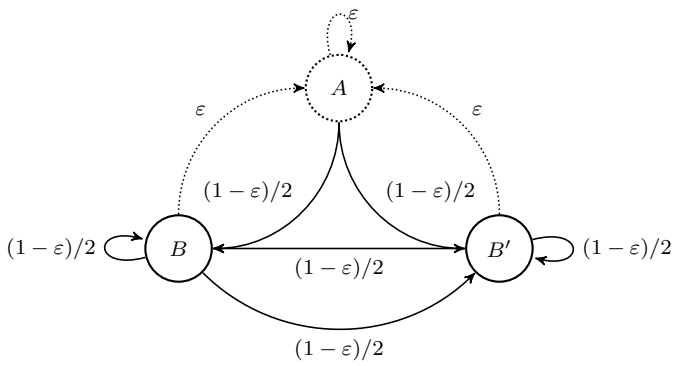


Figure 10.1: A 3-states Markov Process with parameter ε. The state A is repulsive, and its ergodic measure is ε. The states B and B' are symmetric and their ergodic measure are $(1-\varepsilon)/2$

On the contrary, in the same MRP, the bounds in Theorems 10.1 and 10.2 becomes, for a fixed γ and with fixed probability $1-\delta$:

$$\|V_t(s) - V(s)\|_{L_1(\rho)} \leqslant O_{\varepsilon\to 0, t\to\infty}\left(\frac{1}{\sqrt{t}}\right). \tag{10.2.8}$$

Hence, with this example, we see that taking ε very small makes the theoretical bound given by the bound obtained in (Bhandari et al., 2018) vacuous. This is true for other sample efficiency bounds for policy evaluation, such as (Lazaric et al., 2012; Tagorti and Scherrer, 2015) in the case of LSTD, even though these bounds are tighter with respect to other parameters of the environment. On the contrary, our bounds in in Theorems 10.1 and 10.2 are invariant to ε, hence still gives a theoretical guarantee when ε is small. This is a desirable result: if a state s is *almost never observed*, our estimate $V(s)$ will be inaccurate, but because we study $L_1(\rho)$ or $L_2(\rho)$ norms which weights the error in state s with ρ_s, this error is s should be controlled.

Similarly, if we add a large number S' of *almost never visited states* (such that $\rho_s \leqslant \varepsilon/S'$) and consider $S' \to \infty$, the bounds in Theorem 10.2 are still non-vacuous, while bounds in (Bhandari et al., 2018) or even in Theorem 10.1 are vacuous.

In these case, we thus show that we can provably learn the value function, in finite time, with theoretical guarantees, even with arbitrarily large or infinite state spaces.

We can now compare the bounds with respect to γ. For a fixed finite MDP $\mathcal{M}$, when $\gamma \to 1$, $t \to \infty$, the bound (10.2.4) becomes asymptotically:

$$\mathbb{E}\left[\|V^\pi - V_t\|^2_{L_2(\rho)}\right]^{1/2} \leqslant O_{\gamma\to 1, t\to\infty}\left(\frac{\|\sigma_V\|_{L_2(\rho)}}{(1-\gamma)\sqrt{t}}\right) \tag{10.2.9}$$

and we consider $\inf_s \rho(s)$ as a constant here. On the contrary, the bound in Theorem 10.1 is

$$\|V_t(s) - V(s)\|_{L_1(\rho)} \leqslant O_{\gamma\to 1, t\to\infty}\left(\frac{1}{(1-\gamma)^2\sqrt{t}}\right). \tag{10.2.10}$$

In the worst case, $\|\sigma_V\|_{L_2(\rho)}$ is of order $\frac{1}{1-\gamma}$ (because V is of order at most $R_{\max}/(1-\gamma)$ and σ_V is the state-wise variance of the value function at the next step). In that case, the two bounds have the same order of magnitude. But $\|\sigma_V\|_{L_2(\rho)}$ can be of order 1, as in the 3-state process introduced above. Hence the bound obtained in (Bhandari et al., 2018) handles some of the structure of the environment and can be tighter with respect to γ.

There are three main differences between the bounds (10.1.5) and (10.2.4): the algorithms (SSIPE and TD), the norms ($L_1(\rho)$ and $L_2(\rho)$), and the proof techniques we use to derive the bounds. Therefore, the conclusion of this discussion is unclear. Would it be possible to derive a

bound for TD for the $L_2(\rho)$ norm, which would not depend on $\inf_s \rho_s$? If it is not, is it because of the TD algorithm itself? Or would any policy evaluation method in the $L_2(\rho)$ depend on $\inf_s \rho_s$ because this norms gives too much weight to errors in infrequently visited states? These questions would be interesting directions for future works.

10.2.2 A discussion of Theorem 10.1 compared to (Pananjady and Wainwright, 2019)

The plug-in estimate studied by (Pananjady and Wainwright, 2019) is equivalent to the SSIPE algorithm defined in Chapter 9. Their data sampling model is not the same as ours: they use the *synchronous* data sampling model: at every step, a transition starting from every state is observed. This model is less realistic than the i.i.d model used for the bounds in Theorem 10.1 and Theorem 10.2, as well as (10.2.4). It naturally removes the issue of infrequently observed states, and allow a more precise analysis of the convergence rate. We consider the $\|.\|_\infty$ norn defined as $\|x\|_\infty = \sup_s |x_s|$. Then, their Theorem 1.b is: with probability $1-\delta$, the $\|.\|_\infty$ error is:

$$\|V_t(s) - V^\pi(s)\|_\infty \leqslant O_{t\to\infty}\left(\sqrt{\frac{S\log(S/\delta)}{t}}\|M\sigma_V\|_\infty\right). \tag{10.2.11}$$

where t is not number of observed transitions (with the synchronous model, at every step we observed S transitions) and where $\sigma_V(s) := \mathrm{Var}_{s'\sim P(\mathrm{d}s'|s)}(V^\pi(s'))$ as already introduced above. We simplified their statement to highlight the most important part of the bound. Because of the synchronous observation model, there is no issue for handling state with *almost never* observed in the process. Still, this bound cannot handle infinite state space $\mathcal{S}$, while our bounds in Theorem 10.2 can.

Informally, we can compare the factors in the $O(\frac{1}{\sqrt{t}})$ term. Pananjady and Wainwright (2019) obtain a factor $\|M\sigma_V\|_\infty$. Both M and σ_V can be of order $\frac{1}{1-\gamma}$, hence the factor $\|M\sigma_V\|_\infty$ can be of order $\frac{1}{(1-\gamma)^2}$ at most, but can be much smaller is some cases. On the contrary, in our bound (10.1.5), the corresponding factor is $\frac{1}{(1-\gamma)^2}$.

10.3 Proof of convergence bounds

We now prove Theorem 10.1 and Theorem 10.2. This convergence analysis is partially inspired by (Pananjady and Wainwright, 2019). The main differences are the data model and the metrics computed. The main results used to go from Theorem 10.1 to Theorem 10.2 is the study of discrete distributions with infinite support estimation by Cohen et al. (2020).

10.3.1 Proof of Theorem 10.1

Proof. We assume that ρ is an invariant probability measure of P, and that the reward is bounded by $R_{\max}$ with probability 1. We define the empirical distribution of states $\hat{\rho}_t$ as: $(\hat{\rho}_t)_s = \frac{n_s}{t}$, with n_s the number of visits to state s up to time t. We also consider $\hat{P}_t$ and $\hat{R}_t$ as defined in (10.1.2).

The initialization of $\hat{P}$ and $\hat{R}$ does not matter, as it is erased the first time a state is visited. To fix ideas, we initialize $\hat{P}$ and $\hat{R}$ to 0; this helps if $\rho = 0$ for some states. (In particular, $\hat{P}$ may be substochastic: $0 \leqslant \sum_j \hat{P}_{ij} \leqslant 1$ for all i.)

We define $\widehat{\rho P}_t$ as the empirical distribution of transitions: $(\widehat{\rho P}_t)_{s_1 s_2} := \frac{n_{s_1 s_2}}{t}$ where $n_{s_1 s_2}$ is the number of observations of a transition (s_1, s_2) up to time t. We have $(\hat{P}_t)_{s_1 s_2} = \frac{n_{s_1 s_2}}{n_{s1}}$ if $n_{s1} > 0$, or 0 if $n_{s1} = 0$. Hence $\left(\widehat{\rho P}_t\right)_{s_1 s_2} = \hat{\rho}_{s_1}(\hat{P}_t)_{s_1 s_2}$.

The proof strategy is to bound the errors $\|\hat{M} - M\|_{\rho,\mathrm{TV}}$ and $\|\hat{V} - V\|_\rho$ by errors on $\widehat{\rho P}$ and $\hat{R}$. The error on $\widehat{\rho P}$ can then be controlled by concentration inequalities on empirical distributions, and the error on $\hat{R}$ can be bounded via the Hoeffding inequality.

The successor state operator estimate $\hat{M}$ is $(\mathrm{Id} - \gamma\hat{P})^{-1}$. By the Bellman equation for M and $\hat{M}$,

$$\begin{aligned}\hat{M} - M &= \gamma\hat{P}\hat{M} - \gamma PM \\ &= \gamma P(\hat{M} - M) + \gamma(\hat{P} - P)\hat{M}\end{aligned}$$

and therefore

$$(\mathrm{Id} - \gamma P)(\hat{M} - M) = \gamma(\hat{P} - P)\hat{M}$$

and thus

$$\hat{M} - M = \gamma M(\hat{P} - P)\hat{M}$$

by definition of M.

Therefore,

$$\begin{aligned}\|\hat{M} - M\|_{\rho,\mathrm{TV}} &= \gamma\|M(\hat{P} - P)\hat{M}\|_{\rho,\mathrm{TV}} \\ &= \frac{\gamma}{2}\sum_{i,j}\rho_i\left|\sum_{k,l}M_{ik}(\hat{P} - P)_{kl}\hat{M}_{lj}\right| \\ &\leqslant \frac{\gamma}{2}\sum_{i,j,k,l}\rho_i M_{ik}|\hat{P} - P|_{kl}\hat{M}_{lj}.\end{aligned}$$

We know that $(1-\gamma)M$ is a stochastic matrix, and ρ is an invariant probability measure. Therefore, $\sum_i \rho_i M_{ik} = \frac{1}{1-\gamma}\rho_k$. Moreover, if $\hat{P}$ is sub-stochastic, $\sum_j \hat{M}_{lj} \leqslant \frac{1}{1-\gamma}$ (with equality if P is stochastic). Therefore,

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{\gamma}{(1-\gamma)^2}\|\hat{P} - P\|_{\rho,\mathrm{TV}}. \tag{10.3.1}$$

We define (ρP) as the matrix $\mathrm{Diag}(\rho)P$. We now bound the error $\|\hat{P} - P\|_{\rho,\mathrm{TV}}$ by the error $\|\widehat{\rho P} - (\rho P)\|_{\mathrm{TV}}$, in order to use standard concentration inequalities on empirical distributions:

$$\begin{aligned}\|\hat{P} - P\|_{\rho,\mathrm{TV}} &= \frac{1}{2}\|\mathrm{Diag}(\rho)\hat{P} - (\rho P)\|_1 \\ &\leqslant \|\widehat{\rho P} - (\rho P)\|_{\mathrm{TV}} + \frac{1}{2}\|\mathrm{Diag}(\hat{\rho} - \rho)\hat{P}\|_1 \\ &\leqslant \|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} + \|\hat{\rho} - \rho\|_{\mathrm{TV}} \\ &\leqslant \|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} + \frac{1}{2}\sum_i|\sum_j \hat{\rho}_i\hat{P}_{ij} - \rho_i P_{ij}| \\ &\leqslant \|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} + \frac{1}{2}\sum_{i,j}|\hat{\rho}_i\hat{P}_{ij} - \rho_i P_{ij}| \\ &\leqslant 2\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}}\end{aligned}$$

Therefore,

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{2\gamma}{(1-\gamma)^2}\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}}. \tag{10.3.2}$$

We now consider the error on $\hat{V}$.

We have:

$$\begin{aligned}\|\hat{V} - V\|_\rho &= \|\hat{M}\hat{R} - MR\|_\rho \\ &\leqslant \|(\hat{M} - M)\hat{R}\|_\rho + \|M(\hat{R} - R)\|_\rho \\ &\leqslant 2R_{\max}\|\hat{M} - M\|_{\rho,\mathrm{TV}} + \frac{1}{1-\gamma}\|\hat{R} - R\|_\rho \\ &\leqslant \frac{4R_{\max}}{(1-\gamma)^2}\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} + \frac{1}{1-\gamma}\|\widehat{R} - R\|_{\hat{\rho}}\end{aligned} \tag{10.3.3}$$

We now bound $\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}}$ and $\|\widehat{R} - R\|_{\hat{\rho}}$.

We can bound the error $\|\widehat{R} - R\|_{\hat{\rho}}$ with the Hoeffding inequality. $\widehat{R}_s$ is the average of n_s independent samples of expectation R_s. Since the reward is bounded by R_{max} with probability 1, we can use Hoeffding's inequality. For any s with $n_s > 0$, we have:

$$\mathbb{P}(|\widehat{R} - R|_s > u) \leqslant 2\exp\left(-\frac{n_s u^2}{2R_{\max}^2}\right)$$

Hence, for any s with $n_s > 0$, we have with probability $1 - \frac{\delta}{S}$:

$$|\widehat{R_t} - R|_s \leqslant R_{\max}\sqrt{\frac{2}{n_s}\log\frac{2S}{\delta}}$$

and since $\hat{\rho}_s = n_s/t$, this rewrites as

$$\hat{\rho}_s|\widehat{R_t} - R|_s \leqslant \frac{R_{\max}}{t}\sqrt{2n_s \log \frac{2S}{\delta}}. \tag{10.3.4}$$

For states with $n_s = 0$, $\hat{\rho}_s = 0$ and the inequality still holds. If for all s, $\mathbb{P}(|\widehat{R_t} - R|_s \geqslant \varepsilon_s) \leqslant \frac{\delta}{S}$, then

$$\begin{aligned}\mathbb{P}(\|\hat{R} - R\|_{\hat{\rho}} \geqslant \sum_s \varepsilon_s) &\leqslant \sum_s \mathbb{P}(\hat{\rho}_s|\widehat{R_t} - R|_s \geqslant \varepsilon_s) \\ &\leqslant \sum_s \frac{\delta}{S} = \delta\end{aligned}$$

Thus, with probability $1 - \delta$,

$$\begin{aligned}\|\widehat{R} - R\|_{\hat{\rho}} &\leqslant \frac{R_{\max}}{t}\sqrt{2\log\frac{2S}{\delta}}\sum_s \sqrt{n_s} \\ &\leqslant \frac{R_{\max}}{t}\sqrt{2\log\frac{2S}{\delta}}\sqrt{\sum_s n_s}\sqrt{S} \\ &\leqslant \frac{R_{\max}}{\sqrt{t}}\sqrt{2S\log\frac{2S}{\delta}}\end{aligned} \tag{10.3.5}$$

since $\sum_s n_s = t$.

We now bound $\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}}$. $\widehat{\rho P}$ is the empirical distribution over all possible transitions. The set of all possible transitions is of size S^2. However, if $(\rho P)_{s_1 s_2} = 0$, then with probability 1, $\widehat{\rho P}_{s_1 s_2} = 0$. Therefore, if E is the number of edges of the MDP ((s, s') is an edge if $P_{ss'} > 0$), $\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}}$ can be bounded by an inequality on the total variation error of the empirical measure on a set of size E. We use Theorem 2.2 from (Weissman et al., 2003)[a], and have with with probability $1 - \delta$:

$$\|\widehat{\rho P}_t - \rho P\|_{\mathrm{TV}} \leqslant \frac{1}{2\sqrt{t}}\sqrt{2E\log\frac{2}{\delta}} \tag{10.3.6}$$

By plugging equation (10.3.6) into (10.3.2), with probability $1 - \delta$,

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{\gamma}{(1-\gamma)^2\sqrt{t}}\sqrt{2E\log\frac{2}{\delta}}$$

Finally, by plugging (10.3.5) and (10.3.6) into (10.3.3), with probability $1 - \delta$, we obtain

$$\begin{aligned}\|\hat{V} - V\|_\rho &\leqslant \frac{2R_{\max}}{(1-\gamma)^2}\frac{1}{\sqrt{t}}\sqrt{2E\log\frac{4}{\delta}} + \frac{1}{1-\gamma}\frac{R_{\max}}{\sqrt{t}}\sqrt{2S\log\frac{4S}{\delta}} \\ &\leqslant \frac{3R_{\max}}{(1-\gamma)^2}\sqrt{\frac{2E}{t}\log\frac{4S}{\delta}}\end{aligned}$$

which ends the proof.

[a] We use the trivial bound $\varphi(\pi) \geqslant 2$ with the notation of the original paper.

These bounds do not depend on the sampling measure ρ, although the norm used to define the error does. Thus, rarely visited points have no impact on these bounds.

10.3.2 Proof of Theorem 10.2

Proof. From (10.3.2), we know that:

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{2\gamma}{(1-\gamma)^2}\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}}$$

We now use Theorem 2.1 in (Cohen et al., 2020). With probability at least $1 - \delta$, we have:

$$\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} \leqslant \Phi_t(\widehat{\rho P}_t) + 3\sqrt{\frac{\log\frac{2}{\delta}}{2t}} \tag{10.3.7}$$

where Φ_t is defined in (Cohen et al., 2020). And from Theorem 2.3 in (Cohen et al., 2020), we now that with probability at least $1 - \delta$:

$$\Phi_t(\widehat{\rho P}_t) \leqslant 2\Lambda_t(\rho P) + \sqrt{\frac{\log(1/\delta)}{t}} \tag{10.3.8}$$

Hence, using (10.3.7) with probability $2\delta/3$ and (10.3.8) with probability $\delta/3$, we have with probability $1-\delta$:

$$\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} \leqslant 2\Lambda_t(\rho P) + \left(1 + \frac{3}{\sqrt{2}}\right)\sqrt{\frac{\log\frac{3}{\delta}}{t}}$$

Therefore, we can conclude:

$$\|\hat{M} - M\|_{\rho,\mathrm{TV}} \leqslant \frac{4\gamma}{(1-\gamma)^2}\Lambda_t(\rho P) + \frac{2\gamma}{(1-\gamma)^2}\sqrt{\frac{10\log\frac{3}{\delta}}{t}} \tag{10.3.9}$$

We now bound the reward error. For now, we will consider $\hat{\rho}$ and the sequence of observed states $\bar{s} := (s_1, ..., s_t)$ fixed. We first bound $\mathbb{E}_{r_1,...,r_t}\left[\|\hat{R} - R\|_{L_1(\hat{\rho})}\middle|\bar{s}\right]$, then we will concentrate conditionally to the sequence the error $\|\hat{R} - R\|_{L_1(\hat{\rho})}$.

We have:

$$\begin{aligned}
\mathbb{E}_{r_1,...,r_t}\left[\|\hat{R} - R\|_{L_1(\hat{\rho})}|\bar{s}\right] &= \sum_{s|n_s>0}\hat{\rho}_s\mathbb{E}_{r_1,...,r_t}\left[|\hat{R}_t - R|_s\right] \\
&\leqslant \sum_{s|n_s>0}\hat{\rho}_s\sqrt{\mathbb{E}_{r_1,...,r_t}\left[(\hat{R}_t - R_s)^2_s\right]} \\
&\leqslant \sum_{s|n_s>0}\hat{\rho}_s\frac{1}{\sqrt{n_s}}R_{\max} \\
&= R_{\max}\frac{1}{\sqrt{t}}\sum_s\sqrt{\hat{\rho}_s} \\
&\leqslant R_{\max}\Phi_t(\hat{\rho})
\end{aligned}$$

We now use MacDiarmid's inequality to concentrate $\|\hat{R} - R\|_{L_1(\hat{\rho})}$ around its mean, knowing $\bar{s} := (s_1, ..., s_t)$. We consider the function $f(r_1, ..., r_t) = \|\hat{R} - R\|_{L_1(\hat{\rho})}$, where $\hat{R}$ is the empirical model of the rewards after observing $r_1, ..., r_t$ in states $(s_1, ..., s_t)$. We have:

$$|f(r_1, ..., r_{i-1}, r, r_{i+1}, ..., r_t) - f(r_1, ..., r_{i-1}, r', r_{i+1}, ..., r_t)| = \tag{10.3.10}$$

$$= \hat{\rho}_s|\hat{R}(r_1, ..., r_{i-1}, r, r_{i+1}, ..., r_t) - R_s| - \hat{\rho}_s|\hat{R}(r_1, ..., r_{i-1}, r', r_{i+1}, ..., r_t) - R_s| \tag{10.3.11}$$

$$\leqslant \hat{\rho}_s\frac{2R_{\max}}{n_s} = \frac{2R_{\max}}{t} \tag{10.3.12}$$

Hence, using MacDiarmid's inequality, for any $\varepsilon > 0$:

$$\mathbb{P}\left(\|\hat{R} - R\|_{L_1(\hat{\rho})} - \mathbb{E}_{r_1,...,r_t}\left[\|\hat{R} - R\|_{L_1(\hat{\rho})}|\bar{s}\right] > \varepsilon|\bar{s}\right) \leqslant \exp\left(-\frac{2\varepsilon^2}{\sum_{1\leqslant i\leqslant t}\left(\frac{2R_{\max}}{t}\right)^2}\right) \tag{10.3.13}$$

$$= \exp\left(-\frac{t\varepsilon^2}{2R_{\max}^2}\right) \tag{10.3.14}$$

Hence, with fixed $\bar{s}$, we have with probability at least $1-\delta$ that:

$$\|\hat{R} - R\|_{L_1(\hat{\rho})} \leqslant \mathbb{E}_{r_1,...,r_t}\left[\|\hat{R} - R\|_{L_1(\hat{\rho})}|\bar{s}\right] + R_{\max}\sqrt{\frac{2\log(1/\delta)}{t}} \tag{10.3.15}$$

$$= R_{\max}\Phi_t(\hat{\rho}_t) + R_{\max}\sqrt{\frac{2\log(1/\delta)}{t}} \tag{10.3.16}$$

Because, this bound is independant of $\bar{s}$, it is true in expectation with respect to $\bar{s}$. Moreover, we have $\Phi_t(\hat{\rho}) \leqslant \Phi_t(\widehat{\rho P})$, indeed:

$$\Phi_t(\hat{\rho}) = \frac{1}{\sqrt{t}}\sum_s\sqrt{\hat{\rho}_s} = \frac{1}{\sqrt{t}}\sum_s\sqrt{\sum_{s'}\widehat{\rho P}_{ss'}} \leqslant \frac{1}{\sqrt{t}}\sum_{ss'}\sqrt{\widehat{\rho P}_{ss'}} \tag{10.3.17}$$

Hence, we have:

$$\|\hat{R} - R\|_{L_1(\hat{\rho})} = R_{\max}\Phi_t(\widehat{\rho P}_t) + R_{\max}\sqrt{\frac{2\log 1/\delta}{t}} \tag{10.3.18}$$

From (10.3.3) we have:

$$\|\hat{V}_t - V\|_{L_1(\rho)} \leqslant \frac{4R_{\max}}{(1-\gamma)^2}\|\widehat{\rho P} - \rho P\|_{\mathrm{TV}} + \frac{1}{1-\gamma}\|\widehat{R} - R\|_{\hat{\rho}} \tag{10.3.19}$$

Hence, using (10.3.18) with $\delta/3$ and (10.3.7) with $2\delta/3$, we have with probability $1-\delta$:

$$\|\hat{V}_t - V\|_{L_1(\rho)} \leqslant \frac{4R_{\max}}{(1-\gamma)^2}\left(\Phi_t(\widehat{\rho P}_t) + 3\sqrt{\frac{\log\frac{3}{\delta}}{2t}}\right) + \frac{R_{\max}}{1-\gamma}\left(\Phi_t(\widehat{\rho P}_t) + \sqrt{\frac{2\log 3/\delta}{t}}\right) \tag{10.3.20}$$

$$= \frac{5R_{\max}}{(1-\gamma)^2}\left(\Phi_t(\widehat{\rho P}_t) + \sqrt{\frac{\log\frac{3}{\delta}}{2t}}\right) \tag{10.3.21}$$

Moreover, we know that with probability $1-\delta$ (already used in (10.3.8))

$$\Phi_t(\widehat{\rho P}_t) \leqslant 2\Lambda_t(\rho P) + \sqrt{\frac{\log(2/\delta)}{t}} \tag{10.3.22}$$

Hence, using this bound with $\delta/4$ and (10.3.20) with $3\delta/4$, we have with probability at least $1-\delta$:

$$\|\hat{V}_t - V\|_{L_1(\rho)} \leqslant \frac{R_{\max}}{(1-\gamma)^2}\left(10\Lambda_t(\rho P) + 9\sqrt{\frac{\log\frac{4}{\delta}}{t}}\right) \tag{10.3.23}$$

Chapter 11

Matrix Factorization and the Forward-Backward (FB) Representation

In this chapter, we introduce bilinear models for the successor states operator, which learn a low-rank approximation. In Section 11.1 we introduce these parametrizations, and discuss the low rank hypothesis for the successor states operator in n Section 11.2. In Section 11.3 we study temporal difference algorithms for the low rank models, and show how to derive lower variance methods. Then, in Section 11.4 we study the fixed points of these methods and show a relation between the fixed points of this method and the SVD. This statement is necessary but not sufficient to show that the algorithm will converge to the optimal low-rank representation. In practice, we observe that this algorithm converges to the optimal low-rank representation of the successor state operator in simple environments.

The learned representations might be used in other methods. Indeed, representation learning has been an active field in RL. States (or state-actions) representations can be used directly as input of a more simple policy (Ha and Schmidhuber, 2018; Stooke et al., 2021), used to derive a bonus for exploration (Machado et al., 2019), or for reward shaping (Wu et al., 2019). These FB representations only depend on the dynamics and not on other signals (such as pixels), which can be irrelevant for the task and biased representation learning toward ignoring the most important information.

11.1 The Forward-Backward representation

In this section we study a specific parametric model for the successor state operator, which has many advantages: a low-rank representation. We consider the model $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$, with the particular choice

$$m_\theta(s_1, s_2) = \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2)\rangle = \sum_{i=1}^{r} \left(F_{\theta_F}(s_1)\right)_i \left(B_{\theta_B}(s_2)\right)_i \tag{11.1.1}$$

where $F\colon S \to \mathbb{R}^r$ and $B\colon S \to \mathbb{R}^r$ are two learnable functions from the state space to some representation space $\mathbb{R}^r$, parameterized by $\theta = (\theta_F, \theta_B)$. This provides an approximation of M by a rank-r operator. Such a factorization is used for instance in (Schaul et al., 2015) for the goal-dependent Q-function (up to the factor ρ). We will also use the notation $F_{\theta_F}(s_1)^\top B_{\theta_B}(s_2) = \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2)\rangle$.

Intuitively, F is a "*forward*" representation of states and B a "*backward*" representation: if the future of s_1 matches the past of s_2, then $M(s_1, \mathrm{d}s_2)$ is large. The Forward and Backward TD algorithms introduced in Chapters 7 and 8 can be applied directly to the FB parametrization, by

considering m_θ as a standard parametric model, and without leveraging its bi-linear structure. In Section 11.3, we study more specifically the Forward and Backward TD algorithms for the FB parametrization and derive explicit updates, with lower variance.

11.2 A discussion of low rank approximations for the successor states operators

In this section, before presenting algorithms to learn the FB parametrization in Section 11.3, we discuss the advantages and limitations of using a low-rank representation for the successor states operator.

Advantages of the low-rank parametrization Here are some of the main advantages of using the FB parametrization for the successor states.

First, even in the tabular case, when the state space is discrete and unstructured, this provides a form of prior or generalization between states (based on a low-rank prior for the successor state operator). States that are linked by the MDP dynamics get representations F and B that are close. This prior might lead to generalization both in the tabular and continuous case, if the low-rank hypothesis is reasonable. This is discussed in more details below.

Then, it provides a direct estimation of the value function at every state, without learning an additional model of V. Namely,

$$V(s) \approx F(s)^\top B(R), \qquad B(R) := \mathbb{E}_{s\sim\rho}[r_s\, B(s)] \tag{11.2.1}$$

where the "reward representation" $B(R)$ can be directly estimated by an online average of $B(s)$ weighted by the reward r_s at s. This is discussed in Section 12.1.2.

Furthermore, we will see in Section 11.3.2 that this parametrization simplifies the sampling of a pair of states (s, s_2) needed for forward (or backward) TD, and allow variance reduction, and even allows for purely "trajectory-wise" online estimates using only the current transition $s \to s'$, without sampling of another independent state s_2. Moreover, we show in Section 11.5 that in expectation and in some settings, the FB parametrization is equivalent to second-order methods of Chapter 9. Hence it could benefit of the efficiency obtained via the second-order method without the huge variance.

Finally, it produces two (policy-dependent) representations of states, a forward and a backward one, in a natural way from the dynamics of the MDP and the current policy. In Section 11.4, we relate the fixed points of our algorithms for the low rank parametrization to the singular value decomposition of the successor states operator, whose output are the *optimal low rank representation* for the norm $\|.\|_\rho$, and observe in practice in tabular environments that the algorithm converges to the optimal low-rank representation. Hence, this algorithm can be a way to learn the *best* representations containing information on the *future* of the trajectories, in order to be used for other purposes such as exploration.

Limitation of the low-rank approximation The FB parametrization approximates the successor state operator by an operator of rank at most r. This is never an exact representation unless the representation dimension r is at least the number of distinct states. The *best* rank-r approximation of $(\mathrm{Id} - \gamma P)^{-1}$ erases the small singular values of P: thus this representation will tend to erase "high frequencies" in the reward and value function, and provide a spatially smoother approximation focusing on long-range behavior. This is fine as long as the reward is not a "fast-changing" function made up of high frequencies (such as a "checkerboard" reward). This can be expected: learning a reward-agnostic object such as M cannot work equally well for all rewards.

Is the low-rank hypothesis relevant for M? Small-rank approximations of a matrix are relevant when the matrix has a few large eigenvalues and many small eigenvalues (or singular

values, depending on the precise criterion). Since the successor state operator is the inverse of $\mathrm{Id} - \gamma P$, this means the approximation is reasonable if $\mathrm{Id} - \gamma P$ has few small eigenvalues and many large eigenvalues.

The spectrum of Markov operators is a well-studied topic. For continuous-time operators associated with random diffusions, possibly with added drift, the spectrum generally follows *Weyl's law* (Wikipedia, 2021): in dimension d, the continuous-time analogue of $\mathrm{Id} - P$ has roughly $k^{d/2}$ eigenvalues of size $\leqslant k$, thus, few small and many large eigenvalues.

The simplest example is a random walk on a discrete torus $[1; n]$. The operator P is diagonal in Fourier representation, with eigenvectors $e^{2i\pi kx/n}$ with k an integer. The corresponding eigenvalue of P is $\cos(2\pi k/n)$, yielding an eigenvalue $(1-\gamma) + 2\gamma \sin^2(\pi k/n)$ for $\mathrm{Id} - \gamma P$. The largest eigenvalue of P is 1 (for $k = 0$) corresponding to the smallest eigenvalue $1 - \gamma$ for $\mathrm{Id} - \gamma P$. For γ close to 1, $(\mathrm{Id} - \gamma P)^{-1}$ has a very large eigenvalue $1/(1-\gamma)$, then an eigenvalue of order $n^2/2\pi^2$, and the next eigenvalues behave like $n^2/2k^2\pi^2$, thus decreasing like $1/k^2$. In this case, a small-rank approximation is reasonable. A similar computation holds for periodic grids $[1; n]^d$ in higher dimension.

How general is this example? The best studied case is for continuous-time diffusions in continuous spaces such as a subset in $\mathbb{R}^d$. In continuous time, the analogue of the operator $\mathrm{Id} - \gamma P$ is the *infinitesimal generator* operator of the continuous-time Markov process. For the standard Brownian motion, this operator is the Laplacian $\Delta = \sum_{i=1}^d \partial^2/\partial x_i^2$. Its inverse Δ^{-1} plays the role of the successor state operator and provides the value function in continuous time. The spectrum of the Laplacian is well-known and follows Weyl's law: there are about $k^{d/2}$ eigenvalues of size $\leqslant k$ (Wikipedia, 2021). In particular, Δ has few small eigenvalues and many large eigenvalues, so that the successor state operator (given by Δ^{-1}, which provides the value function in continuous time) has few large eigenvalues and many small eigenvalues as needed.

This applies not only to Brownian motion, but to basically any diffusion with drift and variable coefficients on a subset of $\mathbb{R}^d$: indeed, in this case the infinitesimal generator is an *elliptic operator* and also follows Weyl's law (Gårding, 1953). The same law also holds for diffusions on Riemannian manifolds, as the Riemannian Laplace operator also follows Weyl's law (Berger, 2003, Chapter 9.7.2). These continuous estimates are still valid when discretizing the state space (Xu et al., 2017a). So this situation is quite general.

11.3 The Forward-Backward algorithm

In this section, we study more specifically the temporal difference updates with the FB parametrization. First, in Section 11.3.1, we show how to*mix* the Forward and Backward TD updates, by using Forward or Backward TD on F, and independently Forward or Backward TD (but not necessarily the same) on B. This lead to four potential algorithms. We then study the Forward-Backward algorithm, using Forward TD on F and Backward TD on B. In Section 11.3.2 we show that there is a variance-reduced form for this method.

In the following sections we will study two theoretical properties of this approach. In Section 11.4, we study the fixed point representations obtained with the Forward Backward algorithm in the tabular case, and show it corresponds to *truncated SVD* for the $\|.\|_\rho$ norm. Finally in Section 11.5 we show that there is a relation between the FB updates and the second order Bellman–Newton method, in a very simple case.

11.3.1 Mixing Forward-TD and Backward-TD updates with the FB representation

The Forward-Backward representation introduced in Chapters 7 and 8 can be used directly for Forward-TD and Backward-TD as discussed in Section 11.1, without changing anything for this specific parametrization. It is also possible to mix the forward and backward updates. Let

$$M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2) = \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2)\rangle \rho(\mathrm{d}s_2) \tag{11.3.1}$$

be our current model. The Forward-TD update of θ is $\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2)$ defined in Theorem 7.5. As, $\theta = (\theta_F, \theta_B)$, we can split

$$\widehat{\delta\theta}^{\text{F-TD}}(s, s', s_2) = (\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2), \widehat{\delta\theta_B}^{\text{F-TD}}(s, s', s_2)) \tag{11.3.2}$$

Similarly, we can define the backward updates $\widehat{\delta\theta_F}^{\text{B-TD}}(s, s', s_2)$ and $\widehat{\delta\theta_B}^{\text{B-TD}}(s, s', s_2)$ via the Backward-TD update on M_θ defined in Theorem 8.3.

An equivalent way of defining these updates is as follows: The forward update on F $\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)$ can be seen as the forward TD update $\widehat{\delta\theta}^{\text{F-TD}}(s, s', s_2)$ for the model $m_{\theta_F}(s_1, s_2) = \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2)\rangle$ in which θ_B is considered as a constant and we only learn θ_F.

Now, we mix the forward and backward updates together, and define the *forward-forward* (ff), *forward-backward* (fb), *backward-forward* (bf) and *backward-backward* (bb) algorithms:

- Forward-Forward update (ff): Update $\theta = (\theta_F, \theta_B)$ with Forward TD both for F and B: $(\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2), \widehat{\delta\theta_B}^{\text{F-TD}}(s, s', s_2))$. This update is exactly equivalent to forward TD on M_θ with the FB parametrization.
- Backward-Backward update (bb): Update $\theta = (\theta_F, \theta_B)$ with Backward TD both for F and B: $(\widehat{\delta\theta_F}^{\text{B-TD}}(s, s', s_2), \widehat{\delta\theta_B}^{\text{B-TD}}(s, s', s_2))$. This update is exactly equivalent to backward TD on M_θ.
- Forward-Backward update (fb): Update θ with Forward TD for F and Backward TD for B: $(\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2), \widehat{\delta\theta_B}^{\text{B-TD}}(s, s', s_2))$.
- Backward-Forward update (bf): Update θ with Backward TD for F and Forward TD: $(\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2), \widehat{\delta\theta_B}^{\text{F-TD}}(s, s', s_2))$.

We first show that mixing the forward and backward updates doesn't change the fixed point property of the algorithms:

Theorem 11.1. *M is a fixed point of the four algorithms (ff, fb, bf, bb)*

Proof. We assume M that there is θ_F and θ_B such that $M(s_1, \mathrm{d}s_2) = F_{\theta_F}(s_1)^\top B_{\theta_B}(s_2)\rho(\mathrm{d}s_2)$. By definition, we have to show that:

$$\mathbb{E}\left[\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)\right] = \mathbb{E}\left[\widehat{\delta\theta_B}^{\text{F-TD}}(s, s', s_2)\right] \quad = \mathbb{E}\left[\widehat{\delta\theta_B}^{\text{F-TD}}(s, s', s_2)\right] = \mathbb{E}\left[\widehat{\delta\theta_B}^{\text{B-TD}}(s, s', s_2)\right] = 0$$

We know that $\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)$ is the forward TD update $\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)$ for the model $m_{\theta_F}(s_1, s_2) = \langle F_{\theta_F}(s_1), B_{\theta_B}(s_2)\rangle$ in which θ_B is considered as a constant and we only learn θ_F. If $M(s_1, \mathrm{d}s_2) = F_{\theta_F}(s_1)^\top B_{\theta_B}(s_2)\rho(\mathrm{d}s_2)$, then by Theorem 7.5, we have $\mathbb{E}_{s\sim\rho, s'\sim P(\mathrm{d}s'|s), s_2\sim\rho}\left[\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)\right] = 0$. The same argument can be applied to the three other terms (using equivalently Theorem 8.3 for the Backward updates).

In the following, we will especially study the Forward-Backward (fb) algorithm. First, in Section 11.3.2, we show that we can reduce the variance of the algorithm. Then, in Section 11.4, we study the fixed point of the fb algorithm, and show in the tabular setting that these coincides with the truncated SVD of the successor state operator. This is a strong argument for this method, as the SVD is the optimal low-rank representation of a matrix. Finally in Section 11.5, we show that in a simple setting the fb algorithm is equivalent in expectation to the Bellman–Newton algorithm.

The other methods (bf, ff, bb) are more discussed in our preprint (Blier et al., 2021), but this discussion was not included in this thesis for simplicity.

11.3.2 The variance reduced Forward-Backward algorithm

The following Theorem defines fb updates, equal in expectation to the updates defined in the previous section, with a different structure. It uses the covariance matrices of the Forward and

Algorithm 10 The Forward-Backward algorithm with function approximation, in a purely online setting.

Input: Policy $\pi(a|s)$, randomly initialized model $F_{\theta_F}(s)$, $B_{\theta_B}(s)$; `TransitionMemory`, covariance momentum α
Get an initial state s_0 from the environment.
Define $\widehat{\Sigma_F} \leftarrow 0_{r\times r}$, $\widehat{\Sigma_B} \leftarrow 0_{r\times r}$
repeat
  Sample $a_t \sim \pi(.|s_t)$, execute a_t and observe s_{t+1}
  Compute $f_{\theta_F} \leftarrow F_{\theta_F}(s_t)$, $f'_{\theta_F} \leftarrow F_{\theta_F}(s_{t+1})$, $b_{\theta_B} \leftarrow B_{\theta_B}(s_t)$, $b'_{\theta_B} \leftarrow B_{\theta_B}(s_{t+1})$
  $\widehat{\delta\theta_F}^{\text{fb-TD}} \leftarrow (\partial_{\theta_F} f_{\theta_F}^\top) b_{\theta_B} + (\partial_{\theta_F} f_{\theta_F}^\top)\widehat{\Sigma_B}(\gamma f'_{\theta_F} - f_{\theta_F})$
  $\widehat{\delta\theta_B}^{\text{fb-TD}} \leftarrow (\partial_{\theta_B} b_{\theta_B}^\top) f_{\theta_F} + \left(\gamma\, \partial_{\theta_B} b'_{\theta_B} - \partial_{\theta_B} b_{\theta_B}\right)^\top \widehat{\Sigma_F} b_{\theta_B}$
  Gradient steps: $\theta_F \leftarrow \theta_F + \eta\widehat{\delta\theta_F}^{\text{fb-TD}}$ and $\theta_B \leftarrow \theta_B + \eta\widehat{\delta\theta_B}^{\text{fb-TD}}$.
  Update covariance estimates: $\widehat{\Sigma_F} \leftarrow (1-\alpha)\widehat{\Sigma_F} + \alpha f f^\top$ and $\widehat{\Sigma_B} \leftarrow (1-\alpha)\widehat{\Sigma_B} + \alpha b b^\top$
  $t \leftarrow t+1$
until end of learning

Backward representation:

$$\Sigma_F := \mathbb{E}_{s_1\sim\rho}\, F(s_1)F(s_1)^\top \tag{11.3.3}$$

$$\Sigma_B := \mathbb{E}_{s_2\sim\rho}\, B(s_2)B(s_2)^\top. \tag{11.3.4}$$

The matrices Σ_F and Σ_B are $r \times r$ matrices. While we don't directly have access to these matrices, we can easily compute estimates $\widehat{\Sigma_F}$ and $\widehat{\Sigma_B}$. Here are two possible ways to estimate such estimates:

- If we are able to sample states $s \sim \rho$, (typically from a known distribution of states or from a state buffer), we can sample a mini-batch $(s_1, ..., s_B)$ of states each time we need an estimate $\widehat{\Sigma_B}$ or $\widehat{\Sigma_F}$, and define:

$$\widehat{\Sigma_F} := \frac{1}{B}\sum_{1\leqslant b\leqslant B} F(s_b)F(s_b)^\top \quad \text{and} \quad \widehat{\Sigma_B} := \frac{1}{B}\sum_{1\leqslant b\leqslant B} B(s_b)B(s_b)^\top \tag{11.3.5}$$

  With this approach, we know that these estimates are unbiased: $\mathbb{E}_{(s_1,\dots,s_B)}\left[\widehat{\Sigma_F}\right] = \Sigma_F$ and $\mathbb{E}_{(s_1,\dots,s_B)}\left[\widehat{\Sigma_B}\right] = \Sigma_B$

- An other approach is to use a moving average with momentum, which is used in Algorithm 10: when observing a state s , we can update our estimates $\widehat{\Sigma_F}$ and $\widehat{\Sigma_B}$ as:

$$\widehat{\Sigma_F} \leftarrow (1-\alpha)\widehat{\Sigma_F} + \alpha F(s)F(s)^\top$$
$$\widehat{\Sigma_B} \leftarrow (1-\alpha)\widehat{\Sigma_B} + \alpha B(s)B(s)^\top$$

  These estimates are not *unbiased* but their variance might be lower. Moreover, it removes the need for additional state sampling as in TD or Backward TD, and allow purely trajectory-wise online estimation.

We now assume we have some estimates $\widehat{\Sigma_F}$ and $\widehat{\Sigma_B}$, regardless of the method used to estimate it. We can now define the corresponding (fb) udpate:

Theorem 11.2. *Consider the parametrization $m_\theta(s_1, s_2) = F_{\theta_F}(s_1)^\top B_{\theta_B}(s_2)$ of the successor state operator M where F and B are two functions from $\mathcal{S}$ to $\mathbb{R}^r$, parameterized by $\theta = (\theta_F, \theta_B)$.*

We assume $\widehat{\Sigma_F}$ and $\widehat{\Sigma_B}$ are our current estimates of Σ_F and Σ_B defined in Equations (11.3.3) *and* (11.3.4)

When observing a transition (s, s') with $s \sim \rho(\mathrm{d}s)$ and $s' \sim P(\mathrm{d}s'|s)$, we define the fb update $\widehat{\delta\theta}^{fb\text{-}TD}(s, s', \widehat{\Sigma_F}, \widehat{\Sigma_B})$ as

$$\widehat{\delta\theta}^{fb\text{-}TD}(s, s', \widehat{\Sigma_F}, \widehat{\Sigma_B}) = \left(\widehat{\delta\theta_F}^{fb\text{-}TD}(s, s', \widehat{\Sigma_B}), \widehat{\delta\theta_B}^{fb\text{-}TD}(s, s', \widehat{\Sigma_F})\right) \tag{11.3.6}$$

with:

$$\widehat{\delta\theta_F}^{fb\text{-}TD}(s, s', \widehat{\Sigma_B}) = (\partial_{\theta_F} F_{\theta_F}(s))^\top B(s) + (\partial_{\theta_F} F_{\theta_F}(s))^\top \widehat{\Sigma_B}(\gamma F(s') - F(s)) \tag{11.3.7}$$

$$\widehat{\delta\theta_B}^{fb\text{-}TD}(s, s', \widehat{\Sigma_F}) = (\partial_{\theta_B} B_{\theta_B}(s))^\top F(s) + (\gamma\, \partial_{\theta_B} B_{\theta_B}(s') - \partial_{\theta_B} B_{\theta_B}(s))^\top \widehat{\Sigma_F} B(s) \tag{11.3.8}$$

If $\widehat{\Sigma_F} = \Sigma_F$ and $\widehat{\Sigma_B} = \Sigma_B$, the updates (11.3.7)-(11.3.8) are equal to the expectation with respect to s_2 of the (fb) update defined in Section 11.3.1: Formally, for every (s, s'), we have

$$\widehat{\delta\theta_F}^{fb\text{-}TD}(s, s', \Sigma_F) = \mathbb{E}_{s_2\sim\rho}\left[\widehat{\delta\theta_F}^{F\text{-}TD}(s, s', s_2)\right] \tag{11.3.9}$$

$$\widehat{\delta\theta_B}^{fb\text{-}TD}(s, s', \Sigma_B) = \mathbb{E}_{s_2\sim\rho}\left[\widehat{\delta\theta_B}^{B\text{-}TD}(s, s', s_2)\right] \tag{11.3.10}$$

Hence, if $\widehat{\Sigma_F} = \Sigma_F$ and $\widehat{\Sigma_B} = \Sigma_B$, $\widehat{\delta\theta}^{fb\text{-}TD}(s, s', \widehat{\Sigma_F}, \widehat{\Sigma_B})$ is a variance-reduced forward-backward TD update.

