On actions that matter : credit assignment and interpretability in reinforcement learning

Johan Ferret

Université de Lille
École Doctorale MADIS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in **Computer Science**

submitted by

Johan Ferret

ON ACTIONS THAT MATTER: CREDIT ASSIGNMENT AND INTERPRETABILITY IN REINFORCEMENT LEARNING

---

under the supervision of **Olivier Pietquin & Philippe Preux**.

Publicly defended on **July 4th, 2022** at **Villeneuve d'Ascq, France**, to the doctoral committee

Mr. David **Filliat**	ENSTA Paris	Thesis Referee & Committee Chair
Mr. Fabien **Moutarde**	Mines de Paris	Thesis Referee
Ms. Ofra **Amir**	Technion	Thesis Jury
Mr. Théophane **Weber**	DeepMind	Thesis Jury
Mr. Olivier **Pietquin**	Google Research	Thesis Supervisor
Mr. Philippe **Preux**	Inria, Université de Lille	Thesis Supervisor
Mr. Matthieu **Geist**	Google Research	Invited Member

Research Center in Computer Science, Signal and Automatic Control of Lille (CRIStAL),
UMR 9189 Scool team, 59650, Villeneuve d'Ascq, France

Université de Lille
École Doctorale MADIS

Thèse de Doctorat

Spécialité **Informatique**

présentée par
Johan Ferret

De l'Importance des Actions: Assignation de Crédit et Interprétabilité pour l'Apprentissage par Renforcement

---

sous la direction de **Olivier Pietquin** & **Philippe Preux**.

Soutenue publiquement le **4 Juillet 2022** à **Villeneuve d'Ascq, France** devant le jury composé de

M. David **Filliat**	ENSTA Paris	Rapporteur & Président du jury
M. Fabien **Moutarde**	Mines de Paris	Rapporteur
M^{me} Ofra **Amir**	Technion	Examinatrice
M. Théophane **Weber**	DeepMind	Examinateur
M. Olivier **Pietquin**	Google Research	Directeur de thèse
M. Philippe **Preux**	Inria, Université de Lille	Directeur de thèse
M. Matthieu **Geist**	Google Research	Invité

Centre de Recherche en Informatique, Signal et Automatique de Lille (CRIStAL),
UMR 9189 Équipe Scool, 59650, Villeneuve d'Ascq, France

"So high,
So low,
So many things to know."
- Vernor Vinge

"It is not down on any map, true places never are."
- Herman Melville

To my parents, sisters, family and friends.

Contents

Chapter 1

Overview of the thesis

1.1 Short summary

This thesis, written for the qualification of Doctor of Philosophy in Computer Science, studies the question of the individual importance of actions in sequential decision-making, through the lens of Reinforcement Learning, and with diverse applications. An important finding of this work is that two seemingly different open problems in Reinforcement Learning, namely the credit assignment problem and explainability, have partial solutions that involve similar tools that can all be viewed as the estimation of particular forms of action importance. Several algorithms that suit different forms of action importance are proposed and studied from a theoretical point of view, empirically, or both.

1.2 Court résumé

Cette thèse, écrite pour la qualification de Docteur en Informatique, étudie la question de l'importance individuelle des actions dans la prise de décision séquentielle, via le prisme de l'Apprentissage par Renforcement, avec diverses applications. Une découverte importante de ce travail est que deux problèmes ouverts en apparence distincts en Apprentissage par Renforcement, à savoir le problème d'assignation de crédit et l'interprétabilité, ont des solutions partielles qui impliquent des outils communs qui peuvent être vus comme des estimations de formes particulières d'importance des actions. Des algorithmes qui correspondent à des formes distinctes d'importance des actions sont proposées et étudiées empiriquement, d'un point de vue plus théorique, ou les deux.

1.3 Publications

Here is the complete list of research articles that were co-written during the course of this thesis' work.

- **Self-Attentional Credit Assignment for Transfer in Reinforcement Learning** (Ferret et al., 2020)
  Ferret J., Marinier R., Geist M., Pietquin O.
  IJCAI 2020
  Covered in Chapter 6.

- **Self-Imitation Advantage Learning** (Ferret et al., 2021)
  Ferret J., Pietquin O., Geist M.

AAMAS 2021 (oral)
Covered in Chapter 5.

- **Adversarially-Guided Actor-Critic** (Flet-Berliac et al., 2021a)
  Ferret J.*, Flet-Berliac Y.*, Preux Ph., Pietquin O., Geist M.
  ICLR 2021
  Covered in Appendix A.

- **There is no Turning Back: A Self-Supervised Approach to Reversibility-Aware Reinforcement Learning** (Grinsztajn et al., 2021)
  Ferret J.*, Grinsztajn N.*, Preux Ph., Pietquin O., Geist M.
  NeurIPS 2021
  Covered in Chapter 8.

- **Lazy-MDPs: Towards Interpretable Reinforcement Learning by Learning When to Act** (Jacq et al., 2022)
  Ferret J.*, Jacq A.*, Pietquin O., Geist M.
  AAMAS 2022 (oral)
  Covered in Chapter 7.

- **More Efficient Exploration with Symbolic Priors on Action Sequence Equivalences** (Johnstone et al., 2021)
  Johnstone T., Grinsztajn N., **Ferret J.**, Preux Ph.
  Preprint

1.4 Statements of authorship

Self-Attentional Credit Assignment for Transfer in Reinforcement Learning OP proposed the research direction and initial steps. JF did initial experiments and proposed using a Transformer model in place of the LSTM model. JF proposed the Triggers environment for principled study of credit assignment. JF did the Triggers experiments and analysis. RM did the DMLab experiments and analysis. JF produced the figures and videos for the paper. MG and OP provided helpful feedback on the supervised learning procedure. JF and OP wrote the major part of the article. MG provided helpful feedback on writing too.

Self-Imitation Advantage Learning MG proposed the research direction and initial steps. JF performed the experiments, analyzed the results, produced the figures for the paper and wrote the major part of the article. MG and OP supervised the progress, made decisive suggestions and helpful comments on the draft. JF was responsible for open-sourcing the code to reproduce paper results.

Adversarially-Guided Actor-Critic YFB proposed the research direction and the collaboration. YFB and JF led the research project with OP, PP and MG serving as advisors. YFB implemented the method for PPO and came up with the positive initial experimental results. JF reimplemented the method and conducted partial experiments with IMPALA, which were not fruitful, and PPO. JF proposed to simplify the algorithm (removing the Transformer part), found better hyperparameter configurations and provided the heatmap visualization. YFB found that reducing the learning rate for the adversary was beneficial. MG provided the theoretical analysis of a simplified version of AGAC. YFB and JF equally contributed to the writing of the article. MG, PP and OP also contributed to the writing. The final results and experiments used in the paper were conducted by YFB.

There is no Turning Back: A Self-Supervised Approach to Reversibility-Aware Reinforcement Learning JF proposed the research direction and the collaboration. NG and JF led the research project with OP, PP and MG serving as advisors. JF designed the method and performed early experiments to validate the idea. NG came up with the formalism and wrote the theoretical analysis. MG proof checked these. NG did the Turf and Cartpole experiments while JF did Sokoban experiments on a toy instance of Sokoban. JF designed the final Sokoban experiments and analysis. NG and JF equally contributed to the writing of the article. MG, PP and OP also contributed to the writing.

Lazy-MDPs: Towards Interpretable Reinforcement Learning by Learning When to Act JF proposed the research direction and the collaboration. AJ and JF led the research project with OP and MG serving as advisors. JF designed the method and performed early experiments to validate the idea. AJ came up with the formalism and wrote the theoretical analysis. MG proof checked these. AJ did the tabular and KDT experiments. JF designed the final Atari experiments and analysis. AJ and JF equally contributed to the writing of the article. MG and OP also contributed to the writing.

More Efficient Exploration with Symbolic Priors on Action Sequence Equivalences NG and TJ proposed the research direction and the collaboration. TJ conducted experiments and designed the formalism for the method. JF ran Atari experiments and proposed additional experiments. NG and TJ equally contributed to the writing of the article. JF gave feedback on various drafts of the article. PP supervised the project.

Chapter 2

Introduction

2.1 Motivation

Artificial Intelligence (AI) was coined a term by John McCarthy, Claude Shannon and Nathan Rochester in 1956. They defined it as an attempt to "find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves". More than 75 years later, AI is a living subfield of Computer Science that studies computational intelligence, that is programs that can acquire and leverage forms of knowledge autonomously, while relying or not on expert knowledge and hand-crafted rules. It departs from regular programming, whose programs have deterministic (or at least predictable) outputs given all their lines of code. Also, AI programs generally learn (i.e. change their variables and overall behavior) from their inputs, while regular programs generally do not. AI is a broad subfield, that encompasses expert systems, evolutionary computation, robotics and many others. It notably takes inspiration from cognitive sciences, drawing many parallels with human cognition in particular. At the heart of AI, and living at the crossroads of statistics, probabilistic theory, linear algebra and theoretical computer science; Machine Learning (ML) aims at building useful models of numerical phenomena that learn from data. ML models are used as tools in many fields such as Computer Vision, Natural Language Processing and Signal Processing.

One of the hallmarks of AI, and an open problem still, is artificial general intelligence (AGI). AGI will be considered reached when we build a single learning program that reaches human-level cognitive abilities in many activities. A decades long debate exists between AI researchers, parting them among the so-called "symbolists" and "connectionists": for symbolists, intelligence necessarily arises from the manipulation of symbols and logic, while for connectionists, intelligence can emerge in models such as artificial neural networks, that do not explicitly manipulate symbols and do not learn explicit rules. In a sense, it is a debate about whether intelligence requires understanding in the form of acquisition of explicit notions and rules.

Artificial neural networks are models that consist in successions of layers of neurons, each layer being connected to the previous layer, and each neuron computing a weighted sum (followed by a nonlinear function or not) of the output values of the neurons from the previous layer. We say that a neural network is deep when it is composed of many layers of neurons. Artificial neural networks roughly mimic the working of the biological neural networks that compose the brains of animals: the artificial inputs can be assimilated to the signal received by the dendritic synapses in neurons, the weighted sum to the processing by the soma, the result of the nonlinearity to the signal fired along the neuron axon and finally the outputs to the signal transmitted via terminal synapses of the axon. In what is next, we simply refer to artificial neural networks as neural networks.

An interesting property of neural networks is that, due to the combination of several layers of neurons

and intermediary nonlinearities, they can self-organize and learn to compute arbitrary combinations of their inputs. One can show that they are universal approximators (Hornik et al., 1989) with a much as two layers of neurons (if not constrained on the number of neurons in the middle layer), that is they can approximate any function under those assumptions. As a result, due to this property and the fact that they can combine input features to produce new ones, neural networks are both powerful and flexible estimators.

Starting from 2012, with the work of Krizhevsky et al. (2012) that won the ImageNet contest (Russakovsky et al., 2015) using a large neural network, AI and ML experienced a revolution due to the coming of age of Deep Learning (DL), which is about the study of deep neural networks. Neural networks were not new at all in the sense that they were invented before 1960 (Rosenblatt, 1958), (they were called multilayer perceptrons at that point) and the canonical algorithm to train them (i.e. backpropagation) (Rumelhart et al., 1986) around 1980. What was new was the unprecedented scale at which neural networks were operating, being able to predict somehow accurately the class of natural pictures among a thousand classes, outperforming other Computer Vision (CV) methods by a wide margin. The reasons for that change of paradigm for neural networks are multiple: 1) hardware advances including the manufacturing of ever more powerful Graphical Processing Units (GPUs), which are particular chips that can perform a large number of operations in parallel, making them suitable for the matrix multiplications and tensor operations and thus, neural networks; 2) algorithmic advances alleviating long-known issues of neural networks such as overfitting (Srivastava et al., 2014) or vanishing/exploding gradients (Kingma and Ba, 2014; Ioffe and Szegedy, 2015); and 3) the democratization of larger, standard datasets available to the academic world.

This progress on the ImageNet dataset (Deng et al., 2009) fueled the interest for neural networks, which then found new successes on natural language processing, speech processing (and signal processing in general), and many other applications. A very important consequence of the ImageNet success is the democratization of the fine-tuning approach: large neural networks are trained on equally large datasets for a long time (which is usually quite expensive and requires dedicated hardware and expertise), and serve as the starting point of training for neural networks used in downstream tasks (i.e. more specialized applications, with less data available). Successes of this kind first appeared in Computer Vision, fine-tuning neural networks trained using ImageNet data on new image tasks such as other visual classification tasks, image segmentation or image retrieval. The same phenomenon happened for natural language processing a few years after, with large-scale pre-training on Wikipedia data using large, bi-directional sequence neural networks (BERT, (Devlin et al., 2018)) and fine-tuning on tasks such as named entity recognition or question answering. Those successes came with the widespread adoption of neural networks as off-the-shelf tools to deal with images, text, as well as other modalities, and with connectionists quietly scoring points against symbolists.

The aforementioned methods and challenges all supposed that labelled data (i.e. data annotated with the ground truth regarding aspects to be modelled) was readily available, and additionally with quality labelling. Providing labels to data is what is called supervision. A drawback of supervision is that, generally, it involves manual labeling, which is costly, time consuming and not always reliable. As a counterpoint to supervised learning, semi-supervised learning techniques (Chapelle et al., 2009) make use of both labelled and unlabelled data, with the usual hypothesis that unlabelled data is available in large quantities while labeled data is limited. Going even further, at the extreme of the spectrum lies unsupervised learning (also called self-supervised learning in recent works), which only considers unlabelled data. Impressively, recent advances bridge the gap between the performance of self-supervised and supervised approaches (Chen et al., 2021), which is very promising progress for the many tasks where mass supervision is unrealistic or impossible.

While supervised learning (and variants such as self-supervised, semi-supervised, active learning) are an important part of AI, they are not straightforward to apply in the setting of interest of this work: that of sequential decision-making. Roughly speaking, the goal of sequential decision-making is to find

sequences of actions (or strategies) that are optimal according to an objective function. The widely adopted objective function we will consider in this work is the discounted sum of rewards. Note that this is the objective function used in most sequential decision-making works, and that there are others possible. A reward is obtained after each action, and is a scalar such that optimal behaviors maximize the said objective. Rewards generally cannot play the role of supervision because 1) they only hint at the current desirability of the decision taken, instead of determining the exact action to be taken, 2) they can be sparse and 3) the best course of action depends on future behavior as well. As an example of a sparse reward function, there are many scenarios in which a reward is only obtained at the end of the interaction (i.e. when reaching an absorbing state). Generally speaking, the field of sequential decision-making views the aforementioned families of techniques as tools, but not as off-the-shelf solutions to the specific issues that make sequential decision-making difficult. Still, neural networks are useful in sequential decision-making, as we will discuss later on.

There are several ways to address sequential decision-making problems, most of which target specific families of problems. Evolutionary algorithms (Deb, 2011; Mania et al., 2018) involve the perturbation and recombination of a large number of intermediate candidate solutions based on approximate gradients of an objective function. These kinds of approaches are useful but usually quite expensive, and do not leverage the fact that we might be able to compute gradients of quantities of interest. Model predictive control (Camacho and Alba, 2013), and optimal control (Sethi, 2019) in general, make use of a model of the environment the agent operates in, and recursive planning to select actions. Planning consists in simulating possible future scenarios in order to choose the best course of action. This is a powerful approach but it requires having access to a model of the environment in the first place, which might be costly or even unrealistic in complex scenarios. Another family of approaches is bandit algorithms (Auer, 2002; Lattimore and Szepesvári, 2020), which are thought in particular for multi-armed bandit problems. The canonical multi-arm bandit problem can be stated as such: given n single-armed bandit machines with payoffs distributed following distinct, unknown probability distributions; what strategy to adopt in order to maximize profit? A particularity of that problem is that the past decisions do not affect the outcomes of the next ones (e.g. like in a casino, where a new drawing of the roulette is not affected by previous drawings). There exist many refinements to that problem (contextual bandits, combinatorial bandits, etc). There lacks an important aspect to the multi-armed bandit problem, though. In many applications of sequential decision-making, such as in games, past decisions do influence the next ones. For instance, in chess, the state of the board is the result of the decisions of the two players, and it should greatly influence the next move of any player. Accordingly, bandit algorithms (and more generally algorithms that apply preferentially to bandit problems) cannot be the solution to general decision-making problems (which is, of course, fine!).

Reinforcement Learning (RL) (Sutton and Barto, 2018) is a family of techniques that aim to tackle sequential decision-making problems that are instances of a specific family of problems: Markov Decision Problems (MDPs, see Chapter 3.2). MDPs work as follows (see Fig. 2.1): a learning agent observes the current state s_t (or a partial view of the state o_t) of the environment. It decides on an action a_t and acts accordingly, in return the environment updates its state, the agent receives a reward r_t and the new environment state s_{t+1} (or partial view o_{t+1}). In standard MDPs, the reward function and transition kernel are supposed to be Markov. The Markov property implies that the probability of the next reward (resp. next state) given the current state and action is equal to the corresponding probability given all states and actions up to the current point. Interestingly, most problems of interest can be viewed as MDPs, or relaxed form of MDPs. A common relaxation is the partially observable MDP (POMDP, see Chapter 3.2), which is a superset of the MDP. In a POMDP, the agent observes a partial state (we call observation) o_t that does not have to satisfy the Markov property. In many real-world scenarios, the Markov hypothesis is quite limiting. As humans for instance, we only observe a tiny fraction of the overall state of the universe at a time, so we cannot really say that we live in an MDP. Those scenarios can generally be represented as POMDPs.

When a model of the environment is known (that is, the reward function and transition kernel of the

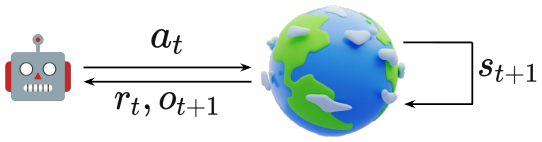


Figure 2.1: The interaction between agent and environment in RL.

environment), Dynamic Programming (Howard, 1960) is a set of techniques that leverage fixed point convergence to obtain deterministic, optimal policies. A drawback is that having access to a model of the environment is generally unrealistic.

Instead, the philosophy of RL is that the components of the objective function, i.e. the individual rewards obtained, contain enough information to find an optimal course of action (Silver et al., 2021). Everything should be learned from scratch, so there should be no need for models, expert knowledge, symbolic logic, handcrafted priors or biases to reach optimal decision-making. RL algorithms gradually learn to improve at the task at hand by reinforcing pertinent actions, which is achieved through trial-and-error. A core problem of RL is the exploration/exploitation dilemma. It can be stated as the following question: given the knowledge acquired about the task and the current situation, should an agent try to gather more information (i.e. explore) or take advantage of what it already knows (i.e. exploit). Another way to put it is that the agent can either act best according to its current estimation, or take a different, maybe under-exploited mode of action. While there are provably optimal exploration algorithms (under constraints) (Brafman and Tennenholtz, 2002), and while exploitation is trivial when how much each action is worth is known, in general finding the right combination between exploration and exploitation is hard and RL is far from solved yet. Open questions include stable learning (i.e. how to guarantee steady progress in many tasks), generalization and transfer (i.e. how to learn in a way that transfers to new, similar tasks), sample-efficiency (i.e. how to learn from the least amount of interaction) and many others.

> One of the objects of study of this thesis is the importance of actions. The question we tackle in this work can be stated as follows: given a sequential decision-making task, what are the decisions and corresponding actions that really matter? Inversely, what are the decisions that do not really matter? These questions are important because they are at the essence of RL: reinforcing actions that matter. On a high-level, it appears that we should tend to try more actions (i.e. explore more!) in decisions that matter, and that we should try to select actions that prove to be important more often (i.e. exploit more!). Nevertheless, such claims should be thoroughly checked. Action importance is a somewhat vague entity, though. Actions can serve distinct purposes: apart from the exploration and exploitation aspects we already discussed, actions can prove crucial for more specific purposes, such as reaching a given state, or reaching a given performance. Extrapolating a bit, we could give importance to actions according to how much learning potential they have for the agent. Also, forgetting about the specific agent at hand, one could ask for a universal notion of importance, i.e. one that measures how important an action is in general. To the best of our knowledge, the question of action importance is often ignored and remains largely unanswered.

There are two natural follow-up questions that are: 1) does knowing about such decisions and appropriate course of action provide an edge (i.e. leads to performance improvements) in the context

of RL? 2) Does knowing about such decisions provide us with elements that can help us get a better grasp of the task at hand, and how autonomous agents choose to solve it? Those two questions, that we conveniently separated and that appear to be seemingly independent, provide the two research directions we study in this work, under the common angle of action importance. The first question is intimately linked to that of credit assignment (Minsky, 1961) and the second one to that of explainability (Molnar, 2019). While the two questions seem to diverge in the studies and solutions that can be envisioned, we will show in this work that there are common tools that we can use to shed more light on both of them.

We now dive deeper into the origin and consequences of the first of these two questions.

2.2 Credit assignment

In ML, credit assignment is a quite general notion. It refers to the problem of updating specific parameters of the models being learned in light of the current subset of data being learned from. We call that *structural credit assignment*, to signify that we change local parameters while keeping in mind the whole structure (that is, the rest of the parameters) of the method. There are various, model-specific or model-agnostic ways to address the structural credit assignment problem. For instance, for differentiable models, a widely adopted solution is to compute the gradient of a loss function (which measures how far the model output is from the ground truth) with respect to the model parameters and update the parameters in the direction that is opposed to the gradient (so as to reduce the loss function).

In RL, there is a credit assignment problem of a different nature. It consists in quantifying the impact of choices or decisions on observed outcomes (Minsky, 1961). We call that problem the *temporal credit assignment problem*. Usually, structural credit assignment and temporal credit assignment are distinguished and treated as different problems. Structural credit assignment, as in ML, aims at measuring the influence of the parameters of the agent on a scalar measure that depends on the output of this function. In that context, the individual values of the parameters are considered as decisions. Structural credit assignment is thus a very general problem since it is involved each time the parameters of the agent are tuned to maximize the objective function. In contrast, temporal credit assignment is entirely specific to sequential decision-making. Indeed, the temporal credit assignment problem (Sutton, 1984), which is also known as the distal reward problem in behavioral science (Hull, 1943), is that of evaluating the effects of a subset of the sequence of decisions made on observed outcomes. In RL, we are particularly interested in measuring the influence of individual decisions (or actions) over the cumulative reward, since the main objective of RL agents is to maximize it. In what is next, and when not ambiguous, we will often refer to the temporal credit assignment problem simply as the credit assignment problem.

In RL, there are several factors that make credit assignment difficult, most of which also contribute to making RL distinct from the supervised learning paradigm. The delay between actions and consequences is the most important factor. A direct implication of delay is that the attribution of observed outcomes is ambiguous, since they are potentially explained by all preceding actions, by a subset of the actions, or by a unique action. Stochasticity in the rewards and/or dynamics of the environment adds variance in the outcomes, making action-outcome identification harder. Partial observability means that the Markov assumption on rewards and dynamics does not hold anymore, and that unless the underlying state of the environment can be recovered, outcomes cannot be accurately estimated without memorizing past data. Finally, non-stationarity adds a dependence on time to rewards and dynamics, further complexifying action-outcome identification.

The consequences of credit assignment being difficult when combinations of previous factors occur is that standard RL methods are not efficient at learning useful policies, when they do manage to learn those. This can be explained by taking a high-level look at RL algorithms. Most are based on Dynamic Programming (DP) (Bellman, 1957; Howard, 1960). DP methods propagate the value of states (i.e. how much discounted cumulative reward one can hope to collect from a given state) to previous states. The

value is then used to guide or dictate the action selection process. This mode of propagation (from a state to the preceding states) is called Temporal Differences (TD). In practice, when the state space becomes large enough, DP is traded for approximate (i.e. sampling-based) methods equipped with function approximation. But the core workings of DP (i.e. the Bellman equation) are still at play, hence temporal delays between actions and corresponding rewards can cause exponential slowdowns in the learning of agents. This is a problem because in complex tasks, delays between actions and consequences can be arbitrarily long. Additionally, Harutyunyan et al. (2019) mention fundamental issues of the value function and the problem it embodies (*"how does choosing an action a in state s affect future return?"*), often tackled using TD learning. A first problem is variance, for Monte-Carlo estimation of the value averages the returns observed after taking a given action in a given state and is inefficient due to the compounding variance of trajectories. Another is partial observability, which makes rewards dependent on the full sequence of observation-action pairs and introduces bias to TD learning (Singh et al., 1994), which may cause divergence. A third problem is the reliance of TD on time as a credit assignment mechanism, which is not universal and could be improved by learning. Last but not least, values are typically updated for the actions observed, while other actions could be updated at the same time. While there are modifications to TD that alleviate these issues (such as TD(λ) (Sutton, 1988), which we introduce and discuss later on), thee are generally not enough to solve the credit assignment issue efficiently.

Instead, credit assignment methods, by identifying the contribution of actions to potentially farsighted outcomes, could alleviate the exponential slowdown problem by linking outcomes to actions that can then be reinforced during the learning procedure.

2.3 Explainability

Broadly speaking, explainability assesses the quality of the justification of decisions. Ideally, both domain experts and non-experts should be able to make sense of those justifications.

In ML, explainability has two complementary definitions (Molnar, 2019): the first one refers to the degree to which a human can understand a model's result (Miller, 2019), or the degree to which a human can consistently predict a model's result (Kim et al., 2016).

It is important, for what follows, to note that we equate explainability with interpretability, and that we use both terms with equal meaning in this manuscript. It is equally important to understand the distinction between explainability and explanations. It is indeed commonly accepted that explanations focus on individual predictions, while explainability is more general and can look at several predictions at once, or even at the overall workings of a model.

In the same vein, explainability does not necessarily come under the form of natural language answers to user-formulated queries. Examples include the widely used saliency maps (Simonyan et al., 2013) that represent the per-pixel gradient of the output of a model with respect to its image input. In natural language processing, their equivalents are heatmaps representing the per-token attention weight. In general, for differentiable models, the gradient is a very useful object for interpretability, because it quantifies how much local perturbations affect the model decision. Usually, we deem the directions of perturbation associated with the largest gradient values as important to the model decision.

A major difference that opposes RL to general ML is that, usually, the whole decision-making process of RL agents is not differentiable, because the internal mechanisms of the environment (i.e. the dynamics and reward functions) are not. Even if they were, we make the assumption that we do not have direct access to those. As a result, it is generally possible to apply similar explainability techniques as in supervised learning, which for the most part make use of gradients, at the scale of a single agent output (i.e. an individual action); but these do not apply at the scale of a whole trajectory of interaction.

The finality is different too. The goal of explainable RL (XRL) (Puiutta and Veith, 2020) is to provide intelligible insights about RL agents' behavior, ranging from visualizations to proper explanations. This

can be useful for plenty of reasons, the main ones being 1) to explain and summarize agent behavior, 2) to better understand what and how agents learn, 3) to gain transparency and trust regarding individual decisions or the agent as a whole. These are crucial features if we want to apply RL in the real world, where decisions can directly or indirectly impact the actual lives of people. In any case, to make sense of agents' behaviors, one should take a look at sequences of actions instead of individual actions, which calls for different techniques as in standard ML.

Another aspect that we take an interest on in this work is getting insights about the task at hand. While we have an instinctive understanding of most tasks we benchmark RL agents in, we might want to push our understanding of what is possible to do and achieve in those tasks. Conversely, we might have applications where we have a poor understanding of what the objective is. Accordingly, behaviors learned via RL can reveal precious information about the task and environment that might have escaped human predictions. For instance, RL agents have discovered and exploited known and less known bugs in Atari 2600 video games, reaching superhuman scores doing so. Also, in physical simulators, agents have been shown to abuse the physics (for instance throwing a hammer towards a button instead of using the hammer to hit said button) in order to complete the objective. Such cases are spectacular but not very interesting from the perspective of explainability since they tend to show that the original reward functions of the tasks considered were misspecified, allowing optimal behaviors that were not anticipated and should not be. More interesting examples in that line of research include the work of Tomašev et al. (2020), which uses an RL algorithm to explore novel, alternative sets of rules for the game of chess; deriving a measure of game balance from the analysis of games played by the RL agent. Another interesting work is that of McGrath et al. (2021), which looks at how RL agents acquire knowledge related to chess, which provides a different look to skill acquisition in chess compared to the human player studies.

We refer readers eager to know more about interpretability in the context of ML to the work of Molnar (2019).

2.4 Organization of the manuscript

The manuscript is organized as follows. Chapter 3 covers the background and related work that is important to get the most of this manuscript. It starts by introducing foundational ML concepts that our work largely builds upon. It then moves to a short background on RL, elements of which are also part of the cement that made our work possible. It then briefly reviews the existing literature on credit assignment for RL in Chapter 3.3, with further details in Chapter 4. It also briefly reviews the literature on interpretability in RL in Chapter 3.4. The two next chapters (Chapter 5, 6) take distinct perspectives on the question of *action importance for the credit assignment problem in RL*. Chapter 5 studies the question of *action importance regarding the overall performance of RL agents*. It introduces a simple modification to the learning loss of off-policy value-based RL methods that can be viewed as adding an optimistic version of the advantage to the reward function. It establishes connections with several existing methods from the literature (Self-Imitation Learning (Oh et al., 2018) and Advantage Learning (Baird, 1995) most notably). Empirically, it shows that this modified loss results in improved credit assignment in various tasks including Atari 2600 games, and especially in hard exploration tasks. Chapter 6 studies the question of *action importance within episodes of interaction*. It adopts a sequential view and proposes an offline credit assignment algorithm powered by an attention mechanism over sequences of observations and actions. It addresses the problem of explaining away by artificially reducing the amount of information available in observations to the sequential estimator. It then studies the potential for transfer of the credit assigned and shows successes in environments with both modified state spaces and modified dynamics. The two chapters that follow (Chapter 7, 8) focus on the question of *action importance for explainability in RL*. Chapter 7 studies the question of *action importance in explaining performance improvements between two policies*, and develops the idea that explainability can come from how we cast our sequential

decision-making tasks. More precisely, it proposes an abstraction over environments that requires a default policy and a cost, and in turn allows agents to be lazy, that is defer their actions to the default policy. It characterizes the equivalent of various standard quantities in this new type of environments. It then shows how the abstraction can be used to get interpretability at the level of the task or at the level of specific policy improvements. Finally, Chapter 8 studies how we can assess the general reversibility of actions, which relates to the question of *action importance for reversible behaviors*. It proposes the notion of precedence as a proxy quantity for how reversible specific actions are, and binds the two quantities together from a theoretical perspective. It then studies a practical, self-supervised approach to learn to establish precedence from interaction data, and proposes two distinct ways to incorporate the resulting estimated reversibility into general RL procedures. In experiments, it shows the versatility of the approach in various use cases ranging from safety to structured exploration in the game of Sokoban.