Proof. We consider the update $\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)$. We have, from the definition of $\widehat{\delta\theta}_{\text{F-TD}}(s, s', s_2)$ in Theorem 7.5 and the definition of $\widehat{\delta\theta_F}^{\text{F-TD}}$:

$$\begin{aligned}\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2) &= \partial_{\theta_F} m_{\theta_F}(s, s) + \partial_{\theta_F} m_{\theta_F}(s, s_2)\left(\gamma m_{\theta_F}(s', s_2) - m_{\theta_F}(s, s_2)\right)\\ &= (\partial_{\theta_F} F_{\theta_F}(s))^\top B_{\theta_B}(s) + (\partial_{\theta_F} F_{\theta_F}(s))^\top B_{\theta_B}(s_2)\left(\gamma F_{\theta_F}(s') - F_{\theta_F}(s)\right)^\top B_{\theta_B}(s_2)\end{aligned}$$

We can rewrite the second term:

$$(\partial_{\theta_F} F_{\theta_F}(s))^\top B_{\theta_B}(s_2)\left(\gamma F_{\theta_F}(s') - F_{\theta_F}(s)\right)^\top B_{\theta_B}(s_2) = (\partial_{\theta_F} F_{\theta_F}(s))^\top \left(B_{\theta_B}(s_2) B_{\theta_B}(s_2)^\top\right)\left(\gamma F_{\theta_F}(s') - F_{\theta_F}(s)\right)$$

Hence:

$$\begin{aligned}\mathbb{E}_{s_2\sim\rho}\left[\widehat{\delta\theta_F}^{\text{F-TD}}(s, s', s_2)\right] &= (\partial_{\theta_F} F_{\theta_F}(s))^\top B_{\theta_B}(s) + (\partial_{\theta_F} F_{\theta_F}(s))^\top \mathbb{E}_{s_2\sim\rho}\left[B_{\theta_B}(s_2) B_{\theta_B}(s_2)^\top\right]\left(\gamma F_{\theta_F}(s') - F_{\theta_F}(s)\right)\\ &= (\partial_{\theta_F} F_{\theta_F}(s))^\top B_{\theta_B}(s) + (\partial_{\theta_F} F_{\theta_F}(s))^\top \Sigma_B\left(\gamma F_{\theta_F}(s') - F_{\theta_F}(s)\right)\\ &= \widehat{\delta\theta_F}^{\text{fb-TD}}(s, s', \Sigma_F)\end{aligned}$$

The proof is similar for $\widehat{\delta\theta_B}^{\text{fb-TD}}(s, s', \Sigma_B)$.

The algorithm corresponding to this update is defined in Algorithm 10. Is Sections 11.4 and 11.5 we will study theoretical properties of the forward-backward algorithm, first on its fixed points and their relation to the SVD, then on its relation with the Bellman–Newton method. In Section 11.4.2, we will show experimentally in tabular environments that the (fb) algorithm converges in practice in practice to the *optimal low rank* approximation.

11.3.3 Relationship with successor representation learning and with linear TD with learned features.

For fixed and orthonormal B, the forward update of F corresponds to standard successor representation learning with state representation (features) B.

To simplify things, in this paragraph we consider the "tabular-FB" setting, in which F and B are parameterized just by listing the value of $F(s)$ and $B(s)$ on every state s, assuming a finite state space. [1] For instance, the forward TD update (11.3.7) for F, with learning rate $\eta > 0$, becomes $F(s) \leftarrow F(s) + \eta\, \delta F(s)$ where

$$\delta F(s) = B(s) + \Sigma_B(\gamma F(s') - F(s)) \tag{11.3.11}$$

[1] This is different from a tabular setting for M, which would parametrize M by listing the values $M(s_1, s_2)$ for every pair of states.

upon sampling a transition $s \to s'$.

If B is a fixed, $L^2(\rho)$-orthonormal collection of feature functions (namely, if $\Sigma_B = \mathrm{Id}$), then this forward TD equation to learn F is identical to standard deep successor representation learning using B as the representation. Indeed, standard deep successor representation learning (Kulkarni et al., 2016) starts with given features $\varphi(s)$ on the state space, and learns the successor features m as the expected discounted future value of φ along a trajectory (s_t): $m(s) = \sum_{t \geqslant 0} \gamma^t \mathbb{E}[\varphi(s_t)|s_0 = s]$. Such an m is the fixed point of the Bellman equation $m = \varphi + \gamma P m$. Via identifying $m = F$ and $\varphi = B$, ordinary TD for this Bellman equation is equivalent to (11.3.11) when $\Sigma_B = \mathrm{Id}$. However, this is not the case if $\Sigma_B \neq \mathrm{Id}$. This is because scalings are different: With the successor state operator, if B is doubled, then F is halved so that $M = F^\top B$ is fixed. With successor representations, if the state representation φ is doubled, then m is doubled as well.

11.4 Fixed Points for the FB Representation of M

We now study the fixed points of the (fb) algorithm, which are the low-rank representations possibly learned at convergence. We want to know if these final representations are well-approximating the true successor states operator M^π. We will consider the finite state space case, in which M^π is a matrix. In that case, we can compare the low-rank representation of M^π learned via (fb) with the *optimal* low-rank representation for the $L_2(\rho)$ norm. This optimal low-rank representation is obtained by computing the SVD of M for the $L_2(\rho)$ norm:: find U, D, V three $S \times S$ matrices with $M^\pi = UDV^\top$, $U^\top \mathrm{Diag}(\rho) U = V^\top \mathrm{Diag}(\rho) V = \mathrm{Id}_{\mathcal{S}}$, and D is a diagonal matrix with entries $d_1 \geqslant ... \geqslant d_S \geqslant 0$. Then, the optimal approximation of M^π of rank r for the $L_2(\rho)$ norm is the matrix $\tilde{M}$ of rank at most r which minimizes the norm $\|\tilde{M} - M^\pi\|_\rho^2$. This optimum is achieved with the truncated SVD: $\tilde{M} = U\tilde{D}V^\top$ with $\tilde{D} = \mathrm{Diag}(d_1, ..., d_r, 0, ..., 0)$. More details on the SVD and how it depends on ρ is given in Section 11.4.3.

In the following, we first prove that the fixed points of the (fb) algorithms are truncated SVDs for the $L_2(\rho)$ norm, which means the fixed points are matrices $U\tilde{D}V^\top$ where $\tilde{D}$ is a diagonal matrix obtained by keeping only r values of D and setting other values to 0: $\tilde{D} = \mathrm{Diag}(\varepsilon_1 d_1, ..., \varepsilon_S d_S)$, with $\varepsilon_i \in \{0, 1\}$ and $\sum_i \varepsilon_i = r$. This is not sufficient condition for convergence toward the global minima of $\|\tilde{M} - M^\pi\|_\rho^2$ for $\tilde{M}$ of rank at most r, but only for convergence to local minima. Then, we show in some experiments that in practice, the (fb) algorithm does converge to the optimal low-rank approximation.

11.4.1 Describing the fixed point of the (fb) algorithm

Here we state precisely, and prove, the fixed points properties for the fb-FB algorithm in the tabular case. The "tabular" case for F and B means that the state space is finite and the values of $F(s)$ and $B(s)$ are stored explicitly for every state s.

In this section, we abuse notation by considering F and B both as functions from the state space $\mathcal{S}$ to $\mathbb{R}^r$ and as $r \times S$-matrices. The model $M(s_1, \mathrm{d}s_2) = F(s_1)^\top B(s_2)\rho(\mathrm{d}s_2)$ rewrites as $M = F^\top B \,\mathrm{diag}(\rho)$ or $m = F^\top B$, viewing everything as matrices with entries indexed by the state space. We also assume that $\rho_s > 0$ for every state s: every state is sampled with nonzero probability.

Theorem 11.3. *We assume the state space $\mathcal{S}$ is finite, and F, B are $r \times S$ matrices. Let (F, B) be a fixed point of the (fb) algorithm, F and B are local extrema of the error*

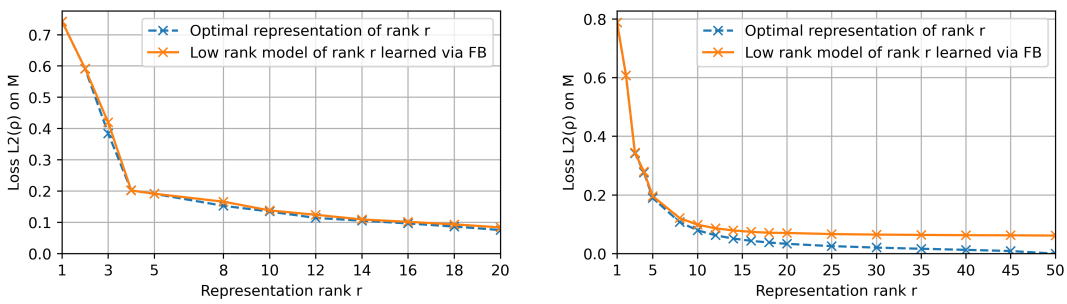
$$\ell(F, B) := \|F^\top(s_1)B(s_2)\rho(\mathrm{d}s_2) - M^\pi(s_1, \mathrm{d}s_2)\|_\rho^2 \tag{11.4.1}$$

In that case, $F^\top B \,\mathrm{diag}(\rho)$ is a truncated singular value decomposition of the operator M^π acting on the space of functions over $\mathcal{S}$ equipped with the $L^2(\rho)$ norm.

This theorem shows that the (fb) algorithm satisfies a necessary but not sufficient condition in order to guarantee convergence to the optimal low rank representation for the $L_2(\rho)$ norm. The principle of looking for the optimal low-rank approximation via he SVD is similar to the approach of Wu et al. (2019), which directly tries to approximate eigenvectors of the Laplacian. In their approach, the main difficulty is to ensure the orthonormality of the learned vectors, and they add a second objective, corresponding to a soft version of this constraint. On the contrary, with the methods described above, no additional objective is needed.

We now provide experiments in tabular environments, showing that in practice, the (fb) algorithm converges to the optimal low-rank approximation.

11.4.2 Experiments



(a) Random walk in a maze with 104 states

(b) Random walk on a circle with 50 states.

Figure 11.1: Error of the low-rank approximations learned via the (fb) algorithm after convergence (10^8 steps), as a function of the rank r, compared to the optimal low-rank approximation of rank r for the norm $L_2(\rho)$, obtained with the SVD.

In this section, we study the low-rank approximation learned via the (fb) algorithm after convergence, as a function of the rank r. We consider two environments: First, a small non-reversible maze, already introduced in Chapter 9, in Figure 11.1a, with 104 states. The policy is a uniform random walk.

Then, we consider a random walk on a circle, with 50 states (already introduced in Chapter 9). At step t, the agent is in $s_t \in \{0, ..., 49\}$, and $s_{t+1} = s_t + \varepsilon_t \mod 50$, where ε_t is sampled uniformly in $\{-1, 0, 1\}$.

We use Algorithm 10, in a purely online setting, with a single trajectory. We use a learning rate schedule $\eta_t = \frac{\beta}{t+t_0}$. For every environment, we selected β, t_0 and α, from a grid (respectively $\{10^k, \quad k \in [-4, 4]\}$, $\{10^k, \quad k \in [0, 9]\}$, and $\{10^k, \quad k \in [3, 5]\}$), for a single rank $r = 20$ and kept the same parameters for every other ranks. In practice, for both environment, we used $\beta = 100$ and $t_0 = 10^5$, and $\alpha = 10^{-4}$. Every parameter of F and B are initialized independantly as $\mathcal{N}(0, \frac{S}{r})$.

We measure for both environments the errors for the $L_2(\rho)$ on the successor states estimate, at the end of training (10^8 steps). We compare this value with the optimal representation for the $L_2(\rho)$ norm, obtained by computing the SVD of the operator, and truncate all singular values except the r largests.

In practice, we observe that when $r \ll S$, the error of the representation learned via the (fb) algorithm corresponds to the optimal errorfor a representation of rank r. When r is closer to S, the difference between the error of the (fb) algorithm and the optimal error gets larger. This can probably be explained by the expected asymptotic error of order $O(1/\sqrt{t})$: even after 10^8 steps the method has not completely converged..

Theorem 11.3 proves that the fixed points of the method are truncated SVD for the $L_2(\rho)$ norm. The experiments, in these toy tabular environments, suggests that among all truncated SVDs, the method converge to the one with the largest singular values.

11.4.3 Proof of Theorem 11.3

Background on Singular Value Decompositions In the text, we often work with the space of functions over $\mathcal{S}$ equipped with the $L^2(\rho)$ norm. Since $\rho \neq \mathrm{Id}$, we include here a reminder on how the usual notions of Euclidean vector spaces play out in non-orthonormal bases. We also include details on what constitutes a "truncated singular value decomposition".

A Euclidean vector space E is a finite-dimensional vector space equipped with a dot product; the dot product is given by some symmetric, positive definite matrix q in some basis, namely, $\langle x, y\rangle_E = x^\top q y$ for any vectors x and y.

If $A\colon E_1 \to E_2$ is a linear map between two Euclidean spaces, its adjoint A^* is the map from E_2 to E_1 such that $\langle y, Ax\rangle_{E_2} = \langle A^* y, x\rangle_{E_1}$ for any vectors $x \in E_1$ and $y \in E_2$. Expressed in bases of E_1 and E_2, its matrix is $A^* = q_1^{-1} A^\top q_2$, or just $A^\top$ if the bases are orthonormal.

Such a map A is orthogonal if $AA^* = \mathrm{Id}_{E_2}$ and $A^*A = \mathrm{Id}_{E_1}$.

The Hilbert–Schmidt norm for an operator M on a Euclidean vector space is $\mathrm{Tr}(M^*M)$ where M^* is the adjoint of M. In an orthonormal basis this is $\mathrm{Tr}(M^\top M)$ viewing M as a matrix, but in a non-orthonormal basis this is $\mathrm{Tr}(q^{-1}M^\top q M)$ where q is the matrix defining the norm in the basis.

A *singular value decomposition* of such a map A is a triplet of linear maps $U\colon \mathbb{R}^{\dim(E_2)} \to E_2$, $D\colon \mathbb{R}^{\dim(E_1)} \to \mathbb{R}^{\dim(E_2)}$ and $V\colon \mathbb{R}^{\dim(E_1)} \to E_1$ such that $A = UDV^*$, U and V are orthogonal, and D is rectangular diagonal. Equivalently, a singular value decomposition can be written as $Ax = \sum_i u_i d_i \langle v_i, x\rangle_{E_1}$ where each $d_i > 0$, the u_i's make an orthonormal family in E_2, and the v_i's make an orthonormal family in E_1 (or equivalently, an orthonormal family of linear forms on E_1 by identifying v_i with the map $x \mapsto \langle v_i, x\rangle_{E_1}$).

Definition 11.4 (Truncated SVD)**.** A linear map B is a *truncated singular value decomposition* of a linear map $A\colon E_1 \to E_2$ if there is a singular value decomposition $A = UDV^*$ of A and a rectangular diagonal matrix D' such that D' is obtained from D by replacing some elements with 0, and $B = UD'V^*$.

Lemma 11.5. *A linear map $B\colon E_1 \to E_2$ is a truncated singular value decomposition of $A\colon E_1 \to E_2$ if and only if A and B are equal on $(\mathrm{Ker}\, B)^\perp$ and the image of $\mathrm{Ker}\, B$ by A is orthogonal to the image of B.*

Proof. ($\Leftarrow$) Define $E_1' = \mathrm{Ker}\, B$ and $E_1'' = (\mathrm{Ker}\, B)^\perp$ so that $E_1 = E_1' \oplus E_1''$. Let A' and A'' be the restrictions of A to E_1' and E_1'' respectively, so that $A = A' + A''$. Define B' and B'' likewise.

Since E_1' is $\mathrm{Ker}\, B$, we have $B' = 0$ so $B = B''$.

By assumption, A and B are equal on E_1''. Therefore, $A'' = B''$, so $B = A''$.

By assumption, the image of E_1' by A is orthogonal to the image of B. The former is $\mathrm{Im}\, A'$ while the latter is $\mathrm{Im}\, A''$. Therefore, $\mathrm{Im}\, A' \perp \mathrm{Im}\, A''$.

Consider singular value decompositions of A' and A'' as $A' = \sum_i u_i' d_i' v_i'$ and $A'' = \sum_j u_j'' d_j'' v_j''$, where the d_i' are positive real numbers, the u_i' are an orthonormal basis of $\mathrm{Im}\, A'$, the v_i' are an orthonormal set of linear forms on E_1', and likewise for A''. (Any zero singular values have been dropped in this decomposition.)

Since $\mathrm{Im}\, A' \perp \mathrm{Im}\, A''$, the u_i''s are orthogonal to the u_j'''s. Likewise, since the decomposition $E_1 = E_1' \oplus E_1''$ is orthogonal, the v_i''s are orthogonal to the v_j'''s as linear forms on E_1.

It follows that $\sum_i u_i' d_i' v_i' + \sum_j u_j'' d_j'' v_j''$ is a singular value decomposition of A (with the zero singular values omitted). Since $B = A''$, $\sum_j u_j'' d_j'' v_j''$ is a singular value decomposition of B, so that B is a truncated SVD of A.

($\Rightarrow$) Let $A = UDV^*$ and $B = UD'V^*$ as in Definition 11.4. Up to swapping rows and columns, we can assume that the nonzero entries of D and D' are located in the first rows. Let k be the number of nonzero entries in D'. Then $\mathrm{Ker}\, D'$ is spanned by the last $\dim(E_1) - k$ basis vectors in $\mathbb{R}^{\dim(E_1)}$, and $(\mathrm{Ker}\, D')^\perp$ is spanned by the first k basis vectors. Thus, by construction, D and D' coincide on $(\mathrm{Ker}\, D')^\perp$. Moreover, $\mathrm{Im}\, D'$ is spanned by the first k basis vectors, and $D(\mathrm{Ker}\, D')$ is spanned by the last $\dim(E_1) - k$ basis vectors, so $\mathrm{Im}\, D'$ and $D(\mathrm{Ker}\, D')$ are orthogonal.

Since $A = UDV^*$ and $B = UD'V^*$, and since U is invertible, A and B are equal on $(\mathrm{Ker}\, B)^\perp$ if and only if DV^* and $D'V^*$ are equal on $(\mathrm{Ker}\, B)^\perp$. Since V^* is invertible, this happens if and only if D and D' are equal on $V^*((\mathrm{Ker}\, B)^\perp)$. Since V^* is orthogonal, the latter is $(V^*(\mathrm{Ker}\, B))^\perp$.

Since U and V are orthogonal, hence invertible, one has $\mathrm{Ker}\, B = \mathrm{Ker}(UD'V^*) = \mathrm{Ker}(D'V^*) = V(\mathrm{Ker}\, D')$. Hence $V^*(\mathrm{Ker}\, B) = \mathrm{Ker}\, D'$. Thus, we need D and D' to be orthogonal on $\mathrm{Ker}\, D'$, which we have established above.

Next, let us prove that $A(\mathrm{Ker}\, B) \perp \mathrm{Im}\, B$, namely, that $UDV^*(\mathrm{Ker}\, B) \perp \mathrm{Im}(UD'V^*)$. Since U is orthogonal, this is equivalent to $DV^*(\mathrm{Ker}\, B) \perp \mathrm{Im}(D'V^*)$. We have seen that $V^*(\mathrm{Ker}\, B) = \mathrm{Ker}\, D'$; moreover $\mathrm{Im}(D'V^*) \subset$

$\mathrm{Im}(D')$, so it is enough to prove that $D(\mathrm{Ker}\, D') \perp \mathrm{Im}\, D'$, which we have established above. This proves the first part of the equivalence.

Proof of Theorem 11.3

Proof. First, we derive the tabular updates of F and B for fb-FB. By direct identification in Proposition 11.2, in the tabular case we find the following expressions for the updates of F and B. Abbreviate $\mathrm{diag}(\rho)$ for the diagonal matrix with entries ρ_s for each state s, then the updates $\delta\theta_F$ and $\delta\theta_B$ of Proposition 11.2 for the fb-FB algorithm are equal to

$$\delta F = B\,\mathrm{diag}(\rho) - \Sigma_B F \Delta^\top \mathrm{diag}(\rho) \tag{11.4.2}$$

$$\delta B = F\,\mathrm{diag}(\rho) - \Sigma_F B\,\mathrm{diag}(\rho)\Delta \tag{11.4.3}$$

for backward TD on F and B respectively. Here Δ is the matrix $\mathrm{Id} - \gamma P$, $\Sigma_B = B\,\mathrm{diag}(\rho)B^\top$, and $\Sigma_F = F\,\mathrm{diag}(\rho)F^\top$.

Viewing $\tilde{m}$, F and B as matrices, the loss is

$$\ell(F,B) = \sum_{ij} \rho(i)\rho(j) \left(\sum_k F_{ki} B_{kj} - \tilde{m}_{ij} \right)^2 \tag{11.4.4}$$

so that

$$\frac{\partial \ell(F,B)}{\partial F_{ki}} = 2 \sum_j \rho(i)\rho(j) B_{kj} \left(\sum_{k'} F_{k'i} B_{k'j} - \tilde{m}_{ij} \right) \tag{11.4.5}$$

which is the ki entry of the matrix $2B\,\mathrm{diag}(\rho)(B^\top F - \tilde{m}^\top)\,\mathrm{diag}(\rho)$.

Now, F is a local extremum of this loss if and only if this derivative is 0 for every ki, namely, if and only if the matrix $B\,\mathrm{diag}(\rho)(B^\top F - \tilde{m}^\top)\,\mathrm{diag}(\rho)$ is 0. Now, by definition of $\tilde{m}$ we have $M = \tilde{m}\,\mathrm{diag}(\rho)$, namely, $\tilde{m} = \Delta^{-1}\,\mathrm{diag}(\rho)^{-1}$. So $B\,\mathrm{diag}(\rho)(B^\top F - \tilde{m}^\top)\,\mathrm{diag}(\rho) = 0$ is equivalent to $B\,\mathrm{diag}(\rho)B^\top F\,\mathrm{diag}(\rho) - B(\Delta^{-1})^\top\,\mathrm{diag}(\rho) = 0$. Since $\mathrm{diag}(\rho)$ and Δ are invertible, by multiplying by $\mathrm{diag}(\rho)^{-1}\Delta^\top\,\mathrm{diag}(\rho)$ on the right, this is equivalent to $B\,\mathrm{diag}(\rho)B^\top F\Delta^\top\,\mathrm{diag}(\rho) - B\,\mathrm{diag}(\rho) = 0$. This is equivalent to $\delta F = 0$ in (11.4.2), namely, to F being a fixed point of forward TD.

A similar computation with B proves that $\partial\ell(F,B)/\partial B = 0$ if and only if $\delta B = 0$ in (11.4.3), namely, if and only if B is a fixed point of *backward* TD. Therefore, F and B are a local optimum of ℓ if and only if they are a fixed point of the fb-FB algorithm.

Let us turn to the statement about singular value decompositions. Generally speaking, we know that the matrices of a given rank which are local extrema of the matrix norm of the difference with $\tilde{m}$ are truncated singular value decompositions of $\tilde{m}$; however, here these matrices are parameterized as $F^\top B$, and a priori this parametrization might change the local extrema, so we give a full proof.

By Lemma 11.5, the matrix $F^\top B\,\mathrm{diag}(\rho)$ is a truncated SVD of M if and only if $F^\top B\,\mathrm{diag}(\rho)$ and M coincide on $(\mathrm{Ker}\, F^\top B\,\mathrm{diag}(\rho))^\perp$ and $M(\mathrm{Ker}\, F^\top B\,\mathrm{diag}(\rho)) \perp \mathrm{Im}\, F^\top B\,\mathrm{diag}(\rho)$. Here all orthogonality relations are defined with respect to the $L^2(\rho)$ inner product, namely, $\langle x, y\rangle = x^\top\,\mathrm{diag}(\rho)y$.

If F is a fixed point of (11.4.2), then $0 = B\,\mathrm{diag}(\rho) - \Sigma_B F\Delta^\top\,\mathrm{diag}(\rho)$. Since $\mathrm{diag}(\rho)$ is invertible and since $\Sigma_B = B\,\mathrm{diag}(\rho)B^\top$, this rewrites as $B(\mathrm{Id} - \mathrm{diag}(\rho)B^\top F\Delta^\top) = 0$. Taking transposes, this is $(\mathrm{Id} - \Delta F^\top B\,\mathrm{diag}(\rho))B^\top = 0$. By definition, M is the inverse of Δ; multiplying by M, we find $(M - F^\top B\,\mathrm{diag}(\rho))B^\top = 0$. This implies that M and $F^\top B\,\mathrm{diag}(\rho)$ coincide on the image of $B^\top$. A fortiori, they coincide on the image of $B^\top F\,\mathrm{diag}(\rho)$, which is included in the image of $B^\top$.

In general, for an operator A on a Euclidean space, $\mathrm{Im}\, A = (\mathrm{Ker}\, A^*)^\perp$ with A^* the adjoint of A. Here, with the inner product from $L^2(\rho)$, the adjoint of A is $\mathrm{diag}(\rho)^{-1}A^\top\,\mathrm{diag}(\rho)$ (Appendix 11.4.3). So the adjoint of $B^\top F\,\mathrm{diag}(\rho)$ is $F^\top B\,\mathrm{diag}(\rho)$. Therefore, $\mathrm{Im}\, B^\top F\,\mathrm{diag}(\rho)$ is $(\mathrm{Ker}\, F^\top B\,\mathrm{diag}(\rho))^\perp$. So M and $F^\top B\,\mathrm{diag}(\rho)$ coincide on $(\mathrm{Ker}\, F^\top B\,\mathrm{diag}(\rho))^\perp$. This was the first point to be proved.

Next, if B is a fixed point of (11.4.3), then $0 = F\,\mathrm{diag}(\rho) - F\,\mathrm{diag}(\rho)F^\top B\,\mathrm{diag}(\rho)\Delta$. Multiplying on the right by $M = \Delta^{-1}$ this is equivalent to $F\,\mathrm{diag}(\rho)(M - F^\top B\,\mathrm{diag}(\rho)) = 0$. This states that the image of $M - F^\top B\,\mathrm{diag}(\rho)$ is ρ-orthogonal to the image of $F^\top$. So for any x, $(M - F^\top B\,\mathrm{diag}(\rho))x \perp \mathrm{Im}\, F^\top$. Take $x \in \mathrm{Ker}\, F^\top B\,\mathrm{diag}(\rho)$. Then $Mx \perp \mathrm{Im}\, F^\top$. Since $\mathrm{Im}\, F^\top B\,\mathrm{diag}(\rho) \subset \mathrm{Im}\, F^\top$, we have $Mx \perp \mathrm{Im}\, F^\top B\,\mathrm{diag}(\rho)$ as well. Therefore, the image of $\mathrm{Ker}\, F^\top B\,\mathrm{diag}(\rho)$ by M is orthogonal to the image of $F^\top B\,\mathrm{diag}(\rho)$. This was the second point to be proved.

11.5 The FB Representation and Bellman–Newton

In this section, we show that there is a relation between the FB updates and the second order Bellman–Newton method. We prove this relation in a simple case: we assume ρ is the invariant measure of the process, and is the uniform measure. Moreover, we assume P is symmetric: $P_{s_1 s_2} = P(s_2 s_1)$. In that case, we prove that the tabular FB updates (all four versions) coincide with the Bellman–Newton update in the small-learning-rate (continuous-time) limit, on-policy, with suitable initializations.

While this theorem only holds in a specific case, it suggests that FB updates might enjoy Bellman–Newton increased asymptotic convergence, while having a smaller variance.

Theorem 11.6 (The FB update is Bellman–Newton for symmetric P)**.** *Assume that the state space $\mathcal{S}$ is finite with $S := |\mathcal{S}|$, and that the transition matrix P is symmetric. Let ρ be the uniform distribution on $\mathcal{S}$, which is invariant under the Markov process. Let $\operatorname{diag}(\rho) = \frac{1}{S}\operatorname{Id}$ be the diagonal matrix with entries ρ.*

We assume $r = S$ and initialize F_0 and B_0 as $F_0 = B_0 = \operatorname{Id}$. Consider the continuous-time equation

$$\frac{\mathrm{d}F_t}{\mathrm{d}t} = \delta F, \qquad \frac{\mathrm{d}B_t}{\mathrm{d}t} = \delta B \tag{11.5.1}$$

where δF and δB are the tabular FB updates in equations (11.4.2) *and* (11.4.3)*, corresponding to the Tabular case in Proposition 11.2, computed at F_t and B_t. Any of the four variants ff-FB, fb-FB, bf-FB, bb-FB may be used.*

Let $M_t := F_t^\top B_t \operatorname{diag}(\rho)$ be the resulting estimate of the successor state matrix. Then M_t evolves according to the Bellman–Newton update

$$\frac{\mathrm{d}M_t}{\mathrm{d}t} = \eta M_t - \eta M_t(\operatorname{Id} - \gamma P)M_t \tag{11.5.2}$$

with learning rate $\eta = 2/S$.

Proof. We abbreviate F_t' for $\mathrm{d}F_t/\,\mathrm{d}t$ and likewise for all other quantities.

According to equations (11.4.2) and (11.4.3), the forward-TD equations for F and B are

$$F_t' = B_t \operatorname{diag}(\rho) - B_t \operatorname{diag}(\rho) B_t^\top F_t \Delta^\top \operatorname{diag}(\rho), \qquad B_t' = F_t \operatorname{diag}(\rho) - F_t \operatorname{diag}(\rho) \Delta F_t^\top B_t \operatorname{diag}(\rho) \tag{11.5.3}$$

and the backward-TD equations are

$$F_t' = B_t \operatorname{diag}(\rho) - B_t (\operatorname{diag}(\rho)\Delta)^\top B_t^\top F_t \operatorname{diag}(\rho), \qquad B_t' = F_t \operatorname{diag}(\rho) - F_t \operatorname{diag}(\rho) F_t^\top B_t \operatorname{diag}(\rho)\Delta \tag{11.5.4}$$

Here we have $\operatorname{diag}(\rho) = \frac{1}{\#S}\operatorname{Id}$. Moreover, since P is symmetric, we have $\Delta = \Delta^\top$.

Let us start with the bf-FB variant (backward-TD on F and forward-TD on B). In that case, the evolution equations are symmetric between F and B, because $\Delta = \Delta^\top$. Therefore, if $F = B$ at startup then $F = B$ at all times. Thus, we have $M_t = F_t^\top F_t \operatorname{diag}(\rho)$. Since $\operatorname{diag}(\rho)$ is proportional to Id, it commutes with all other terms. Thus, using $F_t = B_t$ and $\Delta = \Delta^\top$, we find

$$\begin{aligned}
M_t' &= (F_t')^\top F_t \operatorname{diag}(\rho) + F_t^\top F_t' \operatorname{diag}(\rho) && (11.5.5)\\
&= 2F_t^\top F_t \operatorname{diag}(\rho)^2 - 2F_t^\top F_t \Delta F_t^\top F_t \operatorname{diag}(\rho)^3 && (11.5.6)\\
&= 2M_t \operatorname{diag}(\rho) - 2M_t \Delta M_t \operatorname{diag}(\rho) && (11.5.7)\\
&= \frac{2}{S}(M_t - M_t \Delta M_t) && (11.5.8)
\end{aligned}$$

as $\operatorname{diag}(\rho) = \frac{1}{S}\operatorname{Id}$. This is the Bellman–Newton update.

In the other cases there is one more argument, after which the computation is similar. At startup, by assumption we have $F = B$ and Δ commutes with $F^\top B$. Assume that, at some particular time t, we have $F_t = B_t$ and Δ commutes with $F_t^\top B_t$. Then, since $\Delta = \Delta^\top$ and $\operatorname{diag}(\rho)$ commutes with everything, all the updates of F and B at that time t amount to just

$$F_t' = F_t \operatorname{diag}(\rho) - F_t F_t^\top F_t \Delta \operatorname{diag}(\rho)^2. \tag{11.5.9}$$

Therefore, at that time t, the derivative of the commutator between Δ and $F_t^\top B_t$ is

$$\begin{aligned}
[\Delta, F_t^\top B_t]' &= [\Delta, (F_t^\top F_t)'] && (11.5.10)\\
&= [\Delta, 2F_t^\top F_t \operatorname{diag}(\rho) - F_t^\top F_t F_t^\top F_t \Delta \operatorname{diag}(\rho)^2 - \Delta F_t^\top F_t F_t^\top F_t \operatorname{diag}(\rho)^2] && (11.5.11)\\
&= 0 && (11.5.12)
\end{aligned}$$

since Δ commutes with $F_t^\top F_t$ and $\operatorname{diag}(\rho)$ commutes with everything.

Thus, if at some time t one has $F_t = B_t$ and Δ commutes with $F_t^\top B_t$, then F_t and B_t have the same derivative at time t, and the derivative of the commutator of Δ and $F_t^\top B_t$ is 0. Therefore, if these conditions hold at startup, they hold at all times t. In that case, the evolution equations become identical to the bf-FB case, and the conclusion holds as above.

Chapter 12

Learning Value Functions via Successor States

There are a few possible ways to learn the value function V using use a model M_θ of the successor state operator. We mainly define two approaches: first, in Section 12.1 by using the equation $V^\pi(s) = (M^\pi \cdot R)(s)$ and estimating the integral $\int_{s_2} \rho(\mathrm{d}s_2) m_\theta(s, s_2) R(s_2)$. Then, in Section 12.2, by using m_θ as a way to propagate the Bellman error of Temporal Difference in the environment, similarly to TD(λ) with eligibility traces.

12.1 Learn V using directly $V = MR$

The simplest way to compute the value function V via a model $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ of the successor state operator is to estimate the matrix vector product $V = MR$. We consider three specific cases: first is the reward is located in a single goal state $R(s) = \mathbb{1}_{s=s_{\mathrm{tar}}}$, then if we use the FB parametrization defined in chapter 11, finally for a more general dense reward setting.

12.1.1 The sparse Dirac reward case

If the reward is located in a single state s_{tar}, we can directly estimate V as:

$$V_\theta(s) = m_\theta(s, s_{\mathrm{tar}}) \tag{12.1.1}$$

Indeed, in that case the reward in a continuous state space is usually defined up to a precision ε: $R_\varepsilon(s) = \mathbb{1}_{\|s - s_{\mathrm{tar}}\| \leqslant \varepsilon}$. In that case, in the limit $\varepsilon \to 0$, then the value function goes to $m(s, s_{\mathrm{tar}})$, up to a multiplicative constant. This case is very related to the goal-oriented setting developed in Part V, hence will not be described with more details here. The main difference is that here we learn a single policy $\pi(a|s)$ which does not depend on the target state, whereas in Part V we learn a goal dependent policy $\pi(a|s, g)$. In particular, the statement on the limit $\varepsilon \to 0$ informally stated above corresponds to Theorem 13.5 in the goal-oriented setting.

The sparse Dirac reward case can be extended to sparse rewards in a feature space of the state: if $\varphi : \mathcal{S} \to \mathbb{R}^k$ is a feature function of the states and the reward is $R_\varepsilon(s) = \mathbb{1}_{\|\varphi(s) - g\| \leqslant \varepsilon}$. This can be useful for instance in an environment in which the agent gets a reward if it brings an object to a goal: the state contains information on the position of both the agent and the object, but the reward only depend of the position of the object. In that case, we can learn the *successor feature operator* $M_\theta^\varphi(s_1, \mathrm{d}g) = m_\theta^\varphi(s_1, g)$ defined in Section 7.8. (In the example, $\varphi(\text{agent position}, \text{object position}) = \text{object position}$.) Then the value function can be estimated through:

$$V_\theta(s) = m_\theta^\varphi(s, g) \tag{12.1.2}$$

Algorithm 11 Policy evaluation via successor states with the FB parametrization.

Input: Policy $\pi(a|s)$, randomly initialized model $F_{\theta_F}(s)$, $B_{\theta_B}(s)$; an algorithm for learning $(F_{\theta_F}, B_{\theta_B})$ `FBAlgorithm` (such as Algorithm 10), momentum parameter α
Get an initial state s_0 from the environment.
repeat
  Sample $a_t \sim \pi(.|s_t)$, execute a_t and observe s_{t+1}, r_{t+1}
  Update $\theta_F, \theta_B \leftarrow \texttt{FBAlgorithm}(\texttt{s}_\texttt{t}, \texttt{s}_\texttt{t+1})$
  Update br: $br \leftarrow (1-\alpha)br + \alpha.r_t B_{\theta_B}(s_t)$
  \\ *Then, the value model is* $V(s) = \langle F_{\theta_F}(s), br\rangle$
  $t \leftarrow t+1$
until end of learning

12.1.2 Learn V with the FB parametrization

The FB representation of Chapter 11 directly provides a representation of the value function as

$$V(s) \approx F(s)^\top B(R), \qquad B(R) := \mathbb{E}_{s\sim\rho}[r_s\, B(s)] \tag{12.1.3}$$

where $B(R)$ is a "representation of the reward", which can be sampled by weighting the representation $B(s)$ of states by their reward. Thus $B(R)$ can be estimated online (Algorithm 11).

Since the FB representation will focus on low frequencies (long-range) features, it might be useful to used a "mixed" model for V, with $F(s)^\top B(R)$ as one component, and another component learned via ordinary TD; see (12.1.6) below.

12.1.3 General case, with a dense reward

We consider a general dense reward. We say that a reward is *dense* if, with an initial random policy, the probability of observing a reward is not 0. This covers all standard environments. Still, using a sparse reward which is non zero only on a specific target state s^{tar} can be a very important use case. In that case, in a continuous stochastic environment, the probability of reaching exactly the state s^{tar} hence observing a reward is 0. This is very related to multi-goal RL, discussed in Part V). In that case, a simple option is to learn a model of V based on $V = MR$. This becomes a supervised learning problem. No matrix product is necessary: we can perform a stochastic gradient descent of $\|V - MR\|^2_{L^2(\rho)}$ with respect to the parameters of V, just by sampling states, either with discrete or continuous states.

With V parameterized as V_φ, and with M parameterized by the model m_θ, we have

$$-\partial_\varphi \|V_\varphi - MR\|^2_{L^2(\rho)} = 2\mathbb{E}_{s\sim\rho,\, s_1\sim\rho}\left[\partial_\varphi V_\varphi(s_1)(r_s\, m(s_1, s) - V_\varphi(s_1))\right] \tag{12.1.4}$$

where r_s is the reward obtained when visiting state s. As for other algorithms presented here, this requires sampling one or several additional states s_1 in addition to the state s currently visited. This is formalized in Algorithm 12.

12.1.4 Mixed approach: mitigate the approximation errors of M with an additional component

Learning V via $V = MR$ assumes that the model of M is reasonably accurate: any error on M shows up on V. Another option is to just use MR as a component in the model of V, or as an initialization to V. For instance, V may be parameterized as

$$V := V_{\varphi_1} + V_{\varphi_2} \tag{12.1.5}$$

where V_{φ_1} is trained to match MR using (12.1.4), and V_{φ_2} is learned via ordinary TD.

In the FB representation this would yield

$$V(s) = F(s)^\top B(R) + V_{\varphi_2}(s) \tag{12.1.6}$$

Algorithm 12 Policy evaluation via M with a dense reward.

Input: Policy $\pi(a|s)$, randomly initialized M_{θ_M}, V_φ models, any algorithm for updating M_θ `SSLAlgorithm`; `TransitionMemory`, maximum number of time steps T, parameters $K, L \geqslant 0$
repeat
  for K trajectories **do**
    Get an initial state s_0, r_0 from the environment.
    for $0 \leqslant t \leqslant T$ steps do **do**
      Sample $a_t \sim \pi(.|s_t)$, execute a_t and observe s_{t+1}, r_{t+1}
      Store in the transition memory the transition `TransitionMemory` $\leftarrow (s_t, r_t, s_{t+1})$
    end for
  end for
  for L gradient steps **do**
    Sample a transition $(s, r, s') \sim$ `TransitionMemory`.
    Sample a state $(s_1, \_, \_) \sim$ `TransitionMemory`.
    Compute $\widehat{\delta\varphi} = \frac{1}{2}\partial_\varphi \left(r.m_\theta(s_1, s) - V_\varphi(s_1)\right)^2$
    Update M_θ with `SSLAlgorithm`
    Gradient steps: $\varphi \leftarrow \delta + \eta\widehat{\delta\theta_F}^{\text{fb-TD}}$
  end for
until end of learning

where $B(R)$ is estimated online as above, and φ_2 is estimated by ordinary TD.

This makes particular sense for the FB representation: in Section 11.4 we prove that the fb-FB algorithm minimizes a loss producing a truncated SVD of M, thus focusing on large eigenvalues of M (large eigenvalues of P, long-range dependencies in the environment). Thus $F(s)^\top B(R)$ will focus on large eigenvalues of P. The training of F and B is reward-independent ("unsupervised" reinforcement learning). Thus, ordinary TD on V_{φ_2} may be useful to catch short-range (high-frequency) behavior in the reward.

12.2 Using M for credit assignment: estimate the expected SSIPE and expected $TD(\lambda)$ updates

We now consider an other approach for policy evaluation via the successor states operator, in which the successor state model m_θ is used to propagate the Bellman error in the environment, or in other words to improve the *credit assignment* when observing a transition (s, r, s'). The algorithm is formalized in Algorithm 13. This method has two interesting interpretations:

First, we derive this method from the expected value update via SSIPE in the tabular case, presented in section 12.2.1. In Section 12.2.2, we define the corresponding algorithm with function approximations. Hence, this method can be seen as an approximation of the online update of the value function for the process estimation method described in Section 9.2.

We then show in Section 12.2.3 that this update corresponds to an estimate of the expected eligibility traces update in TD(λ). Eligibility traces, introduced in Section 1.4.2 are a way to improve credit assignment by propagating the Bellman error to the states recently visited in the current trajectory. We show that our approach is tackling credit assignment by propagating the Bellman error to all states which could have been visited from the current state s, according to the distribution of *predecessor* states, which is equivalent to the expected traces for a state s. This method is closely related to *expected eligibility traces* (van Hasselt et al., 2020), and *source traces* (Pitis, 2018), both discussed in Section 12.2.3.

12.2.1 Expected SSIPE update for the value function

In Chapter 9, we considered the tabular case and the process estimation algorithm defined in Equation (9.1.1): with this approach, we estimate the successor matrix and value function of a

finite MRP by $M_t = (\mathrm{Id} - \gamma P_t)^{-1}$ and $V_t = M_t R_t$ where P_t and R_t are estimated directly by the empirical averages. From this approach, in Section 9.4.1, we computed the corresponding *expected* update, and proved that in expectation over the transition (s_t, s'_t) observed at step t, conditionally to the current estimate M_t, we have: $\mathbb{E}_{s_t \sim \rho, s'_t \sim P(.|s_t)}[M_{t+1}] = M_t + \frac{1}{t}\delta M + o(1/t)$ with

$$\delta M = M_t - M_t(\mathrm{Id} - \gamma P)M_t \tag{12.2.1}$$

This approach lead to the Bellman–Newton update in Section 9.5.

We now consider the same approach, but for the value function. We first study in Theorem 12.1 the expected value update $\mathbb{E}[V_{t+1}]$, in expectation over the transition (s_t, r_t, s'_t) observed at step t. Then, in Section 12.2.2, we derive the corresponding update with function approximators. We will see that with this approach, the successor states operator is used to tackle the *credit assigment* problem and propagate the Bellman error in the environment.

Theorem 12.1. *Estimate the successor matrix of a finite MRP by $M_t = (\mathrm{Id} - \gamma P_t)^{-1}$ and $V_t = M_t R_t$ where P_t and R_t are estimated directly by the empirical averages* (9.1.1)*. Then, in expectation over the transition (s_t, r, s'_t) observed at step t, conditionally to the current estimates, we have:*

$$\mathbb{E}_{s_t \sim \rho, s'_t \sim P(.|s_t), r \sim \mathcal{R}(.|s)}[V_{t+1}] = V_t + \frac{1}{t}\delta V + o(1/t) \tag{12.2.2}$$

where

$$\delta V = M_t(R + \gamma P V_t - V_t) \tag{12.2.3}$$

Proof. First, note that the expectation in the statement is averaged over the next step, but conditional to all quantities $\hat{M}$, $\hat{V}$, etc., computed in the previous steps. In this proof, we will just write $\mathbb{E}$ for short.

To compute $V_{t+1} = M_t \cdot R_t$, let us first compute the update of R_{t+1}. By (9.1.1), the latter is $\mathbb{R}_{t+1} \leftarrow R_t + \delta R$ with

$$\delta R = \frac{1}{n_s}(r_s - (R_t)_s)\mathbb{1}_s = \frac{1}{t\rho_s}(r_s - (R_t)_s)\mathbb{1}_s + o(1/t). \tag{12.2.4}$$

Now, the update of $V_{t+1} = M_{t+1}R_{t+1}$ is $\delta V = \delta M R_t + M_t\,\delta R + \delta M\,\delta R$. The last term $\delta M\,\delta R$ is $O(1/t^2)$, so we can drop it. We find

$$\mathbb{E}[\delta V] = \mathbb{E}[\delta M R_t] + \mathbb{E}[M_t\,\delta R] + o(1/t) \tag{12.2.5}$$
$$= \mathbb{E}[\delta M]R_t + M_t\mathbb{E}[\delta R] + o(1/t) \tag{12.2.6}$$

since the expectations are averaged over the next step but conditional on the previous steps, which comprises the previous values R_t and M_t. Next,

$$\mathbb{E}[\delta M]R_t = \frac{1}{t}M_t(\gamma P M_t - M_t + \mathrm{Id})R_t + o(1/t) \tag{12.2.7}$$
$$= \frac{1}{t}M_t(\gamma P V_t - V_t + R_t) + o(1/t) \tag{12.2.8}$$

since $V_t = M_t R_t$. Next,

$$M_t\mathbb{E}[\delta R] = M_t \sum_s \rho_s \frac{1}{t\rho_s}(\mathbb{E}[r_s] - (R_t)_s)\mathbb{1}_s + o(1/t) \tag{12.2.9}$$
$$= \frac{1}{t}M_t(R - R_t) + o(1/t) \tag{12.2.10}$$

since $\sum_s \mathbb{E}[r_s]\mathbb{1}_s = R$ and $\sum_s (R_t)_s \mathbb{1}_s = R_t$. Summing, we find $\mathbb{E}[\delta V] = \frac{1}{t}M_t(\gamma P V_t - V_t + +R) + o(1/t)$ as needed.

Thus, the online update of the value function via the process estimation approach (equivalently SSIPE) corresponds to a propagation of the Bellman error via M: it estimates the Bellman error for the model V_t for transition $(s, r, s') : \delta = r + \gamma V_t(s') - V(s)$. Then, for every state s_1, it propagates the error with weight $M_t(s_1, s)$ to update the value $V_t(s_1)$ as $V_{t+1}(s_1) \leftarrow V_t(s_1) + M_t(s_1, s)\delta$.