Chapter 3

Background and related work

In this section, we provide the mathematical definitions and tools that the reader will need to follow the course of this thesis. We start by introducing general Machine Learning concepts (in Chapter 3.1) that the experimented reader can choose to skip. In Chapter 3.2, we detail general Reinforcement Learning concepts that the versed reader might, again, choose to skip. We then go on presenting more advanced, specific concepts (in Chapter 3.3 and 3.4, notably related to credit assignment and interpretability) that we advise the reader to get familiar with for their later comfort.

3.1 Machine Learning background

Here, we introduce important ML concepts that will be used in throughout the text. We do not go into much detail for the sake of brevity, but the interested reader may refer to Murphy (2012) for a rigorous introduction to ML, and to Goodfellow et al. (2016) for an exhaustive treatment of Deep Learning.

Estimation problems An estimator is a function $\hat{f} : \mathbb{R}^d \to \mathcal{X}$ where $\mathcal{X}$ is its output space. Its role is to approximate an unknown function f based on a set $\mathcal{D} = (x, y)$ that we call dataset. For a given pair (x, y) we call x a data point and y a label. An estimation problem is a problem of the form:

$$\hat{f} = \min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \Big[\mathcal{L}(y, f(x)) \Big],$$

where $\mathcal{F}$ is the set of functions considered, and $\mathcal{L} : (\mathcal{X}, \mathcal{X}) \to \mathbb{R}$ is a loss function that penalizes function outputs proportionally to an error between $f(x)$ and y. How the error is calculated is problem dependent, as different types of errors suit different types of problems. Classical problems include the regression problem, where $\mathcal{X} = \mathbb{R}$ and the loss is the squared error:

$$\mathcal{L}(y, \hat{f}(x)) = \frac{1}{2} \Big(y - \hat{f}(x) \Big)^2,$$

and the multi-class classification problem, where $\mathcal{X} = \Delta_{[\![1,n]\!]}$ for n-class classification and the loss is the negative log-likelihood:

$$\mathcal{L}(y, \hat{f}(x)) = \sum_{i=1}^{n} \mathbb{1}\{y = i\} \log \hat{f}(x)_i.$$

Here Δ_X is the simplex over the set X, that is the set:

$$\Delta_X = \left\{ \big(\rho_i \in [0, 1]\big)_{i \in [\![1,|X|]\!]} \text{ such that } \sum_{i=1}^{|X|} \rho_i = 1 \right\}.$$

In multi-class classification, the estimated class is defined as:

$$\hat{y} = \arg\max_i \hat{f}(x)_i.$$

Both problems admit sequential forms. The sequential regression problem has an output space $\mathcal{X} = \mathbb{R}^T$. The sequential multi-class classification problem has an output space $\mathcal{X} = \Delta_n^T$. Both sequential losses (noted $\mathcal{L}_T$ here) can be derived from their respective non-sequential losses:

$$\mathcal{L}_T(y, f(x)) = \frac{1}{T}\sum_{i=1}^{T} \mathcal{L}\Big(y_i, \hat{f}(x_{\leq i})\Big).$$

In some cases, including cases of interest, the length of the output can depend on the length of the input. The output space is then $\mathcal{X} = \mathbb{R}^{T_{max}}$, with T_{max} being the maximum input length, and some outputs being non-admissible (for instance any output such that $|f(x)| \neq |x|$). Losses are also adapted accordingly by treating the length of the sequence as a variable quantity.

Training differentiable models Most ML models we consider are parametric, that is their output depends on their input and on parameters that are learned using a training dataset. In what comes next, we indicate the dependency on parameters θ by the notation $\hat{f}_\theta(x)$. Additionally, most of them are said to be differentiable: their output is differentiable with respect to the input and to model parameters. We note the gradient of the output with respect to parameters $\nabla_\theta \hat{f}_\theta(x)$. Following the direction that is opposed to the one defined by the loss gradient in the space of parameters (locally) leads to loss decrease. This is called gradient descent (Cauchy et al., 1847) and will be our method of choice to train such models. We distinguish three variants of gradient descent. The first one is called batch gradient descent and follows a gradient computed on the whole dataset:

$$\theta \leftarrow^\alpha \nabla_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \mathcal{L}(y, \hat{f}_\theta(x)),$$

with $x \leftarrow^\alpha y := x \leftarrow x - \alpha y$. The second one is called stochastic gradient descent (Kiefer and Wolfowitz, 1952) and follows a gradient computed on a single example sampled uniformly from the dataset:

$$\theta \leftarrow^\alpha \nabla_\theta \mathcal{L}(Y, \hat{f}_\theta(X)),$$

where $(X, Y) \sim \mathcal{U}(\mathcal{D})$. The third one, which is the one we will use in practice, is called minibatch gradient descent and sits in the middle of both:

$$\theta \leftarrow^\alpha \nabla_\theta \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} \mathcal{L}(Y_i, \hat{f}_\theta(X_i)),$$

where $\mathcal{B} = \{(X_i, Y_i) \sim \mathcal{U}(\mathcal{D})\}_{i=1}^{|\mathcal{B}|}$. It is our method of choice because computationally speaking, batch gradient descent requires to fit the whole dataset into memory (often not realistic), and stochastic gradient descent suffers from too much variance. Minibatch gradient descent still has variance, albeit reduced compared to the stochastic version, and it was showed that the residual variance helped with exploration in the space of parameters, acting as a form of regularizer. In what follows, we use two distinct gradient descent optimizers: RMSProp (Tieleman and Hinton, 2012) and Adam (Kingma and Ba, 2014). These optimizers adapt the learning rate α based on approximations of the first and second moments of past gradients. RMSProp divides the learning rate by an exponentially decaying estimate $\hat{g}$ of the second moment of past gradients:

$$\hat{g}_t = \beta \hat{g}_{t-1} + (1-\beta) g_t^2,$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{g}_t + \epsilon}} g_t,$$

where g_t is the gradient at step t of the optimization procedure. Adam is a slightly different optimizer: it divides the learning rate by a quantity that is similar to that of RMSProp (up to a normalization constant), but it also replaces the gradient by an exponentially decaying estimate $\hat{g}_1$ of the first moment of past gradients:

$$\begin{aligned}\hat{g}_{1,t} &= \frac{1}{1-\beta_1^t}\Big(\beta_1 \hat{g}_{1,t-1} + (1-\beta_1) g_t\Big),\\ \hat{g}_{2,t} &= \frac{1}{1-\beta_2^t}\Big(\beta_2 \hat{g}_{2,t-1} + (1-\beta_2) g_t^2\Big),\\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{\hat{g}_{2,t}} + \epsilon}\hat{g}_{1,t},\end{aligned}$$

We refer the reader that would like to have a broader overview of gradient descent optimization algorithms to Ruder (2016).

Neural network estimators A neural network estimator is a function that is a succession of "layers", each layer being associated to a specific parametric function. The most straightforward example is the feed-forward neural network (also called multi-layer perceptron, MLP), which is a function

$$f(x;\theta) = W_n\Big(W_{n-1}\big(\dots W_1 x + b_1 \dots\big) + b_{n-1}\Big) + b_n,$$

where $\theta = (W_1, b_1, \dots, W_n, b_n)$ is a set of weights W_i (matrices) and biases b_i (vectors) for an n-layer neural network and $\{g_i\}_{i\in[\![1,n]\!]}$ is the set of nonlinearities used. We call the functions that follow matrix multiplications nonlinearities since they are the only source of nonlinearity of the neural network output (with respect to its input). Common nonlinearities include the sigmoid $g(x) = \frac{1}{1+\exp(-x)}$, the hyperbolic tangent $g(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$ or the rectified linear unit (ReLU) $g(x) = \max(0, x)$. Layers do not necessarily perform a simple matrix multiplication. A convolutional layer instead performs the convolution of a learned kernel over its input. Networks that combine convolutional layers and feed-forward layers are called convolutional networks. A nice property of 2D convolutions is that they are 1) locally invariant to translations and 2) relate features that are spatially close. As a result, 2D convolutions are very useful for image processing, where functions of interest are usually invariant to translations as well, and it makes sense to rely on spatially close pixels instead of relying on all pixels at the same time. Recurrent neural networks (RNNs) take as input sequences (e.g. videos) instead of single elements (e.g. images). For instance, a standard RNN admits the following recurrence relation:

$$h_{t+1} = g(W_x x_{t+1} + W_h h_t + b),$$

where h_t is what we call the hidden state of the RNN at step t in the sequence. The hidden state captures information in the sequence that the network needs for later predictions. Refinements over this standard RNN include the LSTM (Hochreiter and Schmidhuber, 1997), which supports a second type of hidden state that is updated less frequently and allows to retain information longer. Another improvement comes from the use of attention mechanisms. They relate elements in the same layer or across layers (either all elements or a local subset) by matrix multiplications. They allow representations learned to take elements from the whole sequence as context, at the cost of additional computation. Attention mechanisms are core to Transformer models (Vaswani et al., 2017). In contrast to RNNs and LSTMs, Transformer models are non-autoregressive, i.e. they do not have any recurrence relation as in Equ. 3.1. Instead, they rely only on attention mechanisms to use sequential information in their predictions.

3.2 Reinforcement Learning background

We now present all the basic and advanced RL notions we use in further sections of the thesis. For additional details on the notions presented, we refer the interested reader to Sutton and Barto (2018) for a thorough introduction to RL, and to Puterman (1994) and Bertsekas and Tsitsiklis (1995) for a more theoretical view on RL.

Markov Decision Problems We use the Markov Decision Problem (MDP) formalism (Puterman, 1994). An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \gamma, R, P, \delta_0)$ where $\mathcal{S} \subset \mathbb{R}^{d_\mathcal{S}}$ is a state space, $\mathcal{A} \subset \mathbb{R}^{d_\mathcal{A}}$ is an action space, $\gamma \in [0, 1)$ is the discount factor used in the objective function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [r_{min}, r_{max}]$ is a bounded reward function that maps a transition to the corresponding reward and $\delta_0 \in \Delta_\mathcal{S}$ is the initial state distribution, Δ_S denoting the simplex over S as before. Note that we choose a form that includes the resulting state in the definition of the reward function over the typical $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This is for consistency with some objects defined later on, but by abuse of notation we will sometimes note rewards $R(s, a)$ when the reward function does not depend on the transitioning state. Finally, $P : \mathcal{S} \times \mathcal{A} \to \Delta_S$ is a Markov transition kernel that maps state-action pairs to a probability distribution over resulting states. In what follows, we will consider discrete action spaces only, i.e. $\mathcal{A} = [\![0, |\mathcal{A}| - 1]\!]$. We note that the object we describe is usually called Markov Decision Process in the RL literature, but the discount factor defines an objective function (a problem to solve), hence making it a problem and not a process. This is just a remark since it does not change any of the further results.

Interaction, value functions and optimality An RL agent interacts with an MDP at a given time-step t via its policy π, by choosing an action $a_t \sim \pi(\cdot|s_t)$ and receiving a resulting state $s_{t+1} \sim P(\cdot|s_t, a_t)$ and a reward $r_t = R(s_t, a_t, s_{t+1})$ from the environment. In a partially observable MDP (POMDP), the agent receives instead an observation $o_t \sim \mathcal{O}(\cdot|s_t)$ that contains partial information about the underlying state of the environment. In what follows, we note random variables with uppercase letters (*e.g.* S) and their realization with lowercase letters (*e.g.* s). We define the history as the random variable that summarizes all interaction data so far: $\mathcal{H}_t = \{S_0, A_0, R_0, \ldots, S_{t-1}, A_{t-1}, R_{t-1}, S_t\}$. The performance of a given policy is generally evaluated by its expected discounted cumulative reward:

$$J_\gamma(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 \sim \delta_0\right] = \mathbb{E}_\pi\left[G_{0,\gamma} \middle| S_0 \sim \delta_0\right],$$

where $G_{0,\gamma}$ is the total return, and $G_{t,\gamma}$ is the partial return from time-step t, that is $G_{t,\gamma} = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t$. The value function of a policy π is the function:

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 = s\right].$$

The action-value (or Q-value) function of a policy π is the function:

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 = s, A_0 = a\right].$$

The advantage function of a policy is calculated as $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$. An optimal policy π^* is defined as a policy that has maximal action-values in each state-action couple:

$$\pi^* \in \arg\max_{\pi \in \Pi} Q_\pi(s, a),\ \forall (s, a) \in \mathcal{S} \times \mathcal{A},$$

where Π is the set of all admissible policies. There may be multiple optimal policies depending on the task at hand.

Dynamic Programming Dynamic Programming (DP) (Howard, 1960) uses a model of the environment to learn an optimal policy in any MDP. It makes use of two equations, the first being the Bellman equation (Bellman, 1957). This equation comes from the following identity:

$$V_\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)}[r(s_t, a_t) + \gamma V_\pi(s_{t+1})].$$

Viewing the values of all states as the vector $V_\pi \in \mathbb{R}^{|\mathcal{S}|}$, and introducing R_π such that $R_\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)}[r(s_t, a_t)]$ and P_π such that $P_\pi(s_{t+1}|s_t) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s'}[P(s'|s_t, a_t)\}]$, we can write the previous identity as:

$$V_\pi = R_\pi + \gamma P_\pi V_\pi,$$

which is the Bellman equation and admits the closed form solution, given that $I - \gamma P_\pi$ is invertible:

$$V_\pi = (I - \gamma P_\pi)^{-1} R_\pi.$$

Our second equation, the Bellman optimality equation, instead comes from the identity:

$$V^*(s_t) = \max_{a \in \mathcal{A}} r(s_t, a) + \gamma \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a)}[V^*(s_{t+1})],$$

where V^* is the optimal value. Similarly as before, we have the matrix equation:

$$V^* = \max_\pi R_\pi + \gamma P_\pi V^*,$$

which is the Bellman optimality equation. We do not know a general closed form solution.

A first component of DP methods is what we call *policy evaluation*. It is an algorithm that calculates the value of a given policy. It allows us to measure how good the current policy is, which will in turns allow us to improve it in a further step. Policy evaluation consists in recursively applying the Bellman operator (starting with arbitrary initial values) until we converge to the fixed point, which is V_π. Noting the initial values $\hat{V}_0$ and the Bellman operator $\mathcal{T}_\pi$, we have the recurrence relation:

$$\hat{V}_k = \mathcal{T}_\pi \hat{V}_{k-1} = R_\pi + \gamma P_\pi \hat{V}_{k-1},$$

and the stopping criterion can be that we find k such that $||\hat{V}_k - \hat{V}_{k-1}||_\infty < \epsilon$, where ϵ is a small constant. Given that we have access to the true R_π and P_π we then have convergence guarantees:

$$\hat{V}_k \to_{k \to \infty} V_\pi.$$

A second important component of DP methods is *policy improvement*. It consists in improving over a policy whose values are known, and typically uses the result of policy evaluation as the policy values. A way to obtain a better policy from V_k is by being greedy, i.e. $\pi_{k+1} = \mathcal{G}(V_k)$ meaning that $\forall (s, a) \in \mathcal{S} \times \mathcal{A}$ we have:

$$\pi_{k+1}(a|s) = \mathbb{1}\{a = \arg\max_{a'} Q_k(s, a')\},$$

using the identity:

$$Q_k(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V_k(s')].$$

Policy iteration simply alternates between policy evaluation and policy improvement steps until the policy converges. We can show that policy iteration provably converges to an optimal policy.

Value iteration recursively applies the Bellman optimality operator (starting with arbitrary initial action-values) until the action-values converge to the optimal action-values. Noting the initial values $\hat{V}_0$ and the Bellman optimality operator $\mathcal{T}^*$, we have the recurrence relation:

$$\hat{V}_k = \mathcal{T}^* \hat{V}_{k-1} = \max_\pi R_\pi + \gamma P_\pi \hat{V}_{k-1},$$

with a similar stopping criterion to that of policy evaluation. We can show that value iteration provably converges to the optimal value function V^*, from which we get an optimal policy $\pi^* = \mathcal{G}(V^*)$.

Though seemingly disconnected, policy iteration and value iteration can be seen as special instances of *modified policy iteration*. Basically, an iteration of modified policy iteration looks like this:

$$\pi_k = \mathcal{G}(V_k),$$
$$V_{k+1} = (\mathcal{T}_{\pi_k})^n V_k,$$

with the parameter n controlling the ratio between evaluation and improvement. When $n = 1$, we recover value iteration. When $n \to \infty$, we recover policy iteration.

Reward shaping *Reward shaping* (Skinner, 1951; Mataric, 1994) is a method used to make reward functions more informative. It consists in defining a new MDP $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, \gamma, R', P)$ where $R'(s, a, s') = R(s, a, s') + F(s, a, s')$, and F is a shaping function, that usually measures the desirability of a transition based on expert knowledge. Ng et al. (1999) introduced potential-based reward shaping, i.e. using a potential function $\phi : \mathcal{S} \to \mathbb{R}$ and defining the potential-based shaping as $F(s, a, s') = \gamma\phi(s') - \phi(s)$. They showed that it preserves the set of optimal policies of the original MDP. This property is called policy invariance under reward shaping. Note that Wiewiora et al. (2003) extended this result to state-action potential functions.

Theorem 1 (Policy invariance under reward shaping, Ng et al. (1999))**.** *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \gamma, R, P)$ be an MDP, $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ be a potential-based shaping function and $\phi : \mathcal{S} \to \mathbb{R}$ its potential function such that:*

$$F(s, a, s') = \gamma\phi(s') - \phi(s),$$

then every optimal policy in $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, \gamma, R + F, P)$ is also optimal in $\mathcal{M}$ and vice-versa.

TD-learning We present *temporal differences (TD)* (Sutton, 1988) here. For later convenience, we adopt the backward perspective and place ourselves in a tabular setting, though general temporal differences can be also used in a non-tabular setting by adopting the forward perspective. The one-step TD error is the quantity:

$$\delta_t = r(s_t, a_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t),$$

which is a proxy for the current estimation error, and defines the direction and magnitude of value updates. Eligibility traces (Singh and Sutton, 1996) are defined by the recurrent expression:

$$e_{\lambda,t}(s) = \gamma\lambda e_{\lambda,t-1}(s) + \mathbb{1}\{s = s_t\},$$

where λ controls the bias-variance trade-off. These quantify how much previous state-action couples are responsible for an observed TD error. At a timestep t, TD(λ) (Sutton, 1988) updates the values of each state to $V'_\pi(s) = V_\pi(s) + \alpha\delta_t e_{\lambda,t}(s)$.

Q-learning *Q-learning* (Watkins and Dayan, 1992) is an algorithm that is based on Value Iteration. It initializes all action-values to a chosen starting value and updates them by recursively applying the Bellman optimality equation. Here is the tabular version of the algorithm:

$$\hat{Q}_{k+1}(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_k(s_{t+1}, a')$$

This is the algorithm that inspired Deep Q-Networks (Mnih et al., 2013), which we give details about in the next paragraph.

Deep Q-Networks *Deep Q-Networks* (DQN) (Mnih et al., 2013) is a form of Q-learning using neural networks for function approximation, that is, the action-values are calculated as the output of a neural network that takes as input the current environment state. It proposes the use of two novel elements: *target networks* and a *replay buffer*. Target networks are past versions of the action-value network that are used to update the current action-value in a more stable way:

$$\mathcal{L}(s_t, a_t, s_{t+1}; \theta) = \frac{1}{2}\Big(Q_\theta(s_t, a_t) - \big(r(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_{\theta^-}(s_{t+1}, a')\big)\Big)^2$$
$$\theta \leftarrow^\alpha \nabla_\theta \mathcal{L}(s_t, a_t, s_{t+1}; \theta).$$

A replay buffer $\mathcal{B}$ (Lin, 1992) is a set of previous observed states, actions, rewards and next states that is sampled uniformly from to update the action-value network and avoid "throwing away" past experiences. If we take the previous equation, we can derive the actual loss used in DQN, where transitions are randomly sampled from the replay buffer:

$$\mathcal{L}_{DQN}(\mathcal{D}; \theta) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim \mathcal{B}}\Big[\mathcal{L}(s_t, a_t, s_{t+1}; \theta)\Big]$$

with the caveat that in practice, this loss is replaced by its empirical counterpart (i.e. with actual samples and averaging over a minibatch of samples) and the Huber error replaces the classical squared error.

Advantage Learning and Gap-Increase An interesting modification to Q-learning we refer to later on is *Advantage Learning* (AL) (Baird, 1995; Bellemare et al., 2016b). The advantage of a state-action couple is the quantity $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$, and measures how much value improvement one gains by taking the action a_t in s_t over following the current policy. As an example, if π is deterministic and $\pi(a_t|s_t) = 1$ we have $A(s_t, a_t) = 0$. What AL does is that it implicitly uses the current estimation of the advantage as an auxiliary reward in a classic Q-learning scheme (here with function approximation):

$$\mathcal{L}_{AL}(s_t, a_t, s_{t+1}; \theta) = \frac{1}{2}\Big(Q_\theta(s_t, a_t) - \big(r_{AL,\theta^-}(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_{\theta^-}(s_{t+1}, a')\big)\Big)^2$$
$$\theta \leftarrow^\alpha \nabla_\theta \mathcal{L}(s_t, a_t, s_{t+1}; \theta),$$

where $r_{AL,\theta^-}(s_t, a_t) = r(s_t, a_t) + Q_{\theta^-}(s_t, a_t) - \max_{a \in \mathcal{A}} Q_{\theta^-}(s_t, a)$ and the quantity $\max_{a \in \mathcal{A}} Q_{\theta^-}(s_t, a)$ is the estimated value of the policy that selects actions greedily with respect to the target action-value. An interesting property of AL is that it is gap-increasing, i.e. it provably increases the action-gaps over the standard Q-learning update. The *action-gap* (Farahmand, 2011; Bellemare et al., 2016b; Vieillard et al., 2020b) measures the difference in action-value between best and second-best actions in a given state:

$$\Delta_Q(s_t) = \max_{a_1 \in \mathcal{A}} Q(s_t, a_1) - Q(s_t, a_2),$$

where $a_2 = \arg\max_{a' \in \mathcal{A} \setminus \{a_1\}} Q(s_t, a')$ is the second-best action. It is an important quantity that we will come back to often in this work.

Actor-critics *Actor-critic methods* are different to those that descend from Q-learning. They usually consist of two components: a policy network (which chooses the action based on the current state) and a value network (which estimates the value of the current state). The policy network is updated via policy gradients (Williams, 1992), using the value from the critic as a baseline, i.e. to reduce the variance of the updates. The value network is updated via standard TD updates, bootstrapping itself. Standard actor-critics are on-policy, that is they learn from interaction data generated by the exact current policy only, in contrast to DQN that learns from data from its replay buffer for instance.

Proximal policy optimization for actor-critics A specific instance of actor-critic we will use in further sections is the *Proximal Policy Optimization agent* (PPO) (Schulman et al., 2017). It introduces minimal but key modifications to the standard actor-critic presented in the previous paragraph. A first modification is the use of Generalized Advantage Estimation (GAE) (Schulman et al., 2016) to control the bias-variance trade-off in advantage estimation. GAE replaces standard advantage estimators, like the Monte-Carlo estimator $\hat{A}_{MC}(s_t, a_t) = G_t - \hat{V}(s_t)$ or the one-step TD estimator $\hat{A}_{TD}(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$, by the following estimator:

$$\hat{A}_{GAE,\lambda}(s_t, a_t, \dots, s_{T-1}, a_{T-1}, s_T) = \sum_{i=t}^{T-1} (\gamma\lambda)^i \hat{A}_{TD}(s_i, a_i, s_{i+1}),$$

where λ controls the bias-variance trade-off. Notably, $\hat{A}_{GAE,0} = \hat{A}_{TD}$ (high bias, low variance) and $\hat{A}_{GAE,1} = \hat{A}_{MC}$ (no bias, high variance). A second modification is the use of a clipped policy gradient loss based on importance sampling. With the ratio $r_{t,\theta} = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta,old}}$, we have:

$$\mathcal{L}_{PG,\lambda,\epsilon}(s_t, a_t, \dots, s_{T-1}, a_{T-1}, s_T) = \nabla_\theta \min\Big(r_{t,\theta}\hat{A}_{GAE,\lambda,t},\ \text{clip}\big(r_{\theta,t}, 1-\epsilon, 1+\epsilon\big)\hat{A}_{GAE,\lambda,t}\Big),$$

where $\hat{A}_{GAE,\lambda,t} = \hat{A}_{GAE,\lambda}(s_t, a_t, \dots, s_{T-1}, a_{T-1}, s_T)$ and $\text{clip}(x, a, b) = \min(\max(x, a), b)$. Using an importance ratio in place of the usual log-policy encourages departing from the previous policy, and was first proposed by Kakade and Langford (2002). To avoid going too far off in a single update, it is necessary to constrain these, which is obtained here by clipping the objective.

Importance weighted actor-learner Another variant of actor-critic we use later on is the *Importance Weighted actor-critic* (IMPALA) (Espeholt et al., 2018). It introduces a major difference with previous agents: instead of using a single actor or several actors in parallel synchronized with the learning, the experiences generated by the actors are stored in a queue (following the FIFO order, i.e. first in first out), which means that the speed at which actors are able to generate interaction data is greatly increased due to not having to wait for the learning procedure to catch up. Compared to the single-actor setting used in many RL approaches, it allows to increase the speed of the training procedure up to a certain point (i.e. the maximal speed at which the learning procedure consumes interaction data). This is called the *actor-learner framework*. A consequence is that it introduces off-policyness: the learning procedure consumes data generated by an older version of its current policy, whose age depends on the speed at which data from the queue is consumed. Another important difference with e.g. PPO is the use of *V-trace targets* in policy gradients and value updates. V-trace defines n-step value targets as:

$$V_{target,t} = \hat{V}(s_t) + \sum_{t'=t}^{t+n} \gamma^{t'-t}\big(\Pi_{j=t}^{t'-1} c_j\big)\rho_{t'}\delta_{t'},$$

where $c_t = \min(\bar{c}, \frac{\pi(a_t|s_t)}{\pi_{\theta^-}(a_t|s_t)})$ (noting the previous version of the policy π_{θ^-}) and $\rho_t = \min(\bar{\rho}, \frac{\pi(a_t|s_t)}{\pi_{\theta^-}(a_t|s_t)})$ are truncated importance sampling weights, whose role is to add a correction for the off-policyness. We also recall that δ_t is the one-step TD error at timestep t. V-trace targets act as drop-in replacements for Monte-Carlo or TD targets. They are particularly useful to handle the variance of off-policy updates in an adaptive way: when π and π_{θ^-} are identical, we recover n-step TD updates, when π and π_{θ^-} are different, importance ratios reduce the influence of the farthest steps.

Environments In this work, and depending on what we want to study experimentally, we use distinct environments from the literature. When called for it, we also use our own synthetic tasks, that we will

introduce in the corresponding sections in further text. A standard problem is the *Cartpole* problem (Sutton and Barto, 2018), in which one has to balance a pole by moving a cart among two directions available. This environment emulates the standard single pendulum problem in robotics, where a robot arm with a rotational actuator has to balance the pole rotated by the actuator. It is a simple problem that is useful as a sanity check for discrete control RL algorithms. We use the open-source implementation from OpenAI Gym (Brockman et al., 2016a), in which episodes are capped at 200 timesteps. As a more complex suite of RL environments, we often consider the *Arcade Learning Environment* (Bellemare et al., 2013), which consists of 60 Atari 2600 video games. In each game, the goal of the agent is to maximize the in-game score, with controls that are identical to those of a human player. Atari games are quite diverse and have proven to be a good benchmark for the versatility and stability of RL agents. We use the official implementation from Bellemare et al. (2013) and settings from Machado et al. (2018), including the use of sticky actions (i.e. each action from the agent is repeated the next step with a probability of 0.25).

We now discuss specific related work for the two subjects we deal with in this thesis: credit assignment and explainability.

3.3 Related work in credit assignment for RL

In this section, we give a broad overview of existing methods to tackle the credit assignment problem in RL. We warn the reader that we present some of the algorithms in a longer format in Chapter 4, and indicate so in the concerned paragraphs.

Redistributing rewards to the past A major family of credit assignment approaches can be viewed as redistributing rewards to past actions, either bypassing or complementing TD updates. We review many algorithms of this type in details in Chapter 4.

Policies/values conditioned on the future Another important family of credit assignment approaches conditions policy or value functions on a desired outcome.

Upside-Down RL (Srivastava et al., 2019; Schmidhuber, 2019) is a paradigm in which an estimator is trained to predict the action to be performed so that a given return is reached under a given number of time-steps, and can be used to infer an optimal policy. The estimator (called a *behavior function*) is trained via maximum likelihood estimation, using observed trajectories as training data. The action probability assigned for an outcome $z = (G_{\gamma,0}, H)$ is simply $\pi_\theta(\cdot|s, G_{\gamma,0}, H)$, where π_θ is the behavior function learned, conditioned on both a return $G_{\gamma,0}$ and a time horizon H. Though not a direct measure of the influence of an action over the defined outcome, we argue that policies that are conditioned on a future outcome are a form of credit assignment, since actions that are crucial to reach the outcome should be assigned high probabilities by near-optimal policies.

Similarly, Kumar et al. (2019) condition a policy network on a future return to be reached and learn the policy by maximizing the likelihood of observed data. They also allow advantage conditioning, which consists in conditioning on the difference between the future return and an estimated value of the initial state. Trajectories are generated by sampling a target outcome from a target distribution for the whole episode. In contrast to the previous method, they do not condition the policy on a specific horizon. The action probability assigned for an outcome z (be it a return or an advantage) is simply $\pi_\theta(\cdot|s, z)$, where π_θ is the reward-conditioned policy learned.

Dosovitskiy and Koltun (2016) propose a fully supervised approach to control tasks, based on the related idea that reliable predictions of the future are sufficient to learn how to act. In their setting, the predictor is given an observation, a goal and current measurements as input. It has to predict the future measurements. Measurements are continuous or categorical state variables, and are generally high-level indicators of the agent situation; goals specify which linear combination the agent should optimize. The

predicted measurements are calculated as the sum of the predicted future measurement for the policy and an action-conditional residual. The credit assigned can thus conveniently be expressed as the scalar product of the goal and residual, which roughly measures the advantage of taking action a given the goal g as outcome. A drawback of this approach is that it requires privileged information about the task (i.e. measurements).

We review other algorithms of this type in details in Chapter 4.

Generating trajectories Another approach for implicit credit assignment is data augmentation through backward generative modelling. Given a reliable inverse model (Jordan and Rumelhart, 1992) and an outcome (say a state) to start the generative process from, one can produce imaginary trajectories reaching the outcome.

The advantage of such algorithms is that they have the potential to reduce the sample-complexity of RL algorithms, by producing additional curated data. Generated rollouts can either be used as interaction data in the RL learning loop, or regarded as expert supervision and used for imitation learning. The drawback of generative algorithms is that inverse modelling is a hard and ongoing research topic: errors inevitably compound as the generative process unrolls (Talvitie, 2014; Jiang et al., 2015). As a result, the trajectories produced by the generator can come far-off the manifold of possible trajectories and provide misleading information to RL algorithms consuming the synthetic data (Abbeel et al., 2006). These algorithms are implicit in the sense that they do not measure the influence of actions, only their presence in imagined trajectories.