This update of V is consistent with the view of M as an expected eligibility trace, as formalized in Section 12.2.3. Indeed, eligibility traces also update the value function at states s_1 that are connected to s via a trajectory. Actually, in expectation, these updates are the same: with $\lambda = 1$, the eligibility trace vector at a state s is an unbiased estimator of the column $M_{s_1 s}$ (Theorem 12.3). From this viewpoint, learning M via a parametric model, or using TD(1), are

both ways of estimating the "predecessor states" of a state s. Eligibility traces are unbiased but can have large variance, while the model of M has no variance but may have bias if not learned well.

12.2.2 Expected SSIPE update estimation with parametric models

We can generalize the value update obtained in Theorem 12.1 to any state space, with parametric models $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ and $V_\varphi(s)$. The parametric update is derived in the following theorem, and the corresponding algorithm is specified in Algorithm 13.

Theorem 12.2. *Let V_φ be a smooth parametric model of the value function. Define an update of V by setting $V^{\mathrm{tar}} := V_\varphi + \delta V$ with*

$$\delta V := M_\theta(R + \gamma P V_\varphi - V_\varphi), \tag{12.2.11}$$

as given by (12.2.3).

Let (s, s', r_s) be a sample of the environment such that $s' \sim P(s'|s, a)$, r_s is the reward observed in s, we define:

$$\widehat{\delta\varphi}_{prop\text{-}TD}(s, s', r_s, s_1) = (r_s + \gamma V_\varphi(s') - V_\varphi(s))\, m_\theta(s_1, s)\partial_\varphi V_\varphi(s_1) \tag{12.2.12}$$

Then, $\widehat{\delta\varphi}_{prop\text{-}TD}$ is an unbiased estimate of the error between V_θ and V^{tar}:

$$\mathbb{E}_{s\sim\rho, s'\sim P(\mathrm{d}s'|s), r_s\sim R(s), s_1\sim\rho}\left[\widehat{\delta\varphi}_{prop\text{-}TD}(s, s', r_s, s_1)\right] = -\partial_\varphi \|V_\varphi - V^{\mathrm{tar}}\|^2_{L^2(\rho)} \tag{12.2.13}$$

This involves sampling an additional state $s_1 \sim \rho$ and applying a TD update at that point, with weight depending on M. Notably, even if the model of M is wrong, the true value function is still a fixed point of (12.2.13) in expectation over s' and r_s; it is the only fixed point provided $\hat{M}$ is invertible and $\rho > 0$. This is a theoretical advantage over all other estimates of V described above. However, the sampling of s_1 adds variance, and any negative eigenvalues in the estimate of M will produce divergence.

Proof. We have:

$$\begin{aligned}
&-\frac{1}{2}\partial_\varphi \|V_\varphi - V^{\mathrm{tar}}\|^2_{L^2(\rho)} = \\
&= \int_{s_1} \rho(\mathrm{d}s_1)\partial_\varphi V_\varphi(s_1)\delta V(s_1) \\
&= \int_{s_1, s, s'} \rho(\mathrm{d}s_1)\partial_\varphi V_\varphi(s_1) M_\theta(s_1, \mathrm{d}s)(R + \gamma P V_\varphi - V_\varphi)(s) \\
&= \int_{s_1, s, s'} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s)P(\mathrm{d}s'|s)\partial_\varphi V_\varphi(s_1) m_\theta(s_1, s)(R(s) + \gamma V_\varphi(s') - V_\varphi(s))
\end{aligned}$$

This method is related to $TD(\lambda)$ and eligibility traces, as described in the next secion.

12.2.3 The expected SSIPE value update is an expected eligibility traces update

In this section, we show the relation between the expected SSIPE value update described in Theorem 12.2 and $TD(\lambda)$ with eligibility traces.

Eligibility traces require access to an arbitrarily long trajectory $\tau = (s_t, r_t)_{t\in\mathbb{Z}}$ (which, for convenience, we index with both positive and negative integers, with s_0 the state at the current time). Thus, contrary to the rest of this text, we assume that the Markov process is ergodic and that the data are coming from a stationary random trajectory of the process. In this case, the sampling measure ρ is the stationary distribution, and the law of any sequence of consecutive observations $(s_t, \ldots, s_{t+n})$ from the trajectory is $\rho(\mathrm{d}s_t)P(s_t, \mathrm{d}s_{t+1}) \cdots P(s_{t+n-1}, \mathrm{d}s_{t+n})$.

In the tabular setting, TD(λ) maintains a vector e_t over states; e_t is updated by

$$e_t(\tilde{s}) = \mathbb{1}_{s_t} + \gamma\lambda e_{t-1}(\tilde{s}) \qquad \forall \tilde{s} \tag{12.2.14}$$

$$\delta V(\tilde{s}) = e_t(\tilde{s})(r_t + \gamma V(s_{t+1}) - V(s_t)) \qquad \forall \tilde{s}. \tag{12.2.15}$$

Algorithm 13 Policy evaluation via the expected SSIPE Value update.

Input: Policy $\pi(a|s)$, randomly initialized M_{θ_M}, V_φ models, `SSLAlgorithm`; `TransitionMemory`, maximum number of time steps T, $K, L \geqslant 0$
repeat
  for K trajectories **do**
    Get an initial state s_0, r_0 from the environment.
    for $0 \leqslant t \leqslant T$ steps do **do**
      Sample $a_t \sim \pi(.|s_t)$, execute a_t and observe s_{t+1}, r_{t+1}
      Store in the transition memory the transition `TransitionMemory` $\leftarrow (s_t, r_t, s_{t+1})$
    end for
  end for
  for L gradient steps **do**
    Sample a transition $(s, r, s') \sim$ `TransitionMemory`.
    Sample $(s_1, \_, \_) \sim$ `TransitionMemory`
    Compute $\widehat{\delta\varphi}_{\text{prop-TD}} = (r + \gamma V_\varphi(s') - V_\varphi(s))\, m_\theta(s_1, s)\partial_\varphi V_\varphi(s_1)$
    Update M_θ with `SSLAlgorithm`
    Gradient steps: $\varphi \leftarrow \delta + \eta\widehat{\delta\theta_F}^{\text{fb-TD}}$
  end for
until end of learning

This can be generalized to continuous environments and to a parametric model V_φ of V, by formally defining e as the discounted empirical measure of the past. For a trajectory $\tau = (s_t, r_t)_{t\in\mathbb{Z}}$ we define:

$$e_t(\mathrm{d}\tilde{s}) := \sum_{n\geqslant 0}(\gamma\lambda)^n \delta_{s_{t-n}}(\mathrm{d}\tilde{s}) = \delta_{s_t}(\mathrm{d}\tilde{s}) + \gamma\lambda e_{t-1}(\mathrm{d}\tilde{s}) \tag{12.2.16}$$

and we can update corresponding to the parametric update of V_φ similarly to the tabular case in equation (12.2.15) by

$$\begin{aligned}\delta\varphi &:= (r_t + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t)) \int_{\tilde{s}} \partial_\varphi V_\varphi(\tilde{s})\, e_t(\mathrm{d}\tilde{s}) \\ &= (r_t + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t)) \sum_{n\geqslant 0}(\gamma\lambda)^n \partial_\varphi V_\varphi(s_{t-n}).\end{aligned}$$

In practice, the standard way of using eligibility traces (Sutton and Barto, 2018, Section 12.2)is to estimate the quantity $\bar{e}_t = \sum_{n\geqslant 0}(\gamma\lambda)^n \partial_\varphi V_\varphi(s_{t-n})$ (which is a vector of the parameter space of V_φ) as:

$$\bar{e}_t = \partial_\varphi V_\varphi(s_t) + \gamma\lambda\bar{e}_{t-1} \tag{12.2.17}$$

then updating the value function as:

$$\widehat{\delta\varphi}_{\mathrm{TD}(\lambda)}(\tau, t) = (r_t + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t))\bar{e}_t \tag{12.2.18}$$

This update propagates the Bellman error to previously observed states, directly via the gradient average computed in these states.

The following statement connects the eligibility traces update and the expected SSIPE value update. This analysis is related to *expected eligibility traces* (van Hasselt et al., 2020) and source traces (Pitis, 2018). It uses the idea of *backward* process introduced in Section 8.4.1, for the study of Backward Temporal Difference. The *backward process* $P_{\text{back}}(s', \mathrm{d}s)$ is the process obtained from P by reversing time: it is the law of s given s' in a transition $s \to s'$. In particular, we have $\rho(\mathrm{d}s)P(s, \mathrm{d}s') = \rho(\mathrm{d}s')P_{\text{back}}(s', \mathrm{d}s)$. We can then define $M^{\text{back}} := (\mathrm{Id} - \gamma P_{\text{back}})^{-1}$ the successor state operator of the backward process. From Lemma 8.4, we know that $\rho(\mathrm{d}s)M(s, \mathrm{d}s') = \rho(\mathrm{d}s')M^{\text{back}}(s', \mathrm{d}s)$. If $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ is a successor states model, we define $M_\theta^{\text{back}}(s_1, \mathrm{d}s_2) = m_\theta(s_2, s_1)\rho(\mathrm{d}s_2)$, such that this definition is consistent with M^{back} and we have $\rho(\mathrm{d}s)M_\theta(s, \mathrm{d}s') = \rho(\mathrm{d}s')M_\theta^{\text{back}}(s, \mathrm{d}s')$. We then have the following statement:

Theorem 12.3. *Let ρ be the invariant measure of the Markov process, and $M_{\gamma\lambda} := (\mathrm{Id} - \gamma\lambda P)^{-1}$ the successor state operator with discount factor $\gamma\lambda$. We observe a transition (s_t, r_t, s_{t+1}). We define $\Delta := r_t + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t)$, the bellman gap for the current model V_φ for the transition (s_t, r_t, s_{t+1}).*

The expected parametric TD(λ) update (12.2.18) *knowing (s_t, r_t, s_{t+1}) is observed is:*

$$\mathbb{E}_\tau[\widehat{\delta\varphi}_{\mathrm{TD}(\lambda)}(\tau, t)|s_t, r_t, s_{t+1}] = \Delta. \left(M^{\mathrm{back}}_{\gamma\lambda} \cdot (\partial_\varphi V_\varphi)\right)(s_t) = \Delta. \int_{\tilde{s}} M^{\mathrm{back}}_{\gamma\lambda}(s_t, \mathrm{d}\tilde{s}) \partial_\varphi V_\varphi(\tilde{s})$$

with ρ-probability 1 *over s_t.*

Similarly, the expected SSIPE value update (defined in Theorem 12.2) with respect to $\tilde{s} \sim \rho$, is equal to:

$$\mathbb{E}_{\tilde{s}\sim\rho}\left[\widehat{\delta\varphi}_{prop\text{-}TD}(s_t, s_{t+1}, r_s, \tilde{s})\right] = \Delta. \left(M^{\mathrm{back}}_{\theta} \cdot (\partial_\varphi V_\varphi)\right)(s_t) = \Delta. \int_{\tilde{s}} M^{\mathrm{back}}_{\theta}(s_t, \mathrm{d}\tilde{s}) \partial_\varphi V_\varphi(\tilde{s})$$

This theorem proves the relation between using M for credit assignment and the TD(λ) algorithm. If $M_\theta = M^\pi$ (we exactly learned the successor states operator), the update $\widehat{\delta\varphi}_{\text{prop-TD}}$ propagates the Bellman error to all the states which could have been observed before arriving in s_t, which is the expectation of the eligibility traces. To summarize, the TD(λ) algorithms use a Monte-Carlo approach to estimate the expected traces, while we learn an approximation of this quantity via M. The first approach will have higher variance but no bias, while the other one will have lower variance but potentially high bias. This difference is similar between a Monte Carlo approach for estimating the value function, and a TD udpate.

From that point of view, this approach is similar to *expected eligibility traces* (van Hasselt et al., 2020). The two approaches estimate the same quantity:

$$\left(M^{\mathrm{back}}_{\gamma\lambda} \cdot (\partial_\varphi V_\varphi)\right)(s) = \int_{\tilde{s}} M^{\mathrm{back}}_{\gamma\lambda}(s, \mathrm{d}\tilde{s}) \partial_\varphi V_\varphi(\tilde{s}) = \mathbb{E}_{\tau|s_t=s}\left[\sum_{n\geqslant 0} (\gamma\lambda)^n \partial_\varphi V_\varphi(s_{t-n})\right] \tag{12.2.19}$$

Then, this quantity is used to propagate the Bellman error to every state which could have been observed when arriving in s, in other words to tackle the credit assignment problem. The difference between the two approaches is the following: Our approach learns $M_\theta \approx M^\pi$, then estimates the integral $\left(M^{\mathrm{back}}_{\theta} \cdot (\partial_\varphi V_\varphi)\right)(s) = \int_{\tilde{s}} \rho(\mathrm{d}\tilde{s}) m_\theta(\tilde{s}, s) \partial_\varphi V_\varphi(\tilde{s})$. The approach from van Hasselt et al. (2020) directly estimates this quantity via a function z_θ, such that $z_\theta(s) \approx \left(M^{\mathrm{back}}_{\gamma\lambda} \cdot (\partial_\varphi V_\varphi)\right)(s)$. The function $z_\theta(s)$ is learned online in a supervised way, via the udpate $\delta\theta = \frac{1}{2}\partial_\theta \|z_\theta(s_t) - \bar{e}_t\|^2$. The approach is similar in *Source Traces* (Pitis, 2018) in the tabular case.

Proof. By definition of eligibility traces, one has $e_t(\mathrm{d}\tilde{s}) = \sum_{n\geqslant 0} (\gamma\lambda)^n \delta_{s_{t-n}}(\mathrm{d}\tilde{s})$. Therefore, the expectation of e_t over the past of s_t knowing s_t is:

$$\begin{aligned}
\mathbb{E}_\tau[\bar{e}_t|s_t = s] &= \mathbb{E}_{\tau|s_t}\left[\sum_{n\geqslant 0} (\gamma\lambda)^n \partial_\varphi V_\varphi(s_{t-n})\right] \\
&= \mathbb{E}_{\tau|s_t}\left[\sum_{n\geqslant 0} (\gamma\lambda)^n \int_{\tilde{s}} \delta_{s_{t-n}}(\mathrm{d}\tilde{s}) \partial_\varphi V_\varphi(\tilde{s})\right] \\
&= \mathbb{E}_{\tau|s_t}\left[\int_{\tilde{s}} \partial_\varphi V_\varphi(\tilde{s}) \sum_{n\geqslant 0} (\gamma\lambda)^n \delta_{s_{t-n}}(\mathrm{d}\tilde{s})\right] \\
&= \int_{\tilde{s}} \sum_{n\geqslant 0} (\gamma\lambda)^n P^n_{\mathrm{back}}(s_t, \mathrm{d}\tilde{s}) \partial_\varphi V_\varphi(\tilde{s}) \\
&= \left(M^{\mathrm{back}}_{\gamma\lambda} \cdot \partial_\varphi V_\varphi\right)(s_t)
\end{aligned}$$

Hence, we have:

$$\begin{aligned}\mathbb{E}_\tau[\widehat{\delta\varphi}_{\mathrm{TD}(\lambda)}(\tau,t)|s_t,r_t,s_{t+1}] &= (r_t+\gamma V_\varphi(s_{t+1})-V_\varphi(s_t))\mathbb{E}_{\tau|s_t}\left[\bar{e}_t\right] \\ &= \Delta\left(M^{\mathrm{back}}_{\gamma\lambda}\cdot\partial_\varphi V_\varphi\right)(s_t)\end{aligned}$$

On the other side, we have:

$$\begin{aligned}\mathbb{E}_{\tilde{s}\sim\rho}\left[\widehat{\delta\varphi}_{\text{prop-TD}}(s_t,s_{t+1},r_s,\tilde{s})\right] &= \mathbb{E}_{\tilde{s}\sim\rho}\left[(r_t+\gamma V_\varphi(s_{t+1})-V_\varphi(s_t))\,m_\theta(\tilde{s},s_t)\partial_\varphi V_\varphi(\tilde{s})\right] \\ &= \Delta\int_{\tilde{s}}\rho(\mathrm{d}\tilde{s})m_\theta(\tilde{s},s_t)\partial_\varphi V_\varphi(\tilde{s}) \\ &= \Delta.\left(M^{\mathrm{back}}_\theta\cdot(\partial_\varphi V_\varphi)\right)(s_t)\end{aligned}$$

Part V

Unbiased Methods for Multi-goal Reinforcement Learning

Chapter 13

The multi-goal RL setting, sparse rewards and infinitely sparse rewards

13.1 An introduction to multi-goal RL

Most standard *reinforcement learning* (RL) methods fail when faced with very sparse reward signals. Multi-task reinforcement learning attempts to solve this problem by presenting agents with a diverse set of tasks and learn a task-dependent policy in the hope that the agent could leverage knowledge from some tasks on others (Jaderberg et al., 2016; Hausman et al., 2018; Nagabandi et al., 2019). *Multi-goal* reinforcement learning, introduced by Kaelbling (1993), is a sub-field of multi-task RL, where the different tasks consist in reaching particular *goals* in the environment. This can typically be applied in robotics tasks: for example, a robotic arm needs to push a cube towards a goal position given as input at the beginning of the episode (Plappert et al., 2018; Andrychowicz et al., 2020). It can also be used in a hierarchical pipeline, to do planning by selecting goals an agent should reach, to solve a more complex task (Nair et al., 2018; Nasiriany et al., 2019).

Multi-goal reinforcement learning was first introduced in the finite MDP setting (Kaelbling, 1993). Since then, the sample efficiency of multi-goal RL has been studied in that setting (Tarbouriech et al., 2020). Universal Value Function Approximators (Schaul et al., 2015) show how to adapt standard RL algorithm with function approximators such as Temporal Difference and Q-learning in the multi-goal setting, by learning multi-goal value functions.

One of the most important issue of the multi-goal setting is the sparsity of the reward. Indeed, the goal-oriented reward is usually defined as $R_\varepsilon(s, g)$ which is 1 if the current agent is at a distance to the goal less than ε, 0 everywhere else. In that case, in a continuous environment of dimension d, the probability of reaching a goal with a random exploration policy scales as $O(\varepsilon^d)$. Because of the *curse of dimensionality*, an agent might never see any goal. In this Part, we call this the *vanishing reward* phenomena. Reward shaping (Ng et al., 1999) can be a way to guide the policy towards goals and remove the sparsity issue, but finding a reward both simpler for the agent and such that than the learned policies transfer to the original reward is not easy, and require a lot of human expert effort. A very successful approach is to re-use trajectories observed while aiming at a goal g to learn how to reach other goals g' with *hindsight*: if during a trajectory aiming at g the agent randomly observes a goal g', then that same trajectory can be used to learn how to reach g' as if g' was the target goal from the beginning. This principle was introduced with *Hindsight Experience Replay* (HER) (Andrychowicz et al., 2017), and lead to other methods (Rauber et al., 2019; Li et al., 2020; Fang et al., 2019; Manela and Biess, 2021). One issue of hindsight methods is their bias (Plappert et al., 2018; Manela and Biess, 2021), which corresponds to a well-identified psychological bias (Fischhoff, 1975) and can lead

to low-return policies. We study the bias of HER in Chapter 14, and show that it is actually unbiased in deterministic environments.

An other approach for multi-goal RL is to learn via *curriculum learning* (Florensa et al., 2017): by giving the agent simple goals to achieve at the beginning (closer to the starting point), the agent will be able to learn how to reach them. Then, a *teacher* can increase the difficulty during training. Multiple approaches developped goal sampling strategies (Eppe et al., 2019; Pitis et al., 2020; Fang et al., 2019; Colas et al., 2019). In particular Venkattaramanujam et al. (2019) introduce a method to sample goals without using the goal space distance, which might not be correlated with the difficulty to reach a goal.

In this chapter, we briefly introduce multi-goal RL. First, we formalize the multi-goal setting, and how it can be seen as standard RL via the augmented multi-goal environment. Then, we introduce standard approaches, especially our two main baselines: *Universal value function approximators* and *Hindsight experience replay.* Then, in Section 13.2, we define the *infinitely sparse reward limit*, and how standard objects (reward, value function, successor states, return, ...) translates when considering infinitely sparse rewards. Moreover, we show that all objects are consistent between the sparse and infinitely sparse settings. Then, in Chapters 16 and 15 we will derive unbiased actor critics and Q-learning methods for multi-goal RL via the infinitely sparse reward viewpoint.

13.1.1 Multi-Goal Reinforcement Learning and Vanishing Rewards

We define a multi-goal RL environment as a variant of a Markov decision process (MDP) including a goal space. The MDP is defined by a state-space $\mathcal{S}$, an action space $\mathcal{A}$ (discrete or continuous), a discount factor γ, and a transition probability measure $P(\mathrm{d}s'|s,a)$ which describes the probability that the next state is s' after taking action a in state s. For stochastic continuous environments, this is generally a continuous probability distribution over s', hence the notation $\mathrm{d}s'$ which represents the probability to be in an infinitesimal set $\mathrm{d}s'$ around s', as we used in Part IV for the successor state operator.

The goal space is a set $\mathcal{G}$ together with a function $\varphi\colon \mathcal{S} \to \mathcal{G}$ defining for every state s a corresponding goal $g = \varphi(s)$, which is the goal *achieved* by state s. The objective of the agent is to *reach* a goal g. This is usually formalized by defining a reward function $R_\varepsilon(s,g)$ as 1 when a given distance between the achieved goal $\varphi(s)$ and the target g is lower than a fixed value ε:

$$R_\varepsilon(s,g) := \mathbb{1}_{\|\varphi(s)-g\|\leqslant\varepsilon} \tag{13.1.1}$$

for a fixed norm $\|.\|$ on $\mathcal{G}$. Thus, each goal $g \in \mathcal{G}$ defines an ordinary MDP with reward $R(s,g)$, and Q and value functions $Q^*_\varepsilon(s,a,g)$, $V^\pi_\varepsilon(s,g)$. A goal-conditioned policy $\pi(a|s,g)$ is a probability distribution over the action space $\mathcal{A}$ for every $(s,g) \in \mathcal{S}\times\mathcal{G}$.

We assume that, for a multi-goal policy $\pi(a|s,g)$, we are able to sample trajectories in the environment by sampling a goal $g \sim \rho_{\mathcal{G}}(\mathrm{d}g)$, a starting state $s_0 \sim \rho_0(\mathrm{d}s_0|g)$, and then by sampling at step t the action $a_t \sim \pi(a|s_t,g)$ and the next state $s_{t+1} \sim P(\mathrm{d}s'|s_t,a_t)$. We use the notation $P^\pi(\mathrm{d}s'|s,g) := \int_a \pi(a|s,g)P(\mathrm{d}s'|s,a)$.

13.1.2 The augmented state-goal process

The multi-goal setting can be seen as a standard multi-goal environment, via the augmented state-goal process. Informally, we assume that the agent observes at every time-step the state s_t together with the target goal g. The goal is now part of the observation. After using action a_t and reaching a state s_{t+1}, the new observation is (s_{t+1},g), as the target goal is unchanged.

Formally, if $\mathcal{M}$ is a multi-goal environment, with the notation above, we define $\tilde{\mathcal{M}}$ the Markov decision process as $\tilde{\mathcal{M}} := \langle\tilde{\mathcal{S}},\mathcal{A},\tilde{P},\tilde{R},\gamma\rangle$, with

- The state space $\tilde{\mathcal{S}}$ is defined as: $\tilde{\mathcal{S}} := \mathcal{S}\times\mathcal{G}$. A state $\tilde{s}$ in the state-goal process is a tuple $\tilde{s} = (s,g)$.
- The action space is unchanged.

- The transition operator is defined as $\tilde{P}(\mathrm{d}\tilde{s}'|\tilde{s}, a) = \tilde{P}(\mathrm{d}s', \mathrm{d}g'|s, g, a) = P(\mathrm{d}s'|s, a)\delta_g(\mathrm{d}g')$
- The reward is $\tilde{R}_\varepsilon(\tilde{s}) = \tilde{R}_\varepsilon((s, g)) = R_\varepsilon(s, g)$.

A policy in the augmented environment $\tilde{\mathcal{M}}$, $\pi(a|\tilde{s})$ corresponds to the *multi-goal* policy $\pi(a|s, g)$ in the multi-goal environment. Similarly, the multi-goal value function $V_\varepsilon(s, g)$ corresponds to a standard value function in the augmented environment.

This mathematical construction is similar to the augmented state-action process defined in Section 7.9.1. In that case, we saw that applying standard policy evaluation method for the value function in the state-action process lead to algorithms to learn the Q-function. Similarly, here, we will see that standard RL algorithms can be directly applied to the multi-goal setting via the augmented state-goal process.

13.1.3 Universal Value Function Approximations.

Universal Value Function Approximators (UVFA) (Schaul et al., 2015) extend the classical Q-learning and Temporal Difference (TD) algorithms to the multi-goal setting. It learns the goal-conditioned value-function $V_\varepsilon^\pi(s, g)$ or Q-function $Q_\varepsilon^*(s, a, g)$ corresponding to the sparse rewards $R_\varepsilon(s, g)$ for every state-goal pair, with function approximation, via a TD algorithm.

In practice, we consider a parametric function $Q_\theta(s, g)$, and we want to learn θ such that $Q_\theta(s, g)$ approximates $Q_\varepsilon^*(s, g)$. If θ is our current estimate and Q_{tar} a target Q-function the Q-learning UVFA stochastic update $\widehat{\delta\theta}_{\text{UVFA}}$ is defined as follows. We consider an exploration policy $\pi_{\text{expl}}(a|s, g)$. When a transition (s, a, s', g) is observed, with $a \sim \pi_{\text{expl}}(.|s, g)$ and $s' \sim P(.|s, g)$, $\widehat{\delta\theta}_{\text{UVFA}}$ is:

$$\widehat{\delta\theta}_{\text{UVFA}}(s, a, s', g) := -\frac{1}{2}\partial_\theta \left(Q_\theta(s, a, g) - R_\varepsilon(s, g) - \gamma \sup_{a'} Q_{\text{tar}}(s', a', g) \right)^2 \tag{13.1.2}$$

Then, we update θ with $\theta \leftarrow \theta + \eta\widehat{\delta\theta}_{\text{UVFA}}$, where η is the learning rate. The update $\widehat{\delta\theta}_{\text{UVFA}}$ is an unbiased estimate of $-1/2\partial_\theta\|Q_\theta - T \cdot Q_{\text{tar}}\|^2$ where T is the optimal Bellman operator, $T \cdot Q(s, a, g) = R_\varepsilon(s, g) + \gamma\mathbb{E}_{s'\sim P(.|s,a)}\left[\sup_{a'} Q(s', a', g)\right]$, whose unique fixed point is Q_ε^*. In particular, in the tabular setting, this guarantees that a function Q_∞ is a fixed point of UVFA if and only if $T \cdot Q_\infty = Q_\infty$, which means $Q_\infty = Q_\varepsilon^*$.

Formally, Q-learning with UVFA exactly corresponds to the DQN algorithm on the augmented state-goal environment defined in the previous section. Let us define $\tilde{Q}_\theta(\tilde{s}, a) = Q_\theta(s, a, g)$ (for $\tilde{s} = (s, g)$), and similarly for $\tilde{Q}_{\text{tar}}$. Consider we observe a transition (s, a, s', g). Then, we have

$$\begin{aligned}\widehat{\delta\theta}_{\text{UVFA}}(s, a, s', g) &= -\frac{1}{2}\partial_\theta \left(\tilde{Q}_\theta(\tilde{s}, a) - \tilde{R}_\varepsilon(\tilde{s}) - \gamma \sup_{a'} \tilde{Q}_{\text{tar}}(\tilde{s}', a') \right)^2 \\ &= \widehat{\delta\theta}_{\text{DQN}}(\tilde{s}, a, \tilde{s}')\end{aligned}$$

where $\tilde{s} = (s, g)$, $\tilde{s}' = (s', g)$, and $\widehat{\delta\theta}_{\text{DQN}}$ is the standard Deep Q-learning update in the augmented state space $\tilde{\mathcal{M}}$, for model $\tilde{Q}_\theta(\tilde{s}, a)$.

This equivalence between UVFA and DQN in the augmented state space has a main limitation: it means the algorithm is considering every problem defined by a goal g as an independent environment. Indeed, the augmented multi-goal environment formalism *forgets* that the transition operator $\tilde{P}(\mathrm{d}s', \mathrm{d}g'|s, a, g)$ can be factorized as a transition operator independent of g $P(\mathrm{d}s'|s, a)$ and that goals stay constant. Or equivalently, it doesn't take into account that a trajectory in $\tilde{M}$, $(\tilde{s}_0, r_0, a_0, \tilde{s}_1, r_1, a_1...) = ((s_0, g), r_0, a_0, (s_1, g), r_1, a_1...)$ can be decomposed into a goal-independent trajectory $(s_0, a_0, s_1, a_1...)$ (with $s_{t+1} \sim P(.|s_t, a_t)$), and a sequence of rewards defined as $r_k = R_\varepsilon(s_k, g)$. Hence, an algorithm which can be defined entirely with the augmented multi-goal environment formalism cannot be leveraging some of the structure of the multi-goal setting. For example, such an algorithm cannot use the structure of a multi-goal environment to compute, from a trajectory $(s_0, a_0, s_1, a_1...)$, the sequence of rewards for *any goal*

in $\mathcal{G}$ (not necessarily the original goal), and thus cannot leverage all the mutual information between goals. In practice, some generalization between goals will occur, because of the choice of the parametric function approximator.

Furthermore, a method defined in the multi-goal augmented environment, such as UVFA, will be limited by the vanishing rewards issue, presented in the next section.

13.1.4 UVFA and vanishing rewards.

A major problem with multi-goal setups is the low probability with which each specific goal g is achieved, since rewards are observed only in a ball of radius ε around the goal. In a continuous noisy environment of dimension n, reaching a goal up to precision ε becomes almost surely impossible when $\varepsilon \to 0$. With noise in dimension n, the probability to exactly reach a predefined goal g scales like $O(\varepsilon^n)$. In particular, the Q and value functions vanish like $O(\varepsilon^n)$ when ε is small. The situation is different in continuous deterministic environments. If it is possible to reach a goal exactly by selecting the right action, then the optimal Q-function Q^*_ε does not vanish, even if $\varepsilon = 0$.

With the UVFA update, the probability to observe a reward $\mathbb{1}_{\|\varphi(s)-g\| \leqslant \varepsilon}$ vanishes like $O(\varepsilon^n)$ for continuous exploration policies. So even if Q^*_ε itself does not vanish, the learning algorithm for Q^*_ε may vanish. In practice, in an environment of dimension $n = 6$, UVFA is not able to learn anymore (experiment in Fig. 16.1). This vanishing issue cannot be solved solely by an exploration strategy: the issue is not the lack of diversity in visited states but rather the state space is too large to be visited by an exploration trajectory (Andrychowicz et al., 2017). Solving the issue of sparse rewards requires gathering some information even from *failing trajectories* which do not reach their initial goal, namely, leveraging the structure of multi-goal environments by using that every state achieves *some* goal. This the case in *Hindsight Experience Replay* but not UVFA.

In this work we study algorithms which leverage the multi-goal structure and do not vanish even in the limit $\varepsilon \to 0$. We will focus on *unbiased* algorithms, which ensure that the true Q or value function is indeed a fixed point, by stochastic gradient arguments. UVFA is unbiased but vanishes when $\varepsilon \to 0$. HER does not vanish, but is known to be biased. In Section 14.2 we prove that HER is unbiased in deterministic environments. Chapters 15 and 16 present non-vanishing, unbiased algorithms for stochastic environments; however, they are less efficient than HER in deterministic environments.

13.1.5 Hindsight Experience Replay

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) is an algorithm for multi-goal RL, which removes the issue of vanishing rewards, and introduce generalization between goals.

It leverages information between goals via the following principle: trajectories aiming at a goal g but reaching a goal g' can be used for learning exactly as if the trajectory had been aiming at g' from start. Formally, HER is defined as follows: first, we consider an off-policy algorithm, such as DQN (Mnih et al., 2015) or DDPG (Lillicrap et al., 2015). These algorithms have two steps: first aquire a set of trajectories, and add them to a *replay buffer*. Then, sample some transitions from this *replay buffer* and update the current estimate $Q_\theta(s, a, g)$ of Q^*_ε. HER is acquiring trajectories similarly, but changes the way transitions are sampled from the replay buffer: first, samples a trajectory τ from the buffer, $\tau = (g, s_0, a_0, s_1, a_1, ...)$. Then, it samples two random integers $0 \leqslant K \leqslant L$, and return the transition (s_K, a_K, s_{K+1}, g'), where g' is a re-sampled goal that is, with some probability, either $g' = g$ (the original goal) or $g' = \varphi(s_L)$, the goal achieved by the L-th state in the trajectory. This transition is then used to perform a standard update for the transition (s_K, a_K, s_{K+1}, g') for instance with DQN:

$$\widehat{\delta\theta}_{\text{HER}}(\tau, K, L) := \frac{1}{2}\partial_\theta \left(Q_\theta(s_K, a_K, g') - R_\varepsilon(s_K, g') - \gamma \sup_{a'} Q_{\text{tar}}(s_{K+1}, a', g') \right)^2 \tag{13.1.3}$$

$$= \widehat{\delta\theta}_{\text{UVFA}}(s_K, a_K, s_{K+1}, g') \tag{13.1.4}$$

where $g' = g$ or $g' = \varphi(s_L)$. A more formal definition is given in Chapter 14.

First, we can remark that the HER update does not vanish even for $\varepsilon = 0$: with nonzero probability, $K = L$ and $g' = \varphi(s_L)$, so that $R_\varepsilon(s_K, g') = 1$. Then, we can add that HER introduces some generalization between goals: a trajectory τ sampled while aiming at a goal g is then used to learn how to reach every goal $g' = \varphi(s_L)$ achieved during that trajectory.

This strategy has proved successful in practice, but is known to be *biased* (Manela and Biess, 2021; Lanka and Wu, 2018). In their request for research for robotic multi-goal environments, Plappert et al. (2018) list the necessity for an unbiased version of HER, as such bias can lead to low-return policies. In Chapter 14, we study the bias of HER. First, we show (theoretically and empirically) that HER is biased, and can converge to low-return policies. Then, we show that it is actually *unbiased* in deterministic environments, which covers many standard cases such as robotics.

13.2 The Infinitely Sparse Reward Limit

While *Hindsight Experience Replay* is performing well in many environments, it is known to be biased in general stochastic environments and can learn low-return policies. In this part of the thesis, we will define unbiased algorithm for multi-goal RL, resilient to the vanishing reward issue presented above, and introducing some generalization between goals. Our approach is to replace *sparse* rewards $R_\varepsilon(s, g) = \mathbb{1}_{\|\varphi(s)-g\| \leqslant \varepsilon}$ by *infinitely sparse* Dirac rewards $R(s, \mathrm{d}g) := \delta_{\varphi(s)}(\mathrm{d}g)$, and then use our knowledge on the Dirac function to derive our updates.

In continuous state spaces, the reward is usually defined as $R_\varepsilon(s, g) = \mathbb{1}_{\|\varphi(s)-g\| \leqslant \varepsilon}$. When $\varepsilon \to 0$, the probability of reaching the reward with a stochastic policy goes to 0, and for any stochastic policy, the value function $V^\pi_\varepsilon(s, g)$ converges to 0 as well. To avoid this vanishing issue, we need a scaling factor, and consider the reward $\frac{1}{\lambda(\varepsilon)} R_\varepsilon(s, g)$, with $\lambda(\varepsilon)$ the volume of the ball of size ε in goal space. When $\varepsilon \to 0$, this rescaled reward *converges* to the *Dirac reward*:

$$R(s, \mathrm{d}g) := \delta_{\varphi(s)}(\mathrm{d}g), \tag{13.2.1}$$

where δ_x is the Dirac measure at x. Intuitively, the Dirac reward $R(s, \mathrm{d}g)$ is infinite if the goal is reached ($\varphi(s) = g$) and 0 elsewhere. Formally, the reward is not a function but a *measure* on the goal space $\mathcal{G}$ parametrized by the state s. This is formalized in the following proposition. This result well-known, but still formalize it in this text as we will extend this result on *limit reward* when $\varepsilon \to 0$ to a *limit return*, or a *limit value function*.

Proposition 13.1. *Consider for every s the reward* measure *over $\mathcal{G}$:*

$$\frac{1}{\lambda(\varepsilon)} R_\varepsilon(s, g)\lambda(\mathrm{d}g) = \frac{1}{\lambda(\varepsilon)} \mathbb{1}_{\|\varphi(s)-g\| \leqslant \varepsilon} \lambda(\mathrm{d}g). \tag{13.2.2}$$

where $\lambda(\varepsilon)$ is the volume of the ball of radius ε for the Lebesgue measure $\lambda(B(., \varepsilon))$. Then, the measure $\frac{1}{\lambda(\varepsilon)} R_\varepsilon(s, g)\lambda(\mathrm{d}g)$ converges weakly to $\delta_{\varphi(s)}(\mathrm{d}g)$ when $\varepsilon \to 0$.

Proof. Let $f(g)$ be a 1-Lipschitz test function. We have:

$$\begin{aligned}
\left| \int_{g \in \mathcal{G}} f(g) \left(\frac{1}{\lambda(\varepsilon)} R_\varepsilon(s, g)\lambda(\mathrm{d}g) - \delta_{\varphi(s)}(\mathrm{d}g) \right) \right| &= \left| f(\varphi(s)) - \int_{g \in \mathcal{G}} f(g) \frac{1}{\lambda(\varepsilon)} R_\varepsilon(s, g)\lambda(\mathrm{d}g) \right| \\
&\leqslant \int_{g \in B(\varphi(s), \varepsilon)} |f(\varphi(s)) - f(g)| \frac{1}{\lambda(\varepsilon)} \lambda(\mathrm{d}g) \\
&\leqslant \int_{g \in B(\varphi(s), \varepsilon)} \varepsilon \frac{1}{\lambda(\varepsilon)} \lambda(\mathrm{d}g) = \varepsilon
\end{aligned}$$

Hence:

$$\int_{g \in \mathcal{G}} f(g) \frac{1}{\lambda(\varepsilon)} R_\varepsilon(s, g)\lambda(\mathrm{d}g) \to_{\varepsilon \to 0} \int_{g \in \mathcal{G}} f(g) \delta_{\varphi(s)}(\mathrm{d}g) \tag{13.2.3}$$

However, even after such a scaling, the UVFA update still vanishes with high probability for small ε (this just scales things by $1/\lambda(\varepsilon)$). We will build algorithms that work directly in the limit

$\varepsilon = 0$: replacing the sparse reward $R_\varepsilon(s, g)$ by the *infinitely sparse* reward $R(s, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g)$ will allow us to leverage the Dirac structure to remove the vanishing rewards issue.

Interestingly, the infinitely sparse reward is independent of the goal space norm, which might not be correlated to the real task we want to solve. For instance, if a wall is between the agent and its target goal, the distance might be below ε in the goal space $\mathcal{G}$, while the task is not solved. This was already pointed out and solved via learned distances in goal space in Venkattaramanujam et al. (2019).

13.2.1 Motivation: Computing the exact contribution of sparse rewards.

In this section, we explain how to leverage the multi-goal sparse reward structure, via the infinitely sparse reward setting. The key idea is that, with $\varepsilon = 0$, the contribution of the reward term in the Bellman equation can be computed exactly in expectation. Infinitely sparse rewards can be treated algebraically.

The following derivation is informal, but gives an intuition on our methods. A formal approach requires a proper definition of value functions and Q-functions with infinitely sparse rewards, as detailed in Chapters 15 and 16.

Let us start with the expectation of the UVFA update (13.1.2) with $\varepsilon > 0$ and rewards rescaled by $1/\lambda(\varepsilon)$:

$$\begin{aligned}
\delta\theta_{\mathrm{UVFA}} &= \mathbb{E}_{s,a,s',g}\left[\widehat{\delta\theta}_{\mathrm{UVFA}}(s, a, s', g)\right] \\
&= -\frac{1}{2}\partial_\theta \mathbb{E}_{s,a,g,s'}\left[\left(Q_\theta(s, a, g) - \frac{1}{\lambda(\varepsilon)}R_\varepsilon(s, g) - \gamma \max_{a'} Q_{\mathrm{tar}}(s', a', g)\right)^2\right] \\
&= \mathbb{E}_{s,a,g}\left[\partial_\theta Q_\theta(s, a, g)\frac{1}{\lambda(\varepsilon)}R_\varepsilon(s, g)\right] - \mathbb{E}_{s,a,g,s'}\left[\partial_\theta Q_\theta(s, a, g)\left(Q_\theta(s, a, g) - \gamma \max_{a'} Q_{\mathrm{tar}}(s', a', g)\right)\right].
\end{aligned}$$

This update will not be efficient for small ε, because $R_\varepsilon(s, g)$ is 0 most of the time, hence have high variance. While the expectation of the first term $\mathbb{E}_{s,a,g}\left[\partial_\theta Q_\theta(s, a, g)\frac{1}{\lambda(\varepsilon)}R_\varepsilon(s, g)\right]$ is nonzero because of the scaling factor $1/\lambda(\varepsilon)$, the reward is almost never observed. The agent will almost never reach its goal, but once in a while, with low probability, observe a huge reward. This is a variance issue caused by the vanishing rewards

But when $\varepsilon \to 0$, the rescaled reward $\frac{1}{\lambda(\varepsilon)}R_\varepsilon(s, g)$ converges to the Dirac reward $\delta_{\varphi(s)}$ (Proposition 13.1). Therefore, we can rewrite this first term as

$$\begin{aligned}
\frac{1}{\lambda(\varepsilon)}\mathbb{E}_{s,a,g}\left[\partial_\theta Q_\theta(s, a, g)R_\varepsilon(s, g)\right] &\to_{\varepsilon\to 0} \mathbb{E}_{s,a,g}\left[\partial_\theta Q_\theta(s, a, g)\delta_{\varphi(s)}(\mathrm{d}g)\right] \\
&= \mathbb{E}_{s,a}\left[\partial_\theta Q_\theta(s, a, \varphi(s))\right].
\end{aligned}$$

In this expression, sparse reward issues are avoided, just by taking the goal $g = \varphi(s)$ associated with the currently visited state s. Instead of waiting to reach a goal to update the Q-function, this updates the Q-function for the currently realized goal. The resulting algorithm, δ-DQN, is described in Chapter 15. The proper mathematical treatment A similar treatment holds for actor-critic methods (Chapter 16). (Sections 16.1–16.2).

The two following sections are technical results for multi-goal RL with infinitely sparse rewards. In Section 13.2.2, we define the goal-conditioned successor states, which corresponds to the successor states operator (as studied in Part IV) when the agent follow goal-conditionned policy $\pi(.|., g)$ for a fixed goal g. Then, we define the successor goals measure $M^\pi(s, g, \mathrm{d}g')$, which measures the expected discounted time spent in $\mathrm{d}g'$ when starting from s and following the policy $\pi(.|., g)$. Then, in Section 13.3, we consider the *continuous density assumption*: informally, it assumes that every probability measure $P, \rho, ...$ are continuous with respect to Lebesgue measure (see Assumption 13.1). Under this assumption, we define the multi-goal value measure in Theorem 13.5, and show in Theorems 13.7, 13.8 and 13.9 that the best policies

for infinitely sparse Dirac rewards corresponds to the asymptotically best policies for sparse reward R_ε when $\varepsilon \to 0$. These results confirms that, even for finding an efficient policy in a multi-goal environment with a sparse (but not infinitely) reward R_ε, finding an efficient policy for infinitely sparse rewards is a coherent strategy.

13.2.2 Goal-conditioned successor states, and successor goals

We now define the goal-conditioned successor states, and the successor goals measure $M^\pi(s, g, \mathrm{d}g')$. These objects can be easily defined via the successor states approach defined in Chapter 6, applied to the augmented multi-goal MDP defined in Chapter 13.

Definition-Theorem 13.2. *We define the goal-conditioned successor measure as:*

$$\nu^\pi(\mathrm{d}s|s_0, g) := (1-\gamma)\psi_* \tilde{M}^\pi(s_0, g, ., .) \tag{13.2.4}$$

where $\psi(s, g) \mapsto s$, and ψ_* is the push-forward measure. The goal-conditioned successor measure $\nu^\pi(\mathrm{d}s|s_0, g)$ is the expected discounted time spent in $\mathrm{d}s$ when starting from s_0 and following the policy conditioned by g: $\pi(.|., g)$.

The goal-conditioned successor measure is a well defined probability measure over $\mathcal{S}$ for every s_0, g. We have:

$$\nu^\pi(\mathrm{d}s|s_0, g) = (1-\gamma)\sum_{t\geqslant 0}\gamma^t (P^\pi)^t(\mathrm{d}s|s_0, g) \tag{13.2.5}$$

It satisfies the fixed-point Bellman equation:

$$\nu^\pi(\mathrm{d}s|s_0, g) = (1-\gamma)\delta_{s_0}(\mathrm{d}s) + \mathbb{E}_{a\sim\pi(\mathrm{d}a|s_0,g), s_1\sim P(\mathrm{d}s_1|s_0,a)}\left[\nu^\pi(\mathrm{d}s|s_0, g)\right] \tag{13.2.6}$$

Proof. Consider the successor state operator $\tilde{M}^\pi(s_1, g_1, \mathrm{d}s_2, \mathrm{d}g_2)$ defined on the goal-augmented state space defined in Section 13.1.2. We have:

$$\nu^\pi(\mathrm{d}s|s_0, g) = (1-\gamma)\sum_{t\geqslant 0}\gamma^t \psi_* \tilde{P}^t(\mathrm{d}s, \mathrm{d}g'|s_0, g) \tag{13.2.7}$$

and we know that $\tilde{P}(\mathrm{d}s, \mathrm{d}g'|s_0, g) = P^\pi(\mathrm{d}s|s_0, g)\delta_g(\mathrm{d}g')$, hence $\tilde{P}^t(\mathrm{d}s, \mathrm{d}g'|s_0, g) = (P^\pi)^t(\mathrm{d}s|s_0, g)\delta_g(\mathrm{d}g')$, and:

$$\nu^\pi(\mathrm{d}s|s_0, g) = (1-\gamma)\sum_{t\geqslant 0}\gamma^t \psi_*\left((P^\pi)^t(\mathrm{d}s|s_0, g)\delta_g(\mathrm{d}g')\right) = (1-\gamma)\sum_{t\geqslant 0}\gamma^t (P^\pi)^t(\mathrm{d}s|s_0, g) \tag{13.2.8}$$

We know that the successor states operator is a measure of total mass $\frac{1}{1-\gamma}$, hence with the $(1-\gamma)$ rescaling, the goal-contitioned successor measure is a probability density.

The Bellman equation is a consequence of Proposition 7.10

We can now define the *successor-goal measure.*

Definition-Theorem 13.3. *The* successor-goal measure *is defined as:*

$$M^\pi(s, g, .) := \frac{1}{1-\gamma}\varphi_*\nu^\pi(.|s, g) \tag{13.2.9}$$

We define the Bellman operator mapping $M(s, g_1, \mathrm{d}g_2)$ to $T_\pi \cdot M$ with

$$(T_\pi \cdot M)(s, g_1, \mathrm{d}g_2) = \delta_\varphi(s)(\mathrm{d}g_2) + \gamma\mathbb{E}_{a\sim\pi(a|s,g_1), s'\sim P(\mathrm{d}s'|s,a)}\left[M(s', g_1, \mathrm{d}g_2)\right], \tag{13.2.10}$$

Then, M^π is a fixed point of T^π.

Proof. This result is a direct consequence of Proposition 7.10

13.3 The continuous density setting

We will now consider two questions:

1. How can we define the *multi-goal* value function with infinitely sparse rewards?
2. We replaced *sparse reward* R_ε (with *small* ε) by infinitely sparse rewards. But are the best policies the same for the two problems, for a small ε?

To tackle these questions, we will consider the *continuous density assumption*: informally, it assumes that every probability measure $P, \rho, ...$ are continuous with respect to Lebesgue measure (see Assumption 13.1). Under this assumption, we can define the multi-goal value measure in Theorem 13.5. Then, we can prove in Theorems 13.7, 13.8 and 13.9 the equivalence between environments with the infinitely sparse Dirac reward, and the sparse rewards R_ε when $\varepsilon \to 0$.

We first introduce and discuss the continuous density assumption in Section 13.3.1. Then in Section 13.3.2, we define the multi-goal value *measure* under the continuous density assumption. Finally, in Section 13.3.3 we study the equivalence between environments. sparse rewards R_ε when $\varepsilon \to 0$ infinitely sparse rewards. All the proofs are in Section 13.3.4.

13.3.1 The continuous density assumption

Here, we introduce the continuity assumption, which will be used in this section, to formalize the relation between the multi-goal formulation with infinitely sparse Dirac rewards with the standard formulation with reward located in a neighborhood of size ε around the goal, and to derive a policy gradient theorem.

Assumption 13.1. We assume that $\mathcal{S}$ and $\mathcal{G}$ are finite dimensional vector spaces, and that $\mathcal{A}$ is a compact subset of a finite dimensional vector space. Moreover, $\rho_{\mathcal{G}}(\mathrm{d}g)$ is absolutely continuous with respect to the Lebesgue measure on $\mathcal{G}$, and we write $p_{\mathcal{G}}$ its density: $p_{\mathcal{G}}(g)\lambda(\mathrm{d}g)$, where $p_{\mathcal{G}}$ is a continuous function. Similarly, $\rho(\mathrm{d}s_0|g)$ the distribution of initial states given a goal is supposed to be absolutely continuous with respect the Lebesgue measure: $\rho(\mathrm{d}s_0|g) = p_0(s_0|g)\lambda(\mathrm{d}g)$, with p_0 continuous. The transition probability measure $P(\mathrm{d}s'|s,a)$ is absolutely continuous with respect to the Lebesgue measure on $\mathcal{S}$, and we write $p(s'|s,a)$ its density, which is continuous.

We assume that $\operatorname{supp}\rho_{\mathcal{G}}$ is compact and that there is a compact subset $K_{\mathcal{S}} \subset \mathcal{S}$ such that for every $s, a \in \mathcal{S}, \mathcal{G}$, $\operatorname{supp} P(\mathrm{d}s'|s,a) \subset K_{\mathcal{S}}$.

We consider only policies in Π, the set of policies π such that $\pi(a|s,g)$ is a continuous function of a, s, g.

We assume $\dim \mathcal{G} \leqslant \mathcal{S}$ and φ is a surjective linear function, and $\varphi(\mathcal{S}) = \mathcal{G}$.

Let us comment Assumption 13.1. First, we require P, $\rho_{\mathcal{G}}$, and ρ_0 to be absolutely continuous with respect to Lebesgue measure. This is typically true in environments such that, at every step, the environment adds a noise absolutely continuous with respect to Lebesgue measure (for instance Gaussian) to the position. On the contrary, in environments such that the agent lies in a submanifold of dimension lower than $\dim \mathcal{S}$, the assumption is not satisfied. The assumption that φ is linear is often satisfied in practice, when the achieved goal of a state corresponds to a some coordinates of s. For instance, in FetchReach, the state s contains information on the position and velocity of the robotic arm, while the achieved goal is the position of the extremity of the robotic arm. This assumption could be generalized to φ a submersion (a differentiable function such that its $\mathrm{d}\varphi_s$ is surjective for every s), but we used the linear assumption for the simplicity of the proof.

13.3.2 The Value Measure Under the Continuous Density Assumption

Under this assumption, we can show that ν^π and M^π introduced Section 13.2.2 can be decomposed in a Dirac part and a second part which is continuous with respect to Lebesgue measure:

Lemma 13.4. *Under Assumption 13.1, there is a function $q^\pi(s|s_0, g)$ such that for any (s_0, g):*

$$\nu^\pi(\mathrm{d}s|s_0, g) = (1-\gamma)\delta_{s_0}(\mathrm{d}s) + q^\pi(s|s_0, g)\lambda(\mathrm{d}s) \tag{13.3.1}$$

and $q^\pi(s|s_0, g)$ is a continuous function of s, s_0, g.

There is a function $\tilde{m}^\pi(s, g, g')$ such that for any s, g:

$$M^\pi(s, g, \mathrm{d}g') = \delta_{\varphi(s)} + \tilde{m}^\pi(s, g, g')\lambda(\mathrm{d}g'). \tag{13.3.2}$$

and $\tilde{m}^\pi(s, g, g')$ is a continuous function of (s, g, g').

The function $\tilde{m}^\pi$ satisfies for every $(s, g, g') \in K_\mathcal{S} \times \mathcal{G} \times \mathcal{G}$ the fixed point equation:

$$\tilde{m}(s, g, g') = \gamma \int_a \lambda(\mathrm{d}a)\pi(a|s, g)\left(\tilde{p}(g'|s, a) + \int_{s'} \lambda(\mathrm{d}s')p(s'|s, a)\tilde{m}^\pi(s', g, g')\right) \tag{13.3.3}$$

The proof of the Lemma is in Section 13.3.4.

We can now rigorously define the value measure $V^\pi(s, \mathrm{d}g)$ as follows.

Theorem 13.5. *Under Assumption 13.1, we can define the value-*measure $V^\pi(s, \mathrm{d}g)$ *as the measure on $\mathcal{G} \times \mathcal{G}$:*

$$V^\pi(s, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \tilde{m}(s, g, g)\lambda(\mathrm{d}g) \tag{13.3.4}$$

The value measure V^π satisfies the fixed point equation:

$$V^\pi(s, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma\mathbb{E}_{s'\sim P(\mathrm{d}s'|s,g)}\left[V^\pi(s', \mathrm{d}g)\right] \tag{13.3.5}$$

Finally, the value-measure is consistent with the value function $V^\varepsilon(s, g)$ when $\varepsilon \to 0$. Formally, the measure on $K_\mathcal{S} \times \mathcal{G}$: $\lambda(\mathrm{d}s)\frac{1}{\lambda(\varepsilon)}V^\pi_\varepsilon(s, g)\lambda(\mathrm{d}g)$ converges weakly to $\lambda(\mathrm{d}s)V^\pi(s, \mathrm{d}g)$ when $\varepsilon \to 0$.

The proof of the Theorem is in Section 13.3.4.

Even though we now have defined the multi-goal value measure together with its Bellman equation, learning directly $V^\pi(s, \mathrm{d}g)$ without bias poses technical issues. This is discussed in Section 16.1.2.

13.3.3 Equivalence Between $\varepsilon \to 0$ and the Dirac Setting under the continuous assumption

We now show the equivalence between environments with infinitely sparse rewards and with sparse rewards R_ε when $\varepsilon \to 0$. As replacing sparse rewards with infinitely sparse rewards is similar to reward shaping (Ng et al., 1999), ensuring that this changing this setting does not change the optimal policies in the environment. All the proofs are in Section 13.3.4.

Definition 13.6. We say that π_2 is *better* than π_1 with infinitely sparse rewards if the two measures $\lambda(\mathrm{d}s)V^{\pi_1}(s, \mathrm{d}g)$ and $\lambda(\mathrm{d}s)V^{\pi_2}(s, \mathrm{d}g)$ on $K_\mathcal{S} \times \mathcal{G}$ satisfy: $\lambda(\mathrm{d}s)V^{\pi_1}(s, \mathrm{d}g) \leqslant \lambda(\mathrm{d}s)V^{\pi_2}(s, .)$.

We say that π_2 is *asymptotically better* than π_1 when $\varepsilon \to 0$ if for all s, g,

$$\liminf_{\varepsilon\to 0} \frac{V^{\pi_2}_\varepsilon(s, g)}{V^{\pi_1}_\varepsilon(s, g)} \geqslant 1.$$

Theorem 13.7. *We assume Assumption 13.1 and take $\pi_1, \pi_2 \in \Pi$.*

Then, π_2 is better than π_1 with infinitely sparse rewards if and only if π_2 is asymptotically better *than π_1 when $\varepsilon \to 0$. In particular, a policy π^* is an optimal policy with infinitely sparse rewards if and only if it is an optimal policy when $\varepsilon \to 0$.*

In the following statement, we introduce 3 definitions of expected return: the return $J(\pi)$ with infinitely sparse reward, the return $J_\varepsilon(\pi)$ with sparse reward R_ε, and the estimated return $J_n(\pi)$ with the value measure approximator v_n. Then, we show that these three definitions are consistent.

Theorem 13.8. *We define $J(\pi)$, the expected return with infinitely sparse rewards for the goal density $p_{\mathcal{G}}$, as:*

$$J(\pi) := \int_{s_0,g} \lambda(\mathrm{d}s_0) p_{\mathcal{G}}(g) p_0(s_0|g) V^{\pi}(s_0, \mathrm{d}g). \tag{13.3.6}$$

We consider the expected return for the reward R_ε and the goal distribution $\rho(\mathrm{d}g)$.

$$J_\varepsilon(\pi) = \mathbb{E}_{g\sim\rho(\mathrm{d}g), s_0, a_0, \ldots}\left[\sum_{t\geqslant 0} \gamma^t R_\varepsilon(s_t, g)\right] = \int_{g,s_0} p_{\mathcal{G}}(g)\lambda(\mathrm{d}s_0, \mathrm{d}g) p_0(s_0|s) V_\varepsilon(s_0, g) \tag{13.3.7}$$

Let $(\hat{v}_n(s,g))_{n\geqslant 0}$ be any sequence of densities on $\mathcal{S}\times\mathcal{G}$ such that the measure on $\mathcal{S}\times\mathcal{G}$: $\lambda(\mathrm{d}s)\hat{v}_n(s,g)\rho_{\mathcal{G}}(\mathrm{d}g)$ converges weakly to $\lambda(\mathrm{d}s)V^{\pi}(s,\mathrm{d}g)$. We define $\tilde{\rho}(\mathrm{d}g) := \frac{1}{c}p_{\mathcal{G}}^2(g)\lambda(\mathrm{d}g)$ with $c := \int_g p_{\mathcal{G}}^2(g)\lambda(\mathrm{d}g)$, and $J_n(\pi)$ the estimator of the average return for the goal distribution $\tilde{\rho}$ with estimator $\hat{v}_n$:

$$J_n(\pi) := \mathbb{E}_{g\sim\tilde{\rho}(\mathrm{d}g), s_0\sim p(s_0|g)}\left[\hat{v}_n(s_0, g)\right] \tag{13.3.8}$$

Then the two estimators J_n and J_ε converge to J:

$$\frac{1}{\lambda(\varepsilon)} J_\varepsilon(\pi) \to_{\varepsilon\to 0} J(\pi) \tag{13.3.9}$$

$$cJ_n(\pi) \to_{n\to\infty} J(\pi) \tag{13.3.10}$$

Finally, we show that the return $J(\pi)$ allow to *rank* policies according to $\prec$, and is a relevant objectif to optimize with an actor-critic method as in Chapter 16.

Proposition 13.9. *We assume Assumption 13.1. Moreover, we assume that $p_{\mathcal{G}}(g) > 0$ for every $g \in \varphi(K_{\mathcal{S}})$, and $p_0(s_0|g) > 0$ for every $(s_0, g) \in K_{\mathcal{S}} \times \mathcal{G}$.*

We consider the partial order $\prec$ defined as: $\pi_1 \prec \pi_2$ if π_2 is strictly better than π_1 with infinitely sparse rewards: $\lambda(\mathrm{d}s)V^{\pi_1}(s,\mathrm{d}g) \prec \lambda(\mathrm{d}s)V^{\pi_2}(s,\mathrm{d}g)$ on $K_{\mathcal{S}}\times\mathcal{G}$.

Then $\pi \mapsto J(\pi)$ is strictly increasing for $\prec$.

13.3.4 Proofs of Lemma 13.4 and Theorems 13.5, 13.7, 13.8, 13.9

Proof of Lemma 13.4

Proof. We have:

$$\nu^{\pi}(\mathrm{d}s|s_0, g) = (1-\gamma)\sum_{k\geqslant 0}\gamma^k (P^{\pi})^k(\mathrm{d}s|s_0, g) \tag{13.3.11}$$

We know that

$$(P^{\pi})(\mathrm{d}s'|s,g) = \lambda(\mathrm{d}s')\int_a \lambda(\mathrm{d}a)\pi(a|s,g)p(s'|s,a),$$

and by induction, for $k \geqslant 1$,

$$(P^{\pi})^k(\mathrm{d}s|s_0,g) = \lambda(\mathrm{d}s)\int_{a_0,\ldots,s_{k-1},a_{k-1}} \pi(a_0|s_0,g)\left(\prod_{i=1}^{k-1} p(s_i|s_{i-1},a_{i-1})\pi(a_i|s_i,g)\right)p(s|s_{k-1},a_{k-1}).$$

We define:

$$q^{\pi}(s|g,s_0) := (1-\gamma)\sum_{k\geqslant 1}\gamma^k \int_{a_0,\ldots,s_{k-1},a_{k-1}} \pi(a_0|s_0,g)\left(\prod_{i=1}^{k-1} p(s_i|s_{i-1},a_{i-1})\pi(a_i|s_i,g)\right)p(s|s_{k-1},a_{k-1}) \tag{13.3.12}$$

We now check that q^{π} is well-defined and continuous. For every $k \geqslant 1$, the function

$$(g, s_0, a_0, \ldots, s_{k-1}, a_{k-1}, s) \mapsto \pi(a_0|s_0,g)\left(\prod_{i=1}^{k-1} p(s_i|s_{i-1},a_{i-1})\pi(a_i|s_i,g)\right)p(s|s_{k-1},a_{k-1})$$

is continuous and the supports of π are p compact sets. Therefore, for every $k \geqslant 0$, the function

$$(g, s_0, s) \mapsto \int_{a_0,s_1,\ldots,s_{k-1},a_{k-1}} \pi(a_0|s_0,g)\left(\prod_{i=1}^{k-1} p(s_i|s_{i-1},a_{i-1})\pi(a_i|s_i,g)\right)p(s|s_{k-1},a_{k-1})$$

is well defined and continuous.

Moreover, for every $k \geqslant 0$, and (s, g):

$$\left|\gamma^k \int_{a_0,\dots,s_{k-1},a_{k-1}} \pi(a_0|s_0,g)\left(\prod_{i=1}^{k-1} p(s_i|s_{i-1},a_{i-1})\pi(a_i|s_i,g)\right) p(s|s_{k-1},a_{k-1})\right| \leqslant \tag{13.3.13}$$

$$\leqslant \gamma^k \int_{a_0,\dots,s_{k-1},a_{k-1}} \pi(a_0|s_0,g)\left(\prod_{i=1}^{k-1} p(s_i|s_{i-1},a_{i-1})\pi(a_i|s_i,g)\right) \|p\|_\infty \tag{13.3.14}$$

$$= \gamma^k \|p\|_\infty \tag{13.3.15}$$

and $\sum_{k\geqslant 0} \gamma^k \|p\|_\infty \leqslant \infty$. Therefore, $q^\pi(s|g, s_0)$ is a continuous function and we have:

$$\nu^\pi(\mathrm{d}s|s_0,g) = (1-\gamma)\delta_{s_0}(\mathrm{d}s) + q^\pi(s|s_0,g)\lambda(\mathrm{d}s). \tag{13.3.16}$$

Moreover, the support of ν^π is compact and for every $s_0 \in K_{\mathcal{S}}$, we have $\mathrm{supp}\,(\nu^\pi(.|s_0,g)) \subset K_{\mathcal{S}}$.

We now show the existence of $\tilde{m}^\pi$. We have:

$$M^\pi(s,g,\mathrm{d}g') = \frac{1}{1-\gamma}\big(\varphi_*\nu^\pi(\mathrm{d}s'|s,g)\big)(\mathrm{d}g') = \varphi_*\left(\left(\delta_s(\mathrm{d}s')\right)(\mathrm{d}g') + \frac{1}{1-\gamma}\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s')\right)(\mathrm{d}g')\right.$$

First, $\varphi_*(\delta_s) = \delta_{\varphi(s)}$. Then, we study the second part $\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s')\right)(\mathrm{d}g')$, and show that there is a continuous function $\tilde{m}(s,g,g')$ such that

$$\frac{1}{1-\gamma}\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s')\right)(\mathrm{d}g') = \tilde{m}(s,g,g')\lambda(\mathrm{d}g') \tag{13.3.17}$$

Let $f(g)$ be a continuous test function. We have:

$$\int_{g'\in\mathcal{G}} f(g')\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s)\right)(\mathrm{d}g') = \int_{s'} f(\varphi(s'))q^\pi(s'|s,g)\lambda(\mathrm{d}s') \tag{13.3.18}$$

We use the change of variable $s' = e + k$ with $k \in \mathrm{Ker}\,\varphi$ and $e \in \mathrm{Ker}\,\varphi^\perp$ and use that $\varphi(s') = \varphi(e)$, and $\varphi_{\mathrm{Ker}\,\varphi^\perp}$ the restriction of φ to $\mathrm{Ker}\,\varphi^\perp$ is invertible. In order to use continuity theorems on integrals, we want to restrict the integral domains to compact sets. We define the orthogonal projections of $K_{\mathcal{S}}$ on $\mathrm{Ker}\,\varphi$ and $\mathrm{Ker}\,\varphi^\perp$: $K = \mathrm{proj}_{\mathrm{Ker}\,\varphi}(\mathrm{K}_{\mathcal{S}})$ and $E = \mathrm{proj}_{\mathrm{Ker}\,\varphi^\perp}(\mathrm{K}_{\mathcal{S}})$. K and E are compact sets and $\mathrm{supp}\,(q^\pi(s'|s,g)) \subset \{e+k\ ,\ (k,e) \in K \times E\}$ for every $s \in K_{\mathcal{S}}$. We have:

$$\int_{g'\in\mathcal{G}} f(g')\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s')\right)(\mathrm{d}g') = \int_{e\in\mathrm{Ker}\,\varphi^\perp, k\in\mathrm{Ker}\,\varphi} f(\varphi(e+k))q^\pi(e+k|s,g)\lambda(\mathrm{d}e,\mathrm{d}k) \tag{13.3.19}$$

$$= \int_{e\in E, k\in K} f(\varphi(e+k))q^\pi(e+k|s,g)\lambda(\mathrm{d}e,\mathrm{d}k) \tag{13.3.20}$$

$$= \int_{e\in E} f(\varphi(e))\lambda(\mathrm{d}e)\int_{k\in K} q^\pi(e+k|s,g)\lambda(\mathrm{d}k) \tag{13.3.21}$$

where we can switch integrals because the sets are compact and the functions continuous. We use the change of variable: $g' = (\varphi_{|\,\mathrm{Ker}\,\varphi^\perp})^{-1}(e)$. For simplicity, we use the notation $\varphi^{-1} = (\varphi_{|\,\mathrm{Ker}\,\varphi^\perp})^{-1}$.

$$\int_{g'\in\mathcal{G}} f(g')\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s')\right)(\mathrm{d}g') = \int_{g'\in\mathcal{G}} f(g')\lambda(\mathrm{d}g')\left(\det(\varphi^{-1})\int_{k\in K} \mathbb{1}_E(\varphi^{-1}(g')q^\pi(\varphi^{-1}(g')+k|s,g)\lambda(\mathrm{d}k)\right) \tag{13.3.22}$$

$$= \int_{g'\in\mathcal{G}} f(g')\lambda(\mathrm{d}g')\left(\det(\varphi^{-1})\int_{k\in K} q^\pi(\varphi^{-1}(g')+k|s,g)\lambda(\mathrm{d}k)\right) \tag{13.3.23}$$

where the last line is obtained by using that $\mathbb{1}_E(s')q^\pi(s'|.,.) = q^\pi(s'|.,.)$ because $s' \notin E \Rightarrow q^\pi(s'|.,.) = 0$. We define $\tilde{m}^\pi(s,g,g') = \frac{1}{1-\gamma}\det(\varphi)^{-1}\int_{k\in K} q^\pi(\varphi^{-1}(g')+k|s,g)\lambda(\mathrm{d}k)$. The function $(s,g,k,g') \to q^\pi(\varphi^{-1}(g')+k|s,g)$ is continuous and K is compact. Therefore, $\tilde{m}^\pi$ is continuous and bounded, and:

$$\frac{1}{1-\gamma}\varphi_*\left(q^\pi(s'|s,g)\lambda(\mathrm{d}s')\right)(\mathrm{d}g') = \tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}g')$$

Moreover, the support of $\tilde{m}(s,g,g')\lambda(\mathrm{d}g')$ is compact and $\mathrm{supp}\,(\tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}g')) \subset \varphi(K_{\mathcal{S}})$.

We now prove the fixed point equation on $\tilde{m}^\pi$. We consider the Bellman equation on $M^\pi(s,g,\mathrm{d}g')$. We have:

$$M^\pi(s,g,\mathrm{d}g') = \delta_{\varphi(s)}(\mathrm{d}g') + \gamma\int_{s',a} \lambda(\mathrm{d}s',\mathrm{d}a)\pi(a|s,g)p(s'|s,a)M^\pi(s',g,\mathrm{d}g') \tag{13.3.24}$$

By using $M^\pi(s,g,\mathrm{d}g') = \delta_{\varphi(s)}(\mathrm{d}g') + \tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}g')$, we have:

$$\tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}g') = \gamma \int_{s',a} \lambda(\mathrm{d}s',\mathrm{d}a)\pi(a|s,g)p(s'|s,a)\left(\delta_{\varphi(s')}(\mathrm{d}g') + \tilde{m}(s,g,g')\lambda(\mathrm{d}g')\right) \tag{13.3.25}$$

Let $f(g')$ be a continuous test function, we have:

$$\int_{g'} f(g')\tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}g') = \tag{13.3.26}$$

$$= \gamma \int_{s',a,g'} \lambda(\mathrm{d}s',\mathrm{d}a)f(g')\pi(a|s,g)p(s'|s,a)\left(\delta_{\varphi(s')}(\mathrm{d}g') + \tilde{m}(s,g,g')\lambda(\mathrm{d}g')\right) \tag{13.3.27}$$

$$= \gamma \int_{s',a} \lambda(\mathrm{d}s',\mathrm{d}a)\pi(a|s,g)p(s'|s,a)\left(f(\varphi(s')) + \int_{g'} \lambda(\mathrm{d}g')f(g')\pi(a|s,g)p(s'|s,a)\tilde{m}(s',g,g')\right) \tag{13.3.28}$$

$$= \gamma \int_{a,g'} f(g')\pi(a|s,g)\tilde{p}(g'|s,a) + \gamma \int_{a,s',g'} \lambda(\mathrm{d}a,\mathrm{d}s',\mathrm{d}g')f(g')\pi(a|s,g)p(s'|s,a)\tilde{m}(s',g,g')f(g') \tag{13.3.29}$$

where $\tilde{p}(g|s,a)$ is the density with respect to Lebesgue measure $\lambda(\mathrm{d}g)$ of $\varphi_* P(\mathrm{d}s'|s,a)$. Formally, the existence proof of $\tilde{p}$ is the same than for $\tilde{m}$ in equation (13.3.17), and is using the fact that P is continuous with respect to $\lambda(\mathrm{d}s)$ and φ is a surjective linear operator. Therefore, we have, for λ-almost s,g,g':

$$\tilde{m}(s,g,g') = \gamma \int_a \lambda(\mathrm{d}a)\pi(a|s,g)\left(\tilde{p}(g'|s,a) + \gamma \int_{s'} \lambda(\mathrm{d}s')p(s'|s,a)\tilde{m}^\pi(s',g,g')\right) \tag{13.3.30}$$

Because $\tilde{m}^\pi$ is continuous, this relation is true for every s,g,g', in particular if $g = g'$.

Proof of Theorem 13.5

Proof. Let $f(g)$ be a continuous test function. We have:

$$\int_g V^\pi(s,\mathrm{d}g)f(g) = f(\varphi(s)) + \int_g \tilde{m}^\pi(s,g,g)f(g)\lambda(\mathrm{d}g) \tag{13.3.31}$$

Moreover, we know from Lemma 13.4 that

$$\tilde{m}^\pi(s,g,g) = \gamma \int_a \lambda(\mathrm{d}a)\pi(a|s,g)\left(\tilde{p}(g|s,a) + \int_{s'} \lambda(\mathrm{d}s')p(s'|s,a)\tilde{m}^\pi(s',g,g)\right) \tag{13.3.32}$$

Therefore:

$$\int_g V^\pi(s,\mathrm{d}g)f(g) = f(\varphi(s)) + \gamma \int_{g,a} \lambda(\mathrm{d}a,\mathrm{d}g)f(g)\pi(a|s,g)\left(\tilde{p}(g|s,a) + \int_{s'} \lambda(\mathrm{d}s')p(s'|s,a)\tilde{m}^\pi(s',g,g)\right) \tag{13.3.33}$$

On the other side, we have:

$$\int_g f(g)\mathbb{E}_{a\sim\pi(.|s,g),s'\sim P(\mathrm{d}s'|s,a)}\left[V^\pi(s',\mathrm{d}g)\right] = \tag{13.3.34}$$

$$= \int_{g,a,s'} \lambda(\mathrm{d}a,\mathrm{d}s')f(g)\pi(a|s,g)p(s'|s,a)\left(\delta_{\varphi(s')}(\mathrm{d}g) + \tilde{m}^\pi(s',g,g)\lambda(\mathrm{d}g)\right) \tag{13.3.35}$$

$$= \int_{a,s'} \lambda(\mathrm{d}a,\mathrm{d}s')\pi(a|s,g)p(s'|s,a)f(\varphi(s')) + \int_{a,s',g} \lambda(\mathrm{d}a,\mathrm{d}s',\mathrm{d}g)f(g)\pi(a|s,g)p(s'|s,a)\tilde{m}^\pi(s',g,g) \tag{13.3.36}$$

For the first part, we use the change of variable $g = \varphi(s')$, and we have:

$$\int_g f(g)\mathbb{E}_{a\sim\pi(.|s,g),s'\sim P(\mathrm{d}s'|s,a)}\left[V^\pi(s',\mathrm{d}g)\right] = \tag{13.3.37}$$

$$= \int_{a,g} \lambda(\mathrm{d}a,\mathrm{d}g)\pi(a|s,g)\tilde{p}(g|s,a)f(g) + \int_{a,s',g} \lambda(\mathrm{d}a,\mathrm{d}s',\mathrm{d}g)f(g)\pi(a|s,g)p(s'|s,a)\tilde{m}^\pi(s',g,g) \tag{13.3.38}$$

where $\tilde{p}(.|s,a)$ is the density of $\varphi_* P(.|s,a)$ (where φ_* is the push-forward operator) with respect to Lebesgue measure. Therefore, we have:

$$\int_g V(s,\mathrm{d}g)f(g) = \int_g f(g)\left(\delta_{\varphi(s)}(\mathrm{d}g) + \gamma\mathbb{E}_{s'\sim P^\pi(\mathrm{d}s'|s,g)}\left[V^\pi(s',\mathrm{d}g)\right]\right) \tag{13.3.39}$$

and we can conclude:

$$V^\pi(s,\mathrm{d}g)f(g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma\mathbb{E}_{s'\sim P^\pi(\mathrm{d}s'|s,g)}\left[V^\pi(s',\mathrm{d}g)\right] \tag{13.3.40}$$

We now show that know that the measure on $K_{\mathcal{S}} \times \mathcal{G}$: $\lambda(\mathrm{d}s)\frac{1}{\lambda(\varepsilon)}V_\varepsilon^\pi(s,g)\lambda(\mathrm{d}g)$ converges weakly to $\lambda(\mathrm{d}s)V^\pi(s,\mathrm{d}g)$ when $\varepsilon \to 0$. We know that:

$$V_\varepsilon^\pi(s,g) = \mathbb{E}\left[\sum_{k\geqslant 0}\gamma^k R_\varepsilon(s_k,g)|s_0 = s\right] \tag{13.3.41}$$

$$= \frac{1}{1-\gamma}\int_{s'\in\mathcal{S}} \nu^\pi(\mathrm{d}s'|s,g)R_\varepsilon(s',g) \tag{13.3.42}$$

$$= \int_{s'\in\mathcal{S}} \nu^\pi(\mathrm{d}s'|s,g)\mathbb{1}_{\|\varphi(s')-g\|\leqslant\varepsilon} \tag{13.3.43}$$

We use the change of variable $g' = \varphi(s')$, and we have (with φ_* the push-forward operator):

$$V_\varepsilon^\pi(s,g) = \int_{g'\in\mathcal{S}} (\varphi_*\nu^\pi)(\mathrm{d}g'|s,g)\mathbb{1}_{\|g'-g\|\leqslant\varepsilon} \tag{13.3.44}$$

$$= \int_{g'\in\mathcal{S}} M^\pi(s,g,\mathrm{d}g')\mathbb{1}_{\|g'-g\|\leqslant\varepsilon} \tag{13.3.45}$$

$$= M^\pi(s,g,B(g,\varepsilon)) \tag{13.3.46}$$

Let $F := \{g \in \mathcal{G}, \inf_{s\in K_{\mathcal{S}}} \|g-\varphi(s)\| < 1\}$. Therefore, for every $\varepsilon < 1$, the support of $\lambda(\mathrm{d}s)\frac{1}{\lambda(\varepsilon)}V_\varepsilon^\pi(s,g)\lambda(\mathrm{d}g)$ is compact and is a subset of F. Let $f(s,g)$ be a continuous bounded test function and $0 < \varepsilon < 1$. We have:

$$\int_{s\in K_{\mathcal{S}},g\in\mathcal{G}} f(s,g)\frac{1}{\lambda(\varepsilon)}V_\varepsilon^\pi(s,g)\lambda(\mathrm{d}s,\mathrm{d}g) = \int_{s\in K_{\mathcal{S}},g\in F} f(s,g)\frac{1}{\lambda(\varepsilon)}M^\pi(s,g,B(g,\varepsilon))\lambda(\mathrm{d}s,\mathrm{d}g) \tag{13.3.47}$$

We know that $M^\pi(s,g,\mathrm{d}g') = \delta_{\varphi(s)}(\mathrm{d}g') + \tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}g')$. Therefore, $M^\pi(s,g,B(g,\varepsilon)) = \mathbb{1}_{\|g-\varphi(s)\|\leqslant\varepsilon} + \int_{g'} \frac{\mathbb{1}_{\|g-g'\|\leqslant\varepsilon}}{\lambda(\varepsilon)}\tilde{m}^\pi(s,g,g')$, and we have:

$$\int_{s\in K_{\mathcal{S}},g\in\mathcal{G}} f(s,g)\frac{1}{\lambda(\varepsilon)}V_\varepsilon^\pi(s,g)\lambda(\mathrm{d}s,\mathrm{d}g) = \tag{13.3.48}$$

$$= \int_{s\in K_{\mathcal{S}},g\in F} \frac{\mathbb{1}_{\|g-\varphi(s)\|\leqslant\varepsilon}}{\lambda(\varepsilon)} f(s,g)\lambda(\mathrm{d}s,\mathrm{d}g) + \int_{s\in K_{\mathcal{S}},g\in F,g'\in\mathcal{G}} f(s,g)\frac{\mathbb{1}_{\|g-g'\|\leqslant\varepsilon}}{\lambda(\varepsilon)}\tilde{m}^\pi(s,g,g')\lambda(\mathrm{d}s,\mathrm{d}g,\mathrm{d}g') \tag{13.3.49}$$

$$= \int_u \left(\int_{s\in K_{\mathcal{S}}} \lambda(\mathrm{d}s)f(s,\varphi(s)+u) + \int_{s\in K_{\mathcal{S}},g\in\mathcal{G}} f(s,g)\tilde{m}^\pi(s,g,g+u)\lambda(\mathrm{d}s,\mathrm{d}g)\right) U_\varepsilon(\mathrm{d}u) \tag{13.3.50}$$

where $U_\varepsilon(\mathrm{d}u)$ is the uniform measure on $B(0,\varepsilon)$ the ball of size ε around 0: $U_\varepsilon(\mathrm{d}u) := \frac{\mathbb{1}_{\|u\|\leqslant\varepsilon}}{\lambda(\varepsilon)}\lambda(\mathrm{d}u)$. We can switch the order of integration because f, $\tilde{m}^\pi$ are continuous, bounded, and the integral is computed on compact sets. The function $u \to \int_{s\in K_{\mathcal{S}}} \lambda(\mathrm{d}s)f(s,\varphi(s)+u) + \int_{s\in K_{\mathcal{S}},g\in F} f(s,g)\tilde{m}(s,g,g+u)\lambda(\mathrm{d}s,\mathrm{d}g)$ is bounded and continuous. Since $U_\varepsilon(\mathrm{d}u)$ converges weakly to $\delta_0(\mathrm{d}u)$, we have:

$$\lim_{\varepsilon\to 0}\int_{s\in K_{\mathcal{S}},g\in\mathcal{G}} f(s,g)\frac{1}{\lambda(\varepsilon)}V_\varepsilon^\pi(s,g)\lambda(\mathrm{d}s,\mathrm{d}g) = \tag{13.3.51}$$

$$= \lim_{\varepsilon\to 0}\int_u \left(\int_{s\in K_{\mathcal{S}}} \lambda(\mathrm{d}s)f(s,\varphi(s)+u) + \int_{s\in K_{\mathcal{S}},g\in F} f(s,g)\tilde{m}^\pi(s,g,g+u)\lambda(\mathrm{d}s,\mathrm{d}g)\right) U_\varepsilon(\mathrm{d}u) \tag{13.3.52}$$

$$= \int_u \left(\int_{s\in K_{\mathcal{S}}} \lambda(\mathrm{d}s)f(s,\varphi(s)+u) + \int_{s\in K_{\mathcal{S}},g\in\mathcal{G}} f(s,g)\tilde{m}^\pi(s,g,g+u)\lambda(\mathrm{d}s\,\mathrm{d}g)\right)\delta_0(\mathrm{d}u) \tag{13.3.53}$$

$$= \int_{s\in K_{\mathcal{S}}} \lambda(\mathrm{d}s)f(s,\varphi(s)) + \int_{s\in K_{\mathcal{S}},g} f(s,g)\tilde{m}^\pi(s,g,g)\lambda(\mathrm{d}s,\mathrm{d}g) \tag{13.3.54}$$

$$= \int_{s\in K_{\mathcal{S}},g} f(s,g)V^\pi(s,\mathrm{d}g)\lambda(\mathrm{d}s) \tag{13.3.55}$$

This concludes the proof.

Proof of Theorem 13.7

Proof.

We know that $V^\pi(s,\mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \tilde{m}(s,g,g)^\pi\lambda(\mathrm{d}g)$. Moreover:

$$V_\varepsilon^\pi(s_0,g) = M(s_0,g,B(g,\varepsilon)) \tag{13.3.56}$$

$$= \mathbb{1}_{\varphi(s_0)=g} + \lambda(\varepsilon)\tilde{m}^\pi(s_0,g,g) + o(\lambda(\varepsilon)) \tag{13.3.57}$$

Therefore, for any policies $\pi_1, \pi_2 \in \Pi$:

$$\frac{V_\varepsilon^{\pi_2}(s,g)}{V_\varepsilon^{\pi_1}(s,g)} = \frac{\mathbb{1}_{\varphi(s)=g} + \tilde{m}^{\pi_2}(s,g,g)\lambda(\varepsilon) + o(\lambda(\varepsilon))}{\mathbb{1}_{\varphi(s)=g} + \tilde{m}^{\pi_1}(s,g,g)\lambda(\varepsilon) + o(\lambda(\varepsilon))} \tag{13.3.58}$$

$$= \mathbb{1}_{\varphi(s)=g} + \mathbb{1}_{\varphi(s)\neq g}\frac{\tilde{m}^{\pi_2}(s,g,g)}{\tilde{m}^{\pi_1}(s,g,g)} + o_{\varepsilon\to 0}(1) \tag{13.3.59}$$

Therefore, by definition, π_2 is asymptotically better than π_1 when $\varepsilon \to 0$ if and only if, for all $(s,g) \in \mathcal{S} \times \mathcal{G}$:

$$\mathbb{1}_{\varphi(s)=g} + \mathbb{1}_{\varphi(s)\neq g}\frac{\tilde{m}^{\pi_2}(s,g,g)}{\tilde{m}^{\pi_1}(s,g,g)} \geqslant 1 \tag{13.3.60}$$

If $\varphi(s) \neq g$, this inequality is equivalent to $\tilde{m}^{\pi_2}(s,g,g) \geqslant m^{\pi_1}(s,g,g)$. Because $\tilde{m}^{\pi_1}$ and $\tilde{m}^{\pi_2}$ are continuous, $\tilde{m}^{\pi_2}(s,g,g) \geqslant m^{\pi_1}(s,g,g)$ for all $\varphi(s) \neq g$ is equivalent to $\tilde{m}^{\pi_2}(s,g,g) \geqslant m^{\pi_1}(s,g,g)$ for every (s,g). Therefore, π_2 is asymptotically better than π_1 when $\varepsilon \to 0$ if and only if, for all (s,g), $m^{\pi_2}(s,g,g) \geqslant m^{\pi_1}(s,g,g)$.

On the other side π_2 is better than π_1 with infinitely sparse rewards if and only if:

$$\lambda(\mathrm{d}s)V^{\pi_1}(s,\mathrm{d}g) \preceq \lambda(\mathrm{d}s)V^{\pi_2}(s,\mathrm{d}g) \tag{13.3.61}$$

$$\Leftrightarrow \quad \lambda(\mathrm{d}s)\delta_{\varphi(s)}(\mathrm{d}g) + m_{\pi_1}(s,g,g)\lambda(\mathrm{d}s,\mathrm{d}g) \preceq \lambda(\mathrm{d}s)\delta_{\varphi(s)}(\mathrm{d}g) + m_{\pi_2}(s,g,g)\lambda(\mathrm{d}s,\mathrm{d}g) \tag{13.3.62}$$

$$\Leftrightarrow \quad m_{\pi_1}(s,g,g)\lambda(\mathrm{d}s,\mathrm{d}g) \leqslant m_{\pi_2}(s,g,g)\lambda(\mathrm{d}s,\mathrm{d}g) \tag{13.3.63}$$

Therefore, for λ-almost every (s,g), $m_{\pi_1}(s,g,g) \leqslant m_{\pi_2}(s,g,g)$. Therefore: π_2 is better than π_1 with infinitely sparse rewards if and only if $\tilde{m}^{\pi_2}(s,g,g) \geqslant m^{\pi_1}(s,g,g)$ for λ-almost every s,g. This concludes the proof.

Proof of Theorem 13.8

Proof. We have:

$$J_\varepsilon(\pi) = \int_{s_0,g} V_\varepsilon(s_0,g)p_{\mathcal{G}}(g)p_0(s_0|g)\lambda(\mathrm{d}s_0,\mathrm{d}g) \tag{13.3.64}$$

$$J_n(\pi) = \int_{s_0,g} \hat{v}_n(s_0,g)\frac{1}{c}p_{\mathcal{G}}^2(g)p_0(s_0|g)\lambda(\mathrm{d}s_0,\mathrm{d}g) \tag{13.3.65}$$

and whe know from Theorem 13.5 that $\frac{V_\varepsilon(s,g)}{\lambda(\varepsilon)}\lambda(\mathrm{d}s,\mathrm{d}g)$ and $\hat{v}_n(s,g)p_{\mathcal{G}}(\mathrm{d}g)\lambda(\mathrm{d}s,\mathrm{d}g)$ converge weakly to $\lambda(\mathrm{d}s)V^\pi(s,\mathrm{d}g)$ on $K_{\mathcal{S}} \times \mathcal{G}$ when $\varepsilon \to 0$ and $n \to \infty$. Therefore, because p_0 and $p_{\mathcal{G}}$ are continuous bounded functions,

$$\lim_{\varepsilon\to 0}\frac{1}{\lambda(\varepsilon)}J_\varepsilon(\pi) = \int_{s_0,g} V^\pi(s_0,\mathrm{d}g)p_{\mathcal{G}}(g)p_0(s_0|g)\lambda(\mathrm{d}s_0) = J(\pi) \tag{13.3.66}$$

Similarly:

$$\lim_{n\to\infty} J_n(\pi) = \int_{s_0,g} V^\pi(s,\mathrm{d}g)\frac{1}{c}p_{\mathcal{G}}(g)p_0(s_0|g)\lambda(\mathrm{d}s_0) \tag{13.3.67}$$

$$= \frac{1}{c}J(\pi) \tag{13.3.68}$$

Proof of Proposition 13.9

Proof. The function $J(\pi)$ is clearly non-decreasing, and we have to check that we cannot have $\pi_1 \prec \pi_2$ with $J(\pi_1) = J(\pi_2)$. Let $\pi_1, \pi_2 \in \Pi$ such that $\pi_1 \prec \pi_2$. Therefore, there is $U \subset K_{\mathcal{S}} \times \mathcal{G}$ such that $(\lambda \otimes V^{\pi_2}(.,.))(U) > (\lambda \otimes V^{\pi_1}(.,.))(U)$. Moreover, because $\mathrm{supp}\,(\lambda(\mathrm{d}s)V^\pi(s,\mathrm{d}g)) \subset K_{\mathcal{S}} \times \varphi(K_{\mathcal{S}})$, therefore we can suppose $U \subset K_{\mathcal{S}} \times \varphi(K_{\mathcal{S}})$, and we have:

$$\int_{(s,g)\in U} \lambda(\mathrm{d}s,\mathrm{d}g)(\tilde{m}^{\pi_2}(s,g,g) - \tilde{m}^{\pi_1}(s,g,g)) > 0 \tag{13.3.69}$$

We already know that $\tilde{m}^{\pi_2}(s,g,g) - \tilde{m}^{\pi_1}(s,g,g) \geqslant 0$ for almost every s,g (see proof of Theorem 13.7). Therefore, there is $\varepsilon' > 0$ and $V \subset U$ with $\mathrm{d}\lambda(V) > 0$ such that for every $s,g \in V$, $\tilde{m}^{\pi_2}(s,g,g) - \tilde{m}^{\pi_1}(s,g,g) > \varepsilon'$.

We have:

$$J(\pi_2) - J(\pi_1) = \int_{s \in K_{\mathcal{S}}, g \in \mathcal{G}} p_0(s|g) p_{\mathcal{G}}(g) (V^{\pi_2}(s, \mathrm{d}g) - V^{\pi_1}(s, \mathrm{d}g) \tag{13.3.70}$$

$$\geqslant \int_{(s,g) \in V} p_0(s|g) p_{\mathcal{G}}(g) \left(\tilde{m}^{\pi_2}(s, g, g) - \tilde{m}^{\pi_1}(s, g, g)\right) \tag{13.3.71}$$

$$\geqslant \varepsilon' \int_{(s,g) \in V} p_0(s|g) p_{\mathcal{G}}(g) \tag{13.3.72}$$

$$> 0 \tag{13.3.73}$$

because $p_{\mathcal{G}}(g) > 0$ for λ-almost every g in $\varphi(K_{\mathcal{S}})$, and $p_0(s|g) > 0$ for λ-almost every s, g in $K_{\mathcal{S}} \times \mathcal{G}$. This concludes the proof.

Chapter 14

A study of Hindsight Experience Replay's bias

Hindsight Experience Replay (HER) is known to be biased in a general setting (Manela and Biess, 2021; Lanka and Wu, 2018; Plappert et al., 2018), and this bias corresponds to a well-known psychological bias (Fischhoff, 1975). In this chapter, we study HER's bias. In Section 14.2 we formally prove that HER is biased. While it is a known fact, there were no formal proof of it to our knowledge. In Section 14.1, we show that HER is actually *unbiased* in deterministic environments. This result vindicates HER for deterministic environments: HER leverages the structure of multi-goal environments, is not vanishing when the rewards are sparse, and is mathematically well-grounded.

14.1 HER is unbiased in deterministic environments.

Despite its bias, HER is efficient in practice, especially in continuous control environments. We vindicate HER theoretically by showing that HER is unbiased in *deterministic* environments. We say that an environment is *deterministic* is the next state s_{t+1} is uniquely determined by the current state s_t and an action a_t. This covers many usual environments such as robotic environments.

In order to define HER, we assume access to samples of trajectories $(g, s_0, a_0, s_1, a_1, ...) \sim \rho(g, s_0, a_0, s_1, a_1, ...)$ with $g \sim \rho_{\mathcal{G}}(\mathrm{d}g)$, $s_0 \sim \rho_0(\mathrm{d}s_0|g)$, and for every $k \geqslant 0$, $a_k \sim \pi_{\text{expl}}(a|s_k, g)$ where π_{expl} is an exploration policy, $s_{k+1} \sim P(\mathrm{d}s|s_k, a_k)$. For simplicity, we will assume the trajectories are infinite.