Goyal et al. (2019) propose an inverse model that estimates the probability distribution of the immediate previous state and action given a resulting state. To reduce the variance, they model the density of state residuals instead of actual states. The estimated probability distribution can then be used to generate new plausible trajectories (that they call *recall traces*) from experienced high-value or high-reward states. The inverse model is trained online via maximum likelihood.

Edwards et al. (2018) also propose an inverse dynamics model that learns to predict the previous state s_t based on a state-action pair (s_{t+1}, a_t): $\tilde{s}_t = s_{t+1} - \Delta\tilde{s}_t$, with $\Delta\tilde{s}_t \sim q_\Delta(\cdot|s_{t+1}, a_t)$. Similarly to Goyal et al. (2019), the inverse model operates on state residuals and is learned via maximum likelihood estimation. Within the data generation process, a starting state is sampled from a goal state distribution, previous actions are either sampled at random or using a directed scheme; and state residuals are sampled from the inverse model to construct new transitions backward.

Relabeling transitions Relabeling algorithms are pseudo-generative algorithms: they modify attributes of existing trajectories to create alternate transitions or trajectories to be consumed by RL algorithms. Relabeling usually happens in hindsight, that is after an often imperfect outcome is reached, and turns suboptimal data into optimal data for that suboptimal outcome. While most algorithms we present were designed for multi-goal RL (Schaul et al., 2015a), we think relabeling has applications outside this particular setting, which is in fact corroborated by the work of Buesing et al. (2018).

In a goal-conditional setting, Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) modifies experienced trajectories in the replay buffer, changing the original goal for the goal that was achieved instead, and attributing a reward in the end-of-trajectory transition. Doing so, the agent benefits from "free" supervision about actions to be performed to reach suboptimal outcomes, which in turn make its policy more generalizable to new goals.

In the goal-conditional setting still, Hindsight Policy Gradients (HPG) (Rauber et al., 2017) use relabeling to modify the policy gradient update, providing additional learning signal for goals that were encountered in trajectories, also using importance corrections to account for the difference in behavior when conditioning on other goals.

Buesing et al. (2018) propose a model-based algorithm that generalizes Guided Policy Search (Levine and Koltun, 2013) but explicitly considers alternative outcomes from counterfactual actions, that is other

actions performed in place of the actions observed, in experienced trajectories. Their method requires knowledge of the structural causal model of the environment (Pearl, 2009), which then allows to sample under the counterfactual distribution induced by the interventions (performing alternative actions).

Graph-based methods Proto-value functions (Mahadevan, 2005; Mahadevan and Maggioni, 2007) collectively span the space of all possible value functions in a given state space:

$$V^\pi = \alpha_1 V_1^G + \ldots + \alpha_n V_n^G$$

They are learned in a coordinate-free approach: assume the state space is represented as a known undirected graph $G = (S, E)$. The graph Laplacian L is $D - A$ where A is the adjacency matrix of the graph and D is a diagonal matrix whose diagonal values are row sums of A. An eigenfunction f of the Laplacian verifies $Lf = \lambda f$. If the graph G is known, the Laplacian can be calculated, and the eigenfunctions as well via spectral analysis. We note $\phi_G = \{V_1^G, \ldots, V_k^G\}$ these eigenfunctions ($V_i^G = (V_i^G(1), \ldots, V_i^G(|S|))$), which can express any function $f : \mathcal{S} \to \mathbb{R}$ represented as a vector of its evaluations on the graph vertices. The idea of proto-value functions can be leveraged to learn an approximate optimal value through Representation Policy Iteration (RPI) (Mahadevan, 2005), which repeats the following steps: it uses the eigenfunctions of the Laplacian as basis functions in LSTDQ (Lagoudakis and Parr, 2003) to estimate the Q-values (policy evaluation), computes the greedy function (policy improvement) and updates the graph representation from new interactions sampled. A drawback of this approach is that it is costly and cannot realistically scale to high-dimensional problems. Indeed, the graph Laplacian L lies in $\mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$, and is unsuitable to large state spaces.

Klissarov and Precup (2018) propose a more practical approach to using Proto-Value functions: they approximate the graph Laplacian of the MDP with a Graph Convolutional Network (Kipf and Welling, 2016; Defferrard et al., 2016) and leverage it to compute an approximate value function that allows for efficient credit assignment.

Prioritizing important data Prioritization schemes propose alternatives to the often-used uniform sampling in the replay buffer for off-policy learning. While not technically credit assignment algorithms, these algorithms assign a priority to actions, since they propose sampling distributions that weigh transitions according to the current prediction error, which could correlate with the actual importance of actions. Prioritized sweeping (Moore and Atkeson, 1993) was the first of this kind, defining a state priority based on the magnitude of the TD error and only sampling the highest priority state instead of uniformly, with application for Dynamic Programming. *Prioritized experience replay* (PER) (Schaul et al., 2015b) uses the same priority measure but a different sampling strategy: each transition is sampled from the replay buffer with a probability that is proportional to the latest TD error observed for that transition. Then, Horgan et al. (2018) and Kapturowski et al. (2018) respectively introduced a distributed and a recurrent version of PER. Self-imitation learning (Oh et al., 2018), in contrast to PER, prioritizes transitions proportionally to the difference between the return from the starting state (as observed in the trajectory the transition is a part of) and its estimated value. Lee et al. (2018) take a different approach and propose to update value functions starting from end-of-trajectory states to propagate the experienced rewards to earlier actions faster, mimicking the propagation mechanism of Dynamic Programming.

By revisiting important transitions or trajectories more often (or in the right order), the propagation of the reward signal from the future is facilitated, which alleviates the temporal credit assignment problem.

Meta-learning rewards Zou et al. (2019) propose to learn rewards based on *meta-learning* algorithms, for their fast adaptation properties. They use MAML (Finn et al., 2017a) to learn a meta Q-function over a distribution of environments, and then use the associate value function as a prior potential function in novel environments. They show that their method allows for fast adaptation to previously unseen

continuous control tasks. Here, the use case is quite different because they optimize credit assignment for generalization and transfer. Note that a requirement for their method is that all environments have the same state space.

Hierarchical RL Hierarchical RL (HRL) (Sutton et al., 1999; Dietterich, 2000; Kulkarni et al., 2016; Kipf et al., 2018; Nachum et al., 2018) makes use of a hierarchy of components to exploit the natural decomposability of tasks. They present promising aspects for credit assignment: segmenting the task into sub-tasks reduces the delay between actions and feedback. Additionally, some algorithms decentralizes the credit assignment problem thanks to a manager-worker decomposition (Dayan and Hinton, 1993; Vezhnevets et al., 2017). Indeed, the local manager encourages pertinent behavior from its worker(s) by rewarding them when getting closer to its own goal. In practice though, to the best of our knowledge, HRL is ongoing research and hard to scale beyond two or three levels of hierarchy (Levy et al., 2019).

Model-based planning Planning methods (Kocsis and Szepesvári, 2006; Coulom, 2006), either from a given model (Silver et al., 2016, 2017a) or a model learned from experience (Tamar et al., 2016; Vezhnevets et al., 2016; Silver et al., 2017b; Oh et al., 2017; Hansen et al., 2018; Savinov et al., 2018; Guez et al., 2018, 2019; Eysenbach et al., 2019; Schrittwieser et al., 2019) have an interesting connection to credit assignment. As the best immediate action is evaluated by the sampling value, obtained by sampling trajectories from the model and propagating the value of descendent nodes to the current node in the search tree; search evaluates the influence of the first action towards an outcome that is not known at the start of the procedure and corresponds to the best achievable expected return (as estimated through search). As for some algorithms that condition on the future, the search procedure only decides the first action on the basis that it makes reaching the evaluated best outcome possible, and likewise it is an implicit form of credit assignment.

Specifically, Bakker (2007) proposes a model-based approach to credit assignment. First, a model of the environment and a value function are learned from episodes sampled by acting randomly. In a second time, a policy is learned from imagined episodes. The learning signal for the policy is obtained by backpropagation through the critic: an arbitrary high value is used as a reference value, and the gradient of a mean squared error loss between the value estimated by the critic and the fixed reference value flows back to the policy, whose parameters are updated by gradient ascent. A drawback of this approach is that the model is learned and fixed once and for all, while it could benefit from interactions sampled from improved versions of the starting policy. It also appears likely that it requires a very accurate model to function properly.

Multi-agent credit assignment There is a whole literature on multi-agent credit assignment, which applies to a specific credit assignment problem in multi-agent RL: when rewards are shared among all agents, how to properly attribute credit to the specific actions of specific agents? We refer the interested reader to the related work sections of Son et al. (2019) and other works.

3.4 Related work in explainable RL

There are several, distinct subareas of research in explainable RL. We briefly review the main ones here.

Distilling into interpretable policies Due to the interpretability/expressivity trade-off, fully interpretable models are usually not powerful enough to do well on complex RL tasks. To circumvent that limitation, a natural idea is to train a non-interpretable model (say, a neural network) and then distill it into its interpretable counterpart. Among such interpretable models to be explored, decision trees are popular for their complete interpretability and decent expressivity (Bastani et al., 2018; Liu et al.,

2018b; Coppens et al., 2019; Vasic et al., 2019). Finally, programs (as a series of instructions) are also used (Verma et al., 2019b,a). Since programs are not differentiable, projections are employed instead of distillation to go back-and-forth between the non-interpretable model and the inferred programs, obtained via program synthesis.

Looking at policy gradients Gradient-based methods are ubiquitous analysis tools in supervised learning, be it for tabular, visual or textual inputs. Are gradient-based methods useful in the RL context? Saliency maps were used in different flavors in several works (Wang et al., 2015; Zahavy et al., 2016; Greydanus et al., 2018; Puri et al., 2019). Some works (Wang et al., 2015; Zahavy et al., 2016) used policy gradients directly to compute saliency maps, but the results were noisy and not very informative. This is partly due to backpropagated gradients, whose components measure sensitivity to single pixel variations. Instead, more recent methods (Greydanus et al., 2018; Puri et al., 2019) use finite-differences approaches to approximate policy gradients. As a result, they have more control on the perturbation applied to inputs. For instance, they choose to apply localized Gaussian blurs in Atari games, and to remove pieces in Chess or Go games.

Assessing learned representations Instead of looking at the decisions of the agent, one can look at the state representations formed by the agent to make sense of its understanding of the task. Zahavy et al. (2016) use t-SNE on the state representations learned by the last layer of a DQN agent to form clusters of states based on embedding similarity. An issue with embedding clusters is that manual reviewing is generally necessary to get information out of the resulting organization. Topin and Veloso (2019) propose a different state aggregation method based on the agent policy. Their method clusters states based on similar action preferences and a sensitivity to similar state features, which is not scalable to visual inputs.

Inspecting attention/memory mechanisms Attention mechanisms hold value for interpretability, as originally studied in the NLP literature, and are used to derive interpretations (Hung et al., 2019; Ferret et al., 2020; Mott et al., 2019; Tang et al., 2020). The rationale is that the attention weight distribution should tend to peak on embeddings that hold predictive value for the model, which we correlate to importance. Memory retrieval can be analysed similarly to attention mechanisms (Wayne et al., 2018).

Recovering agent preferences Another option to gain interpretability is to study the preferences of RL agents. By preferences, we mean two types of things: 1) either a reward function that fits best the agent behavior (Huang et al., 2019a; Arora and Doshi, 2021), 2) a set of coefficients in the case of decomposed rewards (Juozapaitis et al., 2019; Bica et al., 2021). Inverse RL (IRL) (Ziebart et al., 2008), to some extent, is also relevant to mention here. IRL methods construct a reward function for which expert demonstrations (without the rewards) are optimal, in a sense explaining how agents reward their own behavior.

Leveraging natural language Several works have been building towards human-understandable, natural language explanations of agent policies and tasks. Such works make use of language templates, that is pre-established sentences that have parts that vary depending on the passed inputs (that are themselves the outputs of the algorithms we briefly present). Khan et al. (2009) propose contrastive templated explanations that compare the expected return between the optimal action and other actions. Wang et al. (2016b) propose natural language templated summaries of agent behavior in POMDPs, without justifying actions though. A drawback of these works is that they generally rely on domain knowledge to build language templates and explanations that are not always universally and easily understandable to every human user.

Extracting representative behavior Another focus of interpretability in RL is the extraction of representative examples of agent behavior. A first approach consists in extracting important states and assessing agent behavior in such states, using the action-gap for instance (Huang et al., 2018). Note that similar approaches were discovered for other purposes, such as having agents advising other agents (Torrey and Taylor, 2013). Another approach consists in selecting trajectories (or portions of trajectories) that are representative of the overall behavior of RL agents. One way of selecting trajectories is by using action-value differences (akin to the action-gap) to find important states (Amir and Amir, 2018), another is by using a measure of disagreement between distinct policies (Amitai and Amir, 2021), another is to rank sequences by likelihood according to policy (Sequeira and Gervasio, 2020). Such approaches can be useful to summarize agent behavior to a human user, and potentially establishing trust in the agent policy.

Part I

Action Importance and Credit Assignment

Chapter 4

A Unifying Perspective on Explicit Credit Assignment

We recall that in RL, credit assignment refers to the ability of mapping specific actions to specific outcomes under delay, distractors, noisy feedback and from incomplete data. While crucial to make RL efficient in complex and long-horizon tasks, explicit credit assignment lacks a unified perspective to build upon. In this chapter, we provide a general framework and show that it brings the existing literature together under a common abstraction. More specifically, we separate existing works based on three main aspects: how the actions are associated to outcomes, the nature of the outcomes considered (*e.g.* returns? individual rewards? future states?) and whether grounded on actual trajectories or not. Notably, the separation of these aspects makes it easier to see that credit assignment can be viewed as a form of action importance with respect to the outcomes considered. We leverage this framework and propose a taxonomy of explicit credit assignment methods, including existing methods and others that are yet to be studied theoretically and empirically.

4.1 Introduction

In sequential decision-making, credit assignment (CA) (Minsky, 1961; Sutton, 1984) is broadly viewed as the answer to the following question: *given an observed outcome, how much did previous actions contribute to its realization*? Outcomes are observed results of the interaction process. A useful outcome gives information about the performance of the agent's behavior; which includes returns, future states, individual rewards, among others. In general, this is a complex question for several reasons, among which: 1) the contribution of an action depends on subsequent behavior, which makes credit assignment policy-dependent, and 2) mapping actions to outcomes is generally hard due to a combination of delay, partial observability, and stochasticity from both the environment and the decision-making process. In general, Reinforcement Learning (RL) evades the question by focusing on learning accurate enough value functions, which perform an implicit form of credit

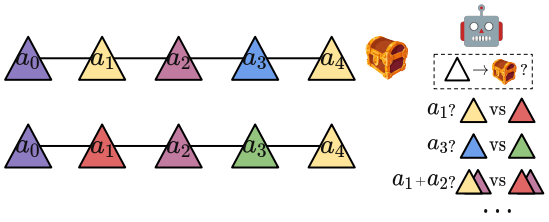
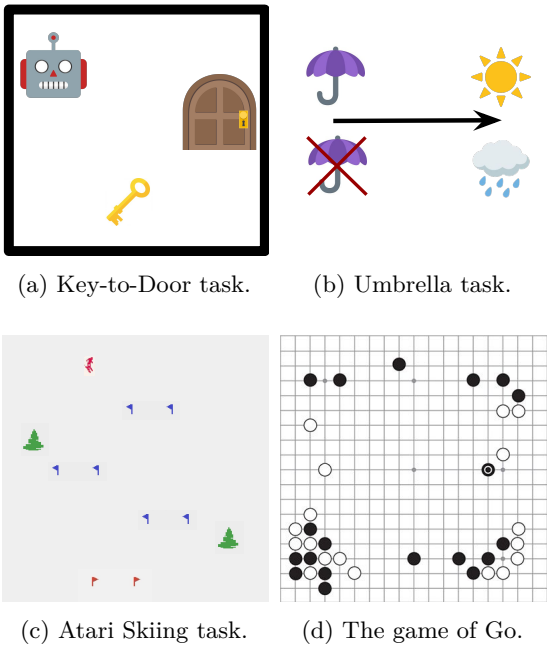


Figure 4.1: A depiction of the credit assignment problem in RL: agents have to map observed outcomes (*treasure*) to actions (*triangles*) under delay, noisy feedback and from incomplete data.

assignment, by evaluating the total contribution of the current and future actions. Indeed, value functions seek the answer to a different question: *given a state (resp. a state-action pair), what is the expected return?* Values are then used to guide policy learning or action selection. Instead, *explicit* credit assignment methods map actions to outcomes in an explicit way: they determine a quantitative notion of the influence of actions over a specific future outcome, and use it to reinforce actions accordingly.

While there is evidence that using values as the only source of credit assignment is enough to solve some complex tasks on its own (Paine et al., 2019; Vinyals et al., 2019; Badia et al., 2020b), recent RL algorithms featuring explicit credit assignment were shown to dramatically improve the sample-efficiency of agents in partially observable, delayed and/or stochastic environments (Hung et al., 2019; Raposo et al., 2021). Among these, hard credit assignment tasks stand out: they feature a small subset of actions or states that determine to a large extent the optimality of behaviors, and generally cause trouble to standard RL algorithms. This is mostly because RL algorithms use various flavors of temporal differences (TD) (Sutton, 1988), which assign credit implicitly based on temporal proximity. In such tasks, acquiring and exploiting knowledge about the relations between actions and desired outcomes can turn difficult RL problems into simple ones (Ferret et al., 2020). We exemplify this in a synthetic task where an agent must open a door with a key picked up along the way (Hung et al., 2019) (see Fig. 4.2a). Eliminating the delay between the action (*i.e.* picking up the key) and the outcome (*i.e.* opening the door) by rewarding the action directly makes the learning problem a lot easier. Aside from the efficiency gains, explicit credit assignment is useful for interpretability: it can help making sense of the behaviors of RL agents. Indeed, complex tasks can require thousands of decisions, hence analysis of the learned behaviors should inspect critical decisions in priority. Accordingly, explicit credit assignment gives us a notion of the importance of each action with respect to an observed outcome, and can be used to identify such decisions.



(a) Key-to-Door task. (b) Umbrella task.

(c) Atari Skiing task. (d) The game of Go.

Figure 4.2: Examples of tasks where accurate credit assignment is crucial.

Nonetheless, despite the apparent promises, the role of credit assignment in RL is undermined by the lack of common ground to build upon. In this work, we aim to propose a general framework that encompasses most existing explicit credit assignment methods. Our motivation is to help reveal key similarities and differences among the existing work, uncover yet-to-be-studied algorithms, and make first steps towards a theory of credit assignment. In the direction of the latter, there is a missing entity in the current state of the credit assignment literature in RL: an oracle method, *i.e.* a quantity that supposes perfect knowledge about the environment and that can act as a point of reference when assessing the relevance of credit assignments. Since we want to identify *actions that cause outcomes* instead of actions that only correlate with outcomes, we advocate for a causally correct oracle.

Our contributions are the following: 1) we provide a general framework for implicit and explicit credit assignment in RL, 2) we propose a causal credit assignment oracle based on counterfactual simulation, 3) we identify and characterize several important axes of variation among practical credit assignment methods; and we use these to establish a taxonomy of the existing credit assignment works. We hope that our work can serve as an entry point for general RL practitioners into credit assignment.

The chapter is organized as such. In Chapter 4.2 we introduce the notations and concepts that we will build upon, including a short overview of existing explicit credit assignment methods (see Chapter 4.2.1). In Chapter 4.3 we present the proposed framework for implicit and explicit credit assignment, which includes a common formalism, criteria for success and an oracle method to establish ground truth credit assignment. In Chapter 4.3.3, we leverage the framework to distinguish several admissible types of approximate credit assignment methods, discuss their relative merits and characteristics, and provide a taxonomy of the existing work.

4.2 Additional background and notations

Here we introduce the concepts and notations that will be of use in further sections. We place ourselves in a finite horizon, discrete control setting, with potentially stochastic rewards.

Reinforcement Learning. A trajectory $\mathcal{T}$ is the random variable $\mathcal{T} = \mathcal{H}_T$, where T is the fixed horizon of the task. In the following sections, τ refers to realizations of $\mathcal{T}$, being sampled from the behavior policy ($\mathcal{T} \sim \pi\mathcal{P}$) unless mentioned otherwise. An outcome Z is a random variable that corresponds to an arbitrary function of a trajectory: $Z = f(\mathcal{T})$. Among notable types of outcomes, the return is the discounted sum of rewards to go: $G_{t,\gamma} = \sum_{t'=t}^{T} \gamma^{t'-t} R(S_{t'}, A_{t'})$. We note $p_\pi(X = x)$ the probability of X being equal to x under the behavior policy π (X being a discrete random variable) and $d_\pi(X)$ the density of X under π (X being a continuous random variable).

4.2.1 Explicit credit assignment methods

Existing explicit credit assignment methods determine the credit of actions in various ways. We now dive into the details of some of these methods, starting with general characteristics. We provide with accompanying figures to summarize the main workings of each method in Fig. 4.3. We give expressions for the credit assignments of a larger set of methods under the proposed framework in Table 4.3, p. 52.

Common aspects. An aspect shared by most explicit credit assignment methods is that interaction data is observed, usually full episodes so that outcomes are available. Then, a predictor is trained to solve a predictive, supervised task; either at the level of the full trajectory or at the level of individual transitions. Intermediate outputs of the predictor (e.g. predicted probabilities, attention, memory reads...) are used to assign approximate credit for observed outcomes to actions. Such credit is then incorporated into the RL learning procedure, either by modifying the reward function, modifying the RL updates or guiding action selection. The principal effect of explicit credit assignment is the mitigation of delays between actions and outcomes. A consequence is improved sample-efficiency: by crediting early actions for their share of the future directly, the propagation of the reward signal is accelerated. Another important effect is variance reduction: by incorporating information about the future, the variance of the various estimations an RL agent can perform (*e.g.* estimating the value of its policy) might decrease.

TVT. Temporal Value Transport (TVT) (Hung et al., 2019) relies on an RL agent with an external memory. Memory vectors are learned representations of the successive states of the environment, which are learned via the autoencoding task of reconstructing the action, immediate reward and next state. When creating a new memory vector, each past memory is assigned a weight that serves a dual purpose: summarizing the past as a weighted average of past memories, and proportionally crediting previous state-action couples. The credit is used in a modified TD scheme, where state-action couples that are retrieved from memory with large weights get an additional term in the bootstrapping target: given the

fixed coefficient η, for a future timestep t' and a previous timestep t with a large weight, the one-step TD loss at timestep t becomes: $\mathcal{L}_{TD,TVT}(\theta) = (r(s_t, a_t) + \gamma V_\theta(s_{t+1}) + \eta V_\theta(s_{t'}) - V_\theta(s_t))^2$.

SECRET. Self-Attentional Credit Assignment (SECRET) (Ferret et al., 2020) uses a Transformer network (Vaswani et al., 2017) to predict the value of individual rewards from previous observations and actions, and view self-attention weights as the explicit credit of previous actions for these rewards. The credit assigned for positive rewards is then used to design auxiliary rewards, using potential-based reward shaping (Ng et al., 1999): $r_\theta(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \gamma\phi_\theta(s_{t+1}) - \phi_\theta(s_t)$, where $\phi_\theta(s)$ is the quantity $(1/|\mathcal{D}|)\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T}\mathbb{1}\{s_t = s\}\sum_{t'=t}^{T}\alpha_\theta(r_{t'} \to t)r_{t'}$ and α_θ is the learned attention function. This is an offline method, *i.e.* the auxiliary reward function is computed ahead of the RL procedure from a dataset of interaction data $\mathcal{D}$, such that the RL agent then experiences rewards from a fixed reward function.

HCA. Hindsight Credit Assignment (HCA) (Harutyunyan et al., 2019) makes use of a neural network to predict observed actions given a state and the value of a future outcome (either a future state or a future return). When the chosen outcome is a future state s', the outcome-conditioned classifier $h_{\theta,\beta}(a|s, s')$ approximates $(1/T)\sum_{t=0}^{T-1}\sum_{k=0}^{T-t}\beta^{k-1}(1-\beta)p_\pi(A_t = a \,|\, S_t = s, S_{t+k} = s')$, the second summation coming from viewing k as a random variable with a geometric distribution. $h_{\theta,\beta}$ is learned online, from episodes sampled from the behavior policy. The explicit credit for a given action is formulated as the ratio $c_\pi(s, a \,|\, s') = h_{\theta,\beta}(a|s, s')/\pi(a|s)$, in accordance with the intuition that actions that are much more probable given a future outcome are direct contributors. The credit is used in a modified policy gradient method, which updates all actions at once: $g_{\theta,HCA} = \sum_{a\in\mathcal{A}}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a|s_t)\psi_{HCA}(s_t, a)$, where $\psi_{HCA} = r_\theta(s_t, a) + \sum_{t'=t}^{T}\gamma^{t'-t}\big(h_{\theta,\beta}(a|s_t, s_{t'})/\pi_\theta(a|s_t)\big)r(s_{t'}, a_{t'})$.

CCA. Counterfactual Credit Assignment (CCA) (Mesnard et al., 2021) leverages a backward recurrent neural network to compute a representation that summarizes future observations and rewards $\phi_{\theta,t}$, for each timestep of the episode. It is optimized to make the estimated value $V_\theta(s_t, \phi_{\theta,t})$ (with hindsight knowledge) most accurate, while maximizing the independence between the action distribution from the behavior policy and that from an outcome-conditioned classifier h_θ (i.e. minimizing the Kullback-Leibler divergence $\mathrm{KL}(\pi_\theta(\cdot|s_t) \,||\, h_\theta(\cdot|s_t, \phi_{\theta,t}))$. h_θ is the same estimator as in HCA. The independence maximization objective serves the purpose of reducing the information available in ϕ about the actual action taken by the agent. As a consequence, ϕ has incentive to only keep information about the future that is not a consequence of the action. The estimated advantage used in the policy gradient features the hindsight value function: $A_\theta(s_t, a_t) = G_{\gamma,t} - V_\theta(s_t, \phi_{\theta,t})$. By subtracting what is not a direct consequence of the action, it estimates the part of the observed return that the action individually contributed to.

RUDDER. Return Decomposition for Delayed Rewards (RUDDER) (Arjona-Medina et al., 2019) employs a recurrent neural network as a return predictor. A discounted return $g_\theta(h_t)$ is predicted at each timestep of the episode from the observed history h_t, and the difference between consecutive predictions is viewed as the assigned credit $c_\theta(h_t) = g_\theta(h_t) - g_\theta(h_{t-1})$, with the rationale that a sharp increase in the expected return correlates with important actions. The credit is used to compute a modified advantage $\hat{A}_\theta(h_t, a_t) = (1-\beta)A_\theta(s_t, a_t) + \beta c_\theta(h_t)$, which replaces the standard advantage estimator in the policy gradient updates.

SR. Synthetic Returns (SR) (Raposo et al., 2021) associate future states to past states by a gated linear regression of the future individual rewards from the observed history h_t: $r_\theta(h_{t+1}) = b_\theta(s_{t+1}) + g_\theta(s_{t+1})\sum_{t'=0}^{t}c_\theta(s_{t'}, a_{t'})$. The coefficients b_θ, g_θ and c_θ are actually the outputs of standalone neural networks, and the latter is used as assigned credit in a modified reward function $\hat{r}_\theta(s_t, a_t) = \alpha r(s_t, a_t) + \beta c_\theta(s_t, a_t)$.

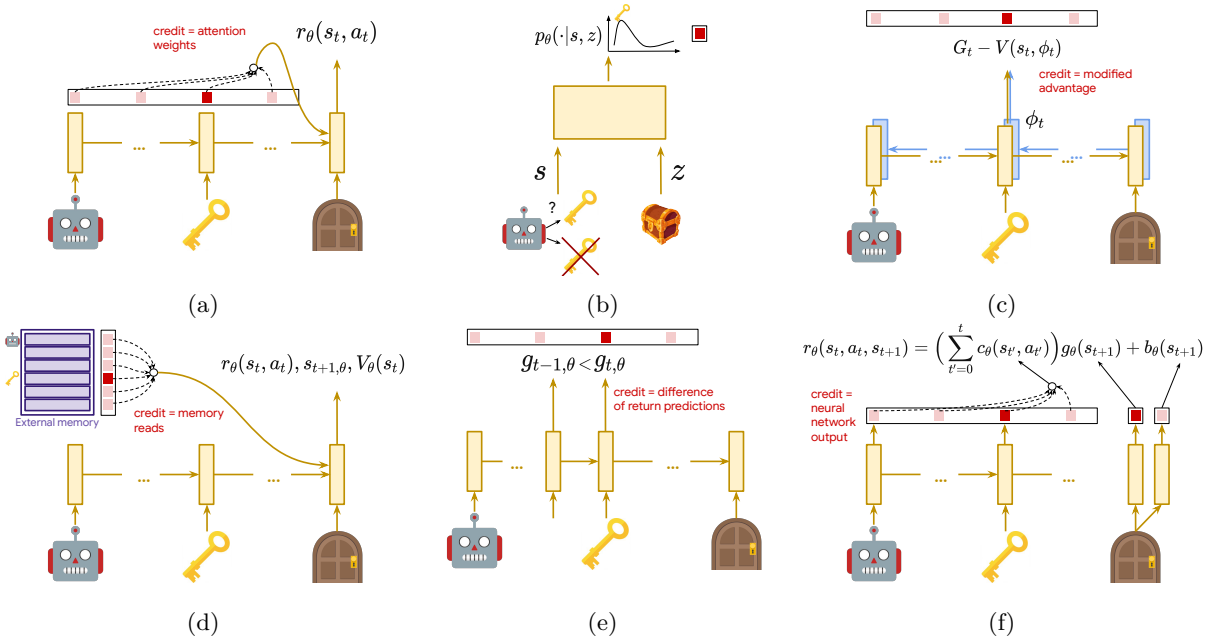


Figure 4.3: High-level depiction of existing credit assignment methods. **a)** SECRET. **b)** HCA. **c)** CCA. **d)** TVT. **e)** RUDDER. **f)** SR.

4.3 A unifying framework for credit assignment in RL

In this section, we propose a unifying formalism for explicit credit assignment in RL. For clarity, we focus on credit assignments to state-action pairs, though definitions can be adapted to states only or to sets of state-action pairs instead. Also, we view credit as a positive quantity, but definitions could be extended to signed quantities.

4.3.1 General credit assignment functions

First, we introduce a very general mathematical object for explicit credit assignment: the credit assignment function. Credit assignment can take different forms, but a common aspect is that credit assignment functions quantify the contribution of actions with respect to a chosen type of outcome, that is either observed from an actual trajectory, predicted, or given as input. We formalize these elements with the next definitions.

Definition 1. *Let z be an outcome value. We define a* ***context-free*** *credit assignment function as a function $c_\pi(s, a \,|\, z) \in \mathbb{R}$.*

Context-free credit assignment functions give the assigned credit for a specific state-action pair and a specific outcome. In their most generic form, such functions do not condition their output on an observed trajectory. As a result, they are defined over the whole state-action space and over the whole outcome space.

Definition 2. *Let τ be a trajectory, and $t \in [0, T]$ a specific timestep. We define a* ***grounded*** *credit assignment function as a function $c_\pi(t \,|\, \tau) \in \mathbb{R}$.*

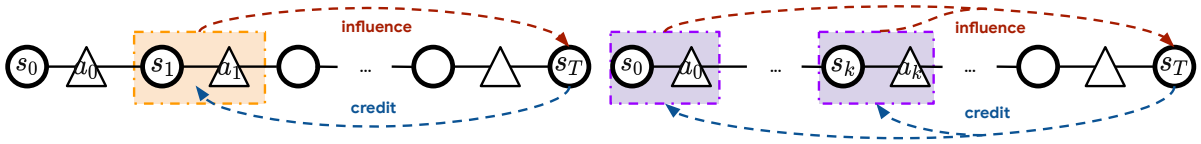


Figure 4.4: **Left:** single outcome to single action credit assignment (*single-to-single*). **Right:** single outcome to many actions credit assignment (*single-to-many*).

Grounded credit assignment functions take as input a trajectory. They give the assigned credit for the state-action pair at the selected timestep in the trajectory and a specific outcome $z = f(\tau)$. Such functions still depend on a behavior policy, since the same trajectory can be observed under much different policies, and distinct credit be assigned.

Definition 3. *We define* ***explicit*** *credit assignment functions as the union of context-free and grounded credit assignment functions.*

Both context-free and grounded credit assignment functions are explicit, in the sense that they quantitatively tie an outcome to a state-action pair. A crucial difference is that grounded functions benefit from the context of the trajectory, which makes it possible to credit actions of the trajectory in a relative way.

Definition 4. *We define an* ***implicit*** *credit assignment function as a function* $c_\pi(s, a) \in \mathbb{R}$.

In contrast to explicit functions, implicit functions do not condition their output on the value of an outcome. Action-value functions are notable implicit functions. This is in line with the fact that value functions might hold information about the contribution of actions to future returns (*i.e.*, important actions might have a large action-value compared to other actions), but are not conditioned on any observed outcome and do not take into account any context other than the current state (*i.e.*, a trajectory).

Definition 5. *We define a* ***proper*** *credit assignment function as a grounded credit assignment function that verifies the following properties:*

- $c_\pi(t \,|\, \tau) \in [0, 1]$ *for all* $t \in [0, T]$ *and all* τ,
- $\sum_{t \in [0,T]} c_\pi(t \,|\, \tau) = 1$ *for all* τ.

Proper functions have an additive property: the credit assigned to individual actions along a trajectory always adds up to 1, meaning that it can be viewed as the fraction of the outcome the action is responsible for.

We now define assignments. An assignment is a set containing the credit assigned individually to the consecutive state-action pairs of a single trajectory, by the specified credit assignment function.

Definition 6. *Given a proper credit assignment function* c_π *and a trajectory* τ*, we define an* ***assignment*** *as the set* $C_\pi(\tau) = \{c_\pi(t \,|\, \tau)\}_{t \in [0,T]}$. *By definition we have* $\sum_{c \in C_\pi(\tau)} c = 1$.

Definition 7. *Given an assignment* $C_\pi(\tau)$*, we consider the following set* $C_{\pi,+}(\tau) = \{c \in C_\pi(\tau) \text{ such that } c > 0\}$. *We call* ***single-to-single assignment*** *an assignment such that* $|C_{\pi,+}(\tau)| = 1$. *We call* ***single-to-many assignment*** *an assignment such that* $|C_{\pi,+}(\tau)| > 1$.

We represent the two types of assignment in Fig. 4.4.

4.3.2 Criteria for principled credit assignment functions

Having defined general forms for credit assignment functions, we now list desiderata for principled credit assignment functions, which stem from the following assertion: the credit for a given outcome should be attributed to actions that matter most to its realization. Informally, influence is akin to causation: *i.e.* maximally influential actions are those that, when not performed, prevent from reaching the chosen outcome. We give several candidate notions of influence along those lines in Appendix B.5. We mention several similarity functions in what comes next, without giving precise expressions. This is for the sake of generality: we think that different similarity functions will suit different contexts. Still, we provide examples for such similarity functions in Appendix B.2.1. Ideally, the following principles should hold in tasks where credit assignment is difficult, either due to increased delay, partial observability, or sources of stochasticity (*i.e.* the policy or the environment dynamics). We qualitatively assess existing methods against the proposed criteria in Table 4.2, p. 52.

1) Sensitivity

- **(a)** Actions that have a large influence on the specified outcome should be assigned high credit.
- **(b)** Actions that have a small influence on the specified outcome should be assigned limited credit.
- **(c)** Actions that have no influence on the specified outcome should be assigned a credit of 0.

A corollary of these is that credit assignment should generally not only be determined by the temporal proximity between actions and outcomes. A corollary of c) is that actions that happen after the outcome is observed should be assigned a credit of 0.

2) Relativity Given two actions, the credit assigned for any action should be larger if the action has more influence on the outcome than the other one. A corollary is that assignment values should have a ranking that matches the ranking of actions in terms of ground truth influence on the selected outcome.

3) Separability Actions that have distinct influence on the specified outcome should have a difference in assigned credit that is as large as possible while satisfying other desiderata.

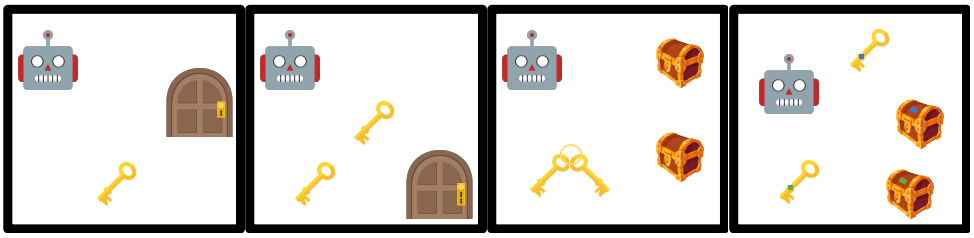
4) Invariance Actions that lead to similar immediate state distributions under the behavior policy should be assigned similar credit. The rationale is that such actions have similar long term effects due to the Markov property.

5) Stability

- **(a)** Given identical outcomes, the assignments for similar trajectories should be close.
- **(b)** Given identical trajectories, the assignments for similar outcomes should be close.

6) Generalizability Given identical trajectories and outcomes, the assignments for similar policies should be close.

7) Expressivity There are several possible cases for credit assignment in RL tasks: tasks with a single outcome of interest either leads to single-to-single or single-to-many assignments; but there might be several outcomes of interest, either many outcomes to single action (*many-to-single*), or many outcomes to many actions (*many-to-many*). We illustrate these cases in Fig. 4.5. In a nutshell, we would like to have credit assignment functions that can deal with any of these tasks.



(a) (b) (c) (d)

Figure 4.5: The types of outcome-action relations that can be featured in RL tasks. **a)** Single-to-single. **b)** Single-to-many. **c)** Many-to-single. **d)** Many-to-many.

8) Scalability We want credit assignment functions to have the potential to extend to multi-step assignments. Multi-step assignments, similarly to options (Sutton et al., 1999), operate on sequences of actions instead of single actions, and might be needed in order to infer useful credit assignment in finer-grained, longer horizon tasks. We elaborate on this aspect in Appendix B.1.

4.3.3 Approximate mappings of outcomes to actions

The credit assignment oracle we introduced in the previous section is often not realistic to compute for practical purposes: it requires a simulator, the ability to reset it to arbitrary states, and access to the stochastic factors of both the environment and the current policy. For a practical use of explicit credit assignment, we are interested in quantities that can be estimated from interactions in the environment, and provide approximate credit assignment that can be leveraged to increase the sample-efficiency of RL agents. All in all, they should identify actions that tend to contribute to observed outcomes, with or without the additional context of a trajectory. In the light of the proposed framework and criteria, we thus provide several classes of approximate credit assignment functions c_π that can be used in practice, including quantities with obvious limitations and previously studied quantities. We discuss the pros and cons of each, and give pointers to corresponding existing works.

Temporal proximity. In RL, most algorithms can be viewed as an implicit form of credit assignment: actions are credited according to their *temporal proximity* to outcomes. We explain this assertion using temporal differences. Recall the backward TD(λ) value update given the TD error at timestep t: $V'_\pi(s) = V_\pi(s) + \alpha\delta_t e_{\lambda,t}(s)$, with $e_{\lambda,t} = \gamma\lambda e_{\lambda,t-1}(s) + \mathbb{1}\{s_t = s\}$ the current eligibility trace and α the learning rate. By recursion we get $e_{\lambda,t}(s) = \sum_{t'=0}^{t}(\gamma\lambda)^{t-t'}\mathbb{1}\{s_{t'} = s\}$. The eligibility trace quantifies how much the value of past states is updated by the TD loss, based on how recently they occurred. We can thus view the eligibility trace as the following implicit credit assignment function: $c_\pi(s,a) = \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[e_{\lambda,t}(s')]$, eligibility traces being defined for states, so we naturally transfer the value to the state-action couple that precedes. Hence, temporal differences amount to implicitly crediting past state-action couples, and updating their associated value accordingly. With $\lambda = 1$, we get Monte-Carlo behavior, that is all previous state-action couples are credited for a given TD error, with a weight that decays at the rate of γ. When additionally $\gamma = 1$, which corresponds to the undiscounted case, all previous state-action couples are uniformly credited for each future TD error. Standard RL updates fall into two major categories: value updates and policy gradient updates. Both are based on various flavors of temporal differences, which makes temporal proximity the implicit credit assignment scheme adopted.

Action-gap. A natural candidate to map decisions to desirable outcomes is the *action-gap* (Farahmand, 2011), which quantifies the difference in action-values between the best and the second-best actions:

$g_Q(s_t) = \max_a Q(s_t, a) - \max_{a' \neq \arg\max_a Q(s_t,a)} Q(s_t, a')$. It was used in previous works to identify important states in the environment (Huang et al., 2019a). Still, the action-gap does not take into account the actual action chosen, which makes it hardly suitable for credit assignment. We propose an alternative expression. Formally, we introduce a modified action-gap, which is an implicit credit assignment function: $c_\pi(s, a) = Q_\pi(s, a) - \mathbb{E}_{\bar{a}\sim\bar{\pi}}[Q_\pi(s, \bar{a})]$. The second term corresponds to the value of the non-stationary policy that acts according to $\bar{\pi}$ and then π, in the state s. This is an interesting quantity because it compares the returns of a given action to those of other, counterfactual actions. Though, an important shortcoming is that this quantity is not grounded in any trajectory and no outcome other than the return can be specified as the outcome to credit actions for. Additionally, such a gap necessitates precise action-values and can still be dominated by temporal proximity, since action-values are learned by temporal differences. An advantage is that most off-policy RL methods make use of action-values, so calculating this gap incurs a small computational cost. Note that gap-increasing RL methods (Baird, 1995; Bellemare et al., 2016b; Van Seijen et al., 2019; Ferret et al., 2020; Vieillard et al., 2020b) make action-gaps wider, thus potentially more informative.

Outcome sensitivity to action. The notion of action-gap presented above essentially measures the sensitivity of the return to an individual action change. We now discuss a generalization, which measures the *sensitivity of (arbitrary) outcomes to actions*. This idea is conceptually close to attribution methods (Lundberg and Lee, 2017; Sundararajan et al., 2017) for supervised learning, which quantify feature importance f for an input x as the sensitivity of the prediction $F(x)$ to changes in the value of the features $(x_i)_{i=1}^{|x|}$. For instance, the most straightforward attribution method for differentiable models is plain gradients (Baehrens et al., 2010), *i.e.* for an input x, the feature importance of the feature x_i is $f_i = \frac{\partial F}{\partial x_i}(x)$. Supposing we are using a continuous outcome, we have the context-free credit assignment function: $c_\pi(s, a \,|\, z) = dist(z, \mathbb{E}_{\bar{a}\sim\bar{\pi},\pi}[Z \,|\, S_t = s, A_t = \bar{a}])$. Note that this matches the expression we gave for the metric counterfactual influence (see Chapter B.5). In practice, due to the lack of a simulator, the expectation would be intractable and would have to be approximated. With a similar argument, attribution methods generally cannot work out-of-the-box in model-free RL. Indeed, though a differentiable model of the outcome is not required (we are in the discrete control setting), a model of the outcome is still needed. This dependence on a partial model of the environment places such methods at the frontier between model-based and model-free. An example of such a hybrid method is RUDDER, which leverages a model of the returns and uses the immediate difference in return predictions as a form of credit assignment. The oracle we introduced in the previous section is another example of such mapping, given that it simulates how much the outcome changes under counterfactual actions. Its model is the perfect model given by the simulator.

Action-to-outcome specificity. Another way to map actions to a specific outcome is to compare the likelihood of actions when conditioned on this outcome to their likelihood when not. The former can be learned in a supervised way, using a replay buffer of trajectories from the learning agent, while the latter is simply the behavior policy, that is learned via policy gradients. Intuitively, an action that is more likely than usual when conditioned on an outcome might be of special importance to the outcome. We call this type of mapping *action-to-outcome specificity*. This is the idea that is exploited in HCA (see Chapter 4.2.1) and follow-ups (Young, 2019; Alipov et al., 2021). HCA assigns credit as the ratio between the policy and a hindsight distribution over actions, which is equivalent to a policy that is conditioned on the outcome value. For instance, the return-conditioned version of HCA makes use of the following context-free credit assignment function: $c_\pi(s, a \,|\, g_{0,\gamma}) = 1 - \frac{\pi(a \,|\, s)}{h(a \,|\, s, g_{0,\gamma})}$, where h is by definition the empirical action probability distribution conditioned on the state and outcome and $g_{0,\gamma}$ is the observed discounted return. A drawback of this kind of approaches is that the assigned credit depends on the action distribution induced by the policy, which can have undesirable effects. As an example, suppose

there is an action a in a state s that is sufficient and necessary for success (i.e. getting a return of $g^*_{0,\gamma}$). If $\pi(a|s) \simeq 1$, and $h(a|s, g^*_{0,\gamma}) \simeq 1$, then $c_\pi(s, a \,|\, g^*_{0,\gamma}) = 0$. It goes against the Sensitivity criterion from 4.3.2, which calls for $c_\pi(s, a \,|\, g^*_{0,\gamma}) \simeq 1$ instead.

Predictiveness of outcome. In some scenarios, outcomes can be predicted from a small subset of the state-action couples of a trajectory. In the same way, such a subset can be crucial to the ability of predicting outcomes. In those cases, it is natural to associate outcomes to the corresponding actions. This is the idea behind many of the existing works (Hung et al., 2019; Seo et al., 2019; Ferret et al., 2020; Raposo et al., 2021). We call this type of mapping *predictiveness of outcome*. Predictiveness usually involves a parametric estimator of the outcome, and an attribution mechanism. The attribution mechanism is a by-product of the prediction of the outcome, and quantifies how much individual state-action couples contribute to the overall prediction. For a trajectory τ and a general attribution mechanism α_θ we have the following grounded credit assignment function: $c_\pi(t \,|\, \tau) = \alpha_\theta(f(\tau) \to (s_t, a_t))$. Attribution mechanisms usually depend on the type of estimator used to predict the outcome. Existing attribution mechanisms include retrieval mechanisms: attention mechanisms (Ferret et al., 2020) (which require a sequence model) or memory accesses (Hung et al., 2019) (which require a memory-augmented model). Both assign credit to sequence elements whose content is retrieved to predict the outcome. Alternatives include linear regressions (Raposo et al., 2021), which attribute credit proportionally to the coefficients of the regression of the outcome. By construction, regressions only work for scalar outcomes. A crucial difference between predictiveness and other types of mappings is that it relies on a parametric component (*i.e.* the outcome estimator) while other methods can be non-parametric (*e.g.* using counts, when in the tabular setting).

Generativeness of outcome. Mapping actions to outcomes can be viewed as the following generative problem: producing the next action or the sequence of actions leading to a given outcome. This is achieved by maximizing the likelihood of generated actions in states sampled from a dataset of trajectories, while being conditioned on the value of the outcome. This idea was studied in several works (Kumar et al., 2019; Srivastava et al., 2019; Schmidhuber, 2019). As such, this is an implicit form of credit assignment, since it does not quantify the individual contributions of the generated actions. Nevertheless, such generative models provide estimates of the probability of the next action given the previous states, actions and rewards; as well as the outcome of the trajectory (Edwards et al., 2018; Goyal et al., 2019; Chen et al., 2021; Janner et al., 2021). Credit assignments can be derived from these learned action probabilities, since actions that are highly probable given a desirable outcome should be reinforced. We call this type of mapping *generativeness of outcome*. Two distinct classes of generators can be used: stepwise models and sequence models. Stepwise models estimate the probability of the next action given the current state and desired outcome, while sequence models access the history up to the current point. In POMDPs, sequence models are preferable since the hidden environment state is a function of the history. We thus have the context-free, explicit credit assignment function for generative methods: $c_\pi(s, a \,|\, z) \propto p_\pi(a|s, z)$, where π is either a behavior policy (on-policy setting) or the generative mixture of policies corresponding to a replay buffer (off-policy setting).

Informativeness about outcome. Another form of mapping consists in measuring the information individual actions hold about future outcomes, which we call *informativeness about outcome*. A useful tool for that is the mutual information between actions A and an outcome Z, which is the quantity: $\mathcal{I}_\pi(A; Z \,|\, S) = D_{KL}\big(d_\pi(Z \,|\, S, A) \,||\, d_\pi(Z \,|\, S)\big)$. In the proposed framework, we obtain an implicit credit assignment function by considering the following modification (Arumugam et al., 2021): $c_\pi(s, a) = D_{KL}\big(d_\pi(Z \,|\, S = s, A = a) \,||\, d_\pi(Z \,|\, S = s)\big)$. Though this quantity is not conditioned on an outcome value, it does depend on the outcome whose distribution is modeled. Here, we assume that the outcome variable Z is a continuous random variable (*e.g.* the return). A drawback of this mapping is its applicability:

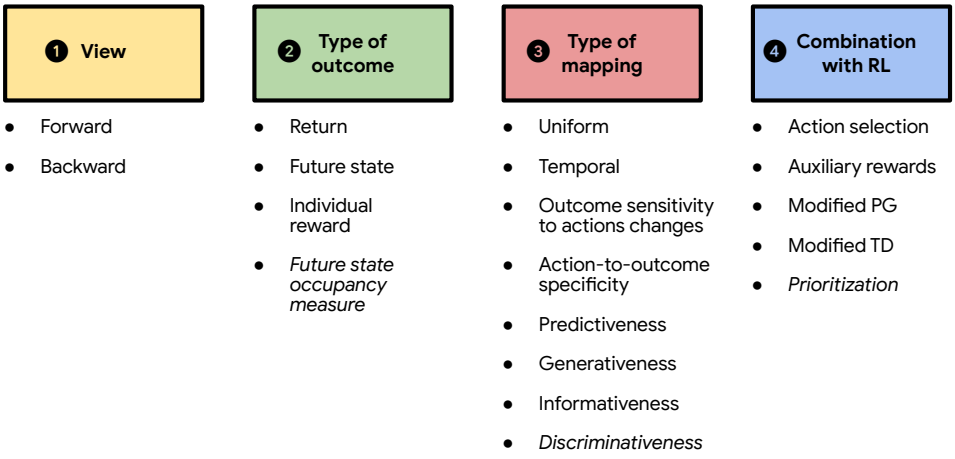


Figure 4.6: The proposed major characteristics of credit assignment methods. Items in *italic* correspond to future directions, see Chapter 4.3.5.

it requires to explicitly model the probability distribution of Z. While there are tools available in the context of RL, notably distributional RL (Bellemare et al., 2017; Dabney et al., 2018), it might limit the practical applications of such a mapping. To the best of our knowledge, no existing method makes use of this type of mapping at the time of writing. We refer the interested reader to Arumugam et al. (2021) for additional elements.

4.3.4 Other practical design choices for credit assignment methods

In the previous section we distinguished several ways of mapping outcomes to actions, and practical implications. We now discuss several important design choices and characteristics of practical credit assignment methods, some of which being constrained by the type of mapping used, others being independent. We suppose neural networks are used for function approximation. Importantly, we provide a taxonomy of existing credit assignment methods by distinguishing several major axes of variation. The taxonomy can be found in Table 4.1. We hypothetically assess these methods against the proposed criteria in Table 4.2. We provide partial expressions for the associated credit assignment functions in Table 4.3.

Forward and backward view. We call forward methods those that assign credit to actions when they are performed, and are either implicit or context-free. They can take advantage of hindsight information, *i.e.* information about the observed future, but only during training. Backward methods assign credit when an outcome is observed, and are grounded functions if they make use of the extended context provided by the trajectory. Both views hold advantages and drawbacks that will suit different scenarios, similarly to the forward and backward view in planning (Chelu et al., 2020). A notable difference is that backward methods have knowledge about the future, which can facilitate estimation by reducing variance at the cost of increased bias. Backward, grounded methods rely naturally on sequence models, which build representations that contain information about the whole trajectory. Sequence models usually incur an increased computational cost (with respect to stepwise models) and might require more data to train, but they also might help with credit assignment via attention mechanisms or return decomposition. Forward and backward explicit methods often rely on forms of conditioning, since they assign credit on the basis of an outcome value. This is an important practical aspect: conditioning was shown to improve the predictive capabilities of various neural networks in the presence of additional, informative inputs (Perez

et al., 2018). In the online RL setting, backward methods can be ill-adapted since they usually require that the outcome is observed before updating estimates.

Type of mapping. See Chapter 4.3.3.

Type of outcome. While any function of the trajectory is an admissible outcome, we describe existing outcomes that are used for explicit credit assignment as well as natural extensions:

- **return:** a common RL objective consists in maximizing returns, so return-specific credit assignment can help reinforcing actions that cause high returns. There is a distinction to make between truncated returns and full returns. While truncated returns only account for a portion of the future, they suffer from reduced variance and have a smaller number of actions to credit. In practice, most existing methods use full trajectory returns, *i.e.* $g_{0,\gamma}$. Since the return is a continuous outcome whose utility increases with its value, meaningful distance functions are easy to find (e.g. the squared distance).

- **future state:** an alternative is to use specific future states as the outcome to credit for. Using future states as outcomes instead of the whole return allows to selectively credit different actions along the same trajectory, since distinct actions can be credited for distinct future states. While this is naturally useful for goal-oriented RL (Andrychowicz et al., 2017), it might also be useful when the return is ambiguous, *e.g.* if two future states are associated to equal returns under a given behavior policy. Note that using the future state as an outcome requires a suitable representation for the state, which might be the corresponding observation or a learned representation.

- **reward:** another alternative is to use individual rewards, when experienced in the context of their state, as outcomes. This is different from using future states as outcomes, since rewards might not be a deterministic function of the state-action couple preceding a given state: one might want to assign credit proportionally to the value of the reward. Note that rewards are equivalent to discounted returns when $\gamma = 0$.

Combination with RL. There are several ways to incorporate the credit assignments into RL algorithms.

- **action selection:** actions that get credit for desirable outcomes are sampled more often. In practice, existing methods condition their behavior policy on the value of an outcome, value which is either sampled from a distribution shaped by previous interactions in the environment (Kumar et al., 2019) or from privileged information about the task at hand (*e.g.* knowing the value of the best return achievable) (Chen et al., 2021).

- **auxiliary rewards:** the reward is augmented proportionally to a function of the assigned credit. If the function is the identity, this is a regular reward bonus. Alternatively, one might use other bonuses, such as potential-based reward shaping (Ng et al., 1999), look-ahead/look-back advice (Wiewiora et al., 2003) or reward redistribution (Arjona-Medina et al., 2019).

- **modified policy gradient:** the assigned credit is directly integrated in the policy gradient updates. Existing modifications either substitute the advantage function (Mesnard et al., 2021) or weight individual rewards (Harutyunyan et al., 2019).

- **modified TD:** the assigned credit is directly integrated in the temporal differences updates. Existing modifications either add long-term connections in the bootstrapping target (Hung et al., 2019) or weight bootstrapping terms according to state importance (Sutton et al., 2016).

Method \Aspect	View	Mapping	Outcome	Combination w/ RL	Agnostic	Dealing with EA
SECRET (Ferret et al., 2020)	Backward	Predictiveness	Reward	Auxiliary rewards	Yes	Info. removal
HCA-Return (Harutyunyan et al., 2019)	Backward	Specificity	Return	Modified PG	No	
HCA-State (Harutyunyan et al., 2019)	Backward	Specificity	State	Modified PG	No	
CCA (Mesnard et al., 2021)	Backward	Action-gap	Return	Modified PG	No	Info. bottleneck
TVT (Hung et al., 2019)	Backward	Predictiveness	Return	Modified TD	No	Dim. reduction
IRCR (Gangwani et al., 2020)	Backward	Uniform	Return	Auxiliary rewards	Yes	
SMTCA (Liu et al., 2019)	Backward	Predictiveness	Return	Auxiliary rewards	Yes	
SR (Raposo et al., 2021)	Forward	Predictiveness	Reward	Auxiliary rewards	Yes	
RCP (Kumar et al., 2019)	Forward	Generativeness	Return	Action selection	No	
UDRL (Srivastava et al., 2019)	Forward	Generativeness	Return	Action selection	No	
DT (Chen et al., 2021)	Forward	Generativeness	Return	Action selection	No	
RPBCA (Seo et al., 2019)	Backward	Sensitivity	Return	Auxiliary rewards	Yes	
RUDDER (Arjona-Medina et al., 2019)	Backward	Sensitivity	Return	Modified PG	Yes	Info. removal
PWTD / PWR (Zheng et al., 2021)	Backward		State	Modified PG	Yes	

Table 4.1: Taxonomy of credit assignment methods. EA stands for Explaining Away.

Agnosticism to the type of agent. While *ad-hoc* methods necessitate a specific RL agent or agent architecture, agnostic methods impose no such constraints. Agnostic methods tend to feature a credit assignment module that is either fully independent from the RL agent networks or only shares some representations. Agnostic methods have the advantage that they can be combined to arbitrary RL agents.

Explaining away. A common challenge for explicit credit assignment methods is *explaining away* (Kakade, 2001): subsequent states and actions give information about past actions, and credit can be falsely attributed to the subsequent actions, when past actions are potentially the ones that contribute the most. Existing methods address this issue by reducing information about the past in representations of the future, or conversely by reducing information about the future in representations of the past. We detail some of these approaches here:

- **information removal:** information is removed from the observations that are used to compute the credit assignments. For instance, Ferret et al. (2020) apply an ad-hoc transformation to the observations of the trajectory (*e.g.* only keeping the agent surroundings), while Arjona-Medina et al. (2019) use the difference between consecutive observations instead of observations as inputs for predictions.
- **dimensionality reduction:** observations (or more generally, the agent's representation of its current state) are mapped to lower dimensional representations, with potential auxiliary losses to control which kind of information they retain (Hung et al., 2019).
- **information bottleneck:** during training, the agent receives aggregated information about the future (Guez et al., 2020; Lee et al., 2020b; Mesnard et al., 2021; Venuto et al., 2021). The amount of information is regularized so that the agent does not rely too much on privileged information about the future (which is only available as an approximation at inference time), and instead focuses on the current observation.

4.3.5 Future directions

We briefly review future directions of interest and how they can be integrated in the described framework.

Method \Criterion	Sensitivity	Relativity	Separability	Invariance	Stability	Generalizability
SECRET (Ferret et al., 2020)	✓	✓	✓			✓
HCA (Harutyunyan et al., 2019)				✓	✓	
CCA (Mesnard et al., 2021)	✓	✓	✓			✓
TVT (Hung et al., 2019)	✓	✓	✓			✓
IRCR (Gangwani et al., 2020)				✓	✓	
SMTCA (Liu et al., 2019)				✓	✓	✓
SR (Raposo et al., 2021)	✓	✓		✓	✓	
RCP (Kumar et al., 2019)				✓	✓	
UDRL (Srivastava et al., 2019)				✓	✓	
DT (Chen et al., 2021)				✓	✓	
RPBCA (Seo et al., 2019)				✓	✓	
RUDDER (Arjona-Medina et al., 2019)	✓	✓	✓			✓
PWTD / PWR (Zheng et al., 2021)				✓	✓	✓

Table 4.2: Criteria fulfilled by credit assignment methods. When relying on estimators, we suppose that the estimator is able to provide the exact quantity desired, so this is a hypothetical assessment. We give more details in Appendix B.2.

Method	Credit function	Additional details
SECRET (Ferret et al., 2020)	$c_\pi(t \mid \tau, t') = \alpha_{\theta,\tau}(r_{t'} \to (s_t, a_t))$	$\alpha_{\theta,\tau}$ is the self-attention weight when predicting individual rewards.
HCA-Return (Harutyunyan et al., 2019)	$c_\pi(s, a \mid g_{0,\gamma}) = 1 - \frac{\pi_\theta(a\mid s)}{h_\theta(a \mid s, g_{0,\gamma})}$	h_θ is an action distribution conditioned on the outcome.
HCA-State (Harutyunyan et al., 2019)	$c_\pi(s, a \mid s') = \frac{h_\theta(a \mid s, s')}{\pi_\theta(a\mid s)} - 1$	
CCA (Mesnard et al., 2021)	$c_\pi(t \mid \tau) = G_{\gamma,t} - V_\theta(s, \phi_{\theta,t}(\tau))$	$\phi_{\theta,t}$ is an embedding of the remainder of the trajectory.
TVT (Hung et al., 2019)	$c_\pi(t \mid \tau, t') = \alpha_{\theta,\tau}(s_{t'} \to (s_t, a_t))$	$\alpha_{\theta,\tau}$ is the memory retrieval weight when reconstructing current items.
IRCR (Gangwani et al., 2020)	$c_\pi(t \mid \tau) = \frac{1}{T}$	
SMTCA (Liu et al., 2019)	$c_\pi(t \mid \tau) = \alpha_{\theta,\tau}(g_{0,\gamma} \to (s_t, a_t))$	$\alpha_{\theta,\tau}$ is the stepwise predicted reward when predicting the trajectory return.
SR (Raposo et al., 2021)	$c_\pi(s, a) = c_\theta(s, a)$	c_θ is the coefficient in the linear regression of the reward.
RCP (Kumar et al., 2019)	$c_\pi(s, a \mid g_{0,\gamma}) = \pi_\theta(a \mid s, g_{0,\gamma})$	
UDRL (Srivastava et al., 2019)	$c_\pi(s, a \mid g_{0,\gamma}) = \pi_\theta(a \mid s, g_{0,\gamma})$	
DT (Chen et al., 2021)	$c_\pi(s, a \mid g_{0,\gamma}) = \pi_\theta(a \mid s, g_{0,\gamma})$	
RPBCA (Seo et al., 2019)	$c_\pi(s, a \mid g_{0,\gamma}) = \alpha_{\theta,\tau}(g_{0,\gamma} \to (s, a))$	$\alpha_{\theta,\tau}$ is the argmax of the sum of predicted reward differences.
RUDDER (Arjona-Medina et al., 2019)	$c_\pi(s, a \mid g_{0,\gamma}) = \alpha_{\theta,\tau}(g_{0,\gamma} \to (s, a))$	$\alpha_{\theta,\tau}$ is the redistributed reward when predicting the trajectory return.
PWTD / PWR (Zheng et al., 2021)	$c_\pi(s_t, a_t \mid s_{t'}) = f_\theta(s_t, a_t, s_{t'}, t' - t)$	f_θ is meta-learned to accelerate objective maximization.