Here we consider HER with the `future` strategy described in the original paper: goals are re-sampled from a trajectory as goal reached later in the trajectory. We formalize HER as follows: we sample a trajectory $\tau = (g, s_0, a_0, s_1, a_1, ...) \sim \rho(g, s_0, a_0, s_1, a_1, ...)$, a Bernoulli variable $U \sim \mathcal{B}(\alpha)$, and two independent integer random variables K, L, from distributions p_K and p_L, such that for every k, l, $p_K(k) > 0$ and $p_L(l) > 0$. The bernoulli variable U represents the random choice of using the standard Q-learning update, or the HER update with a resampled goal. The random variable K represents the timestep of the transition we will use for the Q-learning update, and L represents the timestep used to sample a new goal g' for the `future` sampling strategy. Then, the update $\widehat{\delta\theta_{\text{HER}}}(\tau, U, K, L)$ is defined as:

- If $U = 0$:

$$\widehat{\delta\theta_{\text{HER}}}(\tau, U = 0, K, L) := \partial_\theta \frac{1}{2}(Q_\theta(s_K, a_K, g) - R(s_K, g) - \gamma \sup_{a'} Q(s_{K+1}, a', g))^2,$$

  which corresponds to the usual Q-learning update as defined in UVFA (Schaul et al., 2015).

- If $U = 1$ we set $g' = \varphi(s_{K+L+1})$ and:

$$\widehat{\delta\theta_{\text{HER}}}(\tau, U = 1, K, L) := \partial_\theta \frac{1}{2}(Q_\theta(s_K, a_K, g') - R(s_K, g') - \gamma \sup_{a'} Q(s_K, a', g'))^2,$$

  which corresponds to a Q-learning update for a re-sampled goal $g' = \varphi(s_{K+L})$, a goal achieved later in the trajectory.

We say that environment is a *continuous deterministic environment* if there is a continuous function $\psi : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ such that for every $(s, a) \in \mathcal{S} \times \mathcal{A}$, $P(\mathrm{d}s'|s, a) = \delta_{\psi(s,a)}(\mathrm{d}s')$. In particular, any discrete deterministic environment is a continuous deterministic environment for the discrete topology. Therefore, the following theorem can be applied to discrete environments.

Theorem 14.1 (HER is unbiased in deterministic environments)**.** *We assume the environment is a continuous deterministic environment. We also assume that for every pair of states (s, s'), s' is reachable from s, which means there is a sequence of actions $(a_1, ..., a_k)$ such that applying these actions from s leads to s'. Finally, we assume that the support of the exploration policy $\pi_{\text{expl}}(a|s, g)$ is the entire action space $\mathcal{A}$ for every s, g.*

Then, there is an euclidean norm $\|.\|$ such that, for every θ, the HER update with the `future` *sampling strategy at θ, $\widehat{\delta\theta_{HER}}$ is an unbiased estimate of the gradient step between Q_θ and the target function $Q_{target} := T_{\max} Q_\theta$:*

$$\mathbb{E}\left[\widehat{\delta\theta_{HER}}\right] = \partial_\theta \frac{1}{2}\|Q_\theta - Q^{\text{tar}}\|^2 \tag{14.1.1}$$

If the state space $\mathcal{S}$ is finite, HER has a single fixed point Q_∞, which is equal to Q^*.

The euclidean norm $\|.\|$ in the theorem will depend on the exploration policy $\pi_{\text{expl}}(a|s, g)$. Therefore, if the exploration policy is changing during learning, the norm will will be changing as well.

Proof. The principle of the proof is the following. We study the sampling distribution of transitions $\mu_{\text{HER}}(s, a, s', g)$ with HER. The bias of HER comes from the fact that the sampling of goals g with $\mu_{\text{HER}}(s, a, s', g)$ is not independent of s' knowing (s, a). On the contrary, in deterministic environments, the distribution of g knowing (s, a) is independent of s' because s' is uniquely determined by (s, a).

We study the sampling distribution of transitions (s, a, s', g) used in HER. Formally, we sample a transition (s, a, s', g) by sampling τ, U, K, L and defining $(s, a, s', g') := \Phi(\tau, U, K, L)$ as:

- If $U = 0$, $\Phi(\tau, U = 1, K, L) = (s_k, a_k, s_{k+1}, g)$
- If $U = 1$, $\Phi(\tau, U = 1, K, L) = (s_k, a_k, s_{k+1}, \varphi(s_{K+L}))$

Then, HER update can be equivalently defined as: sample (τ, U, K, L) as described above, define $(s, a, s', g) = \Phi(\tau, U, K, L)$, and:

$$\widehat{\delta\theta_{\text{HER}}}(s, a, s', g) := \partial_\theta \frac{1}{2}(Q_\theta(s, a, s', g) - R(s, g) - \gamma \sup_{a'} Q(s', a', g))^2 \tag{14.1.2}$$

Therefore:

$$\mathbb{E}\left[\widehat{\delta\theta_{\text{HER}}}\right] = \partial_\theta \mathbb{E}_{(s,a,s',g)\sim\mu_{\text{HER}}} \frac{1}{2}(Q_\theta(s, a, g) - R(s, g) - \gamma \sup_{a'} Q(s', a', g))^2 \tag{14.1.3}$$

where we define μ_{HER} to be the distribution of (s, a, s', g) given by the distribution of $\Phi_*(\rho \otimes p_U \otimes p_L \otimes p_K)$, where Φ_* is the *push-forward* operator on measures. We now compute μ_{HER}. Let $f : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ be a test function, we have:

$$\mathbb{E}_{s,a,s',g\sim\mu_{\text{HER}}}\left[f(s, a, s', g)\right] = \mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau, U, K, L))\right] \tag{14.1.4}$$

$$\begin{aligned} &= (1-\alpha)\mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau, U, K, L))|U = 0\right] \\ &\quad + \alpha\mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau, U, K, L))|U = 1\right] \end{aligned} \tag{14.1.5}$$

Moreover:

$$\mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau,U,K,L))|U=0\right] = \sum_K p_K(k)\int_{g,s_0,a_0,\dots}\rho(g,s_0,a_0,\dots)f(s_k,a_k,s_{k+1},g) \tag{14.1.6}$$

$$= \sum_k p_K(k)\int_{g,s_0,a_0,\dots}\rho_{\mathcal{G}}(g)\rho_0(s_0|g)(P^{\pi_{\text{exp}}})^k(s|s_0,g)\pi_{\text{expl}}(a|s,g)P(s'|s,a)f(s,a,s',g) \tag{14.1.7}$$

$$= \int_{s,a,s',g} f(s,a,s',g)\left(\rho_{\mathcal{G}}(g)\int_{s_0}\sum_k p_K(k)\rho_0(s_0|g)(P^{\pi_{\text{exp}}})^k(s|s_0,g)\pi_{\text{expl}}(a|s,g)P(s'|s,a)\right) \tag{14.1.8}$$

$$= \int_{s,a,s',g} f(s,a,s',g)\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)P(s'|s,a) \tag{14.1.9}$$

with

$$\nu(s|g) := \rho_{\mathcal{G}}(g)\int_{s_0}\rho_0(s_0|g)\sum_k p_K(k)(P^{\pi_{\text{exp}}})^k(s|s_0,g) \tag{14.1.10}$$

which is the future distribution of states s when sampling a goal g and following the exploration policy $\pi_{\text{expl}}(.|.,g)$, with p_K as the distribution of future timesteps. If $p_K(k) = (1-\gamma)\gamma^k$, this definition of ν coincides with the definition of ν^π in the following sections. This is the reason why we use the same notation, even though ν is here slightly more general.

We now compute:

$$\mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau,U,K,L))|U=1\right] = \sum_{k,l} p_K(k)p_L(l)\int_{g,s_0,a_0,\dots}\rho(g,s_0,a_0,\dots)f(s_k,a_k,s_{k+1},\varphi(s_{k+l})) \tag{14.1.11}$$

If $l=0$, the re-sampled goal is $g'=\varphi(s)$. Else, the law of g' knowing s_k,a_k,s_{k+1} is the law of $\varphi(s_{k+l})$, which by using the Markov property is the law of $\varphi(\tilde{s})$ if $\tilde{s}$ is sampled as $(P^{\pi_{\text{expl}}})^{l-1}(.|s_{k+1},g)$. Therefore:

$$\mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau,U,K,L))|U=1\right] = \sum_{k,l\geqslant 0} p_K(k)p_L(l)\int_{g,s_0,a_0,\dots}\rho(g,s_0,a_0,\dots)f(s_k,a_k,s_{k+1},\varphi(s_{k+l})) \tag{14.1.12}$$

$$= \sum_k p_K(k)\int_{g,s_0,\dots,s_{k+1}}\rho(g,s_0,\dots,s_{k+1})\left(p_L(0)f(s_k,a_k,s_{k+1},\varphi(s_k))\right) + \\ + \sum_k p_K(k)\int_{g,s_0,\dots,s_{k+1}}\rho(g,s_0,\dots,s_{k+1})\left(\sum_{l\geqslant 1}p_L(l)\int_{\tilde{s}}(P^{\pi_{\text{expl}}})^{l-1}(\tilde{s}|s_{k+1},g)f(s_k,a_k,s_{k+1},\varphi(\tilde{s}))\right) \tag{14.1.13}$$

We define $\mu_{\texttt{future}}(\mathrm{d}g'|s,s',g) := p_L(0)\delta_{\varphi(s)}(\mathrm{d}g') + \sum_{l\geqslant 1}p_L(l)\varphi_*(\pi_{\text{exp}} * P)^{l-1}(g'|s',g)$, where φ_* is the *push-forward* on measures, and we have:

$$\mathbb{E}_{\tau,U,K,L}\left[f(\Phi(\tau,U,K,L))|U=1\right] = \tag{14.1.14}$$

$$= \sum_k p_K(k)\int_{g,s_0,a_0,\dots,s_{K+1},\tilde{s}}\rho(g,s_0,a_0,\dots,s_{k+1})\mu_{\texttt{future}}(g'|s_k,s_{k+1},g)f(s_k,a_k,s_{k+1},g') \tag{14.1.15}$$

$$= \int_{s,a,s',g'}\left(\int_g \rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)\mu_{\texttt{future}}(g'|s,s',g)\right)P(s'|s,a)f(s,a,s',g'). \tag{14.1.16}$$

Therefore,

$$\mu_{\text{HER}}(s,a,s',g) = (1-\alpha)\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)P(s'|s,a) + \alpha\left(\int_{\tilde{g}}\rho_{\mathcal{G}}(\tilde{g})\nu(s|\tilde{g})\pi_{\text{expl}}(a|s,\tilde{g})\mu_{\texttt{future}}(g|s,s',\tilde{g})\right)P(s'|s,a) \tag{14.1.17}$$

We now use the deterministic hypothesis. We know that for every s,a, $P(\mathrm{d}s'|s,a) = \delta_{\psi(s,a)}(\mathrm{d}s')$. We have, for any s,a:

$$P(\mathrm{d}s'|s,a)\mu_{\texttt{future}}(g|s,s',\tilde{g}) = \delta_{\psi(s,a)}(\mathrm{d}s')\mu_{\texttt{future}}(g|s,s',\tilde{g}) \tag{14.1.18}$$

$$= \delta_{\psi(s,a)}(\mathrm{d}s')\mu_{\texttt{future}}(g|s,\psi(s,a),\tilde{g}) \tag{14.1.19}$$

Therefore:

$$\mu_{\text{HER}}(s,a,s',g) = \tag{14.1.20}$$

$$= (1-\alpha)\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)P(s'|s,a) + \alpha\left(\int_{\tilde{g}} \rho_{\mathcal{G}}(\tilde{g})\nu(s|\tilde{g})\pi_{\text{expl}}(a|s,\tilde{g})\mu_{\texttt{future}}(g|s,\psi(s,a),\tilde{g})\right)P(s'|s,a) \tag{14.1.21}$$

$$= \tilde{\mu}(s,a,g)P(s'|s,a) \tag{14.1.22}$$

where

$$\tilde{\mu}(s,a,g) := (1-\alpha)\rho_{\mathcal{G}}(g)\nu(s,g)\pi_{\text{expl}}(a|s,g) + \alpha\left(\int_{\tilde{g}} \rho_{\mathcal{G}}(g)\nu(s|\tilde{g})\pi_{\text{expl}}(a|s,\tilde{g})\mu_{\texttt{future}}(g|s,\psi(s,a),\tilde{g})\right)$$

Therefore:

$$\mathbb{E}\left[\widehat{\delta\theta_{\text{HER}}}\right] = \partial_\theta \int_{s,a,s',g} \tilde{\mu}(s,a,g)P(s'|s,a)(Q(s,a,g) - R(s,g') - \gamma \sup_{a'} Q(s',a',g))^2 \tag{14.1.23}$$

$$= \partial_\theta \int_{s,a,s',g} \tilde{\mu}(s,a,g)\delta_{\psi(s,a)}(s')(Q(s,a,g) - R(s,g') - \gamma \sup_{a'} Q(s',a',g))^2 \tag{14.1.24}$$

$$= \partial_\theta \int_{s,a,s',g} \tilde{\mu}(s,a,g)(Q(s,a,g) - R(s,g) - \gamma \sup_{a'} Q(\psi(s,a),a',g))^2 \tag{14.1.25}$$

$$= \partial_\theta \int_{s,a,g} \tilde{\mu}(s,a,g)(Q(s,a,g) - R(s,g) - \gamma \mathbb{E}_{s'\sim P(\mathrm{d}s'|s,a)} \sup_{a'} Q(s',a',g))^2 \tag{14.1.26}$$

$$= \partial_\theta \int_{s,a,g} \tilde{\mu}(s,a,g)(Q(s,a,g) - T\cdot Q(s,a,g))^2 \tag{14.1.27}$$

We define $\|Q\|_{\tilde{\mu}}$ as:

$$\|Q\|_{\tilde{\mu}}^2 := \int_{s,a,g} \tilde{\mu}(s,a,g)Q(s,a,g)^2. \tag{14.1.28}$$

We now prove that $\|.\|_{\tilde{\mu}}$ is a norm for the space of continuous functions on $\mathcal{S}\times\mathcal{A}\times\mathcal{G}$. This is equivalent to showing that the support of the probability measure $\tilde{\mu}$, $\text{supp}(\tilde{\mu})$ is equal to $\mathcal{S}\times\mathcal{A}\times\mathcal{G}$. Because $\tilde{\mu}(s,a,g) \geqslant (1-\alpha)\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)$, we know that $\text{supp}(\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)) \subset \text{supp}(\tilde{\mu})$. Since for every s,g, $\text{supp}(\pi_{\text{expl}}(a|s,g)) = \mathcal{A}$, $\text{supp}(\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)) = \text{supp}(\rho_{\mathcal{G}}(g)\nu(s|g)) \times \mathcal{A}$. Moreover, $\text{supp}\,\rho_{\mathcal{G}} = \mathcal{G}$. Therefore, we only need to prove that for every g, $\text{supp}(\nu(.|g)) = \mathcal{S}$.

Let $g\in\mathcal{G}$. Because of the definition of ν and because $p_K(k) > 0$ for every k, we have $\text{supp}(\nu(s|g)) = \bigcup_{k\geqslant 0, s_0\in\mathcal{S}} \text{supp}\left((P^{\pi_{\text{expl}}})^k(s|s_0,g)\right)$.

We define the function $\Psi : \mathcal{S}\times\left(\cup_{k\geqslant 1}\mathcal{A}^k\right) \to \mathcal{S}$, corresponding to the action of sequences of action , as follows: for every a, $\Psi(s,a) = \psi(s,a)$, and for every k, $(a_1,...,a_k)\in\mathcal{A}^k$, $\Psi(s,(a_1,...,a_{k+1})) := \psi(\Psi(s,(a_1,...,a_k)),a_{k+1})$. Ψ is continuous. Moreover, we assumed that for any pair of states (s,s'), there is $k\geqslant 0$ and a sequence of actions $(a_0,...,a_k)$ such that applying this sequence of actions from s leads to s'. This means that for every s, $\Psi(s,.)$ is a surjective continuous function.

Moreover, with

$$\text{supp}(P^{\pi_{\text{expl}}})^{k+1}(s|s_0,g) = \cup_{s\in\text{supp}(P^{\pi_{\text{expl}}})^k(s|s_0,g)} \text{supp}\left(\psi(s,\cdot)_*\pi_{\text{expl}}(.|s,g)\right) \tag{14.1.29}$$

$$\supseteq \cup_{s\in\text{supp}(P^{\pi_{\text{expl}}})^k(s|s_0,g)} \left(\psi(s,\text{supp}(\pi_{\text{expl}}(.|s,g)))\right) \tag{14.1.30}$$

by using the continuity of $\psi(s,.)$. Then:

$$\text{supp}(P^{\pi_{\text{expl}}})^{k+1}(s|s_0,g) \supseteq \cup_{s\in\text{supp}(P^{\pi_{\text{expl}}})^k(s|s_0,g)} (\psi(s,\mathcal{A})) \tag{14.1.31}$$

$$= \psi(\text{supp}(P^{\pi_{\text{expl}}})^k(s|s_0,g)\times\mathcal{A}) \tag{14.1.32}$$

By induction, we have: $\text{supp}(P^{\pi_{\text{expl}}})^k(s|s_0,g) \supseteq \Psi(s,\mathcal{A}^k)$. Therefore:

$$\text{supp}(\nu(s|g)) = \bigcup_{k\geqslant 0, s_0\in\mathcal{S}} \text{supp}(P^{\pi_{\text{expl}}})^k(s|s_0,g) \tag{14.1.33}$$

$$\supseteq \bigcup_{k\geqslant 0, s_0\in\mathcal{S}} \Psi(s_0,\mathcal{A}^k) \tag{14.1.34}$$

$$= \bigcup_{s_0\in\mathcal{S}} \Psi(s_0, \bigcup_{k\geqslant 0}\mathcal{A}^k) \tag{14.1.35}$$

$$= \mathcal{S} \tag{14.1.36}$$

This concludes the proof. The main property we use in the theorem is that $\mu_{\texttt{future}}(g'|s,s',g)$ is independent of s'. Therefore, a simple way to remove HER bias is to define $p_L(l) = \mathbb{1}_{l=0}$. Still, this would not remove the issue of vanishing rewards, since the fixed point of HER are the same than those of UVFA.

In the following, we will use again the results derived above. In particular, we know that:

$$\mathbb{E}\left[\widehat{\delta\theta}_{\text{HER}}\right] = \mathbb{E}_{(s,a,s',g)\sim\mu_{\text{HER}}}\left[\partial_\theta \frac{1}{2}(Q_\theta(s,a,g) - R(s,g) - \gamma \sup_{a'} Q(s',a',g))^2\right] \tag{14.1.37}$$

with

$$\mu_{\text{HER}}(s,a,s',g) = (1-\alpha)\rho_{\mathcal{G}}(g)\nu(s|g)\pi_{\text{expl}}(a|s,g)P(s'|s,a) + \alpha\left(\int_{\tilde{g}} \rho_{\mathcal{G}}(\tilde{g})\nu(s|\tilde{g})\pi_{\text{expl}}(a|s,\tilde{g})\mu_{\texttt{future}}(g|s',\tilde{g})\right)P(s'|s,a) \tag{14.1.38}$$

$$\mu_{\texttt{future}}(g'|s',g) = \sum_l p_L(l)\varphi_*(\pi_{\text{exp}} * P)^l(g'|s',g) \tag{14.1.39}$$

$$\nu(s|g) = \rho_{\mathcal{G}}(g)\int_{s_0} \rho_0(s_0|g)\sum_k p_K(k)(\pi_{\text{exp}} * P)^k(s|s_0,g) \tag{14.1.40}$$

14.2 Bias of HER in stochastic environments.

Here is a simple way to design *counter-examples* environments which exhibit this HER bias. Consider a finite multi goal environment and add a single action a^* which, from any state s, sends the agent to a uniform random state s' and then *freezes* it, which means the agent will always stay at s'.

Both in theory and practice, HER will learn to always select the action a^* (third plot in Fig. 16.1). The intuition is the following: when the agent acts with a^* and reaches a random state s', HER reinforces a^* as a good way to reach s' from s, while this was purely random. We now formally state a corresponding theorem.

Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{G}, \mathcal{A}, P, R, \gamma\rangle$ be a multi-goal finite Markov Decision Process, with $\mathcal{G} = \mathcal{S}$ and $R(s,g) = \mathbb{1}_{s=g}$. We define $S = |\mathcal{S}|$ the number of states.

Let $\tilde{\mathcal{M}}$ be the augmented MDP with a *freeze* action a^*, defined as:

- The augmented state space $\tilde{\mathcal{S}} = \mathcal{S} \times \{0,1\}$, where $\tilde{s} = (s,x)$ is said to be frozen if $x = 1$.
- The augmented action space $\tilde{\mathcal{A}} = \mathcal{A} \cup \{a^*\}$, where a^* is the *freeze* action.
- The goal space does not change ($\tilde{\mathcal{G}} = \mathcal{G} = \mathcal{S}$). For an augmented state $\tilde{s} = (s,x)$, the reward is $\tilde{R}(\tilde{s},g) = \tilde{R}((s,x),g) = R(s,g)$
- If $\tilde{s} = (s,x)$ and $\tilde{s}' = (s',x')$ are two augmented states, the transition operator $\tilde{P}(\tilde{s}'|\tilde{s},a)$:
  - If the state is frozen ($x = 1$), the agent can't move: $\tilde{P}((s',y)|(s,x),a) = \mathbb{1}_{s'=s}\mathbb{1}_{y=1}$
  - If the state is not frozen ($x = 0$) and $a = a^*$, the agent is sent to a uniformly random frozen state: $\tilde{P}((s',y)|(s,0),a) = \mathbb{1}_{y=1}\frac{1}{\mathcal{S}}$
  - Else, the dynamic is the same than for $\mathcal{M}$: if $x = 0$ and $a \neq a^*$, then $P((s',y)|(s,0),a) = \mathbb{1}_{y=0}P(s'|s,a)$.

We can now prove the existence of MDPs such that HER will be biased in these environments.

Theorem 14.2. *Let $\mathcal{M}$ be a finite MDP, and $\tilde{\mathcal{M}}$ the augmented MDP with the freeze action a^* defined above. We assume that for every s,a,g the exploration policy satisfies $\pi_{\text{expl}}(a|s,g) > 0$, and that for every every s,g, $\nu(s|g) > 0$, where ν is defined in equation (14.1.10). This means that from the given distribution, every state s has a non-zero probability of being reached when following the exploration policy conditioned by g: $\pi_{\text{expl}}(a|s,g)$.*

Let Q_∞ be a fixed point of tabular HER, and Q^* the true optimal Q-function. Then, for every unfrozen state $(s,0)$ and goal g, HER overstimates the value of action a^*:

$$Q_\infty((s,0),a^*,g) > Q^*((s,0),a^*,g) \tag{14.2.1}$$

Proof.

The principle of the proof is the following. First, we prove that for *frozen* states $\tilde{s} = (s, 1)$, HER converge converge to the true value $Q_\infty(\tilde{s}, a, g) = Q^*(\tilde{s}, a, g)$. Then, we compute the action-value of action a^* for every *unfrozen* state for the true Q^* and for the fixed point Q_∞. HER samples transitions $((s, 0), a^*, (s', 1), g)$. Let us consider the law of s' knowing s, a^*, g: with probability $(1 - \alpha)$ the goal g was re-sampled from the `future` sampling strategy, therefore, because after a^* the position will be frozen, we know that $s' = g$, the goal is reached and the final return is $O(\frac{1}{1-\gamma})$. With probability α, the goal g the original goal, the law of s' is uniform, and the return is of order $O(\frac{1}{S(1-\gamma)})$. Therefore, when estimating the return after action a^* with HER, the computed value will be of order $O(\frac{(1-\alpha)}{1-\gamma})$, while the true value is of order $O(\frac{1}{S(1-\gamma)})$.

We now prove the theorem. We consider Q_∞, a fixed point of the algorithm, which means that starting from Q_∞, the stochastic update defined by HER has mean 0: $\mathbb{E}\left[\widehat{\delta Q}_{\text{HER}}\right] = 0$. We know that

$$\mathbb{E}\left[\widehat{\delta Q}_{\text{HER}}\right] = \mathbb{E}_{(s,a,s',g)\sim\mu_{\text{HER}}}\left[\partial_\theta \frac{1}{2}(Q_\theta(s, a, g) - R(s, g) - \gamma \sup_{a'} Q(s', a', g))^2\right] \tag{14.2.2}$$

$$= \mathbb{E}_{(s,a,s',g)\sim\mu_{\text{HER}}}\left[E_{s,a,g}(Q_\theta(s, a, g) - R(s, g) - \gamma \sup_{a'} Q(s', a', g))\right], \tag{14.2.3}$$

where $(E_{s,a,g})$ is the canonical basis of the tabular model. Therefore, for every (s, a, g) (because $\mu_{\text{HER}}(s, a, g) > 0$ for every (s, a, g)), we have:

$$Q_\infty(s, a, g) = R(s, g) + \gamma \mathbb{E}_{s'\sim\mu_{\text{HER}}(s'|s,a,g)}\left[\sup_{a'} Q_\infty(s', a', g)\right] \tag{14.2.4}$$

First, we prove that the values of frozen states $Q_\infty((s, 1), a, g)$ is equal to the true optimal Q-values. In that case, $\tilde{P}(\tilde{s}'|(s, 1), a) = \delta_{(s,1)}(\tilde{s}')$ we can check that $\mu_{\text{HER}}(\tilde{s}'|(s, 1), a, g) = \delta_{(s,1)}(\tilde{s}')$. Therefore:

$$Q_\infty((s, 1), a, g) = R(s, g) + \gamma \sup_{a'} Q_\infty((s, 1), a', g) \tag{14.2.5}$$

Therefore for every s, a, g, $Q_\infty((s, 1), a, g) = \frac{1}{1-\gamma} R(s, g)$.

Then, we compute the values of $Q_\infty((s, 0), a^*, g)$, with the *freeze* action for an unfrozen state. We have:

$$Q_\infty((s, 0), a^*, g) = R(s, g) + \gamma \mathbb{E}_{(s',y)\sim\mu_{\text{HER}}((s',y)|(s,0),a^*,g)} \sup_{a'} Q_\infty((s', y), a', g) \tag{14.2.6}$$

$$= R(s, g) + \frac{\gamma}{1-\gamma} \mathbb{E}_{(s',y)\sim\mu_{\text{HER}}((s',1)|s,a^*,g)}\left[\mathbb{1}_{s'=g}\right] \tag{14.2.7}$$

$$= R(s, g) + \frac{\gamma}{1-\gamma} \mu_{\text{HER}}((s', y) = (g, 1)|s, a, g) \tag{14.2.8}$$

because $\mu_{\text{HER}}((s', y)|(s, 0), a^*, g)$ is non zero only if $y = 1$, and $Q_\infty((s', 1), a', g) = \frac{1}{1-\gamma} R(s', g) = \frac{1}{1-\gamma} \mathbb{1}_{s'=g}$. We now compute $\mu_{\text{HER}}((s', y) = (g, 1)|s, a, g)$. We use that $P((s', y)|(s, 0), a^*) = \mathbb{1}_{y=1}/S$, and $\mu_{\texttt{future}}(g|(s', y)) = \mathbb{1}_{s'}$ if $y = 1$.

$$\mu_{\text{HER}}((s, 0), a^*, (s', 1), g) = (1 - \alpha)\mu_0((s, 0), g)\pi_{\text{expl}}(a^*|(s, 0), g)\frac{1}{S} + \alpha\left(\int_{\tilde{g}} \mu_0((s, 0), \tilde{g})\pi_{\text{expl}}(a^*|(s, 0), \tilde{g})\right)\frac{1}{S}\mathbb{1}_{g=s'} \tag{14.2.9}$$

Therefore, for every $s' \neq g$: $\mu_{\text{HER}}((s, 0), a^*, (s', 1), g) < \mu_{\text{HER}}((s, 0), a^*, (g, 1), g)$. So:

$$\sum_{s'} \mu_{\text{HER}}((s, 0), a^*, (s', 1), g) < S\mu_{\text{HER}}((s, 0), a^*, (g, 1), g) \tag{14.2.10}$$

and finally $\mu_{\text{HER}}((s', y) = (g, 1)|(s, 0), a^*, g) > \frac{1}{S}$. Then we have:

$$Q_\infty((s, 0), a^*, g) > R(s, g) + \frac{\gamma}{S(1-\gamma)} \tag{14.2.11}$$

On the contrary, we can easily check that for any policy π, $Q^\pi((s, 0), a^*, g) = R(s, g) + \frac{\gamma}{S(1-\gamma)}$. In particular, by taking $\pi = \pi^*$, we have:

$$Q_\infty((s, 0), a^*, g) > Q^*((s, 0), a^*, g) \tag{14.2.12}$$

Generally, HER is overestimating chancy outcomes, by estimating that any action (even random) that led to some goal was a good way to reach that goal. This is clear in the example of the freeze-after-random-jump actions in Theorem 14.2. Thus, HER has no reason to learn reliably in a stochastic environment. Other hindsight methods such as (Rauber et al., 2019) experience a similar bias.

Chapter 15

Unbiased Multi-Goal Q-learning with Infinitely Sparse Rewards

Our goal here is to formally define multi-goal Q-learning with infinitely sparse rewards. In general, the probability of reaching any goal *exactly* is 0: instead we will learn the probability *distribution* of the goals reached by a policy, and compute the probability to reach each infinitesimal element $\mathrm{d}g$ in goal space. This is done by treating everything as measures over $\mathcal{G}$: the reward $\delta_{\varphi(s)}(\mathrm{d}g)$ is a measure, and the value functions $V^\pi(s, \mathrm{d}g)$ or optimal action-value function $Q^*(s, a, \mathrm{d}g)$ are measures on $\mathcal{G}$ as well. In the following, we define these objects in detail, and show how to learn them in practice.

First, in Section 15.1 we define an *optimal Bellman operator* T on *action-value measures*, and the optimal action-value measure $Q^*(s, a, \mathrm{d}g)$. Then in Section 15.2, we derive δ-DQN, a deep Q-learning algorithm with infinitely sparse rewards for multi-goal RL. Some experimental results are in the next chapter, together with the results on unbiased actor-critic with infinitely sparse rewards.

15.1 Optimal Bellman equation and optimal action-value measure.

15.1.1 The optimal action-value measure

We first define $Q^*(s, a, \mathrm{d}g)$, the *optimal action-value measure*, the mathematical object corresponding to the usual optimal Q-function Q^* but infinitely sparse rewards. The following theorem defines the optimal Bellman operator for action-value measures, and $Q^*(s, a, \mathrm{d}g)$ as its fixed point.

With continuous states and goals, in a stochastic environment, the goal-dependent optimal Q-function Q^*_ε with reward $R_\varepsilon(s, g) = \mathbb{1}_{\|\varphi(s)-g\| \leqslant \varepsilon}$ vanishes when $\varepsilon \to 0$: the probability of exactly reaching a goal state is usually 0. Likewise, a direct application of TD would never learn anything because rewards would likely never be observed.

Instead, the goal-dependent Q-function is a *measure* over goals. Intuitively, for every infinitesimally small set of goals $\mathrm{d}g$, the quantity $Q^*(s, a, \mathrm{d}g)$ is the expected amount of time spent in $\mathrm{d}g$ by the policy that tries to maximize time spent in $\mathrm{d}g$, starting at (s, a).

Formally, for every state-action (s, a), $Q^*(s, a, \cdot)$ is a measure over goals, solution to the Bellman equation

$$Q^*(s, a, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma\, \mathbb{E}_{s' \sim P(\mathrm{d}s'|s,a)} \max_{a'} Q^*(s', a', \mathrm{d}g) \tag{15.1.1}$$

where, as above, $\varphi\colon S \to G$ is the function defining the target features, and where $\delta_{\varphi(s)}$ is the Dirac measure at $\varphi(s)$ in goal space. This is an equality between measures, and the supremum is a supremum of measures (Bogachev, 2007, Section 4.7).

Existence and uniqueness of solutions, and a formal derivation of a TD algorithm, are nontrivial in this setting. Uniqueness never holds without restrictions: the infinite measure always solves (15.1.1). But it is not possible to restrict ourselves to finite-mass measures, because sometimes the solution we want has infinite mass, as shown in Section 15.1.3. The need to deal with possibly infinite measures restricts the use of uniqueness proofs by γ-contractivity arguments in some norm.

Intuitively, the total mass $Q^*(s, a, \mathcal{G})$ of the goal state $\mathcal{G}$ describes how much different action sequences result in non-overlapping distributions of states. If the state space $\mathcal{A}$ is finite and $|\mathcal{A}| = A$, the total mass of the horizon-t part of the Q^*-function can be as much as $\gamma^t A^t$: this is realized when every possible sequence of t actions leads to a disjoint part of the state of goals. In section 15.1.3 we provide a simple continuous MDP in which every action sequence leads to a distinct state: as there are an infinite number of action sequences when $t \to \infty$, the total mass $Q^*(s, a, \mathcal{G})$ is infinite.

15.1.2 Existence of a goal-dependent Q-function in continuous spaces

We prove the existence of a canonical solution to the optimal Bellman equation, equal both to the *smallest* solution and to the limit of the horizon-t solution when $t \to \infty$.

Theorem 15.1 (Existence of a goal-dependent Q-function in continuous spaces)**.** *Let $\mathcal{Q}$ be the set of functions from $\mathcal{S} \times \mathcal{A}$ into positive measures over $\mathcal{G}$. Assume the set of actions $\mathcal{A}$ is countable. Let T be the Bellman operator mapping $Q \in \mathcal{Q}$ to $T \cdot Q$ with*

$$T \cdot Q(s, a, \cdot) := \delta_{\varphi(s)}(\cdot) + \gamma\, \mathbb{E}_{s' \sim P(\mathrm{d}s'|s,a)} \sup_{a'} Q(s', a', \cdot) \tag{15.1.2}$$

where the supremum is a supremum of measures and $\delta_{\varphi(s)}$ is the Dirac measure at $\varphi(s) \in \mathcal{G}$.

Let $\mathbf{0} \in \mathcal{Q}$ be the measure 0.

Let $Q_t := T^t\mathbf{0}$. (By expanding the definition of T, this is the solution of the expectimax problem at time horizon t.) Then when $t \to \infty$, for every state-action (s, a) and for every measurable set $G \subset \mathcal{G}$, $Q_t(s, a, G)$ converges to a finite or infinite limit $Q^*(s, a, G)$. This limit Q^* is an element of $\mathcal{Q}$ and solves the Bellman equation $TQ^* = Q^*$. It is the smallest such solution. In finite state spaces, it is the only solution with finite mass. Moreover, for any goal-dependent policy π, its Bellman operator T^π and Q-value $Q^\pi := \lim_{t\to\infty}(T^\pi)^t\mathbf{0}$ can be defined similarly (see equation (15.1.6)*) and satisfy $Q^\pi \leqslant Q^*$ as measures.*

Proof. Assume the action space $\mathcal{A}$ is countable. Let $\mathcal{Q}$ be the set of measurable functions from $\mathcal{S} \times \mathcal{A}$ to the set of measures on $\mathcal{G}$.

For Q_1 and Q_2 in $\mathcal{Q}$, we write $Q_1 \leqslant Q_2$ if $Q_1(s, a, X) \leqslant Q_2(s, a, X)$ for any state-action (s, a) and measurable set $X \subset \mathcal{G}$. The Bellman operator of Definition 15.1.2 acts on $\mathcal{Q}$ and is obviously monotonous: if $Q_1 \leqslant Q_2$ then $TQ_1 \leqslant TQ_2$.

Since the zero measure $\mathbf{0} \in \mathcal{Q}$ is the smallest measure, we have $T\mathbf{0} \geqslant \mathbf{0}$. Since T is monotonous, by induction we have $T^{t+1}\mathbf{0} \geqslant T^t\mathbf{0}$ for any $t \geqslant 0$. Thus, the $(T^t\mathbf{0})_{t\geqslant 0}$ form an increasing sequence of measures. Therefore, for every state-action (s, a) and measurable set X, the sequence $(T^t\mathbf{0})(s, a, X)$ is increasing, and thus converges to a limit. We denote this limit by $Q^*(s, a, X)$. We have to prove that $Q^* \in \mathcal{Q}$, namely, that for each (s, a), $Q^*(s, a, \cdot)$ is a measure. The only non-trivial point is σ-additivity.

Denote $Q_t := T^t\mathbf{0}$. If (X_i) is a countable collection of disjoint measurable sets, we have

$$\begin{aligned} Q^*(s, a, \cup_i X_i) &= \lim_{t\to\infty} Q_t(s, a, \cup_i X_i) = \lim_{t\to\infty} \sum_i Q_t(s, a, X_i) \\ &= \sum_i \lim_{t\to\infty} Q_t(s, a, X_i) = \sum_i Q^*(s, a, X_i) \end{aligned}$$

where the limit commutes with the sum thanks to the monotone convergence theorem, using that Q_t is non-decreasing. Therefore, Q^* is a measure.

Let us prove that $TQ^* = Q^*$. We have

$$TQ^*(s, a, \cdot) = \delta_{\varphi(s)} + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} \sup_{a'} Q^*(s', a', \cdot) \tag{15.1.3}$$

by definition. For any s', denote $\tilde{Q}_t(s', \cdot) := \sup_{a'} Q_t(s', a', \cdot)$ where the supremum is as measures over $\mathcal{G}$. Since Q_t is non-decreasing, so is $\tilde{Q}_t$.

For any state s', we have

$$\sup_{a'} Q^*(s', a', \cdot) = \sup_{a'} \sup_t Q_t(s', a', \cdot) = \sup_t \sup_{a'} Q_t(s', a', \cdot) = \sup_t \tilde{Q}_t(s', \cdot) \tag{15.1.4}$$

since supremums commute. Now, since $\tilde{Q}_t$ is non-decreasing, thanks to the monotone convergence theorem, the supremum commutes with integration over $s' \sim P(s'|s, a)$ (which does not depend on t), namely,

$$\begin{aligned}\mathbb{E}_{s'\sim P(s'|s,a)} \sup_{a'} Q^*(s', a', \cdot) &= \mathbb{E}_{s'\sim P(s'|s,a)} \sup_t \tilde{Q}_t(s', \cdot) \\ &= \sup_t \mathbb{E}_{s'\sim P(s'|s,a)} \tilde{Q}_t(s', \cdot) = \sup_t \mathbb{E}_{s'\sim P(s'|s,a)} \sup_{a'} Q_t(s', a', \cdot)\end{aligned} \tag{15.1.5}$$

and so $TQ^* = \sup_t TQ_t$. Now, since $Q^t = T^t\mathbf{0}$, we have $TQ^t = T^{t+1}\mathbf{0}$, so that $\sup_{t\geqslant 0} TQ^t = \sup_{t\geqslant 1} T^t\mathbf{0} = Q^*$. So Q^* is a fixed point of T.

Let us prove that Q^* is the smallest such fixed point. Let Q' such that $TQ' = Q'$. Since $\mathbf{0} \leqslant Q'$ and T is monotonous, we have $T\mathbf{0} \leqslant TQ' = Q'$. By induction, $T^t\mathbf{0} \leqslant Q'$ for any $t \geqslant 0$. Therefore, $\sup_t T^t\mathbf{0} \leqslant Q'$, i.e., $Q^* \leqslant Q'$.

The statement for finite state spaces reduces to the classical uniqueness property of the usual Q function, separately for each goal state.

Optimality of the policy is proved by following classical arguments. Let $\pi(a|s, g)$ be any goal-dependent policy and let $Q \in \mathcal{Q}$. Define the Bellman operator associated to π by

$$(T^\pi Q)(s, a, \cdot) := \delta_s + \gamma\mathbb{E}_{s'\sim P(s'|s,a)} \sum_{a'} (\pi * Q)(s', a', \cdot) \tag{15.1.6}$$

where for each action a, the measure $(\pi*Q) \in \mathcal{Q}$ is defined via $(\pi*Q)(s', a', X) := \int_{g\in X} \pi(a'|s', g)Q(s', a', \mathrm{d}g)$, so that the sum of $(\pi*Q)$ over all actions a' represents the expected value of $Q(s', a', \cdot)$ under the goal-dependent policy π; this formulation allows the policy to depend on the goal.

Since π is a probability distribution, we have

$$\sum_{a'} (\pi * Q)(s', a', X) \leqslant \max_{a'} Q(s', a', X) \tag{15.1.7}$$

where the right-hand-side is a maximum of measures (thus selecting the best a' for each goal): this is clear from decomposing X into the components where each action a' is optimal.

Therefore, for any $Q \in \mathcal{Q}$, we have the inequality of measures

$$T^\pi Q \leqslant TQ \tag{15.1.8}$$

where T is the optimal Bellman operator from above. Since the latter is monotonous over $Q \in \mathcal{Q}$, for any $Q, Q' \in \mathcal{Q}$ with $Q \leqslant Q'$, we have $T^\pi Q \leqslant TQ'$.

Consequently, by induction, $(T^\pi)^t\mathbf{0} \leqslant T^t\mathbf{0}$ for any horizon $t \geqslant 0$. The monotonous limit $Q^\pi := \lim_{t\to\infty}(T^\pi)^t\mathbf{0}$ exists for the same reasons as $T^t\mathbf{0}$, representing the Q-function (measure) of policy π. Therefore, $Q^\pi = \lim_{t\to\infty}(T^\pi)^t\mathbf{0} \leqslant \lim_{t\to\infty} T^t\mathbf{0} = Q^*$. This proves that the policy π has returns no greater than Q^*.

15.1.3 Examples of MDPs with Infinite Mass for Q^*

Here are two simple examples of MDPs with finite action space, for which the mass of the goal-dependent Q-measure $Q^*(s, a, \mathrm{d}g)$ is infinite. The first has discrete states, the second, continuous ones.

Take for $\mathcal{S}$ an infinite rooted dyadic tree, namely, $\mathcal{S} = \{\varnothing, 0, 1, 00, 01, \ldots\}$ the set of binary strings of finite length $k \geqslant 0$, and $\mathcal{G} = \mathcal{S}$. Consider the two actions "add a 0 at the end" and "add a 1 at the end". Then, for every state s, $Q^*(s, a, \cdot)$ is a measure that gives mass γ^k to all states g that are extensions of s by a length-k string that starts with a. Thus, its mass is $1 + \sum_{k\geqslant 1} \gamma^k 2^{k-1}$. This is infinite as soon as $\gamma \geqslant 1/2$. This extends to any number of actions by considering higher-degree trees.

A similar example with continuous states is obtained as follows. Let $\mathcal{S} = [0; 1) \times [0; 1)$. Let $C = \{\varnothing, 0, 1, 00, 01, \ldots\}$ the dyadic tree above. For each string $w \in X$, consider the set $B_w \subset \mathcal{S}$ defined as follows: B_w is made of those points $(x, y) \in \mathcal{S}$ such that the binary expansion of x starts with w, and $y \in [1 - 1/2^k; 1 - 1/2^{k+1})$ where k is the length of w. Graphically, this creates a tree-like partition of the square $\mathcal{S}$, where the empty string corresponds to the bottom half, the strings $w = 0$ and $w = 1$ correspond to two sets on the left and right above the bottom hald, etc. Define the following MDP with two actions 0 and 1: with action 0, every state $s \in B_w$ goes to a uniform random state in B_{w0}, and with action 1, every state $s \in B_w$ goes to a uniform random state in B_{w1}. The goal-dependent Q-function Q^* is similar to the dyadic

tree above, but is continuous. Its mass is infinite for the same reasons.

15.2 Q-learning with function approximations, with infinitely sparse rewards.

From the fixed point equation for Q^*, we would like to learn a model of $Q^*(s, a, \mathrm{d}g)$ with function approximation. We will represent measures over goals via their *density* with respect to the goal sampling function $\rho_{\mathcal{G}}$ of the environment. Namely, we will approximate $Q^*(s, a, \mathrm{d}g)$ by a model $Q_\theta(s, a, \mathrm{d}g) = q_\theta(s, a, g)\rho_{\mathcal{G}}(\mathrm{d}g)$ where $q_\theta(s, a, g)$ is an ordinary function, and learn q_θ. Hence, q_θ may be approximated by any parametric model, such as a neural network. This is similar to our parametrization $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ for the successor states operator used in Part IV.

In this section, we formally derive the δ-DQN update. Let us consider parametric models for Q:

$$Q_\theta(s, a, \mathrm{d}g) := q_\theta(s, a, g)\rho_{\mathcal{G}}(\mathrm{d}g) \tag{15.2.1}$$

and we will learn q_θ. The factor $\rho_{\mathcal{G}}$, or some other measure, is needed to get a well-defined object in continuous state spaces. In discrete spaces, it results in an g-dependent scaling of the Q function, which still has the same optimal policy for each g.

The resulting parametric update is off-policy: we assume access to a sampling distribution $(s, a, s') \sim \rho_{\mathrm{SA}}(\mathrm{d}s, \mathrm{d}a)P(\mathrm{d}s'|s, a)$ in a Markov decision. Typically, this can correspond to transitions (s_k, a_k, s_{k+1}) from exploration trajectory with $g \sim \rho_{\mathcal{G}}$, $s_0 \sim \rho_0(.|g)$, then $a_t \sim \pi_{\mathrm{expl}}(.|s_t, g)$ and $s_{t+1} \sim P(.|s_t, a_t)$. Here, our statement with a distribution ρ_{SA} is more general.

Similarly to the norms defined for the successor states operator in Section 6.4, we define the following norm on Q-value measure: If $Q_1(s, a, \mathrm{d}g)$ and $Q_2(s, a, \mathrm{d}g)$ are two action-value measures such continuous with respect to $\rho_{\mathcal{G}}$: $Q_1(s, a, \mathrm{d}g) = q_1(s, a, g)\rho_{\mathcal{G}}(\mathrm{d}g)$ and $Q_2(s, a, \mathrm{d}g) = q_2(s, a, g)\rho_{\mathcal{G}}(\mathrm{d}g)$, then

$$\|Q_1 - Q_2\|^2_{\rho_{\mathrm{SA}}, \rho_{\mathcal{G}}} := \mathbb{E}_{(s,a)\sim\rho_{\mathrm{SA}},\, g\sim\rho_{\mathcal{G}}}[(q_1(s, a, g) - q_2(s, a, g))^2] \tag{15.2.2}$$

If Q_1 or Q_2 are not continuous with respect to $\rho_{\mathcal{G}}$, then the norm is infinite.

We can now define the δ-DQN update, which then leads to δ-DQN (Algorithm 14). We present some experimental results in the next chapter, together with experiments with unbiased actor-critic methods with infinitely sparse rewards.

Theorem 15.2. *Let $Q_\theta(s, a, \mathrm{d}g) = q_\theta(s, a, g)\rho_{\mathcal{G}}(\mathrm{d}g)$ be a current estimate of $Q^*(s, a, \mathrm{d}g)$.*

We define $Q_{\mathrm{tar}} := T \cdot Q_\theta$, a terget Q-measure for Q_θ defined via the optimal Bellman equation.