Table 4.3: Functions used in credit assignment methods. Quantities subscripted by θ are learned quantities. $\alpha_{\theta,\tau}$ are grounded mappings of *observed* outcomes to actions. Their nature vary from one method to the other, see the rightmost column for details. We recall that $g_{0,\gamma}$ is the value of the total discounted return.

Discriminativeness of outcome as mapping. A way of mapping actions to outcomes is to learn a conditional discriminator: it is given an observed outcome and has to distinguish the actual data (*i.e.* that was observed and led to the outcome) from negative data (*i.e.* generated or sampled data). Then, a function of the confidence of the discriminator is used as a source of credit. This is an idea that is similar to Adversarial Imitation Learning (Ho and Ermon, 2016; Orsini et al., 2021), where a discriminator has to distinguish between trajectories from the behavior policy and expert trajectories. Given a discriminator $D_\theta(s, a; z)$, we then have the context-free, explicit credit assignment function $c_\pi(s, a|z) = f(D_\theta(s, a; z))$, where f is a continuous and increasing function over $(0, 1)$. Functions used in the Imitation Learning literature include $f(x) = \log(x)$ (Ho and Ermon, 2016), $f(x) = -\log(1 - x)$ (Ghasemipour et al., 2020), and $f(x) = \mathbb{1}\{x > \frac{1}{2}\}$ (Bahdanau et al., 2019). To the best of our knowledge, discriminativeness is yet to be studied in the context of credit assignment for RL.

Other directions. There are other future directions of interest that we review hereafter.

- using future state occupancies as outcome: an unexplored, alternative outcome is the future state occupancy. The discounted state occupancy measure induced by a policy π from a state s_t is $\nu_\pi(s \,|\, s_t) = ((1-\gamma)/(1-\gamma^{T-t+1})) \sum_{t'=t}^{T} \gamma^{t'-t} p_\pi(S_{t'} = s \,|\, S_t = s_t)$. Equivalently, we have the action-conditioned discounted state occupancy measure $\mu_\pi(s \,|\, s_t, a_t) = ((1-\gamma)/(1-\gamma^{T-t+1})) \sum_{t'=t}^{T} \gamma^{t'-t} p_\pi(S_{t'} = s \,|\, S_t = s_t, A_t = a_t)$. This might notably help with continuous or near-continuous state spaces. Note that to the best of our knowledge, this was not used in existing explicit credit assignment methods.
- using prioritization to combine credit with RL: transitions in the replay buffer could be sampled according to a priority distribution (Schaul et al., 2015b), where priorities are proportional to the credit assigned. A parallel can be made with self-imitation (Oh et al., 2018), which prioritizes sampled transitions according to the positive part of an estimated advantage. Note that to the best of our knowledge, this was not used in existing explicit credit assignment methods.

4.4 Conclusion

In this chapter, we provided a general framework for credit assignment in RL, which unifies existing works under a common abstraction. This framework makes apparent that different types of credit assignment methods (either implicit, context-free or grounded methods) can have their assigned credit viewed as action importances with respect to distinct quantities (expected returns or more specific outcomes; and within or outside the context of a trajectory). By providing separate design choices that describe and distinguish credit assignment methods, we revealed that there was room for novel credit assignment methods that are yet to be studied from both theoretical and empirical point of views. We hope that the proposed framework will benefit the research community and spark interest for explicit credit assignment, which we argue is a much-needed RL component, especially when dealing with hard credit assignment problems.

Chapter 5

Implicit Credit Assignment via Self-Imitation

Having presented our unifying framework for credit assignment, we now take an interest in credit assignment methods, starting with implicit credit assignment, which is best understood as a form of action importance with respect to the outcome measuring the global performance of the agent (that is, the expected cumulative discounted reward). In this chapter, hence, we study an implicit form of credit assignment for off-policy RL that incorporates returns in the RL updates. This chapter was presented as a full conference paper at AAMAS 2021 (Ferret et al., 2021).

5.1 Introduction

Self-imitation learning is a Reinforcement Learning (RL) method that encourages actions whose returns were higher than expected, which helps in hard exploration and sparse reward problems. It was shown to improve the performance of on-policy actor-critic methods in several discrete control tasks. Nevertheless, applying self-imitation to the mostly action-value based off-policy RL methods is not straightforward. Hence, we propose SAIL, a novel generalization of self-imitation learning for off-policy RL, based on a modification of the Bellman optimality operator that we connect to Advantage Learning (AL). Crucially, our method mitigates the problem of stale returns by choosing the most optimistic return estimate between the observed return and the current action-value for self-imitation. We demonstrate the empirical effectiveness of SAIL on the Arcade Learning Environment, with a focus on hard exploration games.

Some approaches combine Reinforcement Learning (RL) and learning from (expert) demonstrations Piot et al. (2014); Hester et al. (2018). It is efficient, but having access to expert demonstrations might not be possible in many situations. An interesting and more practical alternative is to learn from oneself, by focusing on one's own instructive experiences. In the context of RL (Sutton and Barto, 2018), self-imitation learning (SIL) is a technique that takes advantage of this idea and proposes to learn from positive experiences, using actions whose payoff was superior to what had been predicted. It was introduced by Oh et al. (2018) for on-policy learning, where interaction data is obtained via a behaviour policy and is used to improve this policy, never to be reused again. SIL was shown to improve the performance of on-policy actor-critic methods in several discrete control tasks. In that context, it provides a practical way to revisit and reinforce interesting actions.

In the off-policy setting, agents learn instead from the behaviour of different policies than their own, while they are still periodically allowed to interact in the environment to generate new interaction data. Off-policy learning is attractive because it would make RL useful in situations where interacting in the

environment is costly or impractical, which is the case in many real-world scenarios (Dulac-Arnold et al., 2019). In the most extreme case, offline RL (Levine et al., 2020), no interaction at all is possible and one must learn a policy just by looking at a dataset of past experiences generated by potentially many different policies. Sadly, the canonical form of SIL relies on a modified policy gradient (Sutton, 1984), which requires the ability to modify the current policy in a given direction. This requirement makes it incompatible with most off-policy methods (Mnih et al., 2013; Hessel et al., 2018a; Kapturowski et al., 2018), whose policy is obtained by applying an exploration method on top of the current estimate of the optimal action-value. Since the policy is implicit, gradient ascent in the policy space is impractical. As an example, a straightforward adaptation of SIL for a standard off-policy RL algorithm such as Deep Q-Networks (DQN, Mnih et al., 2013) seems out-of-reach. Indeed, the action-value network that is central to DQN encapsulates the two aspects SIL targets separately in the actor-critic: value estimation and decision-making.

The focus of this work is on providing a self-imitation method that is applicable across the full spectrum of off-policy methods. A common denominator to all off-policy algorithms is the use of the action-value to inform decision-making, be it with an implicit policy (Mnih et al., 2013), or an explicit one (Haarnoja et al., 2018). Hence, a way to control which actions are reinforced in off-policy learning is to artificially increase the reward of such actions. Staying in the line of reasoning of SIL, a natural idea is to use the difference between the observed return and the estimated value as the reward bonus. It turns out that doing so creates a problem because the observed return becomes stale over time, biasing action-value towards pessimism instead of optimism, and eventually reducing the self-imitation contribution to none. To circumvent this issue, we opt for the simple strategy of using the most optimistic between the return and the estimated action-value, with which we extend the benefits of self-imitation. We show a connection to Advantage Learning (AL, Baird, 1999; Bellemare et al., 2016b), an action-gap increasing off-policy algorithm.

Hence, we propose SAIL, a novel generalization of self-imitation learning for off-policy RL, based on a modification of the Bellman optimality operator that we connect to Advantage Learning (AL). Crucially, our method mitigates the problem of stale returns by choosing the most optimistic return estimate between the observed return and the current action-value for self-imitation. We demonstrate the empirical effectiveness of SAIL on the Arcade Learning Environment, with a focus on hard exploration games.

Our contributions are the following: 1) we propose SAIL, a generalization of self-imitation learning for off-policy methods, 2) we show how it connects to Advantage Learning and complements it, 3) we demonstrate the practicality of our method in terms of simplicity, efficiency and performance on the Arcade Learning Environment (ALE, Bellemare et al., 2013) benchmark, under several base off-policy RL methods. Notably, we report considerable performance gains on hard exploration games.

5.2 Additional background

Actor-critic We recall that actor-critic methods use a policy and a value function. In the standard formulation, the value (from the critic) guides the policy via the policy gradient,

$$\mathcal{L}_{PG} = -\log \pi_\theta(a_t|s_t)\big(G_{\gamma,t} - V_\theta(s_t)\big),$$

while the value itself is updated via temporal difference on data generated by the actor:

$$\mathcal{L}_V = \frac{1}{2}\Big(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)\Big)^2.$$

Off-policy RL We already presented DQN in Chapter 3.2, which we refer the reader to. Rainbow (Hessel et al., 2018a) extends DQN by combining several independently proposed algorithmic innovations (Schaul

et al., 2015b; Van Hasselt et al., 2015; Wang et al., 2015; Bellemare et al., 2017; Fortunato et al., 2018), and showed to be a strong baseline on the Atari Learning Environment benchmark. IQN (Dabney et al., 2018) is a distributional RL approach that estimates the whole distribution of the returns, instead of the mean as in standard Q-learning. More precisely, it approximates the continuous quantile function of the return distribution by sampling input probabilities uniformly. In terms of performance, it (almost) bridges the gap with Rainbow while not having any prioritized replay and using single-step bootstrapping.

Self-Imitation Learning Self-imitation learning (SIL, Oh et al., 2018) provides a set of additional loss functions that complement the existing actor-critic losses, and encourage the agent to mimic rewarding past behavior. It relies on a replay buffer that stores past transitions $\{s_t, a_t, G_{\gamma,t}\}$, where the observed return $G_{\gamma,t} = \sum_{i=0}^{+\infty} \gamma^i r_{t+i}$ is obtained once the episode is over. SIL operates off-policy, since its losses are calculated using past transitions from the replay buffer, implying that on-policy actor-critic methods that benefit from SIL become part on-policy, part off-policy. For a given transition $(s_t, a_t, G_{\gamma,t})$, the additional policy and value losses are:

$$\mathcal{L}_{PG}^{\text{sil}} = -\log \pi_\theta(a_t|s_t)\big(G_{\gamma,t} - V_\theta(s_t)\big)_+ \text{ and } \mathcal{L}_V^{\text{sil}} = \frac{1}{2}\Big(G_{\gamma,t} - V_\theta(s_t)\Big)_+^2,$$

where the notation $(x)_+ = \max(0, x)$ stands for the ReLU operator. The shared term $(G_{\gamma,t} - V_\theta(s_t))_+$ is positive if the observed return outweighs the estimated value, in which case both the action probabilities and the value are increased.

SIL also relies on prioritization (Schaul et al., 2015b), as transitions are sampled from the replay buffer with probability $P(\tau) \propto (G_{\gamma,t} - V_{\theta^-}(s_t))_+$. As a result, transitions in which actions led to an unexpectedly high return get revisited more often, ensuring faster propagation of important reward signals.

A drawback of this formulation is that an explicit policy is required. Also, while it provides good empirical results, self-imitation in general might suffer from stochasticity, both in the environment dynamics and in the variance of the rewards. Indeed, the optimistic updates of SIL can lead to overestimating the value of actions that bring high rewards once in a while. See Chapter 5.4.5 for a more in-depth discussion about the role of stochasticity.

Advantage Learning We refer the reader to our presentation of AL in Chapter 3.2. We add here that AL uses the following modified Bellman operator:

$$\mathcal{T}_{AL}Q(s,a) = \mathcal{T}^*Q(s,a) + \alpha(Q(s,a) - \max_{a'} Q(s,a')),$$

which translates to the following reward modification:

$$\tilde{r}_{AL}(s,a) = r(s,a) + \alpha(Q(s,a) - \max_{a'} Q(s,a')).$$

It can be seen as adding the advantage $Q(s,a) - V(s)$ to the reward, with $V(s) = \max_a Q(s,a)$ (hence the name). Generally speaking, increasing the action-gap makes the RL problem easier, as it facilitates the distinction from experience between the optimal action and the others. In practice, AL was shown to bring consistent performance improvement across the whole support of Atari games for off-policy agents (Bellemare et al., 2016b).

5.3 SAIL: Self-Imitation Advantage Learning

In essence, self-imitation drives the policy towards actions whose returns are unexpectedly good. Off-policy methods usually optimize an action-value function, which only implicitly defines the policy through a

given exploration method. Nevertheless, there is a sensible proxy for increasing the probability of picking an action: increasing its action-value (which is nothing more than a reward increase). Hence, we propose to adapt self-imitation for off-policy using the following modified reward:

$$\tilde{r}_{SAIL}(s_t, a_t) = r(s_t, a_t) + \alpha(\max(G_{\gamma,t}, Q_{\theta^-}(s_t, a_t)) - \max_a Q_{\theta^-}(s_t, a)),$$

The corresponding loss function is:

$$\mathcal{L}_{SAIL} = \frac{1}{2}\left(\tilde{r}_{SAIL}(s_t, a_t) + \gamma \max_a Q_{\theta^-}(s_{t+1}, a) - Q(s_t, a_t)\right)^2.$$

We now motivate this expression in more details.

First, note that if we replace $\max(G_{\gamma,t}, Q(s, a))$ by $G_{\gamma,t}$ and apply the ReLU operator to the additional term we get the modified reward:

$$\tilde{r}(s_t, a_t) = r(s_t, a_t) + \alpha(G_{\gamma,t} - \max_a Q_{\theta^-}(s_t, a))_+,$$

which would be a more straightforward adaptation of SIL, since it increases the action-value by the same term used in SIL for both policy and value updates. Empirically, this formulation compares unfavorably to ours (see Chapter 5.4.6). We posit it could be due to a limitation of self-imitation: the problem of stale returns. This problem is the following: when off-policy, the return used for self-imitation corresponds to the observed return of a policy that becomes increasingly different from the current one, under the hypothesis that the agent is still learning. Inevitably, returns become outdated. More precisely, these returns drift towards pessimism, since the agent mostly improves over time. As a result, the self-imitation bonus disappears for most transitions. In addition to making self-imitation ineffective, it could lead to emphasizing wrong signals and biasing the updates. We illustrate this phenomenon in Fig. 5.1. By exchanging the return for the maximum between the return and the current action-value estimate, our method forms a more optimistic return estimate, which promotes optimism and alleviates staleness. Additionally, by replacing the ReLU operator with the identity, we update our action-value estimate using information from both positive and negative experiences. Finally, compared to SIL, we do not make use of prioritization, as we think it emphasizes the impact of stochasticity on the algorithm. In our experiments, we measure the benefits of our method under increasing stochasticity in Chapter 5.4.5, and show that the gains over the baseline remain even when stochasticity is high. Second, the proposed modified reward can be decomposed as:

$$\tilde{r}_{SAIL}(s_t, a_t) = \tilde{r}_{AL}(s_t, a_t) + \alpha(G_{\gamma,t} - Q_{\theta^-}(s_t, a_t))_+,$$

Thus, our approach actually combines AL, an action-gap increasing method, with SIL, which helps reproducing interesting trajectories from the past. We show next that it outperforms AL.

We name the resulting algorithm Self-Imitation Advantage Learning (SAIL[1]). It is: **1) general**, i.e. compatible across a large spectrum of off-policy methods,**2) easy to use**, as it arranges existing quantities to form a reward bonus, and **3) lightweight**, in the sense that it does not add much to the computational budget of the algorithm (other than computing discounted returns, which is neglectible). The pseudo-code is in Alg. 1.

5.4 Experiments

In this section, we aim to answer the following questions: **1)** How does SAIL perform on various tasks, including hard exploration, using various off-policy algorithms as baselines? (see Chapter 5.4.2 & 5.4.3);

[1] We take the liberty of inverting the middle letters to form a nice acronym.

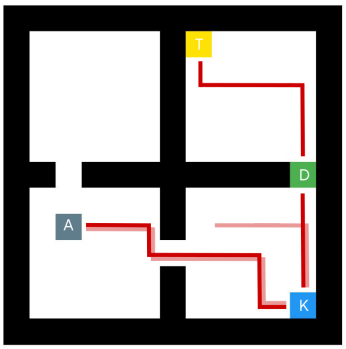


Figure 5.1: Illustration of the stale returns problem, in the Key-Door-Treasure environment (Oh et al., 2018). In past experiences (in faded red), the agent ("A") got the key ("K") but failed to reach the goal ("T"). The corresponding transitions still get sampled from the replay buffer, so actions leading to the key are paired with zero returns. This is inconsistent with the current policy (in red), which gets the key, opens the door and reaches the goal. SAIL avoids undervaluing the role of the key by using the most optimistic return estimate between the observed return and the current action-value.

Algorithm 1: SAIL: Self-Imitation Advantage Learning

Initialize the agent weights θ;
Initialize the replay buffer $\mathcal{B}$;
Initialize the return placeholder $G_{\varnothing}$;
for *each iteration* **do**
  `/* Collect and store interaction data.  */`
  for *each interaction step* **do**
    In s_t, sample $a_t \sim \pi_\theta$, act, observe r_t and s_{t+1};
    $G_{\gamma,t} \leftarrow G_{\varnothing}$;
    Store s_t, a_t, r_t, $s_{t+1}, G_{\gamma,t}$ in $\mathcal{B}$;
    if *the episode is over* **then**
      $G_{\gamma,t} \leftarrow \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ for $t \in [0, T]$;
    end
  end
  `/* Update the agent weights (off-policy).  */`
  for *each training step* **do**
    Sample a minibatch $\mathcal{D}_{batch}$ from $\mathcal{B}$;
    `/* Loss expression in Equation 5.3.  */`
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{SAIL}(\mathcal{D}_{batch})$;
  end
end

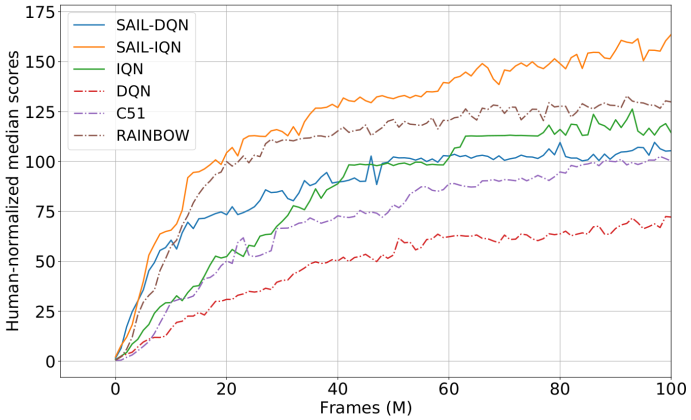


Figure 5.2: Human-normalized median performance of several off-policy methods. SAIL-IQN outperforms every other method, including Rainbow, which uses prioritized experience replay and n-step bootstrapping. SAIL-DQN, though based on DQN, outperforms C51, a distributional RL algorithm that uses a more advanced optimizer (Adam).

2) Does SAIL compare favorably to exploration bonuses? (see Chapter 5.4.4); **3)** What is the impact of stochasticity on SAIL? (see Chapter 5.4.5); and **4)** How does SAIL compare to a more straightforward self-imitation algorithm? (see Chapter 5.4.6).

5.4.1 Experimental setting

We benchmark our method and baselines on the Arcade Learning Environment (Bellemare et al., 2013), with sticky actions and no episode termination on life losses, following (Machado et al., 2018). Sticky actions make the Atari games stochastic by repeating the agent's previous action with a fixed probability (0.25 by default). Hard exploration games are games where local exploration methods fail to achieve high scores. We use the list of hard exploration games from (Bellemare et al., 2016a), and set them apart with a bold font in bar plots. We use the Dopamine framework (Castro et al., 2018) to get reference implementations for the agents and setup our experiments. Whether using the base off-policy algorithms as is or in combination with SAIL, we use Dopamine reference hyperparameters for all methods, unless explicitly mentioned. The only SAIL-specific hyperparameter is α, that we set to $\alpha = 0.9$ in all experiments. The bonus term is also clipped in $[-1, 1]$ (as are the rewards in ALE). We use 6 random seeds for comparative studies on subsets of games, and 3 random seeds for experiments on the 59[2] Atari games, due to the computational demand of such experiments.

For bar plots (e.g. Fig. 5.3), the mean relative improvement metric for a method X compared to the baseline X_{base} is, for one game:

$$\frac{\frac{1}{N}\sum_{t=1}^{200} s_{X,t} - s_{X_{base},t}}{|\frac{1}{N}\sum_{t=1}^{200} s_{X_{base},t}| + \epsilon},$$

where $\{s_{X,t}\}_{t=1,\dots,200}$ are the game scores for method X, collected every million steps in the environment, and averaged across all random seeds. This metric roughly measures by how much the performance of the proposed method increased or decreased compared to the baseline. For aggregated plots (e.g. Fig. 5.2), we report the human-normalized median score, a common metric to evaluate RL methods based on their

[2]We didn't report results for ElevatorAction, whose ROM failed to load in our setup.

Table 5.1: Final scores of several off-policy algorithms based on DQN, in hard exploration games.

	DQN	*AL-DQN*	*SAIL-DQN*
Alien	2586	3514	**4769**
Amidar	1167	**1380**	879
BankHeist	594	909	**915**
Freeway	**25**	22	23
Frostbite	392	2833	**7912**
Gravitar	299	540	**1292**
Hero	17056	7672	**30382**
Montezuma's Revenge	0	0	0
MsPacman	3405	**3973**	3879
Pitfall!	-84	-100	**-27**
PrivateEye	-114	153	**10068**
Qbert	10178	**13426**	9310
Solaris	1437	**2216**	855
Venture	35	371	**989**
WizardOfWor	2020	5930	**6906**
Zaxxon	4713	6221	**10364**

performance **across all games**:

$$\frac{\bar{s}_{X,t} - s_{random})}{|s_{human} - s_{random}|},$$

where $\{\bar{s}_{X,t}\}_{t=1,\dots,200}$ are the median scores across all games for method X, collected every million steps in the environment, and averaged across all random seeds. s_{random} is the end score of a random policy, and s_{human} that of the average human player, as reported in Mnih et al. (2013), and completed for missing games as in Vieillard et al. (2020b).

5.4.2 SAIL-DQN experiments

We report the performance of SAIL combined to DQN (abbreviated SAIL-DQN) against DQN on all Atari games in Fig. 5.3. We use the Dopamine DQN hyperparameters.

We also report the same graph with AL-DQN as the baseline on Fig. 5.3. AL-DQN corresponds to the limit case of SAIL where the action-value estimate is always greater than the observed return, and makes for a stronger baseline than vanilla DQN. With the simple modification provided by SAIL, we get a nice average and median performance increase across all games (+101.4% average and +6.7% median), the gap being more apparent on hard exploration games where self-imitation shines the most (+361.6% average and +24.9% median relative improvement over AL-DQN).

We also zoom in on the performances of the three methods (DQN, AL-DQN and SAIL-DQN) on individual games. We choose four hard exploration games: Frostbite, Venture, Gravitar and PrivateEye. The scores are displayed on Fig. 5.4. On Frostbite and PrivateEye, SAIL-DQN outperforms C51[3] (Bellemare et al., 2017), a distributional RL agent that uses Adam (Kingma and Ba, 2014), a more advanced optimizer than DQN's RMSProp (Tieleman and Hinton, 2012). On PrivateEye, that has particularly sparse rewards, SAIL-DQN also outperforms IQN. Finally, we report the final performances on hard exploration games for all three methods in Table 5.1, and individual curves on all 59 games can be found in Fig. 5.12.

[3]C51 scores 4250 on Frostbite and 4000 on PrivateEye. Scores are available here.

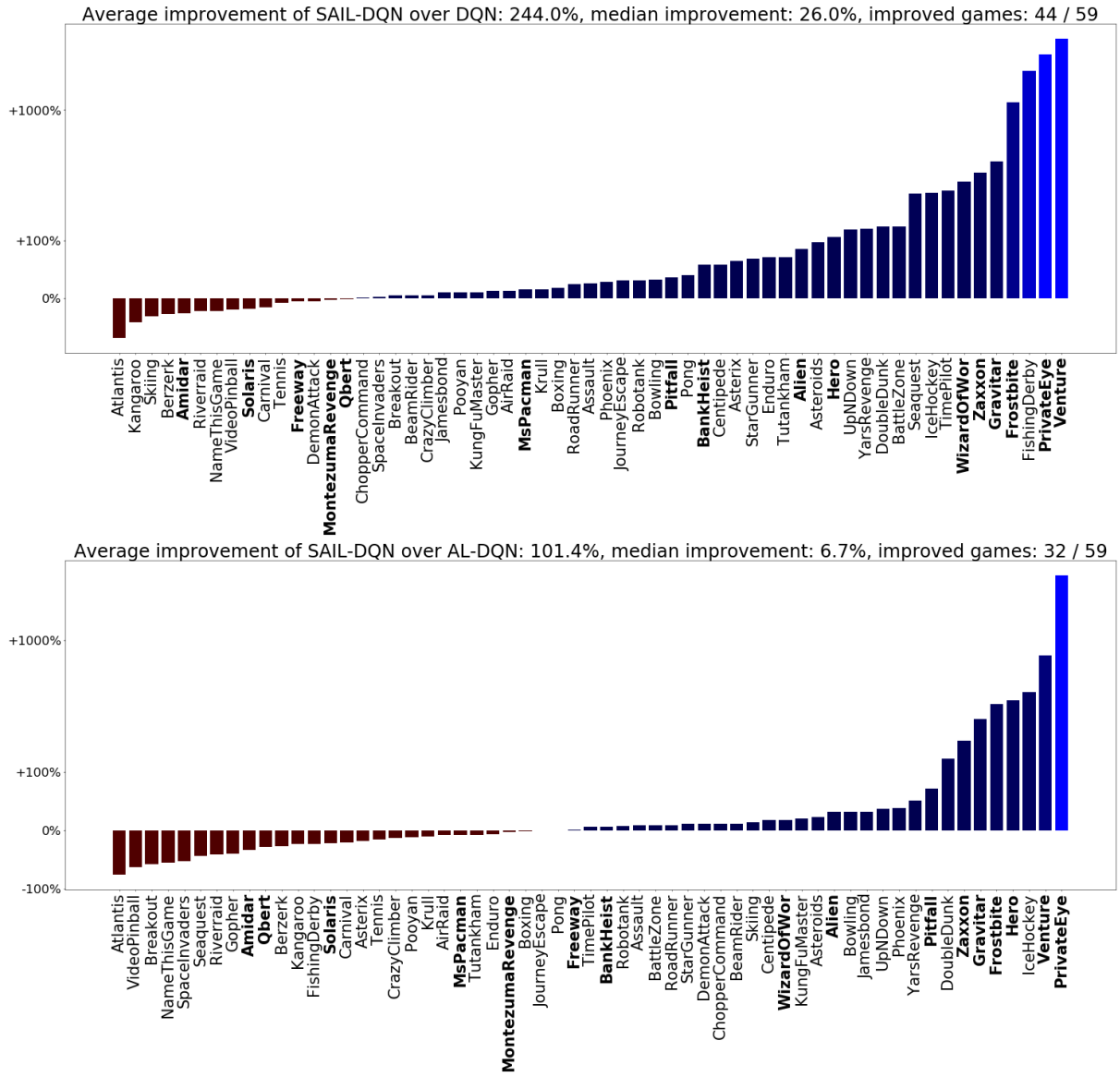


Figure 5.3: SAIL-DQN outperforms both DQN (top) and AL-DQN (bottom), as shown on a relative scale on all Atari games. On hard exploration games (in bold), SAIL-DQN provides a +654.3% average and +71.8% median relative improvement over DQN, and a +361.6% average and +24.9% median relative improvement over AL-DQN.

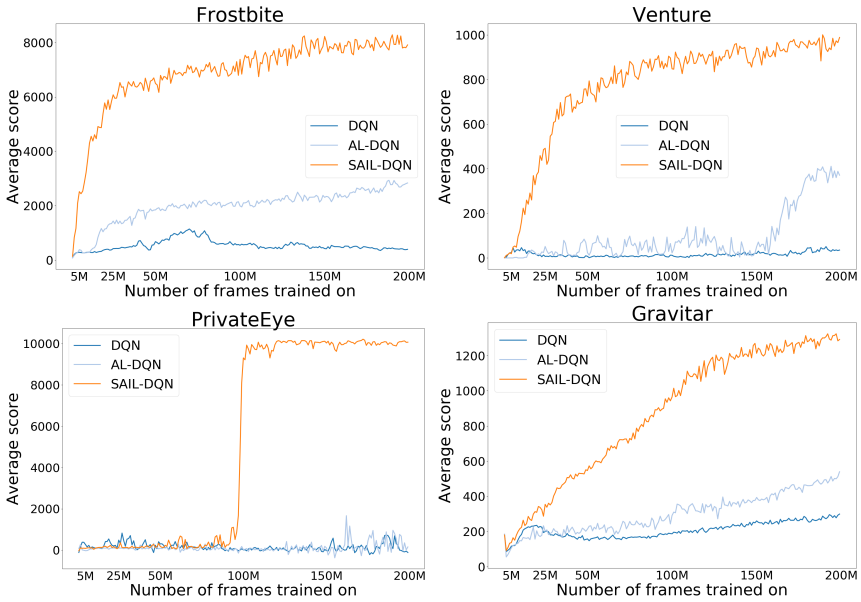


Figure 5.4: Performances of DQN, AL-DQN and SAIL-DQN on four hard exploration Atari games. On PrivateEye, despite sparse rewards, SAIL-DQN outperforms both C51 and IQN.