Let $(s, a, s') \sim \rho_{SA}(\mathrm{d}s, \mathrm{d}a)P(s'|s, a)$ be samples of the environment and $g \sim \rho_{\mathcal{G}}$ sampled independently. Let $\widehat{\delta\theta}_{\delta\text{-DQN}}(s, a, s', g)$ be

$$\widehat{\delta\theta}_{\delta\text{-DQN}}(s, a, s', g) := \partial_\theta q_\theta(s, a, \varphi(s)) + \partial_\theta q_\theta(s, a, g)\left(\gamma \max_{a'} q_{\mathrm{tar}}(s', a', g) - q_\theta(s, a, g)\right) \tag{15.2.3}$$

Then $\widehat{\delta\theta}_{\delta\text{-DQN}}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}\left[\widehat{\delta\theta}_{\delta\text{-DQN}}\right] = -\frac{1}{2}\partial_\theta \|Q_\theta - Q_{\mathrm{tar}}\|^2_{\rho_{\mathrm{SA}}, \rho_{\mathcal{G}}} \tag{15.2.4}$$

In particular, the true optimal state-action measure Q^* is a fixed point of this update: if $Q_\theta = Q_{\mathrm{tar}} = Q^*$ then $\mathbb{E}\left[\widehat{\delta\theta}_{\delta\text{-DQN}}\right] = 0$.

Here we have presented the update using a single function q_θ. Nonetheless, using an additional "target network" with parameter θ_0 (typically a previous value of θ), is a common

practice for parametric Q-learning. This question was addressed for the successor states operator in Section 7.6, and we will not give additional details here.

For this theorem, we sample goals g independently of (s, a, s'). In practice, this could be a source of variance, as sampling goals far from the current state should produce close-to-0 Q-values. If we instead sample goals from a distribution $\mu(g|s, a)$, this introduces an implicit *scaling* factor $\alpha(s, g)$ to the reward. This is discussed 16.1.2 in the case of the V-function.

Proof. By definition of the optimal Bellman operator T and the target Q_{tar}, we have:

$$TQ_{\text{tar}}(s, a, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma\, \mathbb{E}_{s' \sim P(s'|s,a)} \left[\sup_{a'} q_{\text{tar}}(s', a', g)\right] \rho_{\mathcal{G}}(\mathrm{d}g) \tag{15.2.5}$$

By definition of $J'_Q(\theta)$ and of the norm $\|\cdot\|_{\rho_{\text{SA}},\rho_{\mathcal{G}}}$, we have

$$J'_Q(\theta) = \frac{1}{2} \|Q_\theta\|^2_{\rho_{\text{SA}},\rho} - \langle Q_\theta, TQ_{\text{tar}}\rangle_{\rho_{\text{SA}},\rho} \tag{15.2.6}$$

$$= \frac{1}{2} \int_{s,a,g} q_\theta^2(s, a, g)\rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\rho(\mathrm{d}g) - \int_{s,a,g} q_\theta(s, a, g)(T \cdot Q_{\text{tar}})(s, a, \mathrm{d}g)\rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a) \tag{15.2.7}$$

Consequently,

$$\partial_\theta J'(\theta) = \int_{s,a,g} \partial_\theta q_\theta(s, a, g) q_\theta(s, a, g)\rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\rho_{\mathcal{G}}(\mathrm{d}g) - \int_{s,a,g} \partial_\theta q_\theta(s, a, g) TQ_{\text{tar}}(s, a, \mathrm{d}g)\rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a) \tag{15.2.8}$$

$$= \int_{s,a,g} \rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\partial_\theta q_\theta(s, a, g) \left(Q_\theta(s, a, \mathrm{d}g) - TQ_{\text{tar}}(s, a, \mathrm{d}g)\right) \tag{15.2.9}$$

assuming q_θ is smooth enough so that the derivative makes sense and commutes with the integral.

Moreover, we have:

$$TQ_{\text{tar}}(s, a, \mathrm{d}g) - Q_\theta(s, a, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma\, \mathbb{E}_{s' \sim P(s'|s,a)}[\sup_{a'} q_{\text{tar}}(s', a', g) - q_\theta(s, a, g)]\rho_{\mathcal{G}}(\mathrm{d}g) \tag{15.2.10}$$

Therefore,

$$\begin{aligned} -\partial_\theta J'(\theta) = &\int_{s,a} \rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\partial_\theta q_\theta(s, a, g)\delta_{\varphi(s)}(\mathrm{d}g) \\ &+ \int_{s,a,g} \rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\rho_{\mathcal{G}}(\mathrm{d}g) \left(\gamma\, \mathbb{E}_{s' \sim P(s'|s,a)}[\sup_{a'} q_{\theta_0}(s', a', g) - q_\theta(s, a, g)]\right) \end{aligned} \tag{15.2.11}$$

$$\begin{aligned} = &\int_{s,a} \rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\partial_\theta q_\theta(s, a, \varphi(s)) \\ &+ \int_{s,a,g} \rho_{\text{SA}}(\mathrm{d}s, \mathrm{d}a)\rho_{\mathcal{G}}(\mathrm{d}g) \left(\gamma\, \mathbb{E}_{s' \sim P(s'|s,a)}[\sup_{a'} q_{\theta_0}(s', a', g) - q_\theta(s, a, g)]\right) \end{aligned} \tag{15.2.12}$$

This update leads to δ-DQN (Algorithm 14), which corresponds to standard DQN with infinitely sparse rewards. For continuous actions, δ-DQN can be modified similarly to DDPG (Lillicrap et al., 2016). We present some experimental results in the next chapter, together with experiments with unbiased actor-critic methods with infinitely sparse rewards.

15.3 Example: the tabular case.

The tabular case highlights the difference between UVFA and δ-DQN. When a transition (s, a, s', g) is observed, the UVFA update is:

$$Q(s, a, g) \leftarrow Q(s, a, g) + \eta \left(\mathbb{1}_{\varphi(s)=g} + \gamma \max_{a'} Q(s', a', g) - Q(s, a, g)\right) \tag{15.3.1}$$

where η is the learning rate. The only modified value is $Q(s, a, g)$.

With δ-DQN, we learn the density q of $Q(s, a, \mathrm{d}g)$ with respect to $\rho_{\mathcal{G}}$. Assume that $\rho_{\mathcal{G}}(g)$ is the uniform measure over the finite goal space $\mathcal{G}$. Then we learn $q(s, a, g) = |\mathcal{G}| \times Q(s, a, g)$. For a tabular model, the δ-DQN update in Equation (15.2.3) is

$$q(s, a, \varphi(s)) \leftarrow q(s, a, \varphi(s)) + \eta \tag{15.3.2}$$

$$q(s, a, g) \leftarrow q(s, a, g) + \eta \left(\gamma \max_{a'} q(s', a', g) - q(s, a, g)\right). \tag{15.3.3}$$

Algorithm 14 δ-DQN

Input: Randomly initialized model $q_\theta(s, a, g)$; φ; exploration policy $\pi_{\text{expl}}(a|s, g)$; goal function φ; memory buffer `TransitionMemory`, T the maximum trajectory length
repeat
  for K trajectories **do**
    Get a goal g and an initial state s_0
    for $0 \leqslant t \leqslant T$ steps do **do**
      Sample $a_t \sim \pi_{\text{expl}}(.|s_t, g)$, execute a_t and observe s_{t+1}
      Store in the transition memory the transition `TransitionMemory` $\leftarrow (s_t, a_t, s_{t+1})$
    end for
    for L gradient steps **do**
      Sample $(s, a, s') \sim$ `TransitionMemory` and $g \sim \rho_{\mathcal{G}}$
      $\widehat{\delta\theta}_{\delta\text{-DQN}} := \partial_\theta q_\theta(s, a, \varphi(s)) + \partial_\theta q_\theta(s, a, g)\,(\gamma \max_{a'} q_\theta(s', a', g) - q_\theta(s, a, g))$.
      Stochastic gradient step: $\theta \leftarrow \theta + \eta \widehat{\delta\theta}_{\delta\text{-DQN}}$.
    end for
  end for
until end of learning

Here two values are updated: in addition to (s, a, g), the trajectory visiting s is also used to update the value for the goal $\varphi(s)$. The first part always increases q at the goal $\varphi(s)$ achieved by s; the second part at (s, a, g) has no reward contribution, and decreases q at (s, a, g) by a factor $(1 - \eta)$ while propagating the value from s'. In expectation, the decrease at (s, a, g) compensates the increase at $(s, a, \varphi(s))$: this compensation is exact when q is the exact solution.

As a comparison, the tabular HER update works as follows: when observing a trajectory $(g, s_0, a_0, s_1, ...)$, a transition $(s, a, s', g) = (s_K, a_K, s_{K+1}, g)$ for some $K \geqslant 0$ is selected; then HER samples $L \geqslant K$, defines $g' := \varphi(s_L)$ as the re-sampled goal, then applies the UVFA update (15.3.1) but with (s, a, s', g') instead of (s, a, s', g). When $L = K$, the goal sampled by HER is $g' = \varphi(s)$: this is somewhat similar to δ-DQN, except δ-DQN resamples an independent goal instead of g' for the second term instead. Despite this similarity, HER is biased in stochastic environments and can converge to a low-return policy, while δ-DQN is unbiased.

In this chapter, we introduced a Q-learning approach for multi-goal RL via infinitely sparse rewards. In the next chapter, we introduce an actor critic algorithm, using the same principles.

Chapter 16

Unbiased Multi-Goal Actor Critic with Infinitely Sparse Rewards

16.1 Unbiased Policy Evaluation with Infinitely Sparse Rewards

Similarly to δ-DQN, there exists an actor-critic algorithm for multi-goal environments with infinitely sparse rewards. We start with policy evaluation, then derive the policy gradient algorithm.

16.1.1 Policy evaluation via the successor goal measure $M^\pi(s, g, \mathrm{d}g')$

Learning the value measure $V^\pi(s, \mathrm{d}g)$ defined in Section 13.3.2 directly without bias poses technical issues due to the double dependency of $V^\pi(s, \mathrm{d}g)$ on g (first via the location of the reward, second, via the goal-dependent policy $\pi(.|., g)$). This is discussed in Section 16.1.2. Instead, we learn a richer object, $M^\pi(s, g, \mathrm{d}g')$, the successor goal measure defined in Section 13.2.2. It represents the value function of s if the reward is a Dirac at g' but the agent follows the policy $\pi(a|s, g)$ for goal g. Compared to $V^\pi(s, \mathrm{d}g)$, $M^\pi(s, g, \mathrm{d}g')$ splits the two effects of the goal g in two variables g and g'. As explained in Section 13.3.2, $V^\pi(s, \mathrm{d}g)$ can be derived from $M^\pi(s, g, \mathrm{d}g')$ as $V^\pi(s, \mathrm{d}g) = M^\pi(s, g, \mathrm{d}g)$.

We can now derive an unbiased δ-TD update for M^π. The following theorem is similar to the TD update for successor states in Section 7.3. We learn a model $m_\theta(s, g, g')$ of its density with respect to $\rho_\mathcal{G}$, namely, $M_\theta(s, g, \mathrm{d}g') = m_\theta(s, g, g')\rho_\mathcal{G}(\mathrm{d}g')$.

We consider a measure $\rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g)$ of state-goals pairs, without any additional hypothesis. Typically, ρ_{SG} can be a fixed dataset, or the distribution of states s along trajectories observed when aiming at a goal g. In particular, we do not assume that s and g are independent.

We consider the norm $\|.\|_{\rho_{\mathrm{SG}}}$ defined as:

$$\|m(s, g, g')\rho_\mathcal{G}(\mathrm{d}g')\|^2_{\rho_{\mathrm{SG}}} := \int_{s,g,g'} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g)\rho_\mathcal{G}(\mathrm{d}g')m^2(s, g, g').$$

Moreover, we assume that $\rho_\mathcal{G}$ is the law of the achieved goal $\varphi(s)$ for $(s, g) \sim \rho_{\mathrm{SG}}$.

Theorem 16.1. *Let $M_\theta(s, g, \mathrm{d}g') = m_\theta(s, g, g')\rho_\mathcal{G}(\mathrm{d}g')$ be a current estimate of $M^\pi(s, g, \mathrm{d}g')$. Consider $M^{\mathrm{tar}}(s, g, \mathrm{d}g') = \delta_{\varphi(s)}(\mathrm{d}g') + \mathbb{E}_{a\sim\pi(a|s,g), s'\sim P(\mathrm{d}s'|s,a)}\left[m_\theta(s, g, g')\right]\rho(\mathrm{d}g')$, a target estimate for M_θ defined via the Bellman equation* (13.2.10).

Let (s, a, s', g, g') be samples of the environment such that $a \sim \pi(a|s, g)$, $s' \sim P(s'|s, a)$ and $g' \sim \rho_\mathcal{G}$ is a goal sampled independently. Let $\widehat{\delta\theta}_{\delta\text{-TD}}$ be

$$\widehat{\delta\theta}_{\delta\text{-TD}}(s, s', g, g') := \partial_\theta m_\theta(s, g, \varphi(s)) + \partial_\theta m_\theta(s, g, g')\left(\gamma m_{\mathrm{tar}}(s', g, g') - m_\theta(s, g, g')\right) \quad (16.1.1)$$

Then $\widehat{\delta\theta}_{\delta\text{-TD}}$ is an unbiased estimate of the Bellman error:

$$\mathbb{E}_{s,a,s',g,g'}\left[\widehat{\delta\theta}_{\delta\text{-TD}}(s,a,s',g,g')\right] = -\frac{1}{2}\partial_\theta \|M_\theta - T^\pi M_{\text{tar}}\|^2_{\rho_{\text{SG}}} \tag{16.1.2}$$

In particular, the true successor goal measure $M^\pi(s,g,\mathrm{d}g')$ is a fixed point of this update: if $M_\theta = M^\pi$, then $\mathbb{E}\left[\widehat{\delta\theta}_{\delta\text{-TD}}\right] = 0$.

Proof. This Theorem is a particular case of Theorem 7.11 on TD learning for the successor states operator with features of the state. Indeed, we can apply this theorem with:

- $M^\pi(s,g,\mathrm{d}g') = \psi_* \tilde{M}((s,g),(\mathrm{d}s',\mathrm{d}g'))$ corresponds to M^φ in the theorem.
- $M_\theta(\tilde{s},\mathrm{d}g') = m_\theta(\tilde{s},g')\rho_{\mathcal{G}}(\mathrm{d}g') = m_\theta(s,g,g')\rho_{\mathcal{G}}(\mathrm{d}g')$ is our current estimate of $M^\pi(s,g,\mathrm{d}g')$.
- The target M^{tar} corresponds to the target $(M^\varphi)^{\text{tar}}$ Theorem 7.11, by definition in Definition-Theorem 13.3.
- The sampling distribution $\rho_{\mathcal{G}}$ satisfies the constraint that $\rho_{\mathcal{G}} := \varphi_*\rho_{\text{SG}}$.
- The estimator $\widehat{\delta\theta}_{\text{F}-\text{TD}-\varphi}$ in Theorem 7.11 is equal to the update $\widehat{\delta\theta}_{\delta\text{-TD}}$:

$$\begin{aligned}\widehat{\delta\theta}_{\text{F-TD-}\varphi}(\tilde{s},\tilde{s}',g_2) &:= \partial_\theta m_\theta(\tilde{s},\psi(\tilde{s})) + \partial_\theta m_\theta(\tilde{s},g_2)\left(\gamma m_\theta(\tilde{s}',g_2) - m_\theta(\tilde{s},g_2)\right)\\ &= \partial_\theta m_\theta(s,g,\varphi(s)) + \partial_\theta m_\theta(s,g,g_2)\left(\gamma m_\theta(s',g,g_2) - m_\theta(s,g,g_2)\right)\\ &= \widehat{\delta\theta}_{\delta\text{-TD}}(s,s',g,g_2)\end{aligned}$$

- The norm $\|.\|_{\rho\otimes\tau}$ with $\rho = \rho_{\text{SG}}$ and $\tau = \rho_{\mathcal{G}}$ is equal to the norem $\|.\|_{\rho_{\text{SG}}}$.

Hence, we can apply Theorem 7.11.

The update $\widehat{\delta\theta}_{\delta\text{-TD}}$ is similar to the Forward TD for the successor states operator. It has two parts: the first part $\partial_\theta m_\theta(s,g,\varphi(s))$ represents the reward update for the goal achieved in the current state s, and removes the vanishing reward issue. The second part propagates the rewards along transitions.

We can also define a horizon-n δ-TD(n) update if we have access to longer sub-trajectories $\tau = (g, s_0, a_0, s_1, ...)$. The update at a state s_k in the trajectory is

$$\widehat{\delta\theta}_{\delta\text{-TD}(n)}(\tau,k,g') := \sum_{l=0}^{n-1}\gamma^l \partial_\theta m_\theta(s_k,g,\varphi(s_{k+l})) + \partial_\theta m_\theta(s_k,g,g')\left(\gamma^n m_\theta(s_{k+n},g,g') - m_\theta(s_k,g,g')\right) \tag{16.1.3}$$

where $g' \sim \rho_{\mathcal{G}}$ is sampled independently. This result is not formalized here, but is proved for the successor states operator in Theorem 7.7. The first part increases the value estimate at state s_k for every of the n goals $\varphi(s_k), ..., \varphi(s_{k+n-1})$ achieved in the next n steps: this corresponds to the n-step return with Dirac rewards. The second part propagates the value along transitions. This is similar to HER in that future goals achieved along the trajectory are explicitly used, and could thus improve sample efficiency. However, computational complexity is an issue. In non-multi-goal environments, algorithms such as PPO (Schulman et al., 2017) compute the TD(n) update at every step of the trajectory. This is computable with $O(n)$ forward passes through the value model v_θ, because it only requires to compute $v_\theta(s_0), \ldots, v_\theta(s_n)$. Here we have to compute $m_\theta(s_k, g, \varphi(s_{k+l}))$ for every k and l, leading to an $O(n^2)$ complexity (though this could potentially be sub-sampled as in HER). This makes it slow in practice, and δ-TD(n) was not tested experimentally here.

For this theorem, we sample goals g' independently of τ. In practice, this could be a source of variance, as sampling goals far from the current state should produce close-to-0 V-values. If we instead sample goals from a distribution $\mu(g|s,a)$, this introduces an implicit *scaling* factor $\alpha(s,g)$ to the reward. This is discussed in details in the next section in the case of the V-function.

16.1.2 Obstacles for learning the multi-goal value measure $V^\pi(s,\mathrm{d}g)$ directly via a temporal difference algorithm.

We briefly show why learning V^π directly without bias poses technical issues, stemming from the necessity to work on-policy for V and the resulting correlation between visited states and

goals along trajectories in the training set. As a result, the "obvious" analogue of δ-DQN for V introduces uncontrolled bias and implicit preferences among all possible states s that achieve the same goal g. This problem disappears only if the correspondence between s and g is one-to-one (e.g., $\varphi = \mathrm{Id}$). This is why we learn the more complicated object M^π instead of V^π in Section 16.1.

Assume similarly to Theorem 16.1 that we can sample state-goal pairs from a distribution $\rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g)$ over $\mathcal{S} \times \mathcal{G}$, and define the norm $\| \cdot \|_{\rho_{\mathrm{SG}}}$ as

$$\|V\|_{\rho_{\mathrm{SG}}} = \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \left(\frac{V(s, \mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)} \right)^2 \tag{16.1.4}$$

where $\frac{V(s,\mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)}$ is the density of $V(s, \mathrm{d}g)$ with respect to $\rho_{\mathcal{G}}(\mathrm{d}g)$ (if it does not exist, the norm is infinite). We assume we have a model $V_\theta(s, \mathrm{d}g) = v_\theta(s, g)\rho_{\mathcal{G}}(\mathrm{d}g)$, a target $V_{\mathrm{tar}}(s, \mathrm{d}g) = v_{\mathrm{tar}}(s, g)\rho_{\mathcal{G}}(\mathrm{d}g)$, and want to estimate:

$$\frac{1}{2}\partial_\theta \|V_\theta - T^\pi V_{\mathrm{tar}}\|^2_{\rho_{\mathrm{SG}}} \tag{16.1.5}$$

where $T^\pi V(s, \mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \gamma \mathbb{E}_{s' \sim P^\pi(.|s,g)} V(s', \mathrm{d}g)$. Then, informally, we have:

$$\begin{aligned}
\frac{1}{2}\partial_\theta \|V_\theta - T^\pi V_{\mathrm{tar}}\|^2_{\rho_{\mathrm{SG}}} &= \frac{1}{2}\partial_\theta \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \left(\frac{V_\theta(s, \mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)} - \frac{TV_{\mathrm{tar}}(s, \mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)} \right)^2 \\
&= \frac{1}{2}\partial_\theta \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \left(v_\theta(s, g) - \frac{TV_{\mathrm{tar}}(s, \mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)} \right)^2 \\
&= \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \partial_\theta v_\theta(s, g) \left(v_\theta(s, g) - \frac{TV_{\mathrm{tar}}(s, \mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)} \right) \\
&= \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \partial_\theta v_\theta(s, g) \left(v_\theta(s, g) - \gamma \mathbb{E}_{s' \sim P^\pi(.|s,g)} \left[v_{\mathrm{tar}}(s', g) \right] \right) + \\
&+ \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \partial_\theta v_\theta(s, g) \frac{\delta_{\varphi(s)}(\mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)}
\end{aligned}$$

If we assume that $\rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g)$ has a density $\alpha(s, g)$ with respect to $\rho_{\mathrm{SG}}(\mathrm{d}s) \otimes \rho_{\mathcal{G}}(\mathrm{d}g)$, namely, $\rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) = \alpha(s, g)\rho_{\mathrm{SG}}(\mathrm{d}s)\rho_{\mathcal{G}}(\mathrm{d}g)$, then the second part, corresponding to the Dirac reward, is equal to:

$$\begin{aligned}
\int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s, \mathrm{d}g) \partial_\theta v_\theta(s, g) \frac{\delta_{\varphi(s)}(\mathrm{d}g)}{\rho_{\mathcal{G}}(\mathrm{d}g)} &= \int_{s,g} \rho_{\mathrm{SG}}(\mathrm{d}s) \alpha(s, g) \partial_\theta v_\theta(s, g) \delta_{\varphi(s)}(\mathrm{d}g) \\
&= \int_{s} \rho_{\mathrm{SG}}(\mathrm{d}s) \alpha(s, \varphi(s)) \partial_\theta v_\theta(s, \varphi(s))
\end{aligned}$$

If $\alpha(s, g)$ is always equal to 1, the integral $\int_s \rho_{\mathrm{SG}}(\mathrm{d}s) \partial_\theta v_\theta(s, \varphi(s))$ can be estimated without bias by sampling $s \sim \rho_{\mathrm{SG}}(\mathrm{d}s)$ and estimating $v_\theta(s, \varphi(s))$.

However, the case $\alpha(s, g) = 1$ for every s, g corresponds to s and g independent in ρ_{SG}. This is difficult to realize in practice. Learning V requires actions to be selected on-policy (term $\mathbb{E}_{s' \sim P^\pi(.|s,g)}$ above). If we set a goal g and an initial state s_0, and generate an exploration trajectory by following the policy $\pi(.|., g)$ for that goal, obviously the states s visited by the trajectory are going to be correlated to g, by an unknown factor α. Independence could be ensured by re-sampling a new target goal at each step, independently from the current state, and selecting the next action from the policy for this goal. But such an exploration strategy would be essentially random and would not be efficient.

Assume we just ignore this problem and sample exploration trajectories $(g, s_0, s_1, ...)$ as with other methods, namely, with $g \sim \rho_{\mathcal{G}}$, $s_0 \sim \rho_0(\mathrm{d}s_0|g)$ and $s_{t+1} \sim P^\pi(.|s_t, g)$, and define the estimate

$$\widehat{\delta\theta}_V(s, s', g) = \partial_\theta v_\theta(s, \varphi(s)) + \partial_\theta v_\theta(s, g) \left(\gamma v_{\mathrm{tar}}(s', g) v_\theta(s, g) \right) \tag{16.1.6}$$

similarly to updates of δ-DQN or δ-TD. In that case, we have:

$$\mathbb{E}_{s,g\sim\rho_{\mathrm{SG}},s'\sim P^{\pi}(.|s,g)}\left[\widehat{\delta\theta}_V(s,s',g)\right] = -\partial_\theta \frac{1}{2}\|V_\theta - T^{\pi}_{\alpha}V_{\mathrm{tar}}\|_{\rho_{\mathrm{SG}}} \tag{16.1.7}$$

where:

$$T^{\pi}_{\alpha}V = \alpha(s,g)\delta_{\varphi(s)} + \mathbb{E}_{s'\sim P^{\pi}(.|s,g)}\left[V(s',\mathrm{d}g)\right]. \tag{16.1.8}$$

This is an unbiased estimate of the TD error with the *rescaled reward* $\alpha(s,g)\delta_{\varphi(s)}(\mathrm{d}g)$ instead of $\delta_{\varphi(s)}(\mathrm{d}g)$.

If $\mathcal{S} = \mathcal{G}$ and $\varphi = \mathrm{Id}$, such a reward rescaling is not an issue. Indeed, in that case, $\alpha(s,g)\delta_s(\mathrm{d}g) = \alpha(g,g)\delta_s(\mathrm{d}g)$ as the Dirac measure is nonzero only for $s = g$. This means that for every goal g, the value function for that goal is rescaled by a constant $\alpha(g,g)$, and we learn $\alpha(g,g)V(s,\mathrm{d}g)$ instead of $V(s,\mathrm{d}g)$. This does not change the ranking of state values for each goal g, nor the direction of policy improvement for each goal (but it changes the relative importance of learning different goals g).

On the contrary, if $\mathcal{S} \neq \mathcal{G}$, for a fixed goal g, this implicit reward rescaling can favor some states s over others among the set of states s achieving this goal ($\varphi(s) = g$). For instance, assume the the agent starts at s_0 and wants to reach g, and that there are two states s_1, s_2 such that $\varphi(s_1) = \varphi(s_2) = g$. Even if s_1 is easier to reach than s_2 from s_0, the policy π might *prefer* to reach s_2 because its implicitly rescaled reward is higher. Therefore, the algorithm could converge to non-optimal policies and is not unbiased. It would still learn to reach g, but not necessarily in an optimal way.

16.2 Multi-Goal Actor-Critic

We now derive the multi-goal actor-critic update. The standard policy gradient, as presented in the introduction in Section 1.5 states that if $J(\pi)$ is the expected return:

$$J(\pi) = \mathbb{E}_{s_0\sim\rho_0,\tau\sim\mathbb{P}(\tau|s_0,\pi)}\left[\sum_{t\geqslant 0}\gamma^t R(s_t)\right]$$

and π_θ is a parametric policy, then we can estimate the gradient $\partial_\theta J(\pi_\theta)$ via:

$$\partial_{\theta^\pi} J(\pi_{\theta^\pi}) = \mathbb{E}_{s\sim\nu^\pi,a\sim\pi_{\theta^\pi}(.|s),\tau\sim\mathbb{P}(\tau|s,a)}\left[\partial_{\theta^\pi}\log\pi_{\theta^\pi}(a|s)\,G(\tau)\right] \tag{16.2.1}$$

where $\tau = (s_0, a_0, r_0, s_1, ...) \sim \mathbb{P}(\tau|s_0)$ if $a_t \sim \pi(.|s_t)$, $s_{t+1} \sim P(.|s_t, a_t)$, and where $\nu^\pi(\mathrm{d}s)$ is the discounted occupancy measure starting from $\rho_0(\mathrm{d}s)$. For finite state space ν^π is defined as: $\nu^\pi(s) = (1-\gamma)\mathbb{E}_{s_0\sim\rho_0}\sum_{t\geqslant 0}\gamma^t\mathbb{P}(S_t = s|s_0 = s)$.

We want to derive a similar estimate for the *multi-goal policy gradient*. A possible approach is to directly apply the policy gradient theorem on the extended state space $\mathcal{S}\times\mathcal{G}$ as introduced in Section 13.1.2, with sparse reward $R_\varepsilon(s,g)$. This considers the expected return:

$$J_\varepsilon(\pi) = \mathbb{E}_{g\sim p_{\mathcal{G}},s_0\sim p_0(.|g)}\left[\sum_{t\geqslant 0}\gamma^t R_\varepsilon(s_t,g)|s_0 = s\right]$$

with a sampled goal $g \sim \rho_{\mathcal{G}}(\mathrm{d}g)$ and sampled initial state $s_0 \sim \rho_0(\mathrm{d}s_0|g)$, and subsequent actions sampled from the policy for g, which leads to the following gradient:

$$\partial_{\theta^\pi} J_\varepsilon(\pi_{\theta^\pi}) = \mathbb{E}_{s\sim\nu^\pi(s|g),a\sim\pi_{\theta^\pi}(.|s,g),\tau\sim\mathbb{P}(\tau|s,a,g)}\left[\partial_{\theta^\pi}\log\pi_{\theta^\pi}(a|s,g)\,R(\tau,g)\right] \tag{16.2.2}$$

where $\tau = (s_0, a_0, r_0, s_1, ...) \sim \mathbb{P}(\tau|s_0, g)$ if $a_t \sim \pi(.|s_t, g)$, $s_{t+1} \sim P(.|s_t, a_t)$, and where $\nu^\pi(\mathrm{d}s|g)$ (introduced in Theorem 13.2)is the discounted occupancy measure following the policy aiming at g $\pi(.|g)$, and starting from $\rho_0(\mathrm{d}s)$.

This approach has a main limitation: there is no generalization between goals. If a goal g is sampled as a target at the beginning of the trajectory, no learning occurs until g is reached

during that trajectory. Because of the issue of *vanishing rewards*, the probability of reaching a goal is almost 0, and no learning is possible.

Hindsight Policy Gradients (Rauber et al., 2019) tackles this issue via importance sampling. It re-uses trajectories observed while aiming at a goal g for learning how to reach a goal g', by reweighting with factors $\prod_{1\leqslant i\leqslant t} \frac{\pi(a_t|s_t,g')}{\pi(a_t|s_t,g)}$. This approach leads to unbiased policy gradient, and allow some generalization across goals. Still, this approach has two main limitations: first, as explained by the authors, importance sampling on a long trajectory can lead to large variance because of the product of probability ratios. Then, the main issue is that it *does not solve the issue of vanishing reward*. Indeed, if the new goal g' is sampled independently of the observed trajectory, the probability that the goal corresponds to a state in the trajectory is 0. On the contrary, if the goal g' is taken as one of the achieved goals in the trajectory (which is the author's choice in their experiments), the bias occurs as for Hindsight Experience Replay, as discussed in Chapter 14.

Our goal is therefore to derive an unbiased multi-goal policy gradient algorithm, removing the issue of vanishing reward. We now derive an estimate of $\partial_\theta J(\pi_\theta)$ for a parametric policy $\pi_\theta(a|s,g)$. We assume access to transition samples (s,a,s',g) such that $a \sim \pi(.|s,g)$, $s' \sim P(\mathrm{d}s'|s,a)$ and s is sampled from the goal-dependent *discounted visitation frequencies* $\nu^\pi(\mathrm{d}s|g) = (1-\gamma)\sum_{t\geqslant 0}\gamma^t \rho_0(\mathrm{d}s_0|g)(P^\pi)^t(\mathrm{d}s|s_0,g)$.

In the introduction (section 1.5), we introduced the actor-critic update in the non-multi-goal setting: $\widehat{\delta\theta}_{\mathrm{AC}}(s,a,r,s') := \log \pi_{\theta^\pi}(a|s)\,(r_s + \gamma V^\pi(s') - V^\pi(s))$, which is an unbiased policy gradient estimate. In practice, V^π can be estimated with function approximations.

We similarly define the multi-goal actor critic update with infinitely sparse rewards by using the model $m(s,g,g)$ as an estimate of the values. This leads to

$$\widehat{\delta\theta}_{\delta\text{-AC}}(s,a,s',g) := \partial_\theta \log \pi_\theta(a|s,g)\,(\gamma m_{\theta_M}(s',g,g) - m_{\theta_M}(s,g,g)) \tag{16.2.3}$$

where $m_{\theta_M}(s,g,g')$ is the model of the value density learned in Section 16.1. This update, together with the one for m in Theorem 16.1, make up the δ-Actor-Critic algorithm (Algorithm 15). We can similarly define a PPO algorithm (Appendix 16.A), used in the experiments.

Intuitively, if $m_{\theta_M}(s,g,g)\lambda(\mathrm{d}g)$ approximates $V^\pi(s,\mathrm{d}g)$ as a measure, then

$$\mathbb{E}_{s,a,s',g}\left[\widehat{\delta\theta}_{\delta\text{-AC}}(s,a,s',g)\right]$$

approximates $\partial_\theta J(\pi_\theta)$. This is formalized in the following theorem. Here, we suppose that the continuous assumption (Assumption 13.1) introduced in Section 13.3 is true. This means that every probability measure is supposed to be continuous with respect to the Lebesgue measure $\lambda(.)$. Informally, in that case, we show that three policy gradients are equivalent (up to a constant, and a reweighting of the goal distribution):

- The gradient of the expected return with infinitely sparse rewards $\partial_\theta J(\pi_\theta)$, defined in Section 13.3.3.
- The gradient of the expected return with sparse reward $\partial_\theta J_\varepsilon(\pi_\theta)$ when $\varepsilon \to 0$
- The actor critic update $\mathbb{E}\left[\widehat{\delta\theta}_{\delta\text{-AC}}(s,a,s',g)\right]$ when $m_{\theta_M}(s,g,g')$ accurately approximates the successor goal measure.

Theorem 16.2. *We assume Assumption 13.1 (continuous assumption). Let $\pi_\theta(a|s,g)$ be a parametrized goal-dependent policy, defined for every $\theta \in \Theta$. We assume that for every $\theta \in \Theta, s \in \mathcal{S}, g \in \mathcal{G}, a \in \mathcal{A}$, $\pi_\theta(a|s,g) > 0$. Moreover, we assume $\pi_\theta(a|s,g)$ is a continuous function of a,s,g,θ, and continuously differentiable with respect to θ.*

We define the stochastic actor critic $\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}$ for estimate n as:

$$\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g) := \partial_\theta \log \pi_\theta(a|s,g)\,(\gamma \hat{v}_n(s',g) - \hat{v}_n(s,g)) \tag{16.2.4}$$

Algorithm 15 One-step δ-Actor-Critic

Input: Model $m_{\theta_M}(s,g)$; policy π_θ; goal function φ; T the maximum trajectory length
Get a goal g and an initial state s_0 from the environment
for $0 \leqslant t \leqslant T$ steps **do**
  Sample $a_t \sim \pi(a|s_t,g)$
  Execute action a_t and observe the next state s_{t+1}
  Sample an independent goal $g' \sim \rho_{\mathcal{G}}(\mathrm{d}g')$
  $\widehat{\delta\theta}_{\delta\text{-TD}} := \partial_\theta m_{\theta_M}(s_t,g,\varphi(s_t)) + \partial_\theta m_{\theta_M}(s_t,g,g')\left(\gamma m_{\theta_M}(s_{t+1},g,g') - m_{\theta_M}(s_t,g,g')\right)$
  $\widehat{\delta\theta}_{\delta\text{-AC}} = \gamma^t \times \partial_\theta \log \pi_{\theta_\pi}(a_t|s_t,g)\left(\gamma m(s_{t+1},g,g) - m(s_t,g,g)\right)$
  $\theta_M \leftarrow \theta_M + \eta_M \widehat{\delta\theta}_{\delta\text{-TD}}$
  $\theta_\pi \leftarrow \theta_\pi + \eta_\pi \widehat{\delta\theta}_{\delta\text{-AC}}$
end for

We define $\tilde{\rho}(\mathrm{d}g) := \frac{1}{c}p_{\mathcal{G}}^2(g)\lambda(\mathrm{d}g)$ with $c := \int_g p_{\mathcal{G}}^2(g)\lambda(\mathrm{d}g)$. We assume access to samples $g \sim \tilde{\rho}(\mathrm{d}g)$, $s_0 \sim \rho_0(\mathrm{d}s|g) = p_0(s_0|g)\lambda(\mathrm{d}s_0)$, $s \sim \nu^{\pi_\theta}(\ s|s_0,g)$, $a \sim \pi(a|s,g)$ and $s' \sim P(\mathrm{d}s'|s,a)$. Let $(\hat{v}_n(s,g))_{n\geqslant 0}$ be a sequence of densities, such that $\lambda(\mathrm{d}s)\hat{v}_n(s,g)\rho(\mathrm{d}g)$ converges weakly to $\lambda(\mathrm{d}s)V^{\pi_\theta}(s,\mathrm{d}g)$. Then, we have:

$$\lim_{n\to\infty} \mathbb{E}_{g\sim\tilde{\rho},s\sim\nu^\pi(.|g),a\sim\pi_\theta(.|s,g),s'\sim P(.|s,a)}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right] = \frac{1-\gamma}{c}\partial_\theta J(\pi_\theta) \tag{16.2.5}$$

Moreover, we have:

$$\lim_{\varepsilon\to 0}\frac{1}{\lambda(\varepsilon)}\partial_\theta J_\varepsilon(\pi_\theta) = \partial_\theta J(\pi_\theta) \tag{16.2.6}$$

Proof. We first compute $\partial_\theta J(\pi_\theta)$. We have:

$$J(\pi_\theta) = \int_{s_0,g} V^{\pi_\theta}(s_0,\mathrm{d}g)p_{\mathcal{G}}(g)p_0(s_0|g)\lambda(\mathrm{d}s_0) \tag{16.2.7}$$

We know that $V^\pi(s,\mathrm{d}g) = \delta_{\varphi(s)}(\mathrm{d}g) + \tilde{m}^\pi(s,g,g)\lambda(\mathrm{d}g)$. We define for simplicity $v^\pi(s,g) = \tilde{m}^\pi(s,g,g)$. Moreover, we know, by taking $g' = g$ in Equation (13.3.3) in Lemma 13.4 that for every (s,g), we have:

$$v^{\pi_\theta}(s,g) = \gamma\int_a \lambda(\mathrm{d}a)\pi(a|s,g)\left(\tilde{p}(g|s,a) + \int_{s'}\lambda(\mathrm{d}s')p(s'|s,a)v^{\pi_\theta}(s',g)\right) \tag{16.2.8}$$

We define $F(s,g,\theta) = \gamma\int_a \pi_\theta(a|s,g)\tilde{p}(g|s,a)$. The function F_θ is continuous in s and g and continuously differentiable in θ, because $\tilde{p}$ is and π_θ are continuous, π_θ is continuously differentiable, and $\mathcal{A}$ is compact. From the proof of Equation (13.3.3) in Lemma 13.4, we know that $F(s,g,\theta)$ is the density of $\gamma\int_{a,s'}\pi(a|s,g)p(s'|s,a)\delta_{\varphi(s')}(\mathrm{d}g)$ with respect to the Lebesgue measure $\lambda(\mathrm{d}g)$. This remark will be used later in the computation. We now have:

$$v^{\pi_\theta}(s,g) = F(s,g,\theta) + \gamma\int_{a,s'}\pi_\theta(a|s,g)p(s'|s,a)v^{\pi_\theta}(s',g)\lambda(\mathrm{d}a,\mathrm{d}s') \tag{16.2.9}$$

Therefore:

$$v^{\pi_\theta}(s,g) = F(s,g,\theta) + \sum_{k\geqslant 1}\gamma^k\int_{a_0,s_1,\ldots}\lambda(\mathrm{d}a_0,\mathrm{d}s_1,\ldots,\mathrm{d}s_k)\left(\prod_{i=0}^{k-1}\pi_\theta(a_i|s_i,g)p(s_{i+1}|s_i,a_i)\right)F(s_k,g,\theta) \tag{16.2.10}$$

because it is a fixed point of v^π equation, and is the only fixed point which is continuous and bounded, because the space is compact, and π_θ, p are continuous an bounded.

Equation (16.2.10) can also be written:

$$v^{\pi_\theta}(s,g) = \frac{1}{1-\gamma}\int_{s'}\nu^{\pi_\theta}(\mathrm{d}s'|s,g)F(s',g,\theta) \tag{16.2.11}$$

Because $F(s',g,\theta)$ is continuously differentiable in θ and the support of ν^π is compact, $v^{\pi_\theta}(s,g)$ is differentiable. We will now now derive a fixed point equation on $\partial_\theta v^{\pi_\theta}$. We differentiate equation (16.2.9)

and we get:

$$\partial_\theta v^{\pi_\theta}(s,g) = \partial_\theta F(s,g,\theta) + \gamma \int_{a,s} \lambda(\mathrm{d}a,\mathrm{d}s)\partial_\theta \pi_\theta(a|s,g)p(s'|s,a)v^{\pi_\theta}(s',g) + \tag{16.2.12}$$

$$+\gamma \int_{a,s} \lambda(\mathrm{d}a,\mathrm{d}s)\pi_\theta(a|s,g)p(s'|s,a)\partial_\theta v^{\pi_\theta}(s',g) \tag{16.2.13}$$

We define $G(s,g,\theta) := \partial_\theta F(s,g,\theta) + \gamma \int_{a,s'} \lambda(\mathrm{d}a,\mathrm{d}s')\partial_\theta \pi_\theta(a|s,g)p(s'|s,a)v^{\pi_\theta}(s',g)$. We have:

$$\partial_\theta v^{\pi_\theta}(s,g) = G(s,g,\theta) + \gamma \int_{a,s'} \lambda(\mathrm{d}a,\mathrm{d}s')\pi_\theta(a|s,g)p(s'|s,a)\partial_\theta v^{\pi_\theta}(s',g) \tag{16.2.14}$$

Similarly to the derivation of v^π from its fixed point equation (from (16.2.9) to (16.2.10)):

$$\partial_\theta v^{\pi_\theta}(s,g) = G(s,g,\theta) + \sum_{k\geqslant 1}\gamma^k \int_{a_0,s_1,\ldots} \lambda(\mathrm{d}a_0,\mathrm{d}s_1,\ldots,\mathrm{d}s_k)\left(\prod_{i=0}^{k-1}\pi_\theta(a_i|s_i,g)p(s_{i+1}|s_i,a_i)\right)G(s_k,g,\theta) \tag{16.2.15}$$

$$= \frac{1}{1-\gamma}\int_{s'} \nu^{\pi_\theta}(\mathrm{d}s'|s,g)G(s',g,\theta) \tag{16.2.16}$$

We now compute $\partial_\theta J(\pi_\theta)$. We have:

$$\begin{aligned}
\partial_\theta J(\theta) &= \partial_\theta\left(\int_{s_0,g}\lambda(\mathrm{d}s_0)p_{\mathcal{G}}(g)p_0(s_0|g)V^{\pi_\theta}(s_0,\mathrm{d}g)\right)\\
&= \partial_\theta\left(\int_{s_0,g}\lambda(\mathrm{d}s_0)p_{\mathcal{G}}(g)p_0(s_0|g)\left(\delta_{\varphi(s_0)}(\mathrm{d}g) + v^{\pi_\theta}(s_0,g)\lambda(\mathrm{d}g)\right)\right)\\
&= \partial_\theta\left(\int_{s_0,g}\lambda(\mathrm{d}s_0,\mathrm{d}g)p_{\mathcal{G}}(g)p_0(s_0|g)v^{\pi_\theta}(s_0,g)\right)\\
&= \int_{s_0,g}\lambda(\mathrm{d}s_0,\mathrm{d}g)p_{\mathcal{G}}(g)p_0(s_0|g)\partial_\theta v^{\pi_\theta}(s_0,g)\\
&= \frac{1}{1-\gamma}\int_{s_0,s,g}\lambda(\mathrm{d}s_0,\mathrm{d}g)p_{\mathcal{G}}(g)p_0(s_0|g)\nu^{\pi_\theta}(\mathrm{d}s|s_0,g)G(s,g,\theta)
\end{aligned} \tag{16.2.17}$$

We now show that:

$$G(s,g,\theta)\lambda(\mathrm{d}g) = \gamma\int_{s',a} V(s',\mathrm{d}g)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a) \tag{16.2.18}$$

While this result might seem to come out of nowhere, remember that $F(s,g,\theta)$ was derived above as the measure density of $\gamma\int_{s',a}\pi(a|s,g)p(s'|s,g)\delta_{\varphi(s')}(\mathrm{d}g)$ with respect to Lebesgue measure. With the following informal computation, we have:

$$\begin{aligned}
&G(s,g,\theta)\lambda(\mathrm{d}g)\\
&= \lambda(\mathrm{d}g)\partial_\theta\frac{1}{\lambda(\mathrm{d}g)}\int_{s',a}\lambda(\mathrm{d}s',\mathrm{d}a)\gamma\pi_\theta(a|s,g)p(s'|s,a)\delta_{\varphi(s')}(\mathrm{d}g) + \gamma\int_{s',a}\lambda(\mathrm{d}s',\mathrm{d}a)v^{\pi_\theta}(s',g)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)\\
&= \int_{s',a}\lambda(\mathrm{d}s',\mathrm{d}a)\gamma\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)\left(\delta_{\varphi(s)}(\mathrm{d}g) + v^{\pi_\theta}(s',g)\lambda(\mathrm{d}g)\right)\\
&= \int_{s',a}\lambda(\mathrm{d}s',\mathrm{d}a)\gamma V^{\pi_\theta}(s',\mathrm{d}g)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)
\end{aligned}$$

This derivation is informal because we differentiated through a density: we use $\lambda(\mathrm{d}g)\partial_\theta\frac{1}{\lambda(\mathrm{d}g)} = \partial_\theta$. We now derive the result rigorously. Let $f(g)$ be a continuous test function. We have:

$$\begin{aligned}
&\int_g f(g)G(s,g,\theta)\lambda(\mathrm{d}g) =\\
&= \int_g \lambda(\mathrm{d}g)f(g)\left(\gamma\int_a\lambda(\mathrm{d}a)\partial_\theta\pi_\theta(a|s,g)\tilde{p}(g|s,a) + \gamma\int_{a,s'}\lambda(\mathrm{d}s',\mathrm{d}a)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)v^{\pi_\theta}(s',g)\right)
\end{aligned}$$

We consider the first part. The following is the reversed derivation of $\tilde{p}$ in Equations (13.3.26)-(13.3.29).We have:

$$\begin{aligned}
\int_g\lambda(\mathrm{d}g)f(g)\left(\gamma\int_a\partial_\theta\pi_\theta(a|s,g)\tilde{p}(g|s,a)\right) &= \gamma\int_{g,a}\lambda(\mathrm{d}g,\mathrm{d}a)f(g)\partial_\theta\pi_\theta(a|s,g)\tilde{p}(g|s,a)\\
&= \gamma\int_{g,a,s'}\lambda(\mathrm{d}a,\mathrm{d}s')f(g)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)\delta_{\varphi(s')}(\mathrm{d}g)
\end{aligned}$$

Therefore:

$$\int_g f(g)G(s,g,\theta)\lambda(\mathrm{d}g) = \gamma \int_{g,a,s'} \lambda(\mathrm{d}a,\mathrm{d}s') f(g)\partial_\theta \pi_\theta(a|s,g)p(s'|s,a)\left(\delta_{\varphi(s')}(\mathrm{d}g) + v^{\pi_\theta}(s',g)\lambda(\mathrm{d}g)\right)$$
$$= \gamma \int_{g,a,s'} \lambda(\mathrm{d}a,\mathrm{d}s') f(g)\partial_\theta \pi_\theta(a|s,g)p(s'|s,a)V^{\pi_\theta}(s',\mathrm{d}g)$$

This establishes equation (16.2.18). Finally, from (16.2.17) and (16.2.18), we have:

$$\partial_\theta J(\pi_\theta) = \frac{1}{1-\gamma}\int_{g,s_0,s,a,s'} \lambda(\mathrm{d}s_0)\gamma p_{\mathcal{G}}(g)p_0(s_0|g)\nu^{\pi_\theta}(\mathrm{d}s|s_0,g)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)V^{\pi_\theta}(s',\mathrm{d}g)$$

We now show that Then, we have: $\lim_{n\to\infty}\mathbb{E}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right] = \frac{1-\gamma}{c}\partial_\theta J(\pi_\theta)$ and $\lim_{\varepsilon\to 0}\frac{1}{\lambda(\varepsilon)}\partial_\theta J_\varepsilon(\pi_\theta) = \partial_\theta J(\pi_\theta)$.