5.4.3 SAIL-IQN experiments

Since SAIL is a general self-imitation method for off-policy learning, it is straightforward to apply it to other off-policy agents. Hence, having demonstrated strong performance using DQN as a base agent in the previous subsection, we now turn towards a more advanced off-policy algorithm. We choose IQN, a strong distributional RL agent, and use the same IQN hyperparameters as Dabney et al. (2018), which in contrast to those of Dopamine do not use n-step bootstrapping. We report the performance of SAIL combined to IQN (abbreviated SAIL-IQN) against IQN on all Atari games in Fig. 5.5, using the same metric as in previous graphs. We also compare SAIL-IQN to AL-IQN and to Rainbow across all games in the same figure. Despite the fact that Rainbow uses prioritized experience replay and n-step bootstrapping (with $n = 3$), we report +80.3% average and +4% median relative performance increase for SAIL-IQN over Rainbow.

Additionally, we display individual game performances of IQN, AL-IQN and SAIL-IQN in Fig. 5.6. We choose two hard exploration games: Montezuma's Revenge and Venture; and also on an easy exploration game: Seaquest, illustrating the versatility of SAIL. We report the final performances on hard exploration games of all three methods in Table 5.2, and individual curves on all 59 games in Fig. 5.13.

Finally, we compare several of the algorithms mentioned using the human-normalized median scores in Fig. 5.2, aggregating scores from all games. SAIL-IQN outperforms Rainbow (itself outperforming IQN), without using prioritization or n-step bootstrapping. SAIL-DQN outperforms C51 (itself outperforming DQN).

5.4.4 Comparison to intrinsic motivation

We compare SAIL-IQN to IQN with RND (Burda et al., 2019), a popular intrinsic motivation method for exploration. While the two methods have different motivations, both modify the reward function and were shown to help in hard exploration tasks. RND uses two identical neural networks, the first of which

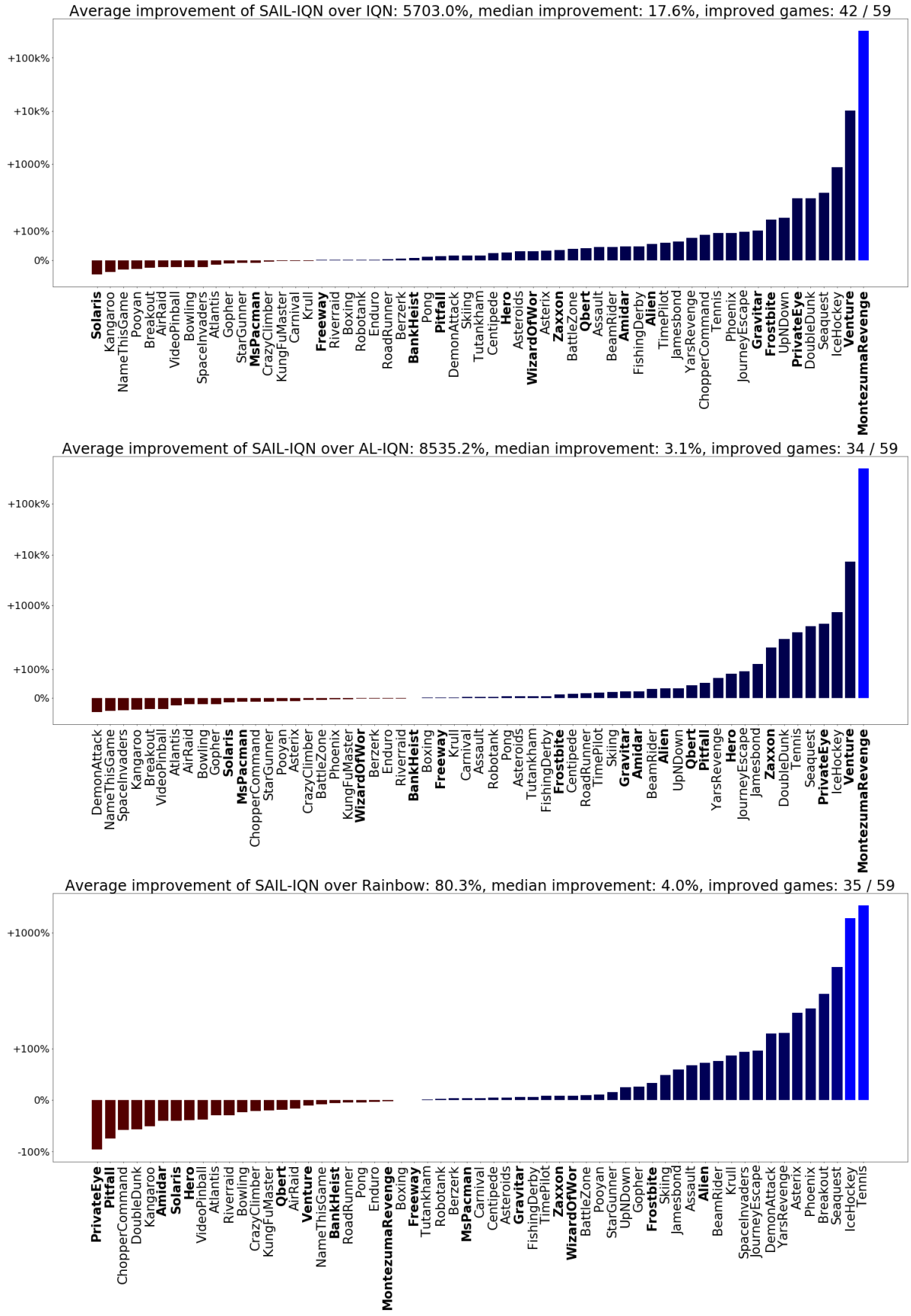


Figure 5.5: SAIL-IQN outperforms both IQN (top), AL-IQN (middle) and Rainbow (bottom), as shown on a relative scale on all Atari games. SAIL is responsible for important gains over IQN and AL-IQN on sparse reward hard exploration games such as Montezuma's Revenge, PrivateEye, or Gravitar.

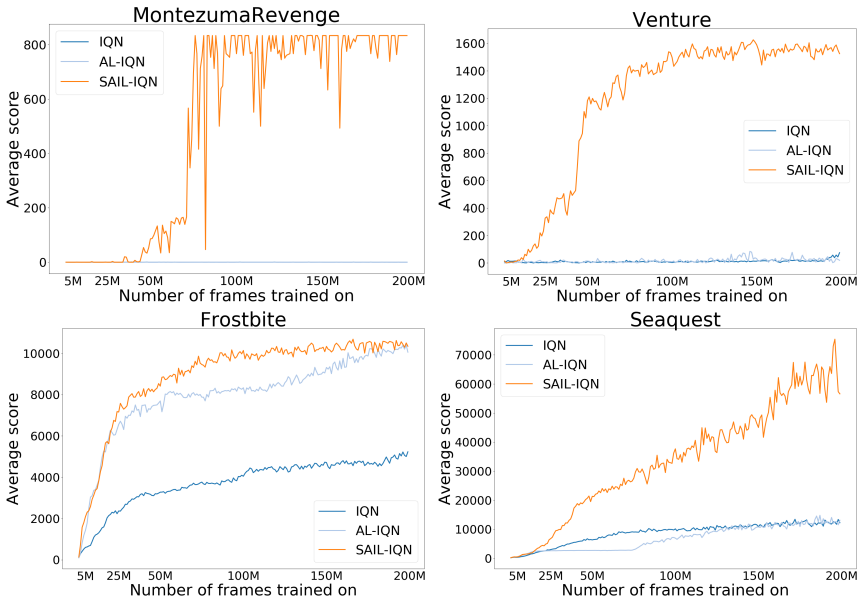


Figure 5.6: Performances of IQN, AL-IQN and SAIL-IQN on three hard exploration Atari games and one easy exploration game (Seaquest), illustrating the versatility of SAIL.

is frozen after initialization. The second network has to predict the output of this random network. The prediction error is then used as a proxy of actual state-visitation counts to reward the agent. For RND, we use code and hyperparameters from Taïga et al. (2019). RND hyperparameters were calibrated for Rainbow, thus to be fair we use the same RND hyperparameters to study the combination of SAIL and RND (see below), and keep equal SAIL hyperparameters across agents. SAIL-IQN shows positive average and median relative improvement scores compared to RND-IQN (+72.2% average and +28.2% median relative improvement over RND-IQN, see Fig. 5.7).

On Montezuma's Revenge, which is an infamously hard exploration game, SAIL-IQN reaches a final score of 833 (for an average score of 513), while RND-IQN reaches a final score of 161 (for an average score of 45), and IQN scores 0 (final and average). SAIL can be combined to RND for further improvements: using RND (as is) and n-step bootstrapping ($n = 3$), SAIL-IQN reaches a final score of 5180 on Montezuma's Revenge. This ties SAIL-IQN with RND-Rainbow, which has the strongest score reported in Taïga et al. (2019). We display the evolution of that score in Fig. 5.8.

While we established the merits of SAIL compared to RND in terms of in-game Atari performance, SAIL has several other advantages over RND. *1) Consistency:* while RND provides with better performance gains on specific games such as Montezuma's Revenge, we find that the gains are not consistent across all hard exploration games, matching the findings of Taïga et al. (2019). More precisely (not shown), using RND with IQN brought a great performance boost on two games (Montezuma's Revenge and Venture), but did not increase the median performance of IQN (-0.2% median relative improvement score). *2) Integration:* SAIL requires two simple modifications to off-policy algorithms, saving returns to the replay buffer and modifying the action-value update (1 hyperparameter). In comparison, RND requires two additional networks, separate optimizers and a reward modification (many hyperparameters). We acknowledge that this comparative benefit might be framework-dependent. *3) Compute:* on the same hardware, we find that standard IQN processes an average of 100 frames per second, against 95 for SAIL-IQN and 45 for RND-IQN, so completing experiments is more than twice faster when choosing SAIL over RND.

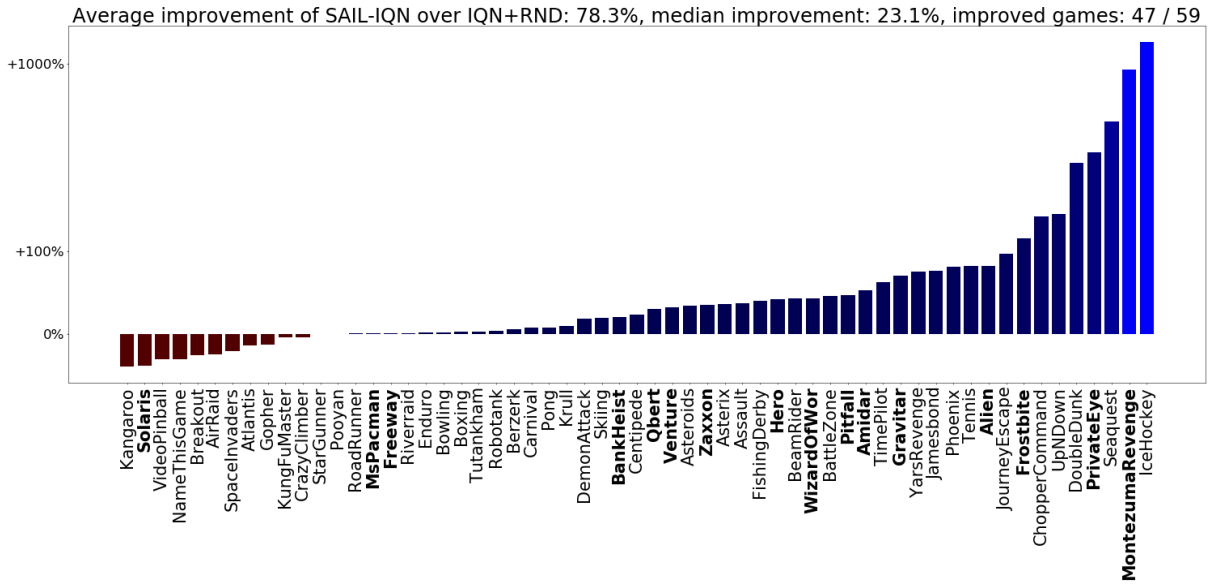


Figure 5.7: SAIL-IQN outperforms RND-IQN, as shown on a relative scale on all Atari games, including on Montezuma's Revenge, an infamously hard exploration game.

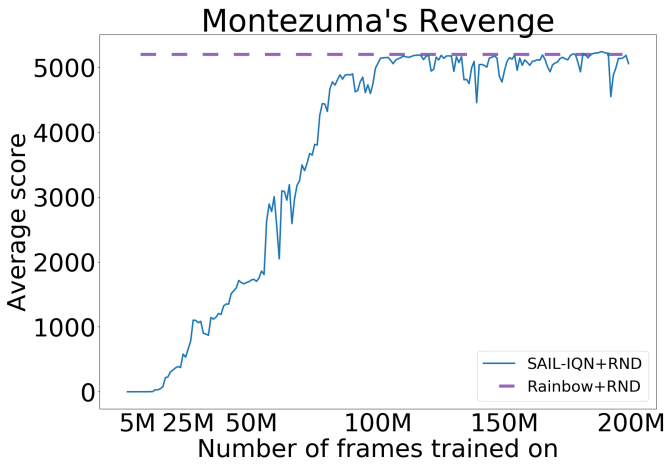


Figure 5.8: Combining SAIL and RND elevates the performance of IQN and matches that of RND-Rainbow, the best reported in Taïga et al. (2019) on Montezuma's Revenge.

Table 5.2: Final scores of several off-policy algorithms based on IQN, in hard exploration games.

	IQN	*AL-IQN*	*SAIL-IQN*
Alien	4262	5318	**6245**
Amidar	1131	1311	**1752**
BankHeist	1081	**1105**	1037
Freeway	34	34	34
Frostbite	5231	10058	**10345**
Gravitar	766	1213	**1375**
Hero	28636	16804	**30083**
Montezuma's Revenge	0	0	**833**
MsPacman	4656	**5340**	4669
Pitfall!	**-19**	-280	-42
PrivateEye	357	49	**4956**
Qbert	10802	11440	**15189**
Solaris	**2165**	1461	992
Venture	74	19	**1525**
WizardOfWor	5700	**10222**	8933
Zaxxon	12024	11843	**15296**

5.4.5 Impact of stochasticity

To quantify the impact of stochasticity on self-imitation, we perform an ablation and deactivate the sticky actions. Sticky actions (Bellemare et al., 2013) introduce stochasticity in otherwise near-deterministic Atari games, by repeating the previous action of the agent with a fixed repeat probability. As in the standard settings of ALE, by default we use a repeat probability of 0.25, that is here set to 0. We report results on all Atari games in Fig. 5.9, similarly to the precedent section.

As expected, SAIL brings a larger performance increase when we fall back to a near-deterministic setting but, interestingly, the results do not diverge much from the ones reported with the sticky actions (+231.6% versus +101.4% average, +12.7% versus +6.7% median).

We extend this experiment and instead increase the repeat probability, making environments more stochastic. We restrict ourselves to hard exploration games to limit the computational cost of the study. As Fig. 5.10 shows, the human-normalized median performance of SAIL-DQN decreases under higher action repeat probabilities. This phenomenon warrants further investigation, as we would have to tell apart the impact of stochasticity on SAIL from the general performance loss due to making the RL problem more difficult, which we leave for future work. Note that the performance of SAIL stays superior to that of DQN in the standard setting.

5.4.6 Comparison to straightforward SIL

We compare SAIL to the more straightforward self-imitation algorithm depicted equ. 5.3, the alternate version of SAIL where we use the standard return alone, and a ReLU operator instead of the identity. We display average and median relative improvement scores with both DQN and IQN in Fig. 5.11.

5.5 Additional related work

Extending self-imitation learning Guo et al. (2018) revisit GAIL (Ho and Ermon, 2016), an adversarial imitation method that encourages the agent to trick a discriminator into taking its behavior for

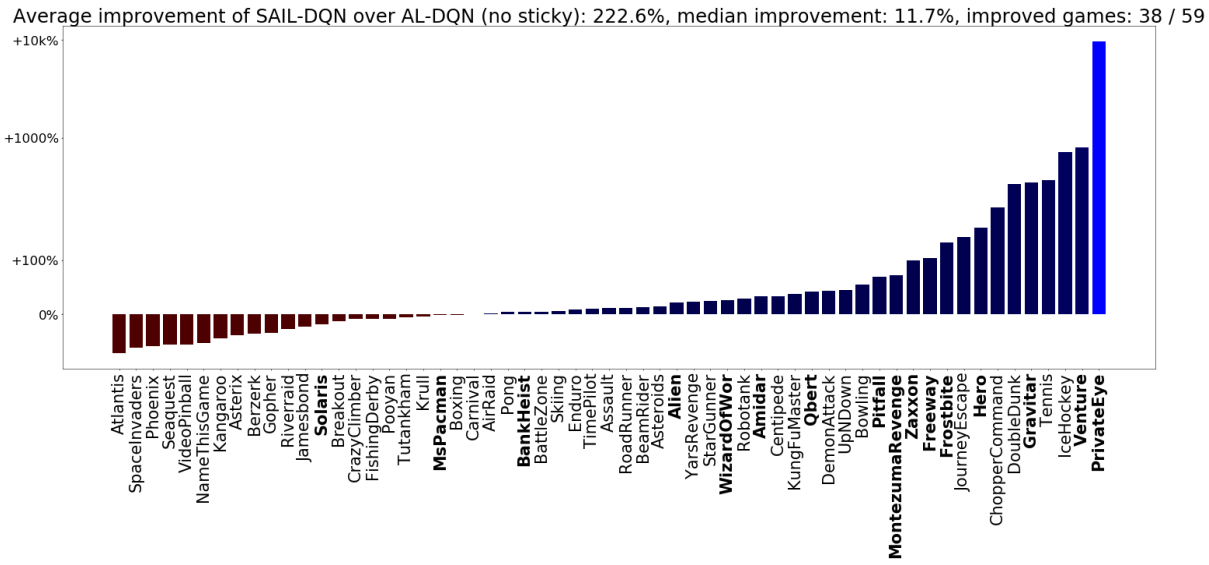


Figure 5.9: Improvement of SAIL-DQN over AL-DQN on all Atari games, without sticky actions. Result do not diverge from those in the standard ALE setting (+222.6% versus +101.4% average, +11.7% versus +6.7% median).

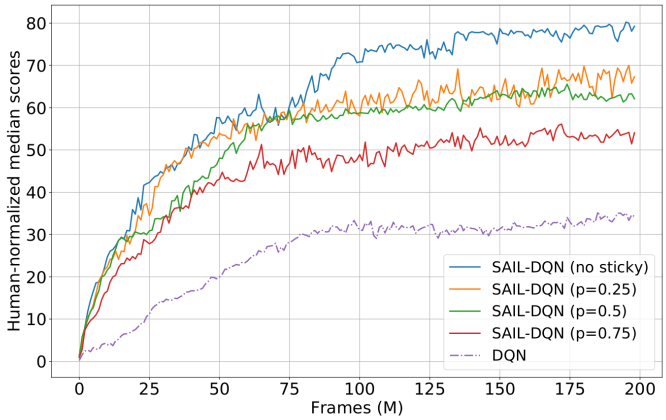


Figure 5.10: Even under increased stochasticity, SAIL-DQN outperforms DQN in the standard setting ($p = 0.25$).

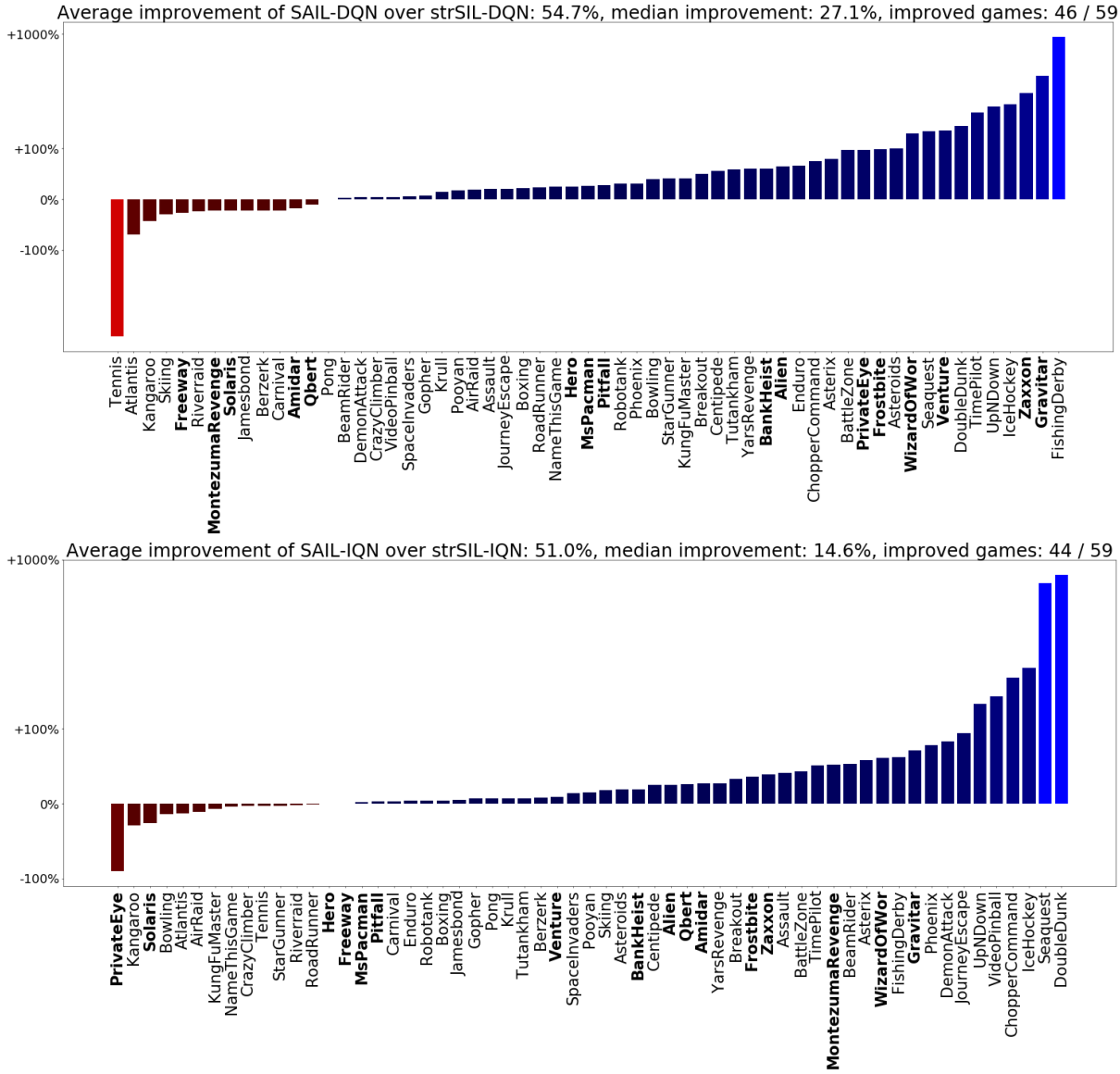


Figure 5.11: Relative improvement of SAIL over self-imitation (strSIL), based on both DQN (top) and IQN (bottom). On hard exploration games (in bold), SAIL-DQN provides a +72.2% average and +45.2% median relative improvement over strSIL-DQN, and SAIL-IQN a +16.0% average and +22.3% median relative improvement over strSIL-IQN.

expert behavior. Their method uses the agent's past behavior as expert behavior, identifying promising behavior in a similar way to standard self-imitation. Guo et al. (2019) use self-imitation over a diverse set of trajectories from the agent's past experience, showing that it helps on games where there are local minima that hinder learning. Tang (2020) studies the impact of importance sampling corrections and using n-step bootstrapping to replace the observed return in a generalized form of self-imitation, which is studied under the operator view. None of these methods explicitly target off-policy learning algorithms, and as such none have straightforward extensions to that setting.

RL for hard exploration tasks Being able to solve hard exploration problems is an important target for RL. So far, quite different strategies have been developed to tackle these problems. Intrinsic motivation methods provide an additional source of reward to the agent, either for going to seldom visited areas of the state space (Bellemare et al., 2016a; Tang et al., 2017), or for experiencing novel things, that challenge its own predictions (Pathak et al., 2017; Burda et al., 2019; Savinov et al., 2019). Aytar et al. (2018) use expert demonstrations from human player-made videos as imitation data and incentivize close-to-expert progression. Ecoffet et al. (2019) propose an exhaustive exploration method that encourages the agent to visit promising areas of the state space while jointly learning how to get back to those areas. Badia et al. (2020b) combine intrinsic motivation with an episodic motivation based on episodic memory. Paine et al. (2019) uses expert demonstrations combined with a recurrent learner to solve partially observable hard exploration tasks. Badia et al. (2020a) uses a neural network to encapsulate multiple policies with different degrees of exploration and switches between policies using a multi-arm bandit algorithm. While SAIL proves to be useful for hard exploration, its main motivation is to learn properly from the agent's own instructive experiences. It is not an exploration method per se, but could generally be combined to said exploration methods for further improvements, as we illustrated with RND.

Sparse reward RL There is a lot of ongoing effort regarding RL for sparse rewards as well, which is related to hard exploration and overlaps on many environments. In the goal-oriented setting, Andrychowicz et al. (2017) propose to modify the replay buffer in hindsight and change the desired goal state by a state the agent actually visited, providing it with free reward signal. Lee et al. (2018) accelerate the propagation of sparse reward signals by sampling whole episodes from the replay buffer and performing updates starting from the end of the sampled episode. In a related way, Hansen et al. (2018) combine standard off-policy with episodic control, estimating the action-value as the convex combination of the action-value learned off-policy and another estimate from episodic memory. Trott et al. (2019) use reward shaping to explore while avoiding local minima. Since the additions of SAIL to off-policy agents are purely restricted to the action-value updates, SAIL and sparse reward RL techniques could synergize well, which we leave for future experiments.

Misc He et al. (2016) show that there are tractable bounds to the optimal action-value function and use them to increase the Q-function when it is inferior to the lower bound (resp. decrease when superior to the upper bound).

5.6 Conclusion

In this chapter, we presented SAIL, a novel self-imitation method for off-policy learning, which leverages an implicit form of credit assignment that directly incorporates outcomes (returns) in RL updates. In comparison with standard self-imitation, SAIL revolves around the action-value function, which makes it compatible with many off-policy algorithms. SAIL is a simple, general and lightweight method that can be equivalently viewed as a modified Bellman optimality operator (that is related to Advantage Learning) or as a modified reward function. Notably, it combines the advantages of self-imitation, that

is better reinforcement in sparse reward scenarios, and those of Advantage Learning, that is increasing the action-gap. We studied its performance in the Arcade Learning Environment and demonstrated that SAIL brought consistent gains, especially on hard exploration games, at virtually no cost.

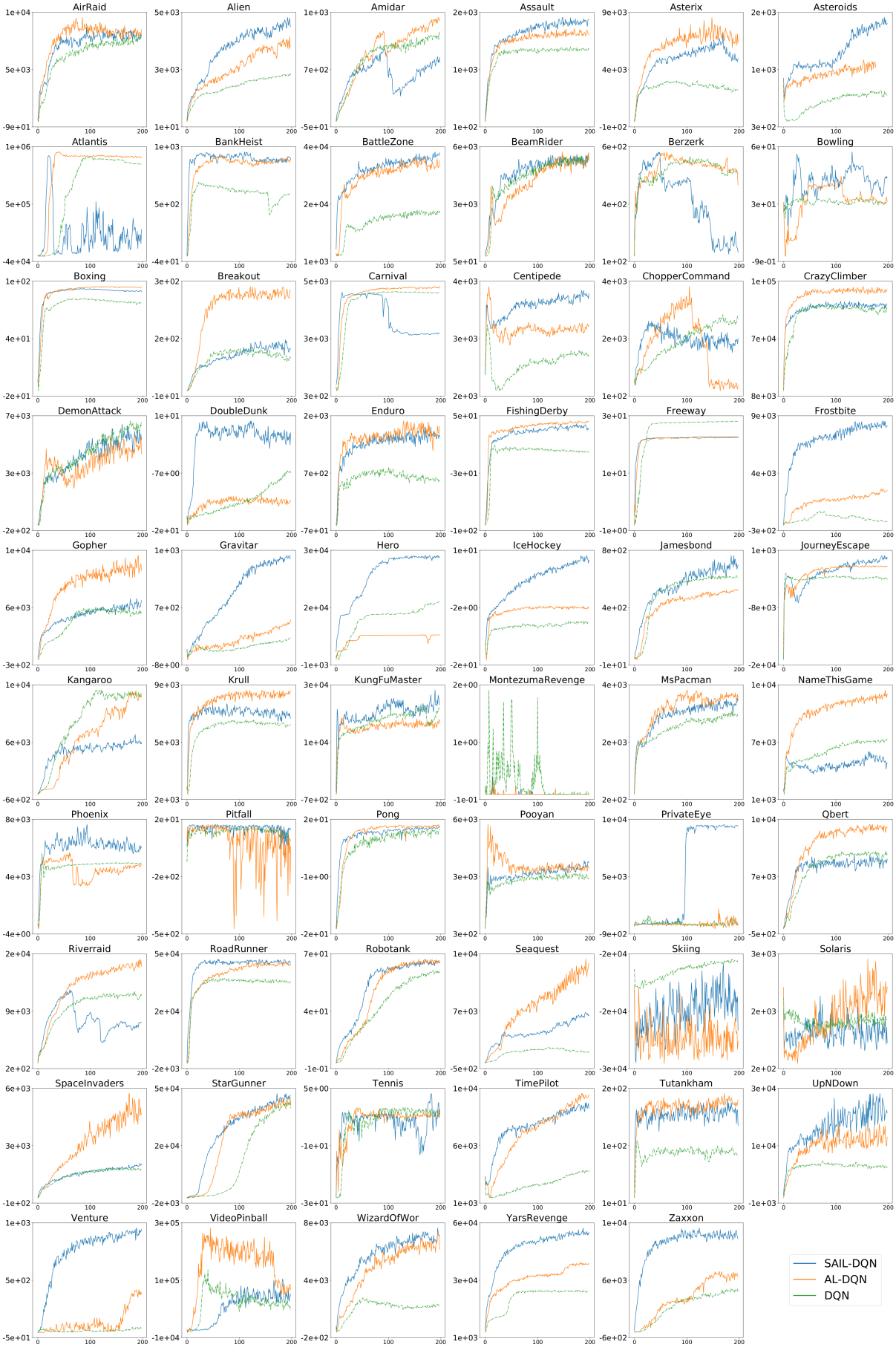


Figure 5.12: Curves on all games for DQN-based methods.