We first compute $\partial_\theta J_\varepsilon(\pi_\theta)$. We apply the policy gradient theorem (Sutton and Barto, 2018) to the augmented state augmented (non-multi goal) environment $\tilde{\mathcal{S}} = \mathcal{S}\times\mathcal{G}$, and we have, for any *baseline* function $b(\tilde{s})$ with $\tilde{s}\in\tilde{\mathcal{S}}$:

$$\partial_\theta J_\varepsilon(\pi_\theta) \tag{16.2.19}$$
$$= \frac{1}{1-\gamma}\int_{\tilde{s}_0,\tilde{s},a,\tilde{s}'} \lambda(\mathrm{d}a)\rho_0(\tilde{s}_0)\nu^{\pi_\theta}(\mathrm{d}\tilde{s}|\tilde{s}_0)\tilde{P}(\mathrm{d}\tilde{s}'|\tilde{s},a)\partial_\theta\pi_\theta(a|\tilde{s})\left(R_\varepsilon(\tilde{s}) + \gamma V^\pi_\varepsilon(\tilde{s}') - b(\tilde{s})\right) \tag{16.2.20}$$
$$= \frac{1}{1-\gamma}\int_{g,s_0,s,a,s'} \lambda(\mathrm{d}a)\rho_{\mathcal{G}}(\mathrm{d}g)\rho_0(\mathrm{d}s_0|g)\nu^{\pi_\theta}(\mathrm{d}s|s_0,g)\tilde{P}(\mathrm{d}s'|s,a)\partial_\theta\pi_\theta(a|s,g)\left(R_\varepsilon(s,g) + \gamma V^\pi_\varepsilon(s',g) - b(s,g)\right) \tag{16.2.21}$$

with the change of variable $\tilde{s} = (s,g)$, $\tilde{s}' = (s',g)$, $\tilde{s}_0 = (s_0,g)$. We use the baseline $b(s,g) = R_\varepsilon(s,g)$, and we have:

$$\frac{1}{\lambda(\varepsilon)}\partial_\theta J_\varepsilon(\pi_\theta) = \frac{1}{1-\gamma}\int_{s_0,s,a,s',g} \lambda(\mathrm{d}s_0,\mathrm{d}a,\mathrm{d}g)p_{\mathcal{G}}(g)p(s_0|g)\nu^\pi(\mathrm{d}s|s_0,g)\partial_\theta\pi_\theta(a|s,g)\left(\frac{\gamma V_\varepsilon(s',g)}{\lambda(\varepsilon)}\right) \tag{16.2.22}$$

We now compute $\mathbb{E}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right]$. We have:

$$\mathbb{E}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right] \tag{16.2.23}$$
$$= \int_{s_0,s,a,s',g} \lambda(\mathrm{d}g,\mathrm{d}s_0,\mathrm{d}s',\mathrm{d}a)\frac{1}{c}p_{\mathcal{G}}(g)^2 p(s_0|g)\nu^\pi(\mathrm{d}s|s_0,g)\pi_\theta(a|s,g)\partial_\theta\log\pi_\theta(a|s,g)(\gamma v_n(s',g) - v_n(s,g)) \tag{16.2.24}$$
$$= \int_{s_0,s,a,s',g} \lambda(\mathrm{d}g,\mathrm{d}s_0,\mathrm{d}s',\mathrm{d}a)\frac{1}{c}p_{\mathcal{G}}(g)^2 p(s_0|g)\nu^\pi(s|s_0,g)\partial_\theta\pi_\theta(a|s,g)(\gamma v_n(s',g) - v_n(s,g)) \tag{16.2.25}$$

We know that for every *baseline* function $b(s,g)$:

$$\int_a \partial_\theta\pi_\theta(a|s,g)b(s,g) = b(s,g)\partial_\theta\int_a \pi_\theta(a|s,g) = 0 \tag{16.2.26}$$

We define $b(s,g) = v_n(s,g)$, and we have:

$$\mathbb{E}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right] = \gamma\int_{s_0,s,a,s',g} \lambda(\mathrm{d}g,\mathrm{d}s_0,\mathrm{d}s',\mathrm{d}a)\frac{1}{c}p_{\mathcal{G}}(g)^2 p_0(s_0|g)\nu^\pi(\mathrm{d}s|s_0,g)\partial_\theta\pi_\theta(a|s,g)p(s'|s,a)v_n(s',g) \tag{16.2.27}$$

We know from Lemma 13.4 that $\nu^\pi(\mathrm{d}s|s_0,g) = (1-\gamma)\delta_{s_0}(\mathrm{d}s) + q^\pi(s|s_0,g)\lambda(\mathrm{d}s)$ where q^π is continuous, bounded, and with compact support as a density. Therefore, for any goal g, if we take the expectation with respect to $s_0 \sim p_0(s_0|g)$:

$$\int_{s_0} p_0(s_0|g)\nu^\pi(\mathrm{d}s|s_0,g) = \int_{s_0}(1-\gamma)p_0(s_0|g)\delta_{s_0}(\mathrm{d}s) + q^\pi(s|s_0,g)\lambda(\mathrm{d}s) \tag{16.2.28}$$
$$= \left((1-\gamma)p_0(s|g) + \int_{s_0} p_0(s_0|g)q^\pi(s|s_0,g)\right)\lambda(\mathrm{d}s) \tag{16.2.29}$$
$$= \tilde{q}^\pi(s|g)\lambda(\mathrm{d}s) \tag{16.2.30}$$

where $\tilde{q}^\pi$ is continuous, bounded and with compact support as a density. Moreover, $p_{\mathcal{G}}$ and $p_0(s_0|g)$ are continuous bounded functions. Therefore:

$$\mathbb{E}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right] = \gamma\int_{s',g}\lambda(\mathrm{d}s',\mathrm{d}g)v_n(s',g)\frac{1}{c}p_{\mathcal{G}}(g)^2\int_{s,a}\lambda(\mathrm{d}s,\mathrm{d}a)\tilde{q}(s,g)p(s'|s,a)\partial_\theta\pi_\theta(a|s,g) \tag{16.2.31}$$

and similarly:

$$\frac{1}{\lambda(\varepsilon)}\partial_\theta J_\varepsilon(\pi_\theta) = \frac{\gamma}{1-\gamma}\int_{s,a,s',g} \lambda(\mathrm{d}s,\mathrm{d}a,\mathrm{d}s',\mathrm{d}g)p_{\mathcal{G}}(g)\tilde{q}(s|g)\partial_\theta\pi(a|s,g)p(s'|s,a)\frac{V_\varepsilon(s',g)}{\lambda(\varepsilon)} \tag{16.2.32}$$

$$= \frac{\gamma}{1-\gamma}\int_{s',g} \lambda(\mathrm{d}s',\mathrm{d}g)\frac{V_\varepsilon(s',g)}{\lambda(\varepsilon)}p_{\mathcal{G}}(g)\int_{s,a}\lambda(\mathrm{d}s,\mathrm{d}a)\tilde{q}(s,g)p(s'|s,a)\partial_\theta\pi_\theta(a|s,g) \tag{16.2.33}$$

We know that the two measures on $K_{\mathcal{S}} \times \mathcal{G}$ defined as $\lambda(\mathrm{d}s,\mathrm{d}g)\frac{V_\varepsilon(s,g)}{\lambda(\varepsilon)}$ and $\lambda(\mathrm{d}s)v_n(s,g)\rho(\mathrm{d}g) = \lambda(\mathrm{d}s,\mathrm{d}g)v_n(s,g)p_{\mathcal{G}}(g)$ converges weakly to $\lambda(\mathrm{d}s)V^{\pi_\theta}(s,\mathrm{d}g)$. Moreover, $(s',g) \to \int_{s,a}\tilde{q}(s,g)p(s'|s,a)\partial_\theta\pi_\theta(a|s,g)$ is continuous and bounded because $\tilde{q}$, p and $\partial_\theta\pi_\theta$ are continuous, bounded, and the supports are compact. Therefore, from equation (16.2.31):

$$\mathbb{E}\left[\widehat{\delta\theta}^{(n)}_{\delta\text{-AC}}(s,a,s',g)\right] \to_{n\to\infty} \frac{\gamma}{c}\int_{s,a,s',g}\lambda(\mathrm{d}s,\mathrm{d}s',\mathrm{d}a)p_{\mathcal{G}}(g)V^{\pi_\theta}(s',\mathrm{d}g)\tilde{q}(s,g)p(s'|s,a)\partial_\theta\pi_\theta(a|s,g) \tag{16.2.34}$$

$$= \frac{\gamma}{c}\int_{s_0,s,a,s',g}\lambda(\mathrm{d}s_0,\mathrm{d}s,\mathrm{d}a,\mathrm{d}s')p_{\mathcal{G}}(g)p_0(s_0|g)\nu^\pi(\mathrm{d}s|s_0,g)V^{\pi_\theta}(s',\mathrm{d}g)\gamma p(s'|s,a)\partial_\theta\pi_\theta(a|s,g) \tag{16.2.35}$$

$$= \frac{1-\gamma}{c}\partial_\theta J(\pi_\theta) \tag{16.2.36}$$

and from equation (16.2.33)

$$\frac{1}{\lambda(\varepsilon)}\partial_\theta J_\varepsilon(\pi_\theta) \to_{\varepsilon\to 0} \partial_\theta J(\pi_\theta) \tag{16.2.37}$$

This concludes the proof.

16.3 Experiments

In this section, we experiment the introduced method δ-DQN introduced in Chapter 15, and the multi-goal actor-critic method introduced in this chapter. We consider two settings. First, the `Torus` environment, then the `FetchReach` robotic arm environment. More experimental details are in Appendix 16.A.

The `Torus` environment. We first define the `Torus`(n) environment, which is a continuous version of the *flipping coin* environment introduced in (Andrychowicz et al., 2017). The state space is the n-th dimensional torus, represented as $\mathcal{S} = [0,1)^n$, and can be obtained from the n-dimensional hypercube by gluing the opposite faces together. The action space is $\mathcal{A} = \{1,\dots,n\}\times\{-\alpha,\alpha\}$ and action $a = (i,u)$ in state s moves the position on the axis i of a quantity u, then the environment adds a Gaussian noise. Formally $s' \sim \big((s + u.e_i + \mathcal{N}(0,\sigma^2)) \mod 1\big)$, where $(e_j)_{1\leqslant j\leqslant n}$ is the canonical basis $(e_i)_k = \mathbb{1}_{i=k}$. We consider the environment in dimensions $n = 4$ and $n = 6$. We also consider the modified environment with the *freeze* action described in Section 14.2. For every environment, we observe trajectories of length 200, and the reported metric is the rescaled negative L1 distance to the goal at the end of trajectory $-\frac{1}{n}\|s-g\|_1$. The experimental details are in Appendix 16.A.

We compare UVFA, HER, δ-DQN, and δ-PPO (defined in Appendix 16.A based on δ-AC). Each algorithm fails in some environment: additional experiments in the Appendix show that δ-DQN and δ-PPO are both failing to learn when the dimension of the torus increases, while HER is still able to learn. This is discussed in Section 16.4. While UVFA, HER and δ-DQN are similar algorithms and can be compared as actor-critic methods handle the trajectory samples in a different way from Q-learning methods. Still, we observe that δ-PPO learns successfully in the same environments as δ-DQN, and also failing when δ-DQN does.

The FetchReach environment. The FetchReach environment (Plappert et al., 2018) is a robotic arm environment in which the objective is for the extremity of the arm to reach a given 3D position. The environment is deterministic, so HER is expected to perform well. Here, all methods learn successfully. We also experimented δ-DQN and δ-PPO on more complex environments of the same robotic suite, such as FetchPush, but both methods fail in this setting, while HER was successful.

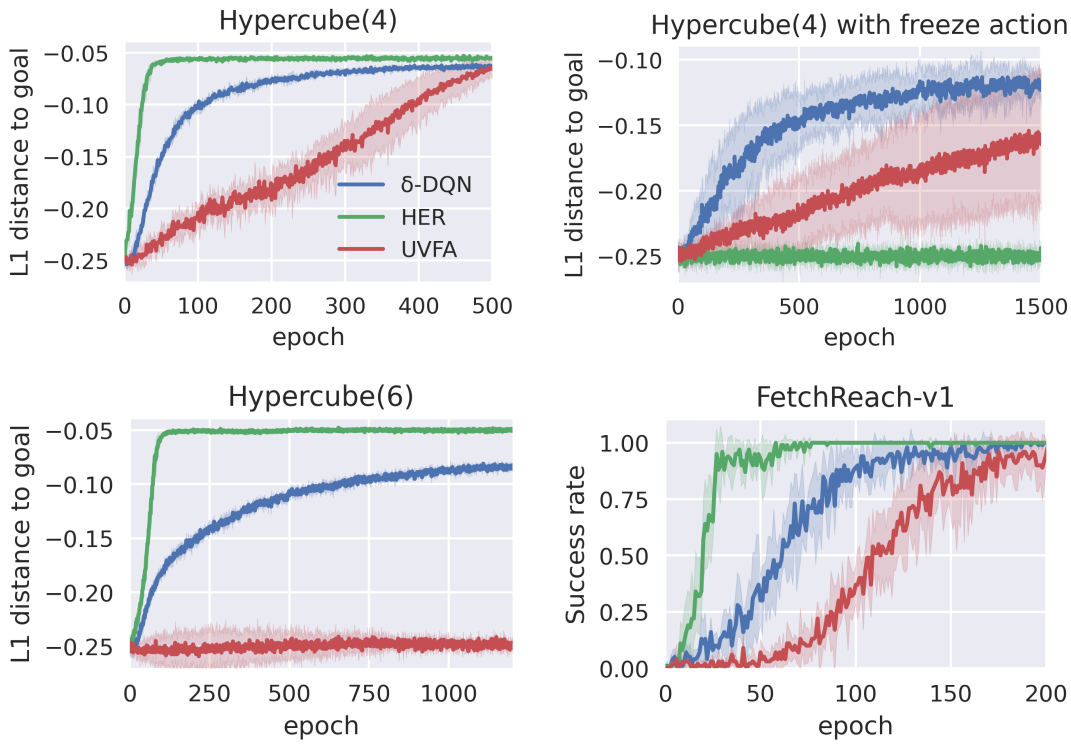


Figure 16.1: We compare UVFA, HER, δ-DQN in toy environments. We observe different regimes: with a highly stochastic environment (`Torus` with freeze action), HER is unable to learn because of its bias, whereas UVFA and δ-DQN are. When the state dimension becomes too large (`Torus`(6)), UVFA is unable to learn because of the vanishing reward issue. In environments in which HER is able to learn, it is the most efficient method, and δ-DQN is always performing better than UVFA.

16.4 Limitations and Future Work

The algorithms using infinitely sparse rewards always perform better than UVFA, and perform better than HER in environments designed to exhibit the HER bias issue. But they do not perform as well as HER in some standard environments, and are unable to learn at all in more complex environments such as FetchPush. We discuss two technical limitations of δ-DQN and δ-Actor-Critic.

The first issue is the function approximation. Learning the models

$$Q_\theta(s, a, \mathrm{d}g) = q_\theta(s, a, g)\rho(\mathrm{d}g)$$

of Q^* and

$$M^\pi_\theta(s, g_1, \mathrm{d}g_2) = m_\theta(s, g_1, g_2)\rho(\mathrm{d}g_2)$$

of M^π requires approximating a Dirac distribution (when $g_2 = \varphi(s)$) with a continuous density. The theorems justify this, but in practice the functions m_θ and q_θ have to reach multiple orders of magnitude (high values close to the goal, low everywhere else), and the values need to be accurate in these two regimes. Representing multiple orders of magnitude in neural networks may require a well-suited family of parametric functions.

A second issue is variance. The Dirac rewards remove the *infinite* variance of vanishing rewards in UVFA when $\varepsilon \to 0$. But the variance of the remaining term can be high. Consider the tabular case (15.3.2)–(15.3.3): δ-DQN learns significantly faster than UVFA on the *diagonal* $Q(s, a, g)$ when $g = s$, thanks to the Diracs. But this does not change the way the reward is

propagated to other states, due to the independent sampling of g in (15.3.3). Selecting goals g more correlated to the state s as in HER could also be helpful, but this is not obvious to do without re-introducing HER-style bias.

Appendix

16.A Experiments Details

In this section, we present the experiment details of Section 16.3. Every experiment was performed on a single GPU.

The `Torus(n)` environment The state space of the `Torus`(n) environment is the n-th dimensional torus, $\mathcal{S} = [0,1)^n$, and can be obtained from the n-dimensional hypercube by gluing the opposite faces together. If the current state is $s = (s_1, ..., s_n)$, we define the observation of the agent as

$$(\cos(2\pi s_1), ..., \cos(2\pi s_n), \sin(2\pi s_1), ..., \sin(2\pi s_n)) \in [-1,1]^{2n}.$$

We use this representation in order to remove the discontinuity of the representation $[0,1)^n$. This representation contains all the information of the state s and the environment is still fully observable (and not partially observable). The action space is $\mathcal{A} = \{1, \dots, n\} \times \{-\alpha, \alpha\}$ and action $a = (i, u)$ in state s moves the position on the axis i of a quantity u, then the environment adds a Gaussian noise. Formally $s' \sim \big((s + u.e_i + \mathcal{N}(0, \sigma^2)) \mod 1\big)$, where $(e_j)_{1 \leqslant j \leqslant n}$ is the canonical basis $(e_i)_k = \mathbb{1}_{i=k}$. In practice, we take $\alpha = 0.1$, and $\sigma = \frac{0.1}{n}$. The reward is $R_\varepsilon(s, g) = \mathbb{1}_{\|s-g\| \leqslant \varepsilon}$ where $\|.\|$ is the rescaled L1 distance in the Torus: $\|s - g\| = \frac{1}{n} \sum_{i=1}^n \min((s_i - g_i) \mod 1, |((s_i - g_i) \mod 1) - 1|)$. In practice, we use $\varepsilon = 0.05$. At the beginning of an episode, we sample a goal uniformly in the environment, then we observe trajectories of length 200. We set $\gamma = .995$.

FetchReach FetchReach is a standard environment from Plappert et al. (2018). The objective is to reach a *goal* position in 3 dimension with the end of the robotic arm. The observation space $\mathcal{S}$ is of dimension 10 and contains positions and velocities, such that the environment is Markov, fully observable, and deterministic. The action space $\mathcal{A}$ is continuous and of dimension 4. The goal space $\mathcal{G}$ is of dimension 3, and the goal represent the position of the end of the robotic arm. Trajectories are of length 50.

Q-learning experiments Here we describe experiments with UVFA, HER and δ-DQN, which have similar structure. For every algorithm, we use the same neural network to learn $Q_\theta(s, a, g)$ or $q_\theta(s, a, g)$. Similarly to DDPG (Lillicrap et al., 2016), if the action space $\mathcal{A}$ is continuous, we additionally learn a deterministic policy $\pi_\theta : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$. We use a dueling architecture (Wang et al., 2015): we learn a *value* network $v_\theta(s, g)$ and an *advantage* network $\mathrm{adv}_\theta(s, a, g)$. We then define $q_\theta(s, a, g) = v_\theta(s, g) + \widetilde{\mathrm{adv}}_\theta(s, a, g)$, where $\widetilde{\mathrm{adv}}_\theta(s, a, g)$ is the *rescaled* advantage, and is defined as $\widetilde{\mathrm{adv}}_\theta(s, a, g) = \mathrm{adv}_\theta(s, a, g) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \mathrm{adv}_\theta(s, a', g)$ if $\mathcal{A}$ is finite, and $\widetilde{\mathrm{adv}}_\theta(s, a, g) = \mathrm{adv}_\theta(s, a, g) - \mathrm{adv}_\theta(s, \pi(s, g), g)$ if $\mathcal{A}$ is continuous. The networks for v_θ, a_θ and π_θ are 3-hidden layers MLP of width 256 and ReLU activations. The inputs of v_θ and π_θ are the concatenation of s and g. If $\mathcal{A}$ is continuous, the input of adv_θ is the concatenation of s, a, g. If $\mathcal{A}$ is discrete, the input of adv_θ is the concatenation of s and g, and its output is of dimension $|\mathcal{A}|$, every dimension corresponding to an action.

Most hypereparameters are shared among the three methods: we observe batches of trajectories of size 16 for the Torus experiments, and of size 2 for the FetchReach environment. At every epoch, we observe a batch of trajectories and store it in a memory buffer of size 10^6 transitions. We use an ε-greedy exploration strategy, with $\varepsilon = 0.2$. At every epoch, we sample 100 batches from the replay buffer for the Torus experiments, and 50 for the FetchReach environment. For HER, we use the `future` sampling strategy for goals: when sampling a transition (s, a, s', g), with probability 0.2 we define $g' = g$, and with probability 0.8 we sample g' uniformly in the future of s. For δ-DQN in the Torus environment, we sample independent goals with $\rho_{\mathcal{G}}$ uniform distribution in the Torus. In FetchReach, we do not assume we have access to the goal sampling distribution. Therefore, we re-sample independent goals from the memory buffer. For every method, observations and goals are normalized. We use a target network with parameter θ_{tar} and update the target as $\theta_{\mathrm{tar}} \leftarrow (1 - \alpha)\theta_{\mathrm{tar}} + \alpha\theta$ with $\alpha = 0.05$ after every epoch. Every model is trained with the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

For every method and environment, the most sensitive hyperparameters were selected with a grid-search. For HER, UVFA and δ-DQN, we selected the learning rate of the optimizer from a range $\{1e-6, 3e-6, 1e-5, 3e-5, 1e-4, 3e-4, 1e-3\}$. For HER and UVFA, we additionally selected R a reward scaling factor, in $\{1e-2, 1e-1, 1, 10, 100, 1000, 1e4\}$. For δ-DQN, we also selected a parameter c_δ corresponding to the scaling of the reward: the scaled infinitely sparse reward is $R(s, \mathrm{d}g) = c_\delta \delta_{\varphi(s)}(\mathrm{d}g)$. We experimented all the

possible hyperparameters of this grid separately on every environment on a single run and selected the best hyperparameters. The values in Figure 16.1 are the mean performance evaluated with 5 different random seeds, and the confidence intervals represent the standard deviation of the reported metric across the 5 independent runs. In practice, the reward scaling factor for UVFA is 10 for all the Torus environments and 100 for FetchReach. The reward factor is 1 for HER for all the Torus environments and 10 for FetchReach. The learning rate for UVFA is $1e-4$ for all the Torus environments and $1e-3$ for FetchReach. The learning rate for HER is $3e-4$ for all the Torus environments and $1e-3$ for HER. For δ-DQN, the learning rate is $1e-5$ for all the Torus environments, and $1e-4$ for the FetchReach environment. The reward scaling coefficient c_δ is $1e-2$ for every environments.

δ-PPO experiments The δ-PPO is defined from δ-AC similarly to PPO (Schulman et al., 2017) from actor critic methods. We learn the model $m_\theta(s,g,g')$ of the density of $M^\pi(s,g,\mathrm{d}g')$ with respect to $\rho_\mathcal{G}$, and $\pi_\theta(a|s,g)$ a parametric policy. We used a shared architecture: we define $h_\theta(s,g,g')$ a network computing a hidden representation of dimension H. Then, we define two linear layers L^m_θ and $L^\pi\theta$, and define $m_\theta(s,g,g') = L^m_\theta(h_\theta(s,g,g'))$ and $\pi_\theta(a|s,g) = L^\pi_\theta(h_\theta(s,g,g'))$. In practice, h_θ is a 2-hidden layers MLP with ReLU activations (except at the last layer), with width $H = 256$ for the internal and output layers.

A step of δ-PPO is defined as follow. We first gather a buffer of trajectories with the current policy π_θ. Then, we define $\theta' := \theta$. For every transition (s,a,s',g) in the buffer and every epoch $e \leqslant E$, we sample an independent goal g' and compute:

$$\widehat{\delta\theta}_M \leftarrow \widehat{\delta\theta}_{\delta\text{-TD}}(s,a,s',g,g') \tag{16.A.1}$$

$$\text{adv} \leftarrow \gamma m_{\theta_M}(s',g,g) - m_{\theta_M}(s,g,g) \tag{16.A.2}$$

$$r(\theta') \leftarrow \frac{\pi_{\theta'}(a|s,g)}{\pi_\theta(a|s,g)} \tag{16.A.3}$$

$$\tilde{r}(\theta') \leftarrow \text{clip}(r, 1-u, 1+u) \tag{16.A.4}$$

$$\widehat{\delta\theta}_\pi \leftarrow \partial_{\theta'}\left(\min\left(\text{adv} \times r(\theta'), \text{adv} \times \tilde{r}(\theta')\right)\right) \tag{16.A.5}$$

$$\widehat{\delta\theta} \leftarrow \widehat{\delta\theta}_\pi + c_M \times \widehat{\delta\theta}_M \tag{16.A.6}$$

where c_M allow to scale the two updates. Then we use $\widehat{\delta\theta}$ and with Adam optimizer to obtain a new value for θ'. We did not use an entropy regularizer aw we observed that the diversity of actions was not an issue in practice.

For the Torus environment, the independent goals g' are sampled from $\rho_\mathcal{G}$ the uniform distribution of goals in the environment. For FetchReach, we do not assume we know $\rho_\mathcal{G}$ and sample goals from the buffer.

In practice, at every step of the δ-PPO algorithm we observe a batch of 2 trajectories for Torus(4) and Torus(6), 100 for the Torus(4) with the freeze action a^*, and 200 for FetchReach. Three hyperparameters were selected independently for every environment via a grid search: E the number of epochs per δ-PPO step, the learning rate of Adam optimizer, and the coefficient c_M. We performed a grid search with a single run per tuple of parameters. Then, the reported results in Figure 16.1 are averaged over 5 different random seeds with the selected hyperparameters. The number of epoch E per step was selected as lowest number which achieved close-to-optimal performance across the range $\{1,2,5,10,20,50,100\}$. In practice, $E = 20$ in the Torus(4) and Torus(6) environments, $E = 10$ in the Torus(4) with the freeze action a^*, and $E = 50$ for FetchReach. The learning rate was selected in the set $\{1e-6, 3e-6, 1e-5, 3e-5, 1e-4, 3e-4, 1e-3\}$, and in practice is $1e-4$ for every environment. The coefficient c_M was selected in $\{1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3\}$ and in practice is $1e-3$ for every Torus environment and $1e-1$ for the FetchReach environment.

Additional experiments We experimented δ-DQN and δ-PPO in more complex environments such Torus of higher dimension, or other environments of OpenAI Robotic suite (Plappert et al., 2018). In the Torus environment, both methods fail when the dimension increases above 15 while HER is still able to learn. More importantly, δ-PPO and δ-DQN did not learn at all in environments such as FetchPush (which is easy to solve with HER) or HandReach, which has similar structure but higher dimension than FetchReach. In the FetchPush environment, the objective is to push a cube with a robotic arm to a given goal. We observed that the issue of our methods was not an exploration issue, since the robotic arm often reaches and pushes the cube randomly. We tried to increase the generalization across goals with the δ-TD(n) update, but it was to computationally expensive, as explained in Section 16.1. Limitations of δ-DQN and δ-PPO which could explain these results are discussed in Section 16.4.

Chapter 17

Adaptive TD updates for successor operators: a comparison with C-learning

In all the algorithms used in this thesis for learning the successor state operator $M^\pi(s, \mathrm{d}s')$, the successor goal measure $M^\pi(s, g, \mathrm{d}g')$, or the multi-goal optimal action-value measure $Q^*(s, a, \mathrm{d}g)$, we derived unbiased estimates of Bellman errors for L_2-norms between the current model and its image under the Bellman operator, typically, if $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$, we used $\|M_\theta(s, \mathrm{d}s')\|_\rho^2 = \int_{s_1,s_2} \rho(\mathrm{d}s_1)\rho(\mathrm{d}s_2)m_\theta^2(s_1, s_2)$.

In this chapter, we derive algorithms for *adaptive* norms, defined using the current parameter estimate θ. Changing the norm leads to algorithms with the same fixed point $M_\theta = M^{\mathrm{tar}}$, but a different numerical behavior: this may be interesting for practical reasons. These algorithms might allow to tackle some of the technical issues raised by learning successor states operator. As they are more general, they open doors for future improvements of these methods. Interestingly, this approach allow us to recover an other known method for multi-goal RL: C-learning (Eysenbach et al., 2021), as discussed in Section 17.3.

We first describe a natural adaptive norm, corresponding to the KL divergence. Then, we describe more generally how to use other divergences for the temporal difference update. This allow us to give a new interpretation of the C-learning update (Eysenbach et al., 2021). Finally, we present first experiments, in order to understand if the adaptive norms as used in the C-learning experiments can directly improve performance compared to the δ-DQN approach.

We will describe these adaptive methods in the general context of successor states operator, as in Part IV, not specifically for the multi-goal setting. As all the multi-goal methods in Part V were derived from techniques for the successor goal operators, these adaptive approaches can be used for multi-goal RL as well. As C-learning was developed in the context of multi-goal RL, our experiments in Section 17.4 are in the multi-goal setting.

17.1 Kullback-Leibler Temporal Difference for the successor operators.

We consider the Forward TD update described in Chapter 7. We assume $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ is our current model of the successor states operator, and we define $M^{\mathrm{tar}} := \mathrm{Id} + \gamma P M_\theta$, the target operator obtained via the Forward Bellman equation obtained in Theorem 7.1.

Since $M_\theta(s, \mathrm{d}s_2)$ is a measure on s_2 for each s, and likewise for M^{tar}, one may use the generalized KL divergence between M and M^{tar} instead of the L^2 norm:

$$L(\theta) := \mathbb{E}_{s\sim\rho}\, \mathrm{KL}\big(M^{\mathrm{tar}}(s, \mathrm{d}s_2) \,\|\, M_\theta(s, \mathrm{d}s_2)\big)\,. \tag{17.1.1}$$

The generalized KL divergence extends the KL divergence to cases where the measures may not sum to 1. It is defined as $\mathrm{KL}(p \,\|\, q) := \int p \log p/q - \int p + \int q \geqslant 0$ and is the Bregman divergence associated to the convex function $\int p \log p$. Mathematically, the dependency on ρ is reduced compared to the L^2 loss, because the KL divergence does not depend on a choice of reference measure on s_2.

We have the following theorem, which corresponds to Theorem 7.5 but for the KL divergence instead of the $L_2(\rho)$ norm.

Theorem 17.1. *Let $M_\theta(s_1, \mathrm{d}s_2) = m_\theta(s_1, s_2)\rho(\mathrm{d}s_2)$ be a current estimate of $M(s_1, \mathrm{d}s_2)$. Consider $M^{\mathrm{tar}} = \mathrm{Id} + \gamma P M_\theta$, a target estimate for M defined via the Forward Bellman equation.*

Let (s, s') be a sample of the environment such that $s' \sim P(s'|s)$ and $s_2 \sim \rho$ is sampled independently, we define $\widehat{\delta\theta}_{KL\text{-}TD}(s, s', s_2)$ as:

$$\widehat{\delta\theta}_{KL\text{-}TD}(s, s', s_2) := \frac{\partial_\theta m_\theta(s, s)}{m_\theta(s, s)} + \partial_\theta m_\theta(s, s_2)\left(\gamma \frac{m_{\theta_t}(s', s_2)}{m_\theta(s, s_2)} - 1\right) \tag{17.1.2}$$

Then $\widehat{\delta\theta}_{KL\text{-}TD}$ is an unbiased estimate of the KL error:

$$\mathbb{E}_{s\sim\rho, s'\sim P(s,\mathrm{d}s'), s_2\sim\rho}\left[\widehat{\delta\theta}_{F\text{-}TD}(s, s', s_2)\right] = \partial_\theta \mathbb{E}_{s\sim\rho}\, \mathrm{KL}\big(M^{\mathrm{tar}}(s, \mathrm{d}s_2) \,\|\, M_\theta(s, \mathrm{d}s_2)\big)\,. \tag{17.1.3}$$

Contrary to the forward TD above, this does not reduce to ordinary TD on each s_2 in the tabular case. Still, the fixed point $M = \mathrm{Id} + \gamma P M$ is the same; the Bellman gaps are just rescaled adaptively for the gradient descent. The principle extends to backward TD and second-order methods (Bellman–Newton), by changing M^{tar} and replace it by the images via the Backward Bellman operator or the Bellman–Newton operator. In Bellman–Newton, the additional factors $1/m$ may help with the numerical instability of the quadratic term, but will not help with the variance from choosing (s, s_1, s_2).

Proof. The proof is similar to the proof of Theorem 7.5. As with the $L_2(\rho)$ norm, the KL divergence $\mathrm{KL}\big(M^{\mathrm{tar}}(s, \mathrm{d}s_2) \,\|\, M_\theta(s, \mathrm{d}s_2)\big)$ is infinite because M^{tar} is not absolutely continuous with respect to ρ. But its the gradient of the KL divergence is well defined. While we formally handled this for the L_2 norm, we only give here an informal explanation here. First, consider $M_0(s_1, \mathrm{d}s_2) = m_0(s_1, s_2)\rho(\mathrm{d}s_2)$ a measure, absolutely continuous with respect to ρ, then we have:

$$\begin{aligned}
&\mathbb{E}_{s\sim\rho}\, \mathrm{KL}(M_0(s, \mathrm{d}s_2) \,\|\, M_\theta(s, \mathrm{d}s_2)) = \\
&= \int_{s,s_2} \rho(\mathrm{d}s)\rho(\mathrm{d}s_2) m_0(s, s_2) \log\left(\frac{m_0(s, s_2)}{m_\theta(s, s_2)}\right) + \int_{s,s_2} \rho(\mathrm{d}s)\rho(\mathrm{d}s_2)(m_\theta(s, s_2) - m_0(s, s_2))
\end{aligned}$$

and by taking the gradient we have:

$$\partial_\theta \mathbb{E}_{s\sim\rho}\, \mathrm{KL}(M_0(s, \mathrm{d}s_2) \,\|\, M_\theta(s, \mathrm{d}s_2)) \tag{17.1.4}$$

$$= -\int_{s,s_2} \rho(\mathrm{d}s)\rho(\mathrm{d}s_2) m_0(s, s_2) \partial_\theta \log\left(m_\theta(s, s_2)\right) + \int_{s,s_2} \rho(\mathrm{d}s)\rho(\mathrm{d}s_2) \partial_\theta m_\theta(s, s_2) \tag{17.1.5}$$

$$= -\int_{s,s_2} \rho(\mathrm{d}s) M_0(s, \mathrm{d}s_2) \partial_\theta \log\left(m_\theta(s, s_2)\right) + \int_{s,s_2} \rho(\mathrm{d}s)\rho(\mathrm{d}s_2) \partial_\theta m_\theta(s, s_2) \tag{17.1.6}$$

We see that the gradient formula is well defined, even if M_0 is not absolutely continuous with respect to ρ. Hence, as fr the $L_2(\rho)$ norm, we say that while the KL divergence is infinite, its gradients are well-defined,

with equation (17.1.6). Hence, we have:

$$\partial_\theta \mathbb{E}_{s\sim\rho}\,\mathrm{KL}\big(M^{\mathrm{tar}}(s,\mathrm{d}s_2)\,\|\,M_\theta(s,\mathrm{d}s_2)\big) =$$
$$= -\int_{s,s_2}\rho(\mathrm{d}s)M^{\mathrm{tar}}(s,\mathrm{d}s_2)\partial_\theta\log\left(m_\theta(s,s_2)\right) + \int_{s,s_2}\rho(\mathrm{d}s)\rho(\mathrm{d}s_2)\partial_\theta m_\theta(s,s_2)$$

We first study the first part:

$$-\int_{s,s_2}\rho(\mathrm{d}s)\left(\delta_s(\mathrm{d}s_2)+\gamma\int_{s'}P(s,\mathrm{d}s')m_\theta(s',s_2)\rho(\mathrm{d}s_2)\right)\partial_\theta\log\left(m_\theta(s,s_2)\right)$$
$$= -\int_{s,s',s_2}\rho(\mathrm{d}s)P(s,\mathrm{d}s')\rho(\mathrm{d}s_2)\left(\partial_\theta\log m_\theta(s,s)+\gamma m_\theta(s',s_2)\partial_\theta\log\left(m_\theta(s,s_2)\right)\right)$$
$$= -\int_{s,s',s_2}\rho(\mathrm{d}s)P(s,\mathrm{d}s')\rho(\mathrm{d}s_2)\left(\partial_\theta\log m_\theta(s,s)+\partial_\theta\log m_\theta(s,s_2)\left(\gamma m_\theta(s',s_2)\right)\right)$$

Hence, we have:

$$\partial_\theta \mathbb{E}_{s\sim\rho}\,\mathrm{KL}(M_0(s,\mathrm{d}s_2)\,\|\,M_\theta(s,\mathrm{d}s_2)) =$$
$$= -\int_{s,s',s_2}\rho(\mathrm{d}s)P(s,\mathrm{d}s')\rho(\mathrm{d}s_2)\left(\partial_\theta\log m_\theta(s,s)+\partial_\theta\log m_\theta(s,s_2)\left(\gamma m_\theta(s',s_2)\right)-\partial_\theta m_\theta(s,s_2)\right)$$
$$= -\int_{s,s',s_2}\rho(\mathrm{d}s)P(s,\mathrm{d}s')\rho(\mathrm{d}s_2)\left(\partial_\theta\log m_\theta(s,s)+\partial_\theta\log m_\theta(s,s_2)\left(\gamma m_\theta(s',s_2)-m_\theta(s',s_2)\right)\right)$$
$$= -\int_{s,s',s_2}\rho(\mathrm{d}s)P(s,\mathrm{d}s')\rho(\mathrm{d}s_2)\widehat{\delta\theta}_{\text{F-TD}}(s,s',s_2)$$

Exponential model Interestingly, we can use an exponential model, and parametrize m_θ as: $m_\theta(s_1,s_2)=\exp(E_\theta(s_1,s_2))$. The first advantage of this parametrization is that m_θ is now always positive. Then, an other advantage is that it might be easier for a model to learn a function taking values in a wide interval (when $s_1\approx s_2$, $m(s_1,s_2)$ goes to infinity, while it is of order $O(1)$ everywhere else). In that case, the KL-TD becomes:

$$\widehat{\delta\theta}_{\text{KL-TD}}(s,s',s_2)=\partial_\theta E_\theta(s,s)+\partial_\theta E_\theta(s,s_2)\left(\gamma\exp(E_\theta(s',s_2))-\exp(E_\theta(s,s_2))\right) \tag{17.1.7}$$

While it was formally possible to learn m_θ with an exponential parametrization with the $L_2(\rho)$ norm, we observed numerical instabilities in practice. With the KL-TD update, the gradient $\partial_\theta m_\theta(s_1,s_2)$ is rescaled by the value of $m_\theta(s_1,s_2)$, which mitigates the numerical issue.

One drawback of this parametrization, is that some of the algorithms obtained in this thesis are not possible to use anymore. In particular, the FB parametrization with reduce variance introduced in Section 11.3.2 is not possible to use with an exponential model, because the bilinear assumption is necessary.

17.2 General form of adaptive TD updates.

In the previous Section, we obtained an adaptive TD update via the KL divergence. Other divergences between the measures M_θ and M^{tar} lead to TD variants, still computing the same fixed point but with different updates. Let $\ell\colon\mathbb{R}^2\to\mathbb{R}$ be a loss function such that for each x, the function $y\mapsto\ell(x,y)$ is convex with a minimum for $y=x$. Define the divergence D_ℓ between two measures $M_1=m_1\rho$ and $M_2=m_2\rho$ by $D_\ell(M_1,M_2):=\mathbb{E}_{s,s_2\sim\rho}[\ell(m_1(s,s_2),m_2(s,s_2))]$. Each such choice leads to a form of TD for M by taking the update $\delta\theta=-\partial_\theta D_\ell(M^{\mathrm{tar}},M_\theta)$. The forward TD introduced in Section 7.3 corresponds to $\ell=\frac{1}{2}(x-y)^2$. The KL-TD update in the previous section corresponds to $\ell(x,y)=x\log x/y-x+y$, for which D_ℓ is the generalized Kullback–Leibler divergence.

Not all choices of ℓ provide a tractable update. Several choices lead to tractable *weighted TD* updates of the form

$$\mathbb{E}_{s\sim\rho,\,s'\sim P(\mathrm{d}s'|s),\,s_2\sim\rho}\left[\frac{\partial_\theta m_\theta(s,s)}{w(\theta,s,s)}+\frac{\partial_\theta m_\theta(s,s_2)}{w(\theta,s,s_2)}(\gamma m_{\theta_t}(s',s_2)-m_\theta(s,s_2))\right] \tag{17.2.1}$$

for some particular choices of w. KL-TD is $w(\theta,s,s_2):=m_\theta(s,s_2)$.

In the next Section, we show that C-learning (Eysenbach et al., 2021) is a particular case of the adaptive TD method.

17.3 Relationship with C-learning

We now turn to the C-learning algorithm (Eysenbach et al., 2021). These authors introduce an algorithm to train m via a logistic regression to tell the difference between samples from ρ and samples from M^{tar}. This turns out to correspond to the weight $w := m_\theta(1+(1-\gamma)m_\theta)/(1-\gamma)$ in the weighted TD update (17.2.1) (equation (17.2.1)), and results from the loss $\ell(x,y) = \frac{1}{C(x)}H(C(x),C(y))$ where H is the cross-entropy $H(p,q) := -p\log q - (1-p)\log(1-q)$ and $C(x) := \frac{1}{1+(1-\gamma)x}$.

Precisely, the algorithm is a logistic regression on pairs (s,s_2), with negative examples sampled by $s\sim\rho$, $s_2\sim\rho$, and positive examples sampled by $s\sim\rho$, $s_2\sim(1-\gamma)M^{\text{tar}}(s,\mathrm{d}s_2)$. Since the mass of the successor state measure M is $\frac{1}{1-\gamma}$, $(1-\gamma)M$ is a probability measure. With an equal number of positive and negative examples, the optimal classifier is

$$p_+^* = (1-\gamma)\frac{M^{\text{tar}}}{\rho+(1-\gamma)M^{\text{tar}}} = (1-\gamma)\frac{m^{\text{tar}}}{1+(1-\gamma)m^{\text{tar}}}$$

and

$$p_-^* = \frac{1}{1+(1-\gamma)m^{\text{tar}}}$$

The predicted values p_+ implicitly correspond to a model m_θ defined such that

$$p_+ =: (1-\gamma)\frac{m_\theta}{1+(1-\gamma)m_\theta}$$

which corresponds to

$$m_\theta(s_1,s_2) = \frac{p_+(s_1,s_2)}{p_-(s_1,s_2)} \tag{17.3.1}$$

The optimum is when $m_\theta = m^{\text{tar}}$: learning moves M_θ away from ρ and towards M^{tar}.

The logistic model is $p_+(s,s_2) = \text{sigmoid}(E_\theta(s,s_2))$ with E_θ the pre-activation value. Using equation (17.3.1), this corresponds to a model m_θ of the successor operator defined via

$$(1-\gamma)m_\theta = \exp E_\theta,$$

which is the exponential parametrization introduced in Section 17.1 with the KL-TD update.

Sampling from $M^{\text{tar}} = \text{Id}+\gamma PM_\theta$ is done by sampling from the Dirac term and from $(1-\gamma)m_\theta(s',s_2)\rho(\mathrm{d}s_2)$. The latter is done by sampling from ρ and reweighting by $(1-\gamma)m_\theta$. In the end, the resulting update amounts to a weighted TD (17.2.1) on m_θ with weight

$$w(\theta,s,s_2) := m_\theta(s,s_2)(1+(1-\gamma)m_\theta(s,s_2)) \tag{17.3.2}$$

although it is derived in a different way. The underlying loss between M_θ and M^{tar} is the following: Letting $H(x,y) := -x\log y - (1-x)\log(1-y)$ be the cross-entropy between two Bernoulli distributions with parameters $x,y\in[0;1]$, this derives from the loss

$$L(\theta) = \mathbb{E}_{s\sim\rho,\, s_2\sim\frac{\rho+(1-\gamma)M^{\text{tar}}}{2}}\left[H\left(\frac{1}{1+(1-\gamma)m^{\text{tar}}(s,s_2)}, \frac{1}{1+(1-\gamma)m_\theta(s,s_2)}\right)\right] \tag{17.3.3}$$

(up to scaling): this is the loss corresponding to learning a density via a classifier that contrasts it with samples from a reference distribution ρ.

In addition, for goal-oriented policy gradient/DDPG, they maximize the expected log-loss to reach the target s_2 instead of the probability, namely, $\mathbb{E}\log p^+ = \mathbb{E}_{s\sim\rho}\log(1-1/(1+(1-\gamma)m(s,s_2)))$ instead of $\mathbb{E}_{s\sim\rho}m(s,s_2)$ as would result from a standard value-based policy gradient with a pointwise Dirac reward at s_2. This is limited to a goal-oriented setup rather than dense

rewards, but can be extended to feature goals $g = \varphi(s_2)$ by learning all models on $(s, \varphi(s_2))$ instead of (s, s_2).