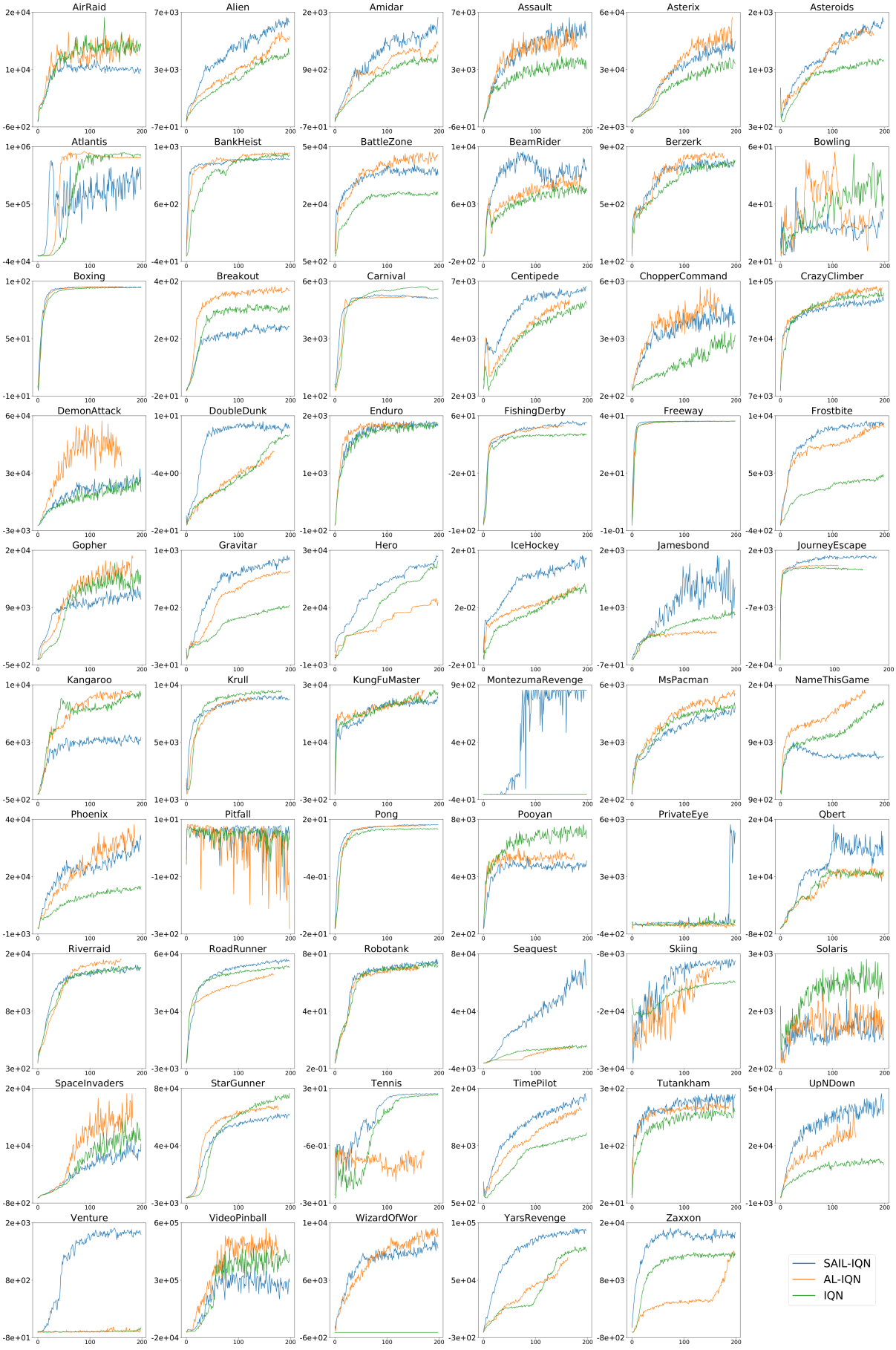


Figure 5.13: Curves on all games for IQN-based methods.

Chapter 6

Grounded Credit Assignment as a Proxy for Transfer

We now depart from implicit credit assignment and take an interest in explicit credit assignment. Explicit credit assignment methods are conditioned on actual outcomes (either observed or generated) and correspond to more targeted forms of action importance. In this chapter, we study a grounded form of credit assignment that is learned offline and transfers well across environments from similar distributions. The ability to transfer knowledge to novel environments and tasks is a sensible desiderata for general learning agents. Despite the apparent promises, transfer in RL is still an open and little exploited research area. Here, we take a brand-new perspective about transfer: we suggest that the ability to assign credit unveils structural invariants in the tasks that can be transferred to make RL more sample efficient. Our main contribution is SECRET, a novel approach to transfer learning for RL that uses a backward-view credit assignment mechanism based on a self-attentive architecture. Two aspects are key to its generality: it learns to assign credit as a separate offline supervised process and exclusively modifies the reward function. Consequently, it can be supplemented by transfer methods that do not modify the reward function and it can be plugged on top of any RL algorithm. This chapter was presented as a full conference paper at IJCAI 2020 (Ferret et al., 2020) and as an oral to the Learning Transferable Skills workshop at NeurIPS 2019.

6.1 Introduction

To some, intelligence is measured as the capability of transferring knowledge to unprecedented situations. While the notion of intellect itself is hard to define, the ability to reuse learned information is a desirable trait for learning agents. The coffee test (Goertzel et al., 2012), presented as a way to assess general intelligence, suggests the task of making coffee in a completely unfamiliar kitchen. It requires a combination of advanced features (planning, control and exploration) that would make the task very difficult if not out of scope for the current state-of-the-art Reinforcement Learning (RL) agents to learn. On the other hand, it is solved trivially by humans, who exploit the universally invariant structure of coffee-making: one needs to fetch a mug, find coffee, power the coffee machine, add water and launch the brewing process by pushing the adequate buttons. Thus, to solve the coffee test, transfer learning appears necessary. Were we to possess a random kitchen simulator and a lot of compute, current transfer methods would still fall short of consistently reusing structural information about the task, hence also falling short of efficient adaptation.

Credit assignment, which in RL refers to measuring the individual contribution of actions to future

rewards, is by definition about understanding the structure of the task. By structure, we mean the relations between elements of the states, actions and environment rewards. In this work, we investigate what credit assignment can bring to transfer. Encouraged by recent successes in transfer based on supervised methods, we propose to learn to assign credit through a separate supervised problem and transfer credit assignment capabilities to new environments. By doing so, we aim at recycling structural information about the underlying task.

To this end, we introduce SECRET (SElf-attentional CREdit assignment for Transfer), a transferable credit assignment mechanism consisting of a self-attentive sequence-to-sequence model whose role is to reconstruct the sequence of rewards from a trajectory of agent-environment interactions. It assigns credit for future reward proportionally to the magnitude of attention paid to past state-action pairs. SECRET can be used to incorporate structural knowledge in the reward function without modifying optimal behavior, as we show in various generalization and transfer scenarios that preserve the structure of the task.

Existing backward-view credit assignment methods (Arjona-Medina et al., 2019; Hung et al., 2019) require to add auxiliary terms to the loss function used to train agents, which can have detrimental effects to the learning process (de Bruin et al., 2018), and rely on an external memory, which hinder the generality of their approach. SECRET does neither. Also, as we show in Chapter 6.3.1, the architecture we consider for SECRET has interesting properties for credit assignment. We elaborate about our novelty with respect to prior work in Chapter 6.4. We insist on the fact that the focus of our work is on transfer and that it is not our point to compete on credit assignment capabilities.

We would like to emphasize several aspects about the generality of SECRET: 1) our method does not require any modification to the RL algorithm used to solve the tasks considered, 2) it does not require any modification to the agent architecture either and 3) it does not alter the set of optimal policies we wish to attain. Moreover, our method for credit assignment is offline, and as a result, it can use interaction data collected by any mean (expert demonstrations, replay memories (Lin, 1992), backup agent trajectories...). We believe that this feature is of importance for real-world use cases where a high number of online interactions is unrealistic but datasets of interactions exist as a byproduct of experimentation.

Additional background We refer the reader to the overall background in Chapter 3.2. A trajectory $\tau = (s_i, a_i, r_i)_{i=1,\dots,T}$ is a set of state-action pairs and resulting rewards accumulated in an episode. A subtrajectory is a portion of trajectory that starts at the beginning of the episode. The performance of an agent is evaluated by its expected discounted cumulative reward $\mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t\big]$. We recall that in a partially observable MDP (POMDP), the agent receives at each timestep t an observation $o_t \sim \mathcal{O}(\cdot|s_t)$ that contains partial information about the underlying state of the environment.

6.2 SECRET: self-attentional credit assignment for transfer

SECRET uses previously collected trajectories from environments in a source distribution. A self-attentive sequence model is trained to predict the final reward in subtrajectories from the sequence of observation-action pairs. The distribution of attention weights from correctly predicted nonzero rewards is viewed as credit assignment. In target environments, the model gets applied to a small set of trajectories. We use the credit assigned to build a denser and more informative reward function that reflects the structure of the (PO)MDP. The case where the target distribution is identical to the source distribution (in which we use held-out environments to assess transfer) will be referred to as generalization or *in-domain transfer*, as opposed to *out-of-domain transfer* where the source and the target distributions differ.

6.2.1 Self-attentional credit assignment

Credit assignment as offline reward prediction We learn to assign credit through an offline reward prediction task, based on saved trajectories of agent-environment interactions. We create a sequence-to-

sequence (seq2seq) model (Sutskever et al., 2014) that takes as input the sequence of observation-action pairs and has to reconstruct the corresponding sequence of environment rewards. Being offline, the reward prediction task is learned separately from the RL task, and the reward prediction model does not share representations with the agent. This way, the representations learned for credit assignment do not affect or get mixed with the representations learned for control. Operating offline brings several advantages: one can directly interact with the replay memory of agents and even use expert demonstrations or arbitrary saved transitions as a source of supervision, which could be useful in settings where on-policy interactions are costly, such as robotics. We equip our seq2seq model with an attention mechanism (Bahdanau et al., 2015) and view the attention weights of the reward reconstruction task as our primary source of assigned credit. The motivation to do so is that the seq2seq model looks into the past to find predictive signal in order to reconstruct the reward, so observation-action pairs it attends to should be those which reduce its uncertainty about the future, in other words those that explain future reward and should be credited.

On the use of observations In MDPs, environment states follow the Markov property: they summarize the history of previous interactions and are sufficient to predict the future. As such, predictive models are highly biased towards focusing on the current sequence element, which hinders credit assignment. Under that consideration, when dealing with MDPs, we turn states into observations by applying transformations that hide a certain amount of information from states and break the Markov assumption. For instance, in gridworlds with visual states, we crop the image and get a player centered image with a given window size. Doing so encourages the model to look into the past to find predictive signal, and allow us to track the relative importance given to each element to reconstruct the credit assigned. In POMDPs, this might be unnecessary depending on the amount of information shared between observations and true states.

Self-attention for credit assignment Unlike other seq2seq architectures, self-attentive models like Transformers (Vaswani et al., 2017) have direct computational paths between pairs of sequence elements, due to their representations that depend on projections of all sequence elements. This feature is key to long-term credit assignment. As an illustration, consider an RL task where the terminal reward depends only on the first observation, which is drawn randomly. Predicting the reward correctly requires to remember the first observation, which would be very challenging for a recurrent architecture whose memory goes through $O(n)$ transformations, n being the size of the sequence. On the other hand, a self-attentive model directly accesses the value of the initial observation, which makes credit assignment easier.

Reward prediction architecture We use a Transformer decoder with a single self-attention layer (Lin et al., 2017) and a single attention head. The model input is a sequence of observation-action couples $(o_t, a_t)_{t=0,\dots,T}$. Each observation goes through a series of convolutional layers (for visual inputs) followed by a series of feed-forward layers. Each action representation, a one-hot vector in the discrete action case, is concatenated to the learned observation embedding. Those representations of dimensionality d_i are combined with positional encoding (PE), fed to a self-attention layer and then to a position-wise feedforward layer that outputs logits for reward prediction classes. PE encodes the relative positions of sequence elements, see Appendix A for details.

Self-attention is an attention mechanism with parameterization (W_k, W_q, W_v), each matrix belonging to $\mathbb{R}^{d_i \times d_k}$, that puts sequence elements in relation by computing non-linear similarity scores for all pairs of elements in the sequence. To do so, each sequence element is mapped to a query vector that is matched against keys and values obtained from the previous elements. To be consistent with the goal of assigning credit, the model should not be able to peek into the future. Thus, we restrict the computational window of each sequence element to the information stored in representations of the previous elements in the sequence and its own by applying a causal mask M_c to the result of the pairwise similarity computations, assigning a value of 0 to masked elements after the softmax.

Let $X = (x_t)_{t=0,\dots,T} \in \mathbb{R}^{T \times d_i}$ denote the input sequence in a matrix form, x_t being the result of internal computations of the model on its t^{th} input. In the same fashion, we note $Z = (z_t)_{t=0,\dots,T} \in \mathbb{R}^{T \times d_k}$ the sequence resulting from the application of self-attention. We then have

$$Z = \text{softmax}\left(\frac{M_c \odot (QK^T) - C(1 - M_c)}{\sqrt{d_k}}\right) V,$$

where $Q = XW_q \in \mathbb{R}^{T \times d_k}$ stores queries, $K = XW_k \in \mathbb{R}^{T \times d_k}$ keys, and $V = XW_v \in \mathbb{R}^{T \times d_k}$ values as linear projections of the input; d_k stands for the dimension of the key vectors, $M_c \in \{0,1\}^{T \times T}$ is a binary matrix that acts as a causal mask (a lower triangular matrix), $\odot$ is the Hadamard product and C is a large constant (10^9 in practice).

Notably, the resulting observation-action representation can be viewed as a linear combination of the values of previous elements: $z_t = \sum_{i=0}^{t} \alpha_{i \leftarrow t} v_i$ where $\alpha_{\cdot \leftarrow t} = (\alpha_{i \leftarrow t})_{i=1,\dots,t} \propto \exp(\langle q_t, k_i \rangle / \sqrt{d_k})$. The vector α_t contains the normalized attention weights for the prediction at timestep t and sums to 1. Since observations contain only a portion of their initial information, the fact that the model succeeds in the prediction task indicates that it reconstructed the missing information from its past. Therefore, attention weights themselves can be viewed as a form of credit assignment, and will be used as such in what follows.

While performing regression on the rewards could also be an option, our experiments found that regression tends to converge to poor local optima. Consequently, we predict the sign of the experienced rewards: $q(r) = \text{sign}(r)$ with $\text{sign}(0) = 0$. We chose the sign as the classification target for its invariance to the scale of the rewards. We use a weighted sequential cross-entropy as the loss function over the class-wise model predictions $f_{\theta,c}$, writing $\tau(o,a)$ the subtrajectory of τ ending with the observation-action couple (o,a) to translate the effect of the binary mask:

$$\mathcal{L}_\theta(\tau) = -\sum_{c \in \{-1,0,1\}} \frac{w(c)}{|\tau|} \sum_{(o,a,r) \in \tau} \mathbb{1}\{q(r) = c\} \log\left(f_{\theta,c}(\tau(o,a))\right).$$

We have found class weighting to be very important to keep the variance of prediction metrics across a variety of datasets of sampled trajectories low.

Generating trajectories To train SECRET, we generate a dataset of trajectories which contains a certain proportion of successful trajectories. If source environments are simple enough so that the task has sufficient chance to be solved by acting randomly, we use a random policy to generate trajectories. For more complex distributions of environments, we use a RL agent (either trained or in the learning phase) to generate trajectories. We think purely exploratory methods like Ecoffet et al. (2019) could have advantages over using an RL agent and leave the study of their use for future work.

6.2.2 Leveraging credit via reward shaping

In this subsection, we explain how we use credit assignment to make learning more sample-efficient.

Reward shaping In RL, agents often deal with sparse rewards that make the learning process slow. Reward shaping Ng et al. (1999) is a technique that aims at densifying the reward so as to improve sample efficiency. It defines a class of reward functions that can be added to the original environment rewards without modifying the set of optimal policies. For a given MDP $M = (S, A, \gamma, R, P)$, we define a new MDP $M' = (S, A, \gamma, R', P)$ where $R' = R + F$ is the shaped reward and F the shaping. The reward shaping theorem states that if there exists a function ϕ such that $F : (s, a, s') \to \gamma\phi(s') - \phi(s)$, then M and M' admit the same set of optimal policies. ϕ is called a potential function. With domain knowledge, one can use reward shaping to design more informative reward functions without encouraging unwanted

behavior. Nevertheless, shaping rewards requires good priors for the task and the potential function must often be engineered manually.

Since SECRET weighs the contribution of observation-action pairs to future reward, we use it to derive a shaped reward that corresponds to the sum of future reward reachable from the underlying state, weighted by the attention calculated by the model. We explain the process in the following.

Computing the potential function We define the redistributed return $R_\tau^{\leftarrow}$ of a trajectory τ as:

$$R_\tau^{\leftarrow}(s,a) = \sum_{t=1}^{T} \mathbb{I}\{s_t = s, a_t = a\} \sum_{i=t}^{T} \alpha_{t \leftarrow i} r(s_i, a_i),$$

where $\alpha_{i \leftarrow j}$ is the attention weight on (o_i, a_i) when predicting the reward r_j and s_i are environment states. Indeed, SECRET uses observations but we keep the states they are constructed from to compute the potential. In POMDPs, we recover an approximate state from the observation, either manually or through inference. In this work, we use a state constructed manually, see Appendix A for details.

To compute the potential function, we generate a set D of trajectories like described in Chapter 6.2.1. Since we operate on trajectories, the same state-action pair can appear twice in a sequence and benefit from a different amount of attention, which is why we must include the first summation. In the reward shaping formalism, the potential function ϕ depends only on the state. To stay within its bounds, we define ϕ as the forwarded redistributed return. It is computed as the following estimate:

$$\hat{\phi}(s) = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=1}^{T} \mathbb{I}\{s_t^{(\tau)} = s\} R_\tau^{\leftarrow}(s_{t-1}^{(\tau)}, a_{t-1}^{(\tau)}).$$

Note that in practice we only redistribute individual rewards that were successfully predicted. Also, some states are generally missing from the data distribution induced by the set of trajectories used. For those states, we set to potential to 0, which results in a $-\hat{\phi}(s)$ additional reward when transitioning to those from the state s. As a result, it gives agents incentive to stay on the support of the data distribution unless they encounter high-reward states.

Because it relies on reward shaping, SECRET conserves optimal policies. We empirically find that agents learn faster with the resulting augmented reward function. A way to look at it is that we densify the learning signal and bias the agent towards behaviors that encourage future rewards.

6.2.3 Transferring credit assignment

We start by conveying intuition as to why SECRET should transfer to new environments. In fields other than RL, seq2seq models similar to that of SECRET have shown outstanding transfer capabilities (Devlin et al., 2018; Peters et al., 2018; Howard and Ruder, 2018), even in low-resource settings (Zoph et al., 2016). In transfer scenarios that preserve the structure of the MDP, the optimal fine-grained control sequence can vary drastically from one environment to another. This is why credit assignment is an interesting alternative to the transfer of weights: given an underlying environment state and a specific action, their contribution to future rewards is not fundamentally altered. Such scenarios include specific changes in the state (or observation) distribution and changes to the reward function that preserve the optimal policies. These also include changes in the dynamics of the environment, and though it affects credit assignment, we show later on that SECRET adapts surprisingly well to such scenarios. Another point that motivates the use of our method for transfer is the fact that we keep the representations learned for credit assignment separate from the control representations learned by agents. Indeed, de Bruin et al. (2018) showed that RL representations were not optimal for transfer.

Transfer setting We argue that transfer should be considered effective when agents learn to solve target tasks efficiently because efficiency gains in the target domain compound while the cost of training in the source is fixed. Hence, we use the Total Target Time Scenario metric (Taylor and Stone, 2009) to assess transfer. Nevertheless, collecting trajectories in the source domain can be costly. We report the number of trajectories used to train SECRET in each scenario.

As before, SECRET is trained on episodes of interaction sampled from the source distribution. In each target environment, we sample multiple trajectories (see the following section for details about the policies used to generate the trajectories). We then compute the attentional potential function by calculating an estimate of the expected redistributed reward, as described in Chapter 6.2.2.

6.3 Experiments

In this section, we aim to answer the following questions: can SECRET improve the sample efficiency of learning for RL agents? Does it generalize and/or transfer? How does it compare to transfer baselines? Is the credit assigned by SECRET interpretable? The results of complementary experiments are presented in Appendix C.

The Triggers environment We introduce Triggers, an interpretable and customizable environment that we use to assess the quality of the credit inferred with our method. In Triggers, the agent is located in a two-dimensional bounded grid. Its actions consist solely of moving of one cell in one of the cardinal directions. Any action that would lead the agent outside the boundaries of the environment (as indicated by the walls in the figure) is ignored but still counted as an action taken by the agent. The goal of the agent (represented as a yellow square) is to activate all the switches (red squares) and then collect all the prizes (pink squares). Prizes are the only source of reward and give a -1 penalty unless all switches are activated, in which case they give a $+1$ bonus. Both prizes and switches disappear once collected. The main feature of Triggers is that every positive reward is conditional to the presence of a known subset of states in the agent history, and thus credit assignment can be assessed in a *rigorous* way. Some instances of Triggers can prove challenging to solve optimally for traditional RL methods since agents have to activate every Triggers before experiencing rewards. Triggers environments being MDPs, we turn their states into observations by cropping the view around the agent. We use 3x3 windows in all our experiments. Trajectories are generated with random policies.

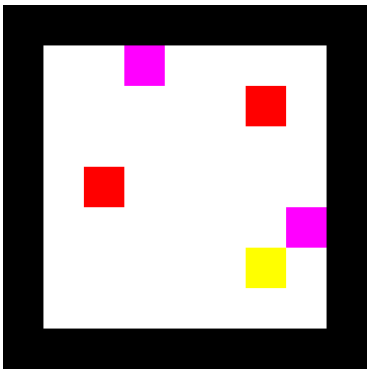


Figure 6.1: Example of a Triggers environment.

DMLab keys doors We use the `keys_doors_puzzle` 3D environment from DMLab (Beattie et al. (2016)) in which the agent must locate keys whose colors indicate the doors they open. It can only possess one key, therefore picking the wrong key prevents it from reaching further rewards. The agent receives as input what would correspond to a first person view of what is in its line of sight. It can move forward, backward and rotate. Each key picked up grants a $+1$ bonus, equally to each door opened. Independently, a cake rewards the agent by a $+50$ increase in score when collected. We do not apply any transformation to the

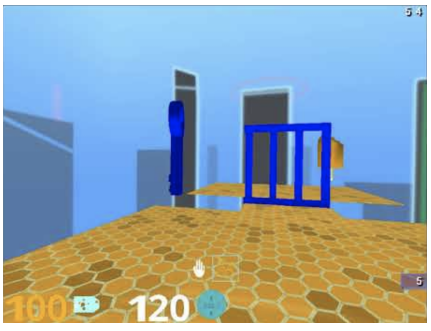


Figure 6.2: Observations from DMLab are first person views.

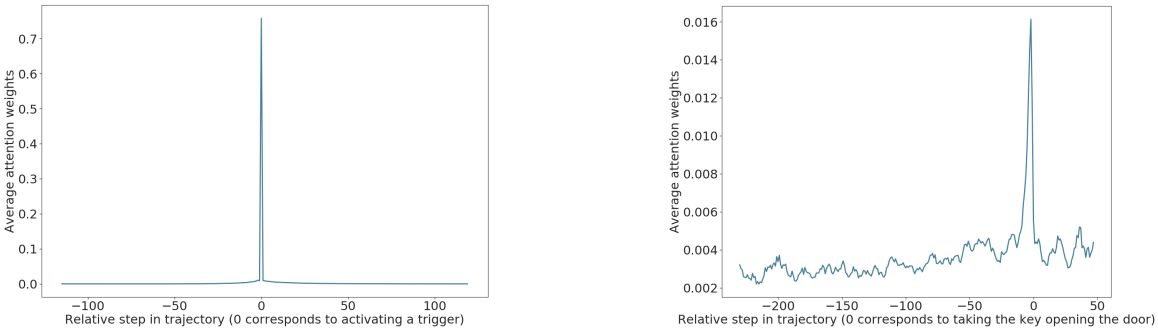


Figure 6.3: On the **left**, the distribution of attention weights around triggers for correct positive reward predictions in a 8x8 Triggers maze with 3 triggers and one reward. The x-axis denotes the signed number of steps between the state-action couple receiving attention and the closest actual moment the agent activated a switch. On the **right**, the distribution of attention weights around keys for correct reward predictions for door traversals in DMLab.

agent inputs. In that setup, agents benefit from understanding the link between keys and doors. We hypothesized that our credit assignment mechanism might identify this relation and reward the agent for picking up keys. To assert this, we modified the setting so that picking up keys does not provide rewards. Additionally, the visual input is richer than the one from Triggers environments and the average number of steps per episode is extended. Finally, agents move and rotate across the room. Since picking up a key does not require to actually see the key, it can be hard to know whether a key was taken and predict further door opening rewards. Trajectories are generated with a trained agent.

Agents used We use tabular Q-learning (Watkins and Dayan, 1992) for in-domain and out-of-domain experiments in Triggers except for the transfer to environments with modified dynamics where we use Deep Q-Networks (DQN) (Mnih et al., 2013). We use Proximal Policy Optimization (PPO) (Schulman et al., 2017) agents for in-domain experiments in DMLab.

6.3.1 Credit assignment

We provide an analysis of the credit inferred by SECRET. The analysis is qualitative and quantitative, since we rely on both visual assessment and binary detection metrics.

The process of evaluating credit assignment in Triggers goes as follows: we first generate trajectories and train the model. We then compare the credit assigned by SECRET on trajectories sampled from held-out environments to a ground truth credit assignment. We build that ground truth by exploiting the exact knowledge of where triggers are. It is a vector that is 0 almost everywhere and 1 on state-action couples that precede the activation of a Triggers. By doing so, we explicitly target the state-action couples whose resulting state is causally linked to the reward experienced later.

We find the redistribution to be near optimal in simple instances of Triggers (see Fig. 6.3-left): attention concentrates quasi exclusively on state-action pairs that enable the collection of future reward. This is confirmed by precision-recall analysis: over the distribution of scenarios considered, a simple binarization heuristic over attention values yields an average precision of 0.96 for an average recall of 0.94. More information on the heuristic is in Appendix A.

In `keys_doors_puzzle`, we adopt the same set of experiments. Since the agent can move backward and spin, in some scenarios it takes a key that is not in is line of sight. In addition, the granularity of the

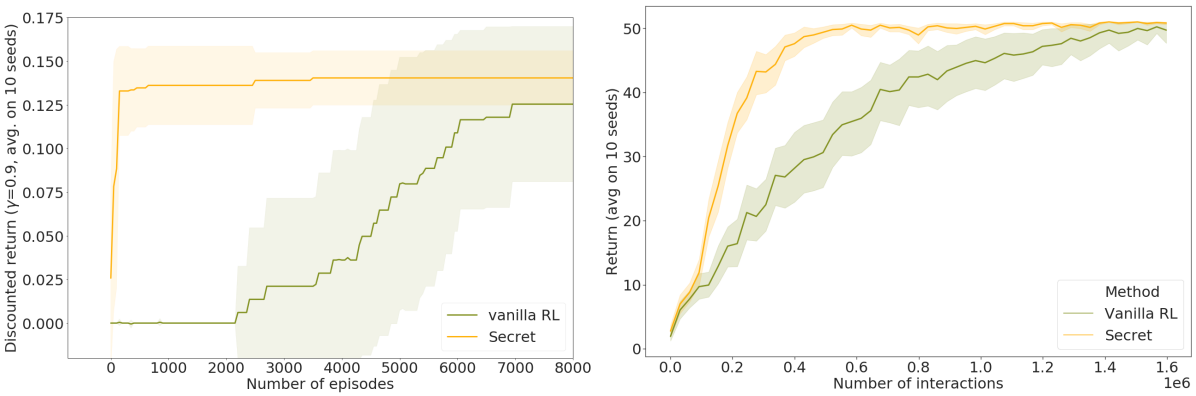


Figure 6.4: **Left:** in-domain transfer results on a 8x8 Triggers with 3 triggers and 1 reward. **Right:** results in DMLab.

state space is such that off-by-one prediction errors are common but do not hinder the credit mechanism: attributing credit to the state-action couple preceding the collection of a key or the previous one leads to imperceptible changes in the resulting shaped rewards. Fig. 6.3-right shows similar results as for Triggers. Appendix A also provides a heatmap for this task that shows that attention concentrates around the keys.

6.3.2 Transfer

We then study how we can leverage the inferred credit and transfer representations that are helpful in new scenarios. We show that agents train faster when using shaped rewards from SECRET. As before, the reward model is trained on episodes of interaction in environments sampled from the source distribution. In transfer environments, we sample multiple trajectories, each using the same maze configuration. We then compute the attentional potential function by calculating an estimate of the expected redistributed reward, as described in Chapter 6.2.2. To evaluate its effect, we compare agents trained from environment rewards to agents that use the resulting shaped reward.

In-domain transfer For in-domain transfer, we transfer the representations for credit assignment to held-out instances of the same distribution over MDPs. For the Triggers environment, the RL agents are tabular Q-learners. For the DMLab environment, we use PPO agents (Schulman et al., 2017) and modify the original task: we do not reward the agent for collecting keys but only to open doors so that the attention can focus on the key positions. Note that this makes the task harder.

As we display in Fig. 6.4 agents learn visibly faster to solve tasks when benefitting from SECRET in both environments.

Out-of-domain transfer For out-of-domain transfer we use the Triggers environment and consider two scenarios that are hard for standard agents: transfer to bigger environments (see Fig. 6.5) and transfer to environments with inverted dynamics (see Fig. 6.6). In the bigger setting, direct weight transfer cannot be used since the visual input has bigger spatial dimensions. On the other hand, SECRET can be used since the transformation we apply to turn states into observations conserves the visual input dimensions. In the inverted dynamics setting, the effect of the agent's actions are inverted, which makes the task hard for transfer methods. In that setting, we compare the transferability of our mechanism to that of the representations learned by an agent equipped with deep function approximation. To this end we use DQN

agents and either train them from scratch in the target environments or start from the set of weights learned in the source environments (WT in Fig. 6.6).

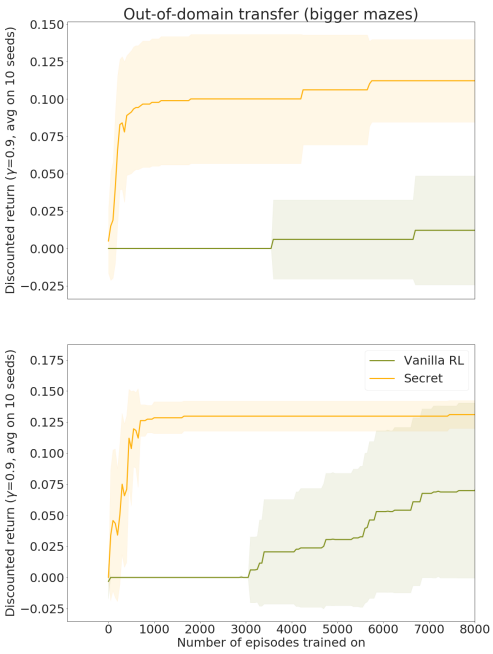


Figure 6.5: Bigger environments we consider are bigger mazes where the structure of the original task is conserved (number of triggers, number of prizes). Environments drawn are 12x12 grids with 1 trigger and 1 prize (**top** figure) versus 2 prizes (**bottom** figure). Environments from the training distribution are 8x8 grids.