This algorithm learns the density m by contrasting it with an arbitrary reference distribution ρ. As a general density estimation method, this makes sense as long as ρ is not too different from m. This is the case here: typically ρ will be the set of all visited states, and m has to predict which of those states lie in the future of which others. Namely, $\rho(\mathrm{d}s_2)$ is typically the average of $M(s, \mathrm{d}s_2)$ over s.

This is formalized in the following Proposition. A minor difference between our setup and Eysenbach–Salakhutdinov–Levine is that they define successor states starting at time $t+1$ whereas we start at time s_t. Here we consider the version of their algorithm starting at time s_t.

Proposition 17.2. *Consider the update of C-learning (Eysenbach et al., 2021) starting at time t instead of time $t+1$, defined by a classifier working on triplets (s, a, g). Let p_θ^+ and p_θ^- be the values returned by the classifier for the positive and negative classes, with parameter θ. Given a sampled transition (s, a, s'), a sampled goal state g, and a sampled action $a' \sim \pi(a'|s', g)$ from the goal-oriented policy, the update $\delta\theta$ is defined by*

$$\kappa := \mathrm{stopgrad}\left(\frac{p_\theta^+(s', a', g)}{p_\theta^-(s', a', g)}\right), \tag{17.3.4}$$

$$L(\theta) := (1-\gamma)\log p_\theta^+(s, a, s) + \log p_\theta^-(s, a, g) + \gamma\,\kappa \log p_\theta^+(s, a, g), \tag{17.3.5}$$

$$\delta\theta := \partial_\theta L(\theta) \tag{17.3.6}$$

where we use (s, a, s) instead of (s, a, s') in the first term due to defining successor states starting at time t instead of $t+1$.

From this classifier, define a model q_θ of the goal-oriented q-function via

$$(1-\gamma)q_\theta(s, a, g) := p_\theta^+(s, a, g)/p_\theta^-(s, a, g). \tag{17.3.7}$$

Then the update above is equal to the goal-oriented TD update,

$$\delta\theta = \frac{\partial q_\theta(s, a, s)}{w(\theta, s, a, s)} + \frac{\partial q_\theta(s, a, g)}{w(\theta, s, a, g)}\left(\gamma q_\theta(s', a', g) - q_\theta(s, a, g)\right) \tag{17.3.8}$$

with weights

$$w(\theta, s, a, g) := q_\theta(s, a, g)\frac{1 + (1-\gamma)q_\theta(s, a, g)}{1-\gamma}. \tag{17.3.9}$$

If the classifier is defined by pre-activation values $E_\theta(s, a, g)$, namely, $p_\theta^+ = \mathrm{sigmoid}(E_\theta(s, a, g))$, then we find $q_\theta(s, a, g) = (\exp E_\theta(s, a, g))/(1-\gamma)$.

Proof. This is a direct computation, writing the classifier as $p_\theta^+ = \mathrm{sigmoid}(E_\theta(s, a, g))$.

17.4 Experiments

We now compare the C-learning algorithm with the δ-DQN algorithm introduced in Chapter 15. Our main question is: is the C-learning update (corresponding to the adaptive TD update described in last section) significantly better than the δ-DQN update? As the technical details in our own experiments with δ-DQN or HER are very different than those of the C-learning original paper, we performed two distincts experiments:

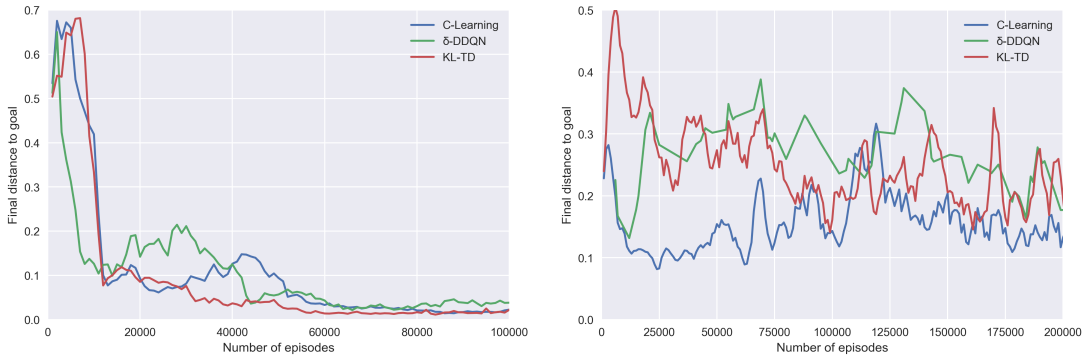
1. First, we implemented δ-DQN and KL-TD in the open source code provided by the authors together with the C-learning original paper (Eysenbach et al., 2021), and compared the three methods together (Figure 17.1a).

2. Then, we implemented the C-learning update in the open-source HER code base we used as a starting point for our δ-DQN experiments (Dai, 2019). The goal was to check whether C-learning was able to solve an environment in which δ-DQN is failing (Figure 17.1b).

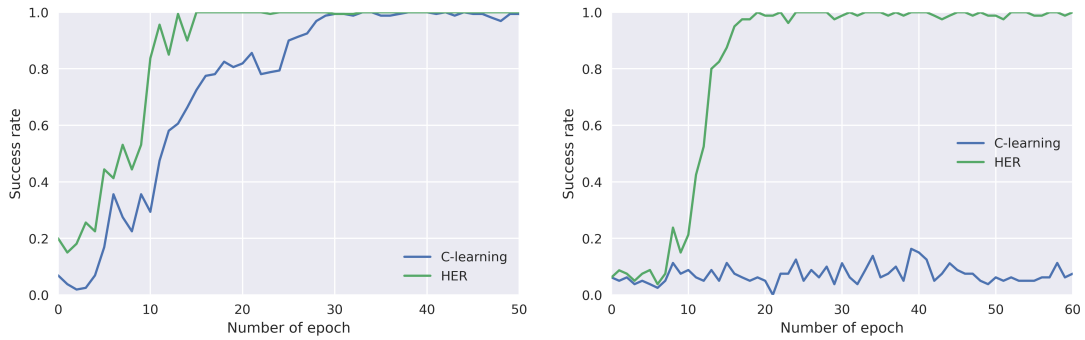
The conclusion is the following: there is no empirical evidence that the C-learning update is significantly better than the δ-DQN update:

1. In the first experimental setting, we observe that C-learning, δ-DQN and KL-TD are performing similarly.
2. In the second one, C-learning is succeeding in the same environments as δ-DQN (FetchReach) and failing in the same too (FetchPush), while HER can solve these two, and more difficult cases.

More experimental details are given in the following.



(a) Comparing the original implementation of C-learning together with the KL-TD and δ-DQN update implemented in the same code base with the same experimental details. We experimented the two environments with available code and hyperparameters in which the C-learning update is tested in the original paper (without Monte-Carlo, which is biased): SawyerReach (Left) and SawyerWindow (right). In SawyerReach, there is no significant difference between the three methods. In SawerWindow, the training is noisy, and there is no clear difference either.



(b) Comparing HER and the C-learning update, both together with DDPG, implemented in an HER open-source implementation (Dai, 2019) we also used for our δ-DQN implementation. In the FetchReach environment (Left), both HER and C-learning are able to learn (as δ-DQN, see figure 16.1). In FetchPush (right), the *next* environment in the OpenAI suite (Plappert et al., 2018), HER is able to learn while C-learning is not (as δ-DQN, see Section 16.4). Hence, in that setting, there is no significant difference between δ-DQN and C-learning.

Figure 17.1: An experimental comparison between δ-DQN, KL-TD, HER, and C-learning.

Experimental details. Our first experiments (Figure 17.1a) started from the C-learning implementation provided by the authors (Eysenbach et al., 2021). In the paper, the authors experiment in eight environments, but specific hyperparameters are provided for four of them in the repository (SawyerReach, SawyerWindow, SawyerPush, SawyerDrawer), hence, we restrict ourselves to these last four. Among these four, only two (SawyerReach, SawyerWindow) are trained with an unbiased algorithm. Indeed, the experiment in SawyerPush and SawyerDrawer

are done with what the authors call the Monte-Carlo update, which is biased, similarly to HER. While this works in practice in these deterministic environments, it might be prevent learning in some settings, similarly to experiments in Figure 16.1. Therefore, we only compare unbiased methods together, and focus on the two environments SawyerReach and SawyerWindow. In these two environments, we first re-wrote the C-learning algorithm into a strictly equivalent method, but written as an algorithm learning $q_\theta(s, a, g)$, as described in the previous Section. In this implementation, $q_\theta(s, a, g)$ is parametrized as $q_\theta(s, a, g) = \frac{1}{1-\gamma} \exp(E_\theta(s, a, g))$, and $E_\theta(s, a, g)$ is a MLP network. In the same setting, keeping exactly the same technical details, we implement the KL-TD update and the δ-DQN update. The KL-TD update is implemented using the same parametrization $q_\theta(s, a, g) = \frac{1}{1-\gamma} \exp(E_\theta(s, a, g))$. The δ-DQN update is too unstable with an exponential parametrization, hence for the δ-DQN update we use an ELU activation $\text{ELU}(x) = \exp(x)$ if $x < 0$ and $\text{ELU}(x) = 1 + x$ if $x \geqslant 0$, and parametrize: $q_\theta(s, a, g) = \frac{1}{1-\gamma}\text{ELU}(E_\theta(s, a, g))$. For C-learning, we kept the settings provided in the author's repository. For KL-TD and δ-DQN, we tuned kept all the original except hyperparameters, except 3 which were tuned separately: the learning rates of the actor and the critic (both trained with Adam (Kingma and Ba, 2015)) in the set $\{1e-3, 1e-4\}$, and the weight of the Dirac term of the gradient (as in the experiments with δ-DQN and δ-AC in Chapter 16), in the set $\{1e-2, 1e-1, 1., 1e1, 1e2\}$. In practice, these hyperparameters had relatively little impact, and the selected learning rates were $1e-4$ and the weight of the Dirac update was $1e1$. We report the final distance to the goal at the end of an episode.

Our second set of experiments took the opposite viewpoint: we implemented the C-learning update in an open-source implementation (Dai, 2019) of Hindsight Experience Replay (Andrychowicz et al., 2017). The goal was to compare experiment C-learning in the setting in which we performed our experiments of δ-DQN and δ-AC, in which we now that the baseline, the hyperparameter of HER were well chosen, such that it is able to solve hard environments. Indeed, in the original C-learning paper (Eysenbach et al., 2021), HER is not able to learn even in the very easy SawyerReach environment, suggesting the method could be more optimized. In these experiments, we learn $q_\theta(s, a, g) = \frac{1}{1-\gamma} \exp(E_\theta(s, a, g))$ with the adaptive TD update equivalent to C-learning, as discussed in the previous section. This update is used with DDPG as in the original code base. We kept all the relevant technical details described in the original C-learning paper (architecture, clipping of the Q-value, optimizers). HER is trained with the hyperparameters provided in the open-source repository. For C-learning, as in the previous experiments, we tuned 3 hyperparameters: the learning rates of the actor and the critic (both trained with Adam (Kingma and Ba, 2015)) in the set $\{1e-3, 1e-4\}$, and the weight of the diract term of the gradient, in the set $\{1e-2, 1e-1, 1., 1e1, 1e2\}$. In practice, as in the previous experiments, these hyperparameters had relatively little impact, and the selected learning rates were $1e-4$ and the weight of the Dirac update was $1e1$. We report the success rate, which is the frequency of reaching the goal in the evaluation trajectories, up to a precision ε specified in the environment. In the FetchReach environment, an epoch is 20 episodes. In the FetchPush environment, an epoch is 800 episodes. For every other hyperparameter we used the default values in the repository.

Conclusion

In this thesis, we introduced several mathematical approaches and principled methods for deep learning and deep reinforcement learning. Our goal was to understand some crucial properties that principled methods should satisfy, in order to be robust, stable, efficient and work in many settings. Then, we tried to define methods which could satisfy these properties. Along that process, we tried to understand the mathematical objects at the core of our learning settings, how to define them, learn them, and their properties.

Our first project (Chapter 3, Blier and Ollivier (2018)) proposed an information theory viewpoint on the complexity of deep learning models. We proved empirically the ability of deep neural networks to compress the training data even when accounting for parameter encoding. Hence, deep learning models, even with a large number of parameters, compress the data well, and these approaches are principled from an information theory point of view: *the number of parameters is not an obstacle to compression.*

We then introduced *All Learning Rates At Once* (Alrao) (Chapter 4, Blier et al. (2019)), an optimization method for neural networks removing the burden of finding the optimal learning rate, by instead assigning to each unit or feature in the network its own learning rate sampled from a random distribution spanning several orders of magnitude. Surprisingly, Alrao performs close to SGD with an optimally tuned learning rate, for various architectures and problems. Alrao is a principled way to define robust optimization method with no hyperparameter tuning, leveraging the redundancy in the neural network architectures.

In Chapter 5, we studied RL in near continuous time environments (Tallec et al. (2019)), and highlighted empirically and theoretically the lack of robustness of Q-learning approaches to time discretization. We detail a principled way to build an off-policy RL algorithm that yields similar performances over a wide range of time discretizations, and confirm this robustness empirically.

After these projects, we focused our work on an in-depth study of the successor states operator (Part IV, Blier et al. (2021)). This mathematical object is at the core of Reinforcement Learning, as it contains the information on all possible value functions for every possible rewards for a fixed policy, or equivalently the information on all expected occupancy measure starting from any initial point, for a fixed policy. This object was studied in finite environments (Dayan, 1993), but the continuous state spaces raises more technical issues, especially because the operator is not a matrix or a function but a measure. While our first motivation for this project was to improve sample efficiency of policy evaluation, at the end, our contribution were especially a better theoretical understanding of this object, its properties and equations. This study then lead to Part V and methods for multi-goal RL. Our main contributions on the successor states operator are as follows.

In Chapter 6, we properly define the successor states operator in continuous state spaces and show the relation between the value function and the successor states operator. In Chapter 7, we derive a Bellman equation and a temporal difference algorithm for the successor states operator with function approximators and show the relation with standard TD on the value function. Then in Chapter 8 we show that the successor states operator also satisfies a backward Bellman equation, which has no equivalent for the value function, and lead to a backward temporal difference algorithm. We show a relation between forward and backward TD, as backward TD corresponds in expectation to forward TD on the time reversed process. Then in Chapter 9, we

introduce second order methods for policy evaluation. These approaches were our first reason to study the successor states operator at the beginning of the project. We prove that the successor states operator satisfies a Bellman–Newton equation on the successor states operator, which also has no equivalent on the value function. We give three interpretations of this new Bellman–Newton equation : first, as a new way to combine paths in the environments, then, as the Newton method for policy evaluation, and finally as a expected exact model based update in the tabular case. This equation leads to a new Bellman–Newton algorithm, with function approximators, in continuous environments. We study this approach in the simpler tabular case, show empirically that it is more sample efficient than TD or TD(λ) in finite environments, and provide non asymptotic convergence bounds for this method in the tabular case (Chapter 10), which interestingly are independent of the number of states, hence are non-vacuous even for very large or infinite environments. This proves that it is possible to learn the value function in finite time, with theoretical guarantees, even in an infinite environment. Still, one of the main practical issues of the Bellman–Newton algorithm with function approximators is its variance, which makes it highly unstable in practice. To overcome this issue, we consider low rank FB representations for the successor states (Chapter 11). Using this FB representation allow us to define updates with lower variance. Additionally, we prove theoretically that the fixed point of these methods are truncated SVDs of the successor states operator, and that in practice it converges to its optimal low-rank representation. Finally in Chapter 12 we described several methods to learn the value function via a model of the successor states operator. In particular, we see that the successor states operator can be used as a model of the expected eligibility traces.

We then extended our approach to the multi-goal RL setting (Part V, Blier and Ollivier (2021)). We tackle the issues of *vanishing rewards* observed with methods as Universal Value Function Approximators (Schaul et al., 2015). While methods such as Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) tackle this issue and achieve great results in practice, they are known to be biased, and can converge to low-return policies in some cases. We first study the bias of HER, prove theoretically and empirically that it is unable to learn in some environments. On the other side, we prove that HER is unbiased and well-founded, in deterministic environments. We then introduce unbiased Q-learning and Actor-critic algorithms for multi-goal RL, in the general stochastic setting, by replacing standard sparse rewards of multi-goal RL by infinitely sparse Dirac rewards, and leveraging their structure.

My goal during this thesis was to improve performance of deep learning and deep RL with principled approaches. Trying to go from theoretical understanding to experimental results was quite a journey, and not always successful in the way we intended.

Some of our conclusion were negative results, impossible to translates in practical methods. In *The description Length of Deep Learning Models*, our first motivation was to use the information theory viewpoint to derive well-founded regularizers via compression bounds. At the end, our best compression bounds were intractable and useless to optimize deep learning models. But our main contribution was to show that standard deep RL approaches were already consistent with the information theory viewpoint, without the use of any additional regularizer.

We also sometimes struggled when comparing to standards methods which benefited from years of careful engineering tuning: in that case, a well-founded method might have nothing to improve. For example in *Making Deep Q-learning approaches robust to time discretization*, we thought that the lack of robustness of standard approaches in near continuous time would be a huge limitation in usual environments, and that our approach could improve these results. Actually, we found out that the time discretization of standard environments was set such that learning was possible, hence we observed almost no improvement in that regime, and had to look for smaller time discretization to highlight the practical issue we identified theoretically. But our contribution gave some interesting insights on some ways to design robust Q-learning agents.

Our study of the successor states operator happened to be even harder. We started this project with the hope of a significant practical impact on policy evaluation, with function approximators, thanks to the Bellman–Newton equation and its different exciting interpretations.

In turned out we quickly encountered experimental obstacles, at first because of the variance of the method then for other reasons. We tried to switch to the multi-goal setting but did not succeed in reaching results comparable to those of HER in standard environments.

During this adventure, we understood a lot. On the successor states operator, which we believe is at the core of reinforcement learning. On the different ways to leverage gathered information for policy evaluation. And some of our most striking results were not in the direction we expected. For example, studying the tabular RL setting was not our main goal. But at some point we wondered what would be the sample efficiency of our method in that case, to understand how much gain we could hope in the general continuous case compared to temporal difference. This lead to a non-vacuous convergence bound for policy evaluation, even for very large or infinite sets. Similarly, in the multi-goal RL setting, one of our goal was to design an efficient and principled methods, solving the bias issue of HER. Surprisingly, we proved that HER is actually unbiased in the deterministic setting.

Many questions remain open for future work: Is there still a way to leverage the path composition of Bellman–Newton to accelerate policy evaluation, with function approximators, without the variance issue? Is policy evaluation still a bottleneck of RL, and how much can we improve it? How could we reliably make use the FB representation in practice? Is it possible to define an unbiased multi-goal algorithm, able to reach similar performance than HER? What are the most natural and efficient ways to generalize across goals in an environment? And many others. I hope that our theoretical insights will be of interest for other researchers in the field, who might tackle some of these open questions, or others I never imagined.

Bibliography

Achille, A. and Soatto, S. (2017). On the Emergence of Invariance and Disentangling in Deep Representations. *arXiv preprint arXiv:1706.01350*.

Amari, S.-i. (1998). Natural gradient works efficiently in learning. *Neural Comput.*, 10:251–276.

Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39:20 – 3.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*.

Azar, M. G., Osband, I., and Munos, R. (2017). Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning*, pages 263–272. PMLR.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Ba, L. J. and Caruana, R. (2014). Do Deep Nets Really Need to be Deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662.

Baird, L. C. (1994). Reinforcement learning in continuous time: Advantage updating. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 4, pages 2448–2453. IEEE.

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Zidek, A., and Munos, R. (2018). Transfer in deep reinforcement learning using successor features and generalised policy improvement. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 501–510. PMLR.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4055–4065. Curran Associates, Inc.

Barreto, A., Hou, S., Borsa, D., Silver, D., and Precup, D. (2020). Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087.

Barron, A. and Yang, Y. (1999). Information-theoretic determination of minimax rates of convergence. *The Annals of Statistics*, 27(5):1564–1599.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. (2018). Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*.

Bellemare, M. G., Dabney, W., and Rowland, M. (2022). *Distributional Reinforcement Learning*. MIT Press. http://www.distributional-rl.org.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

Bengio, Y., Roux, N. L., Vincent, P., Delalleau, O., and Marcotte, P. (2006). Convex neural networks. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 123–130. MIT Press.

Berger, M. (2003). *A panoramic view of Riemannian geometry*. Springer.

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4th edition.

Beyer, H.-G. and Schwefel, H.-P. (2004). Evolution strategies – a comprehensive introduction. *Natural Computing*, 1:3–52.

Bhandari, J., Russo, D., and Singal, R. (2018). A finite time analysis of temporal difference learning with linear function approximation. In Bubeck, S., Perchet, V., and Rigollet, P., editors, *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 1691–1692. PMLR.

Blei, D. M., Kucukelbir, A., and Mcauliffe, J. D. (2017). Variational Inference: A Review for Statisticians.

Blier, L. and Ollivier, Y. (2018). The description length of deep learning models. In *Advances in Neural Information Processing Systems*.

Blier, L. and Ollivier, Y. (2021). Unbiased methods for multi-goal reinforcement learning. *arXiv preprint arXiv:2106.08863*.

Blier, L., Tallec, C., and Ollivier, Y. (2021). Learning successor states and goal-dependent values: A mathematical viewpoint. *arXiv preprint arXiv:2101.07123*.

Blier, L., Wolinski, P., and Ollivier, Y. (2019). Learning with Random Learning Rates. In *ECML PKDD 2019 - European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Blum, A. and Langford, J. (2003). PAC-MDL Bounds. In Schölkopf, B. and Warmuth, M. K., editors, *Learning Theory and Kernel Machines*, pages 344–357, Berlin, Heidelberg. Springer Berlin Heidelberg.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning*, pages 1613–1622.

Bogachev, V. I. (2007). *Measure theory*, volume 1. Springer Science & Business Media.

Borkar, V. S. and Meyn, S. P. (2000). The o.d.e. method for convergence of stochastic approximation and reinforcement learning. *SIAM J. Control. Optim.*, 38:447–469.

Borsa, D., Barreto, A., Quan, J., Mankowitz, D., Munos, R., van Hasselt, H., Silver, D., and Schaul, T. (2018). Universal successor features approximators. *arXiv preprint arXiv:1812.07626*.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1):33–57.

Brandfonbrener, D. and Bruna, J. (2019). Geometric insights into the convergence of nonlinear td learning. *arXiv preprint arXiv:1905.12185*.

Brémaud, P. (1999). *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Chaitin, G. J. (2007). On the intelligibility of the universe and the notions of simplicity, complexity and irreducibility. In *Thinking about Godel and Turing: Essays on Complexity, 1970-2007*. World scientific.

Cohen, D., Kontorovich, A., and Wolfer, G. (2020). Learning discrete distributions with infinite support. *arXiv: Statistics Theory*.

Colas, C., Oudeyer, P.-Y., Sigaud, O., Fournier, P., and Chetouani, M. (2019). Curious: Intrinsically motivated modular multi-goal reinforcement learning. In *ICML*.

Dai, T. (2019). A pytorch implementation of hindsight experience replay (her).

Dalal, G., Szörényi, B., Thoppe, G., and Mannor, S. (2018). Finite sample analyses for td(0) with function approximation. In *AAAI*.

Dawid, A. P. (1984). Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278.

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Denkowski, M. and Neubig, G. (2017). Stronger baselines for trustable results in neural machine translation. *arXiv preprint arXiv:1706.09733*.

Devraj, A. M. and Meyn, S. P. (2017). Fastest convergence for q-learning. *ArXiv*, abs/1707.03770.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159.

Dziugaite, G. K. and Roy, D. M. (2017). Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*, Sydney.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A. X., and Levine, S. (2018). Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *ArXiv*, abs/1812.00568.

Eppe, M., Magg, S., and Wermter, S. (2019). Curriculum goal masking for continuous deep reinforcement learning. *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 183–188.

Erraqabi, A. and Le Roux, N. (2018). Combining adaptive algorithms and hypergradient method: a performance and robustness study.

Eysenbach, B., Salakhutdinov, R., and Levine, S. (2021). C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*.

Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. (2019). Curriculum-guided hindsight experience replay. In *NeurIPS*.

Fischhoff, B. (1975). Hindsight is not equal to foresight: The effect of outcome knowledge on judgment under uncertainty. *Journal of Experimental Psychology: Human perception and performance*, 1(3):288.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pages 482–495. PMLR.

Foster, D. P. and George, E. I. (1994). The Risk Inflation Criterion for Multiple Regression. *The Annals of Statistics*, 22(4):1947–1975.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*.

Frankle, J. and Carbin, M. (2018). The Lottery Ticket Hypothesis: Finding Small, Trainable Neural Networks. *arXiv preprint arXiv:1704.04861*.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019). The lottery ticket hypothesis at scale.

Fujimoto, S., Meger, D., and Precup, D. (2021). A deep reinforcement learning approach to marginalized importance sampling with the successor representation. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3518–3529. PMLR.

Gao, T. and Jojic, V. (2016). Degrees of Freedom in Deep Neural Networks. *arXiv preprint arXiv:1603.09260*.

Gårding, L. (1953). On the asymptotic distribution of the eigenvalues and eigenfunctions of elliptic differential operators. *Math. Scand.*, (1).

Geramifard, A., Bowling, M., and Sutton, R. S. (2006). Incremental least-squares temporal difference learning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 356. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Graves, A. (2011). Practical Variational Inference for Neural Networks. In *Neural Information Processing Systems*.

Greydanus, S. and Olah, C. (2019). The paths perspective on value learning. *Distill*. https://distill.pub/2019/paths-perspective-on-value-learning.

Grinstead, C. M. and Snell, J. L. (1997). *Introduction to probability*. American Mathematical Soc.

Grünwald, P. D. (2007). *The Minimum Description Length principle*. MIT press.

Guyon, I., Chaabane, I., Escalante, H. J., Escalera, S., Jajetic, D., Lloyd, J. R., Macià, N., Ray, B., Romaszko, L., Sebag, M., et al. (2016). A brief review of the ChaLearn AutoML challenge: any-time any-dataset learning without human intervention. In *Workshop on Automatic Machine Learning*, pages 21–30.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Hairer, M. (2006). Ergodic properties of Markov processes. Lecture notes.

Hairer, M. (2010). Convergence of Markov processes. Lecture notes.

Han, S., Mao, H., and Dally, W. J. (2015a). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015b). Learning both Weights and Connections for Efficient Neural Networks. In *Advances in Neural Information Processing Systems*.

Hansen, N. (2016). The cma evolution strategy: A tutorial. *ArXiv*, abs/1604.00772.

Hansen, S. S., Dabney, W., Barreto, A., de Wiele, T. V., Warde-Farley, D., and Mnih, V. (2020). Fast task inference with variational intrinsic successor features. *ArXiv*, abs/1906.05030.

Hausknecht, M. J. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposia*.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *ICCV*, pages 770–778.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017). Deep reinforcement learning that matters. *CoRR*, abs/1709.06560.

Herbster, M. and Warmuth, M. K. (1998). Tracking the best expert. *Machine learning*, 32(2):151–178.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E. and Van Camp, D. (1993). Keeping Neural Networks Simple by Minimizing the Description Length of the Weights. In *Proceedings of the sixth annual conference on Computational learning theory*. ACM.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Honkela, A. and Valpola, H. (2004). Variational Learning and Bits-Back Coding: An Information-Theoretic View to Bayesian Learning. *IEEE transactions on Neural Networks*, 15(4).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, volume 1, page 3.

Hutter, M. (2007). On Universal Prediction and Bayesian Confirmation. *Theoretical Computer Science*, 384(1).

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456.

Islam, R., Ahmed, Z., and Precup, D. (2019). Marginalized state distribution entropy regularization in policy optimization. *ArXiv*, abs/1912.05128.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(4).

Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. (2017). Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*.

Jin, C., Krishnamurthy, A., Simchowitz, M., and Yu, T. (2020). Reward-free exploration for reinforcement learning. *ArXiv*, abs/2002.02794.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350.

Kaelbling, L. P. (1993). Learning to achieve goals. In *IJCAI*.

Kemeny, J. G. and Snell, J. L. (1960). *Finite Markov Chains*. Van Nostrand, New York.

Keskar, N. S. and Socher, R. (2017). Improving generalization performance by switching from Adam to SGD. *arXiv preprint arXiv:1712.07628*.

Khamaru, K., Pananjady, A., Ruan, F., Wainwright, M. J., and Jordan, M. I. (2020). Is temporal difference learning optimal? an instance-dependent analysis. *ArXiv*, abs/2003.07337.

Kianglu (2018). pytorch-cifar.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*.

Kobyzev, I., Prince, S., and Brubaker, M. (2020). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Koolen, W. and De Rooij, S. (2008). Combining expert advice efficiently. *arXiv preprint arXiv:0802.2015*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. (2017). Automatic Differentiation Variational Inference. *Journal of Machine Learning Research*, 18:1–45.

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016). Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.

Kurita, K. (2018). Learning Rate Tuning in Deep Learning: A Practical Guide | Machine Learning Explained.

Lakshminarayanan, C. and Szepesvari, C. (2018). Linear stochastic approximation: How far does constant step-size and iterate averaging go? In *AISTATS*.

Lanka, S. and Wu, T. (2018). Archer: Aggressive rewards to counter bias in hindsight experience replay. *ArXiv*, abs/1809.02070.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2012). Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13(98):3041–3074.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11).

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998b). Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In Touretzky, D. S., editor, *NIPS 2*, pages 598–605. Morgan-Kaufmann.

Lee, D. and He, N. (2019). Target-based temporal-difference learning. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3713–3722. PMLR.

Lehnert, L., Tellex, S., and Littman, M. L. (2017). Advantages and limitations of using successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1708.00102*.

Li, A., Pinto, L., and Abbeel, P. (2020). Generalized hindsight for reinforcement learning. *ArXiv*, abs/2002.11708.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018). Measuring the Intrinsic Dimension of Objective Landscapes. *arXiv preprint arXiv:1804.08838*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18(1):6765–6816.

Li, M. and Vitányi, P. (2008). *An introduction to Kolmogorov complexity*. Springer.

Lillicrap, T., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018a). Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34.

Liu, H., Simonyan, K., and Yang, Y. (2018b). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3290–3300.

Ma, C., Wen, J., and Bengio, Y. (2018). Universal successor representations for transfer reinforcement learning. *arXiv preprint arXiv:1804.03758*.

Machado, M. C., Bellemare, M. G., and Bowling, M. (2019). Count-based exploration with the successor representation.

Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., and Campbell, M. (2018). Eigenoption discovery through the deep successor representation. In *International Conference on Learning Representations*.

Mack, D. (2016). How to pick the best learning rate for your machine learning project.

Mackay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, cambridge edition.

Maclaurin, D., Duvenaud, D., and Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122.

Mahmood, A. R., Sutton, R. S., Degris, T., and Pilarski, P. M. (2012). Tuning-free step-size adaptation. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 2121–2124. IEEE.

Manela, B. and Biess, A. (2021). Bias-reduced hindsight experience replay with virtual goal prioritization. *Neurocomputing*, 451:305–315.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

Massé, P.-Y. and Ollivier, Y. (2015). Speed learning on the fly. *arXiv preprint arXiv:1511.02540*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. In *Neural Information Processing Systems*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Munos, R. and Bourgine, P. (1998). Reinforcement learning for continuous stochastic control problems. In *Advances in neural information processing systems*, pages 1029–1035.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R., Abbeel, P., Levine, S., and Finn, C. (2019). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv: Learning*.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566.

Nair, A., Pong, V. H., Dalal, M., Bahl, S., Lin, S., and Levine, S. (2018). Visual reinforcement learning with imagined goals. In *NeurIPS*.

Nasiriany, S., Pong, V. H., Lin, S., and Levine, S. (2019). Planning with goal-conditioned policies. In *NeurIPS*.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287.

Ollivier, Y. (2014). Auto-encoders: reconstruction versus compression. *arXiv preprint arXiv:1403.7752*.

Ollivier, Y. (2018). Approximate temporal difference learning is a gradient descent for reversible policies.

OpenAI (2018a). Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177.

OpenAI (2018b). Openai five. https://blog.openai.com/openai-five/.

Pan, V. and Schreiber, R. (1991). An improved newton iteration for the generalized inverse of a matrix, with applications. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1109–1130.

Pananjady, A. and Wainwright, M. J. (2019). Value function estimation in Markov reward processes: Instance-dependent ℓ_∞-bounds for policy evaluation.

Parthasarathy, K. R. (2005). *Probability measures on metric spaces*, volume 352. American Mathematical Soc.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Pinto, L. and Gupta, A. (2017). Learning to push by grasping: Using multiple tasks for effective learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2161–2168.

Pitis, S. (2018). Source traces for temporal difference learning. In *AAAI*.

Pitis, S., Chan, H., Zhao, S., Stadie, B. C., and Ba, J. (2020). Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. *ArXiv*, abs/2007.02832.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *Siam Journal on Control and Optimization*, 30:838–855.

Racanière, S., Weber, T., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N. M. O., Li, Y., Pascanu, R., Battaglia, P. W., Hassabis, D., Silver, D., and Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. In *NIPS*.

Rauber, P., Mutz, F. W., and Schmidhuber, J. (2019). Hindsight policy gradients. *ArXiv*, abs/1711.06006.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR.

Rissanen, J. (2007). *Information and complexity in statistical modeling*. Springer Science & Business Media.

Rissanen, J., Speed, T., and Yu, B. (1992). Density estimation by stochastic complexity. *IEEE Transactions on Information Theory*, 38(2):315–323.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). Fitnets: Hints for thin deep nets. In *Proceedings of the International Conference on Learning Representations*.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *ArXiv*, abs/1703.03864.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France. PMLR.

Schaul, T., Zhang, S., and LeCun, Y. (2013). No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351.

Schmidhuber, J. (1997). Discovering Neural Nets with Low Kolmogorov Complexity and High Generalization Capability. *Neural Networks*, 10(5):857–873.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 7839:604–609.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347.

See, A., Luong, M.-T., and Manning, C. D. (2016). Compression of neural machine translation models via pruning. *CoNLL 2016*, page 291.

Shannon, C. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27.

Shwartz-Ziv, R. and Tishby, N. (2017). Opening the Black Box of Deep Neural Networks via Information. *arXiv preprint arXiv:1703.00810*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.

Solomonoff, R. (1964). A formal theory of inductive inference. *Information and control*.

Spaan, M. T. J. (2012). Partially observable markov decision processes. In *Reinforcement Learning*.

Srikant, R. and Ying, L. (2019). Finite-time error bounds for linear stochastic approximation and td learning. *ArXiv*, abs/1902.00923.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2017). The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. (2021). Decoupling representation learning from reinforcement learning. *ArXiv*, abs/2009.08319.

Sun, W., Jiang, N., Krishnamurthy, A., Agarwal, A., and Langford, J. (2019). Model-based RL in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In *COLT*.

Sun, Y., Wierstra, D., Schaul, T., and Schmidhuber, J. (2009). Efficient natural evolution strategies. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*.

Surmenok, P. (2017). Estimating an Optimal Learning Rate For a Deep Neural Network.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. 2nd edition.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. The 10th International Conference on Au- tonomous Agents and Multiagent Systems-Volume 2, pages 761–768.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *ICCV*, pages 1–9.

Tadić, V. Z. B. (2004). On the almost sure rate of convergence of linear stochastic approximation algorithms. *IEEE Transactions on Information Theory*, 50:401–409.

Tagorti, M. and Scherrer, B. (2015). On the rate of convergence and error bounds for lstd(λ). In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1521–1529, Lille, France. PMLR.

Tallec, C. and Blier, L. (2018). Pyvarinf : Variational Inference for PyTorch.

Tallec, C., Blier, L., and Ollivier, Y. (2019). Making Deep Q-learning methods robust to time discretization. In *ICML 2019 - Thirty-sixth International Conference on Machine Learning*.

Tallec, C. and Ollivier, Y. (2018). Can recurrent neural networks warp time? *arXiv preprint arXiv:1804.11188*.

Tarbouriech, J., Pirotta, M., Valko, M., and Lazaric, A. (2020). Improved sample complexity for incremental autonomous exploration in mdps. In *Advances in Neural Information Processing Systems*, volume 33, pages 11273–11284.

Theodoridis, S. (2015). *Machine learning: a Bayesian and optimization perspective*. Academic Press.

Tibshirani, R. and Marchetti-Bowick, M. (2013). Gradient descent: Convergence analysis.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Tishby, N. and Zaslavsky, N. (2015). Deep Learning and the Information Bottleneck Principle. In *Information Theory Workshop*, pages 1–5. IEEE.

Touati, A. and Ollivier, Y. (2021). Learning one representation to optimize all rewards. *ArXiv*, abs/2103.07945.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202.

Tsitsiklis, J. N. and Van Roy, B. (1997). Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081.

Ueno, T., Kawanabe, M., Mori, T., ichi Maeda, S., and Ishii, S. (2008). A semiparametric statistical approach to model-free policy evaluation. In *ICML '08*.

Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the Brownian motion. *Physical review*, 36(5):823.

Ullrich, K., Meeds, E., and Welling, M. (2017). Soft Weight-Sharing for Neural Network Compression. *arXiv preprint arXiv:1702.04008*.

Van Erven, T., Grünwald, P., and De Rooij, S. (2012). Catching Up Faster by Switching Sooner: A predictive approach to adaptive estimation with an application to the AIC-BIC Dilemma. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(3):361–417.

Van Erven, T., Rooij, S. D., and Grünwald, P. (2008). Catching up faster in Bayesian model selection and model averaging. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *NIPS 20*, pages 417–424. Curran Associates, Inc.

van Hasselt, H., Madjiheurem, S., Hessel, M., Silver, D., Barreto, A., and Borsa, D. (2020). Expected eligibility traces. *arXiv preprint arXiv:2007.01839*.

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., and Sutton, R. S. (2016). True online temporal-difference learning. *ArXiv*, abs/1512.04087.

Venkattaramanujam, S., Crawford, E., Doan, T. V., and Precup, D. (2019). Self-supervised learning of distance functions for goal-conditioned reinforcement learning. *ArXiv*, abs/1907.02998.

Voita, E. and Titov, I. (2020). Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, Online. Association for Computational Linguistics.

Volf, P. A. and Willems, F. M. (1998). Switching between two universal source coding algorithms. In *Data Compression Conference, 1998. DCC'98. Proceedings*, pages 491–500. IEEE.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Wasserman, L. (2000). Bayesian Model Selection and Model Averaging. *Journal of Mathematical Psychology*, 44.

Weissman, T., Ordentlich, E., Seroussi, G., Verdu, S., and Weinberger, M. J. (2003). Inequalities for the L1 deviation of the empirical distribution.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Wikipedia (2021). Weyl law.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *NIPS*, pages 4148–4158.

Wu, Y., Tucker, G., and Nachum, O. (2019). The laplacian in rl: Learning representations with efficient approximations. *ArXiv*, abs/1810.04586.

Xu, J., Zhang, H., and Zikatanov, L. (2017a). On the weyl's law for discretized elliptic operators. *arXiv preprint arXiv:1705.07803*.

Xu, T.-B., Yang, P., Zhang, X.-Y., and Liu, C.-L. (2017b). Margin-Aware Binarized Weight Networks for Image Classification. In *International Conference on Image and Graphics*, pages 590–601. Springer, Cham.

Zagoruyko, S. (2015). 92.45% on CIFAR-10 in Torch.

Zhang, A., Wu, Y., and Pineau, J. (2018). Natural environment benchmarks for reinforcement learning. *CoRR*, abs/1811.06032.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017a). Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations*.

Zhang, J., Springenberg, J. T., Boedecker, J., and Burgard, W. (2017b). Deep reinforcement learning with successor features for navigation across similar environments. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)
université PARIS-SACLAY

Titre: Sur certaines méthodes raisonnées pour l'apprentissage par renforcement profond
Mots clés: Apprentissage profond, apprentissage par renforcement, evaluation de politiques, apprentissage multi-objectif

Résumé: Cette thèse développe et étudie certaines méthodes de principe pour l'apprentissage profond (DL) et l'apprentissage par renforcement (RL).

Dans la partie II, nous étudions le DL selon le point de vue du principe de "Minimum Description Length", qui formalise le rasoir d'Occam, et postule qu'un bon modèle prédictif est un modèle capable de compresser sans perte les données (en prenant en compte le coût de la description du modèle lui-même). Les modèles de DL, par le nombre de paramètres à encoder, semblent aller à l'encontre de ce principe. Nous démontrons expérimentalement la capacité de compression des modèles de DL, même en tenant compte de l'encodage des paramètres, montrant ainsi que ces approches sont bien fondées du point de vue de la théorie de l'information.

Dans la partie III, nous étudions deux limitations des approches standard de DL et RL, et nous développons des méthodes mathématiquement bien fondées pour les dépasser; La première concerne l'optimisation des modèles de DL avec SGD, et le coût important du choix d'un bon taux d'apprentissage. Nous introduisons la méthode Alrao (All learning rates at once) : chaque unité (ou neurone) du réseau obtient son propre taux d'apprentissage tiré aléatoirement à partir d'une distribution couvrant de nombreux ordres de grandeur. De façon surprenante, Alrao obtient des résultats proches de ceux de SGD avec un taux d'apprentissage optimal, et ce pour diverses architectures et problèmes. Le second aborde les environnements de RL en temps quasi continu (robotique, contrôle, jeux vidéos,. . .) : nous montrons que la discrétisation temporelle (nombre d'actions par seconde) est un facteur critique, et empiriquement que les approches basées sur Q-learning ne peuvent plus apprendre quand le nombre d'action par seconde devient grand. Formellement, nous prouvons que le Q-learning n'existe pas en temps continu. Nous détaillons une méthode mathématiquement bien fondée pour construire un algorithme RL invariant à la discrétisation temporelle, et confirmons cette approche empiriquement.

La partie principale de cette thèse, (Partie IV), étudie l'opérateur des états successeurs en RL, et comment il peut améliorer l'efficacité de l'apprentissage de la fonction valeur. Dans un environnement où la récompense n'est reçue que très rarement, l'apprentissage de la fonction valeur est un problème difficile. L'opérateur des états successeurs est un objet mathématique qui exprime les fonctions valeur de toutes les fonctions de récompense possibles pour une politique fixe. L'apprentissage de cet opérateur peut se faire sans signaux de récompense et peut extraire des informations de chaque transition observée, illustrant une approche de RL non supervisé. Nous proposons un traitement formel de cet objet dans des espaces finis et continus avec des approximateurs de fonctions, comme les réseaux de neurones. Nous présentons plusieurs algorithmes d'apprentissage et les résultats associés. De même que la fonction valeur, l'opérateur des états successeurs satisfait une équation de Bellman. De plus, il satisfait également deux autres équations à point fixe : une équation de Bellman en arrière et une équation de Bellman-Newton, exprimant la compositionalité des chemins dans le processus de Markov. Ces nouvelles relations nous permettent de généraliser à partir des trajectoires observées de plusieurs façons, ce qui peut conduire à une plus grande efficacité en pratique.

Enfin, (partie V), l'étude de l'opérateur des états successeurs et de ses algorithmes nous permet de dériver des méthodes non biaisées dans le cadre d'un RL à buts multiples. Nous montrons en outre que l'algorithme Hindsight Experience Replay, populaire dans ce cadre mais connu pour être biaisé, est en fait non biaisé dans la classe importante des environnements déterministes.

Title: Some Principled Methods for Deep Reinforcement Learning
Keywords: Deep Learning, Reinforcement Learning, Policy evaluation, Multi-goal reinforcement learning

Abstract: This thesis develops and studies some principled methods for Deep Learning (DL) and deep Reinforcement Learning (RL).

In Part II, we study the efficiency of DL models from the context of the Minimum Description Length principle, which formalize Occam's razor, and holds that a good model of data is a model that is good at losslessly compressing the data, including the cost of describing the model itself. Deep neural networks might seem to go against this principle given the large number of parameters to be encoded. Surprisingly, we demonstrate experimentally the ability of deep neural networks to compress the training data even when accounting for parameter encoding, hence showing that DL approaches are well principled from this information theory viewpoint.

In Part III, we tackle two limitations of standard approaches in DL and RL, and develop principled methods, improving robustness empirically. The first one concerns optimisation of deep learning models with SGD, and the cost of finding the optimal learning rate, which prevents using a new method out of the box without hyperparameter tuning. When design a principled optimisation method for DL, 'All Learning Rates At Once' : each unit or feature in the network gets its own learning rate sampled from a random distribution spanning several orders of magnitude. Perhaps surprisingly, Alrao performs close to SGD with an optimally tuned learning rate, for various architectures and problems. The second one tackles near continuous-time RL environments (such as robotics, control environment, . . .) : we show that time discretization (number of action per second) in as a critical factor, and that empirically, Q-learning-based approaches collapse with small time steps. Formally, we prove that Q-learning does not exist in continuous time. We detail a principled way to build an off-policy RL algorithm that yields similar performances over a wide range of time discretizations, and confirm this robustness empirically.

The main part of this thesis, (Part IV), studies the Successor States Operator in RL, and how it can improve sample efficiency of policy evaluation. In an environment with a very sparse reward, learning the value function is a hard problem. At the beginning of training, no learning will occur until a reward is observed. This highlight the fact that not all the observed information is used. Leveraging this information might lead to better sample efficiency. The Successor State Operator is an object that expresses the value functions of all possible reward functions for a given, fixed policy. Learning the successor state operator can be done without reward signals, and can extract information from every observed transition, illustrating an unsupervised reinforcement learning approach. We offer a formal treatment of these objects in both finite and continuous spaces with function approximators. We present several learning algorithms and associated results. Similarly to the value function, the successor states operator satisfies a Bellman equation. Additionally, it also satisfies two other fixed point equations: a backward Bellman equation and a Bellman-Newton equation, expressing path compositionality in the Markov process. These new relation allow us to generalize from observed trajectories in several ways, potentially leading to more sample efficiency. Every of these equations lead to corresponding algorithms for any function approximators such as neural networks.

Finally, (Part V) the study of the successor states operator and its algorithms allow us to derive unbiased methods in the setting of multi-goal RL, dealing with the issue of extremely sparse rewards. We additionally show that the popular Hindsight Experience Replay algorithm, known to be biased, is actually unbiased in the large class of deterministic environments.