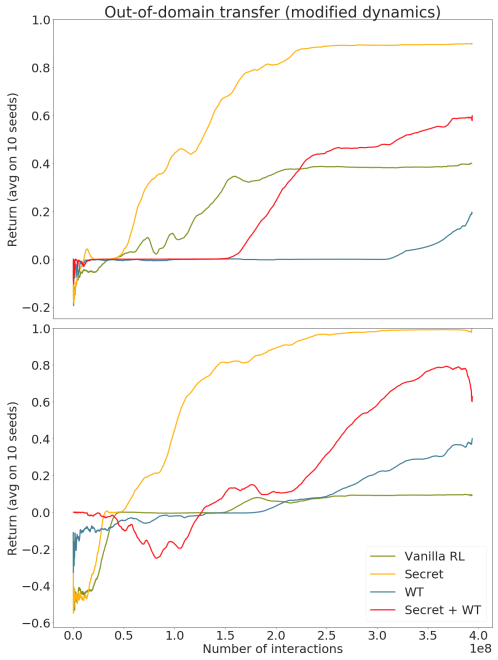


Figure 6.6: The controls of the out-of-domain distribution are inverted (up becomes down, right becomes left). Environments are 8x8 grids with 1 trigger and 1 prize (**top** figure) or 2 prizes (**bottom** figure). The effect of the shaping is exclusively beneficial, while transferring weights from the source task can be detrimental to the learning process.

In both settings, shaping the rewards assists the agent in learning to solve the task. We display some results in Fig. 6.5 and Fig. 6.6. When transferring to bigger environments, the agent benefits very early on from the shaped reward, while also reaching better asymptotical performance. We posit that the positive transfer when inverting the dynamics (which should impact a lot credit assignment methods) is due to SECRET putting more emphasis on observations than actions.

6.4 Additional related work

Transfer in RL While a lot of approaches exist in the transfer literature, to the best of our knowledge none explicitly transfer credit assignment capabilities. Previous works aimed at making the training of an agent in the same task more sample-efficient by using a pretrained model as a teacher (Rusu et al., 2016a; Schmitt et al., 2018). We learn to assign credit as a parallel task that does not modify the representations of the RL agent. Others learn auxiliary reward functions in the hope that they will enable transfer by imposing consistency in the reward (Houthooft et al., 2018; Hessel et al., 2018b; Agarwal et al., 2019). Although we also learn additional reward signal, it is based on a redistribution

of rewards from the environment, which ensures consistency with the original reward function. Transfer is also viewed as learning tasks in a sequential way (Rusu et al., 2016b; Kirkpatrick et al., 2017) and this suggests to introduce inductive bias to the neural architectures of agents to reduce catastrophic forgetting. Our method does not require to alter the agent's architecture. Other explicitly address the problem of transfer through the lens of multitask learning (Parisotto et al., 2016; Teh et al., 2017) while we stick to learning from an initial distribution of environments. Meta-learning approaches aim to train agents on a distribution of tasks or environments so that their learned skills and representations work across the underlying continuum, and allow for fast adaptation of the agents (Duan et al., 2016; Wang et al., 2016a; Finn et al., 2017b; Mishra et al., 2018; Co-Reyes et al., 2019; Zou et al., 2019). In contrast to meta-learning methods, we do not modify the RL algorithm used to train the agent and SECRET is compatible with any core algorithm for RL.

Credit assignment Previous works investigated the role of attention mechanisms for credit assignment. Ke et al. (2018) propose SAB, a sparse attention mechanism used to derive a modified backpropagation algorithm. We draw inspiration from SAB but operate in the RL context without sparsity assumptions about the attention weights. RUDDER (Arjona-Medina et al., 2019) proposes to equip RL agents with an online method for credit assignment based on return decomposition. Our method operates offline and decomposes individual rewards. RUDDER requires a specific exploration scheme, an additional episodic replay buffer, a compute-heavy contribution analysis method and the addition of several auxiliary losses to the objective the RL agent optimizes. In comparison, SECRET is a lightweight method that does not deal with exploration. Crucially, the focus of Arjona-Medina et al. (2019) is on online credit assignment while ours is on transfer. Hung et al. (2019) provide an agent with an external memory and the unsupervised task of reconstructing its inputs (both states and rewards). The agent uses memory reads as a way to identify related elements in sequences, and uses those to transfer the value of states providing delayed rewards to the bootstrapping target of contributing elements. In contrast, SECRET makes use of a non-autoregressive architecture, does not reconstruct states, makes use of reward shaping instead of modifying the update function and most importantly does not rely on an external memory. Recall Traces (Goyal et al., 2019) use a generative model that goes backward from high-reward states and samples state-action pairs that could have led to that state. SECRET also works backward from high-reward states but creates links to previous states from existing trajectories instead of sampling them.

6.5 Conclusion

In this chapter, we investigated the role explicit credit assignment could play in transfer learning and came up with SECRET, a novel transfer learning method that takes advantage of the relational properties of self-attention and transfers credit assignment instead of policy weights. We showed that SECRET led to improved sample efficiency in generalization and transfer scenarios in non-trivial gridworlds and a more complex 3D navigational task. We also showed that it led to identifying important actions, and thus to interpretable auxiliary reward functions. To the best of our knowledge, this is the first line of work in the exciting direction of credit assignment for transfer. We think it would be worth exploring how SECRET could be incorporated into online reinforcement learning methods and leave this for future work.

Part II

Action Importance and Interpretability

Chapter 7

Towards Interpretability by Learning When to Act

We now take an interest in action importance estimation for interpretability. Traditionally, Reinforcement Learning (RL) aims at deciding *how to act* optimally for an artificial agent. In this chapter, we argue that deciding *when to act* is equally important. As humans, we drift from default, instinctive or memorized behaviors to focused, thought-out behaviors when required by the situation. To enhance RL agents with this aptitude, we propose to augment the standard Markov Decision Process and make a new mode of action available: being *lazy*, which defers decision-making to a default policy. In addition, we penalize non-lazy actions in order to encourage minimal effort and have agents focus on critical decisions only. We name the resulting formalism *lazy-MDPs*. We study the theoretical properties of lazy-MDPs, expressing value functions and characterizing optimal solutions. Then we empirically demonstrate that policies learned in lazy-MDPs generally come with a form of interpretability: by construction, they show us the states where the agent takes control over the default policy. We deem those states and corresponding actions important since they explain the difference in performance between the default and the new, lazy policy. In a sense, we quantify the action importance with respect to the performance improvement between these two policies. With suboptimal policies as default (pretrained or random), we observe that agents are able to get competitive performance in Atari games while only taking control in a limited subset of states. The contents of this chapter were submitted and accepted as a full conference paper at AAMAS 2022 (Jacq et al., 2022).

7.1 Introduction

Decision-making is about providing answers to a standard question: *"how to act?"*. While Markov Decision Processes (MDPs) (Puterman, 1994) provide the canonical formalism to ask this question, Reinforcement Learning (RL) provides algorithms to answer it. In this work, we study a different question: ***"when and how to act?"***. There are several motivations for this particular question. First, in many tasks there is only a handful of states that are critical and require complex decision-making, while in other states the action has less impact than the own dynamic of the MDP (for example, the orientation of a falling piece in Tetris has no importance until it reaches the floor). Another motivation is that of learning on top of an existing policy: one might be able to learn a better policy by learning when to take control over this default policy (this is the case in many human-robot interactions (Meresht et al., 2020), for example self-driving cars that would let the human drive except in critical situations, robots assisting surgery, etc). Default policies can take arbitrary forms: controllers, handcrafted policies, programs, and many others.

To study this alternative question, we need an alternative formalism. Instead of starting from scratch, we propose to augment the existing MDP framework (see Fig. 7.1): we extend the action space with a novel action, the *lazy* action; and we modify the reward function to penalize agents when they take control (*i.e.* pick an action from the original action space). Choosing the *lazy* action defers the decision-making process to a default policy. Augmenting the MDP framework means that we can take any decision-making problem that can be expressed as an MDP and turn it into a lazy-MDP. Since defaulting is a discrete action, we chose to focus on discrete actions setting for simplicity, but everything could be adapted to continuous control (for instance using two actors, a default one and a learned one; and one critic for each).

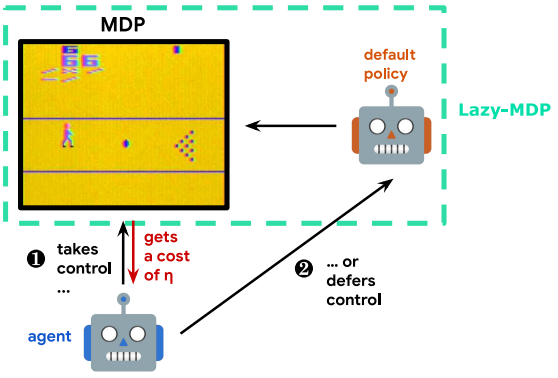


Figure 7.1: High-level overview of a lazy-MDP.

Lazy-MDPs have interesting properties for interpretability: states where the policy diverges from the default hold information. In more details, we leverage the statewise differences between the default policy and the new, lazy-policy to make sense of what is needed to get performance improvement with respect to the default policy (under arbitrary default policies) or to make sense of the overall task (under specific default policies). An important point we want to highlight is that the type of interpretability we consider here is different from explanations (Molnar, 2019). Explanations, in the context of RL, would bring answers to "why" questions (either about agent behavior or the importance of actions), which is not what we tackle here.

Our contributions are the following: 1) we propose a novel formalism called *lazy-MDP* that provides modified decision-making problems where agents have to learn when and how to act, 2) we study lazy-MDPs from a theoretical point of view and prove that we can characterize optimality, which depends on the third-party policy and the value of the penalty for taking control, and 3) we study lazy-MDPs empirically, showing that they lead to learning an interpretable partition of the states. We also study how making control less frequent (by increasing the penalty) impacts the score of agents. In hard exploration tasks, reducing the frequency of controls can even lead to improved performance.

7.2 Additional related work

While the idea of constraining policies to adopt a default behavior as often as possible in MDPs is to the best of our knowledge novel, it lives at the crossroads of several subfields of RL reviewed here.

Residual RL. Residual approaches (Silver et al., 2018; Johannink et al., 2019) consist in learning a residual policy, whose action is added to that of a base policy to get the resulting action. By its nature, residual RL is restricted to continuous control problems, where the sum of two actions is still a valid action. In contrast, the lazy-MDP abstraction is applicable to discrete control problems.

Exploration-conscious RL. Here we discuss related augmented MDPs. Shani et al. (2019) show that exploration-conscious RL (Van Seijen et al., 2009) can be solved via a surrogate MDP, where the dynamics are obtained by a linear interpolation of the dynamics induced by the current policy and those induced by a fixed policy; same for rewards. This amounts to learning a policy that is optimal given that the actual behavior is a fixed mixture of that policy and a base one. In comparison, the policies learned in lazy-MDPs are more controllable in the sense that they can switch between the base policy and a learned policy on the basis of states, and serve the subtly different purpose of learning when and how to act.

Interpretable RL. There are several types of interpretability studied in the literature. Many works try to quantify the influence of parts of the inputs on the decisions of the agents. This can be done via post-hoc gradient methods, using actual policy gradients (Wang et al., 2015; Zahavy et al., 2016) or finite-difference estimates (Greydanus et al., 2018; Puri et al., 2019), coupled with saliency maps as visualizations. Another way to do so is by training ad-hoc interpretable models (Liu et al., 2018a; Coppens et al., 2019) or programs (Verma et al., 2019b,a) to mimic the non-interpretable models used as policy networks. Others try to make sense of the representations learned by agents, using dimensionality reduction techniques (Zahavy et al., 2016) or state aggregation methods (Topin and Veloso, 2019). Another focus is to select trajectories that are representative of the overall behavior of the RL agent (Amir and Amir, 2018; Amitai and Amir, 2021). Finally, some works try to recover the approximate preferences of the agent, under the form of a reward function, or coefficients for a known reward decomposition (Juozapaitis et al., 2019; Bica et al., 2021). While interesting on their own, none of the mentioned works explicitly tackle the question of identifying states that are crucial to the decision-making process. The action-gap (Bellemare et al., 2016b) and importance advising (Torrey and Taylor, 2013) are quantities that hint at this aspect. As we show in Chapter 7.7.2, both suffer from several shortcomings compared to the proposed method.

Credit assignment in RL. Temporal credit assignment consists in associating specific actions to specific results (*i.e.* task success or high returns). Existing approaches complement or modify RL algorithms by either decomposing observed returns as the sum of redistributed rewards along observed trajectories (Arjona-Medina et al., 2019; Ferret et al., 2020; Hung et al., 2019; Raposo et al., 2021) or incorporating hindsight information into the RL process (Harutyunyan et al., 2019; Ferret et al., 2021; Mesnard et al., 2021). Our approach is related but differs in several points: it is tied to (and aims at making sense of) performance improvements instead of outcomes, and it comes from an abstraction over MDPs (which is non-parametric).

Temporal abstractions in RL. Options are common temporal abstractions in RL (Sutton et al., 1999; Precup, 2000; Bacon et al., 2017; Barreto et al., 2019). They consist in triples $(\mathcal{I}, \pi, \beta)$ where $\mathcal{I}$ is a set of states the option can be initiated into, π is the policy that selects actions when the option is active, and β is a random variable that gives the per-state probability of terminating the option. In general, learning options from scratch is hard, prone to collapse to single-action options, and less efficient than standard RL. Huang et al. (2019b) introduce Markov Jump Processes (MJPs), in which the agent both takes action and controls the frequency at which observations are received. Higher frequencies induce an increasing auxiliary cost to model scenarios where observations are limited. In essence, they propose to learn when to observe, while we propose to learn when to take control. In a related way, Biedenkapp et al. (2021) propose skip-MDPs, which decompose policies in the combination of a behavior policy (*i.e.* which selects the action) and skip policy (*i.e.* which selects the number of timesteps the action will be repeated for). Skip-MDPs does not exactly learn when to act as it permanently plays a decided action. This work rather shows that in most of situations an action need to be repeated in consecutive states, but as there are no states where the agent is deferring the control to an independent policy, they do not highlight the states where the agent decisions has a strong impact on the resulting behaviour and reward. Note that dynamic action repetition (Lakshminarayanan et al., 2017; Sharma et al., 2017) is conceptually similar, but is not formalised as an abstraction over MDPs.

Regularized RL. Regularization in RL (Geist et al., 2019) is a well-studied topic. In particular, entropic regularization (Neu et al., 2017) encourages learned policies to be as random as possible in all states. In contrast, when the default policy is uniform random, policies learned in lazy-MDPs are encouraged to be entirely random in a subset of all states only. Also, Kullback-Leibler regularization (Vieillard et al., 2020a) encourages policies to stay close to their previous iterate during learning. In contrast, policies learned in lazy-MDPs act identically to the default policy in a subset of all states, and can act in arbitrary ways in the others. Another way to ensure that the behavior of an agent does not diverge from a baseline behavior is to apply regularization on the state visitation distribution, instead of the action distribution induced by the policy (Lee et al., 2019; Geist et al., 2021).

7.3 Lazy-MDPs

MDP We refer the reader to the overall background in Chapter 3.2. An alternative to the value function that we will use in section 7.5 is $Z^\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}\{s_t \notin \mathcal{S}_{\text{abs}}\} | s_0 = s, a_0 = a]$, which is the expected discounted sum of steps before the agent meets a terminating state.

Lazy-MDP We now introduce the **Lazy Markov Decision Process** (*lazy-MDP*). A lazy-MDP is a tuple $M_+ = (M, \bar{a}, \bar{\pi}, \eta)$, where $M = (\mathcal{S}, \mathcal{A}, \gamma, r, \mathcal{P}, \delta_0)$ is the *base* MDP, $\bar{a}$ the lazy action that defers decision making to the default policy $\bar{\pi} \in \Delta_{\mathcal{A}}^{\mathcal{S}}$ and $\eta \in \mathbb{R}$ is a penalty. The reward function is that of the base MDP, except that all actions but the lazy one incur an additional reward of $-\eta$. Hence, a lazy-MDP is also an MDP. It can be written as $M_+ = (\mathcal{S}, \mathcal{A}_+, \gamma, r_+, \mathcal{P}_+, \delta_0)$. While $\mathcal{S}, \gamma, \delta_0$ are conserved from the base MDP, $\mathcal{A}_+, r_+$ and $\mathcal{P}_+$ depend on their equivalents in the base MDP, and on $\bar{\pi}$ and η:

$$\mathcal{A}_+ = \mathcal{A} \cup \{\bar{a}\},$$

$$r_+(s,a) = \begin{cases} r(s,a) - \eta, & \text{if } a \in \mathcal{A}, \\ \sum_{a \in \mathcal{A}} \bar{\pi}(a|s) r(s,a), & \text{if } a = \bar{a}, \end{cases}$$

$$\mathcal{P}_+(s'|s,a) = \begin{cases} \mathcal{P}(s'|s,a), & \text{if } a \in \mathcal{A}, \\ \sum_{a \in \mathcal{A}} \bar{\pi}(a|s) \mathcal{P}(s'|s,a), & \text{if } a = \bar{a}. \end{cases}$$

In what follows, we will use the notation X_+ to distinguish functions or distributions over the augmented action space $\mathcal{A}_+$.

7.4 Optimality in lazy-MDPs

In this section, we provide a characterization of the optimality in lazy-MDPs, similarly to what is done for regular MDPs. The derived results have two main implications. First (in Chapter 7.4.2 and 7.4.3), we identify what we call the *lazy-gap*, which quantifies the importance of taking control or not in a given state. Second (in Chapter 7.4.4), we show that taking control or not depending on the sole value of this lazy-gap leads to optimal behavior in the lazy-MDP. All statements are proven in the Appendix.

7.4.1 Value functions

Let $\pi_+(a \in \mathcal{A}_+ | s)$ be a policy in the lazy-MDP. If the agent chooses the lazy action $\bar{a}$, the performed action $a \in \mathcal{A}$ is sampled according to the default policy $\bar{\pi}$. We formalize the resulting *lazy policy* (in the base MDP) as follows:

$$\begin{aligned} \pi(a \in \mathcal{A}|s) &= P\Big[(a \sim \pi_+) \cup (\bar{a} \sim \pi_+ \cap a \sim \bar{\pi})\Big], \\ &= \pi_+(a|s) + \pi_+(\bar{a}|s)\bar{\pi}(a|s), \end{aligned}$$

satisfying $\sum_{a \in \mathcal{A}} \pi(a|s) = 1$. A crucial point is that π has the same dynamics in the base MDP as π_+ in the corresponding lazy-MDP. We are interested in the value function $V_+^{\pi_+}(s)$, which is the value of π_+ in the lazy-MDP, and takes the penalties into account. We would like to decompose $V_+^{\pi_+}(s)$ as a function of $V^\pi(s)$ (*i.e.* the value function associated with π in the base MDP) and a cost function $C^{\pi_+}(s)$:

$$V_+^{\pi_+}(s) = V^\pi(s) + C^{\pi_+}(s).$$

Theorem 2. *$C^{\pi_+}(s)$ satisfies the following Bellman equation:*

$$C^{\pi_+}(s) = -\eta(1 - \pi_+(\bar{a}|s)) + \gamma \mathbb{E}_{a \sim \pi, s' \sim \mathcal{P}(\cdot|s,a)} C^{\pi_+}(s').$$

While V^π is the expected discounted sum of rewards obtained by following π in the base MDP, the cost C^{π_+} can be interpreted as the expected discounted sum of the incurred penalties. For instance, a policy that never picks the lazy action (*i.e.* $\forall s,\ \pi_+(\bar{a}|s) = 0$) gets a maximal cost:

$$V_+^{\pi_+} = V^\pi - \frac{\eta}{1-\gamma}.$$

7.4.2 Q-functions

Let $Q_{\setminus\bar{a}}^{\pi_+}(s, a \in \mathcal{A})$ be the value (in the lazy-MDP) of taking another action than the default one:

$$\begin{aligned} Q_{\setminus\bar{a}}^{\pi_+}(s,a) &= r(s,a) - \eta + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s')\Big] \\ &= r(s,a) - \eta + \gamma \mathbb{E}_{s'}\Big[V^\pi(s') + C^{\pi_+}(s')\Big] \\ &= Q^\pi(s,a) - \eta + \gamma \mathbb{E}_{s'}\Big[C^{\pi_+}(s')\Big], \end{aligned}$$

and $\pi_{\setminus\bar{a}}(a \in \mathcal{A}|s)$ be the policy obtained by excluding the lazy action from π_+, *i.e.* assuming $\pi_+(\bar{a}|s) < 1$:

$$\forall a \in \mathcal{A},\ \ \pi_{\setminus\bar{a}}(a|s) = \frac{\pi_+(a|s)}{\sum\limits_{a' \neq \bar{a}} \pi_+(a'|s)} = \frac{\pi_+(a|s)}{1 - \pi_+(\bar{a}|s)}.$$

We can express the value function $V_+^{\pi_+}$ as a function of $Q_{\setminus\bar{a}}^{\pi_+}$:

Property 1.

$$\begin{aligned} V_+^{\pi_+}(s) =& (1 - \pi_+(\bar{a}|s)) \mathbb{E}_{a\sim\pi_{\setminus\bar{a}}}\Big[Q_{\setminus\bar{a}}^{\pi_+}(s,a)\Big] \\ &+ \pi_+(\bar{a}|s)\Big(\mathbb{E}_{a\sim\bar{\pi}}\Big[Q_{\setminus\bar{a}}^{\pi_+}(s,a)\Big] + \eta\Big). \end{aligned}$$

We then have the expression for $Q_+^{\pi_+}(s, a \in \mathcal{A}_+)$, the Q-function of π_+ in the lazy-MDP:

Property 2.

$$Q_+^{\pi_+}(s,a) = \begin{cases} Q_{\setminus\bar{a}}^{\pi_+}(s,a) & \text{if } a \neq \bar{a}, \\ \mathbb{E}_{a\sim\bar{\pi}}\Big[Q_{\setminus\bar{a}}^{\pi_+}(s,a)\Big] + \eta & \text{if } a = \bar{a}. \end{cases}$$

7.4.3 Greediness

A policy π_+ is greedy wrt Q_+ (noted $\pi_+ \in \mathcal{G}(Q_+)$) if and only if:

$$\forall s \in \mathcal{S},\quad \pi_+(\cdot|s) \in \arg\max_{\pi_+(\cdot|s)} \mathbb{E}_{a\sim\pi_+}\Big[Q_+(s,a)\Big].$$

Given a Q-function Q, a useful quantity to construct a greedy policy is what we call the ***lazy-gap***, noted $G_Q(s)$:

$$G_Q(s) = \max_{\mathcal{A}} Q(s,\cdot) - \mathbb{E}_{\bar{\pi}}\Big[Q(s,a)\Big],$$

which is the gap between the value of the best action (in $\mathcal{A}$) and the expected action-value when the immediate next action is picked by the default policy (as defined by Q) *in the lazy-MDP* (*i.e.*, with costs taken into account). In order to be greedy with respect to a policy Q_+, one needs to choose the lazy action if $G_{Q_{+\setminus\bar{a}}}(s) \leq \eta$, and to take the argmax of $Q_{+\setminus\bar{a}}$ otherwise (over $\mathcal{A}$).

Property 3. *The following policy π_+ is greedy with respect to Q_+:*

$$\pi_+(\cdot|s) = \begin{cases} \frac{\mathbb{I}\Big\{a \in \arg\max_{\mathcal{A}} Q_{+\setminus\bar{a}}(s,a)\Big\}}{\Big|\arg\max_{\mathcal{A}} Q_{+\setminus\bar{a}}(s,a)\Big|} & \text{if } G_{Q_{+\setminus\bar{a}}}(s) > \eta, \\ \mathbb{I}\Big\{a = \bar{a}\Big\} & \text{otherwise.} \end{cases}$$

Since the greediness as constructed above does not depend on the value of the lazy action $Q_+(s,\bar{a})$, we can define a greedy policy in the lazy-MDP with respect to a Q-function of the base MDP:

$$\mathcal{G}(Q)(\cdot|s) := \begin{cases} \frac{\mathbb{I}\Big\{a \in \arg\max_{\mathcal{A}} Q(s,a)\Big\}}{\Big|\arg\max_{\mathcal{A}} Q(s,a)\Big|} & \text{if } G_Q(s) > \eta, \\ \mathbb{I}\Big\{a = \bar{a}\Big\} & \text{otherwise.} \end{cases}$$

7.4.4 Optimality

We define the greedy operator $\mathcal{T}$, that maps a Q-function to the immediate reward plus the average value of the next state according to the greedy policy:

$$\mathcal{T}Q(s, a \in \mathcal{A}) := r(s,a) - \eta + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a), a' \sim \mathcal{G}(Q)(\cdot|s')}\Big[Q(s',a')\Big].$$

Theorem 3. *$\mathcal{T}$ is a γ-contraction, and converges to $Q^* := Q^{\pi_+^*}_{\setminus\bar{a}}$ where π_+^* is the optimal policy in the lazy-MDP.*

This allows us to identify the optimal policy to take decisions in the augmented action space $\mathcal{A}_+$.

Corollary 1. *π_+^* is a deterministic policy that verifies $\pi_+^*(\bar{a}|s) > 0$ if and only if $G^*(s) > \eta$, with $G^* = G_{Q^*}$ is the lazy-gap under the optimal action-value in the lazy-MDP.*

7.5 Setting the cost of taking control

Given a known base MDP, we may want to find the minimal cost $\eta_{\min}$ such that an optimal policy takes the lazy action in at least one state, as well as the maximal cost $\eta_{\max}$ such that an optimal policy does not take the lazy action in all states. From Prop. 1:

$$\eta_{\min} = \inf\Big\{\eta > 0 \text{ s.t. } \exists s,\ G^*(s) < \eta\Big\},$$
$$\eta_{\max} = \sup\Big\{\eta > 0 \text{ s.t. } \exists s,\ G^*(s) > \eta\Big\},$$

where G^* is the lazy-gap associated with the optimal value function $Q^{\pi_+^*}_{\setminus\bar{a}}$. Taking η between these two values allows to train agents that decide when to act in a non-trivial way. One can then equate states where the agent takes control as important states, under the right η.

7.5.1 $\eta_{\max}$

When η is equal or larger than the lazy-gap in all states, the optimal policy consists in deferring all actions to the default policy, which induces no cost. Thus, $\eta_{\max}$ is simply equal to the maximal lazy-gap under the default policy.

Theorem 4. *Let $Q^{\bar{\pi}}(s, a \in \mathcal{A})$ be the Q-function of the default policy in the base MDP. Then: $\eta_{max} = \max_s G_{Q^{\bar{\pi}}}(s)$.*

7.5.2 $\eta_{\min}$

One cannot apply a similar treatment to $\eta_{\min}$, due to most actions being non-default and corresponding costs having to be taken into account. However, if η is smaller or equal to the lazy-gap in all states, an optimal agent will follow the optimal policy of the base MDP π^*, and the incurred cost will be equal to $-\eta$ multiplied by the fictitious Q-value associated with π^* for a reward function that is 0 in absorbing states and 1 otherwise:

$$Z^{\pi^*}(s,a) = \mathbb{I}\{s \notin \mathcal{S}_{abs}\} + \gamma \mathbb{E}_{s'}\left[\mathbb{E}_{\pi^*}\left[Z^{\pi^*}(s',\cdot)\right]\right]$$

By construction $C^{\pi^*}(s) = -\eta \mathbb{E}_{\pi^*}[Z^{\pi^*}(s,\cdot)]$, where $C^{\pi^*}(s)$ is the cost for always following π^* from s. If no absorbing state is ever reached, we have $Z^{\pi^*}(s,a) = \frac{1}{1-\gamma}$ for all state s and action a. However, in practice the MDPs we consider have terminal states and eventually end. Z^{π^*} can thus have different values under different state-action couples, impacting the value of $\eta_{\min}$. Its value is given by the following theorem:

Theorem 5. *Let π^* be the optimal policy in the base MDP, and $Q^*(s, a \in \mathcal{A})$ the associated Q-function. Then:*

$$\eta_{min} = \min_s \max_a \frac{Q^*(s,a) - \mathbb{E}_{\bar{\pi}}\left[Q^*(s,\cdot)\right]}{1 + \left(\mathbb{E}_{\pi^*}\left[Z^{\pi^*}(s,\cdot)\right] - \mathbb{E}_{\bar{\pi}}\left[Z^{\pi^*}(s,\cdot)\right]\right)},$$

with $\eta_{min} \geq 0$.

We empirically validate these boundaries over different lazy-MDPs and report the results in Appendix D.8. As expected, when $\eta < \eta_{\min}$ no lazy actions are ever selected, and when $\eta > \eta_{\max}$, the agent always chooses the lazy action. We discuss how to approximate $\eta_{\min}$ and $\eta_{\max}$ in the next section.

7.6 Learning when and how to act in Lazy-MDPs

Since lazy-MDPs can be described as augmented MDPs, standard RL algorithms can still be used to provide policies that maximize the cumulative sum of rewards (which include a cost when taking control). As a result, by converting an MDP into a lazy-MDP, RL agents learn when and how to act without any change to their workings. We now discuss design choices for the two parameters of lazy-MDPs: the cost η and the default policy $\bar{\pi}$.

Value of cost. Regarding η, the explicit expressions for the bounds we provided guarantee meaningful behavior from optimal policies (*i.e.* not always defaulting and not always taking control). Estimating $\eta_{\max}$ is feasible since the default policy $\bar{\pi}$ is supposed available. It requires to estimate its action-values (for instance, using SARSA (Rummery and Niranjan, 1994)) so that the maximal lazy-gap can be approximated (either by taking the maximum gap across the known set of states, or using rollouts to get approximate coverage). To estimate $\eta_{\min}$, usually one does not have access to the optimal Q-function Q^* of the base MDP nor to Z^{π^*}. In that case, a solution is to use value iteration or Q-learning (Watkins and Dayan, 1992) to get approximations Q_θ and Z_θ, where Z_θ is obtained by replacing all the rewards by ones in the loss used to learn Q_θ.

Choice of the default policy. Regarding the choice of $\bar{\pi}$, we argue that taking a random policy (with, say, uniform action probabilities) is the simplest option available: it does not require any knowledge about the task, and is on par with the idea that the agent should take control only when actions actually matter (*i.e.*, a specific action is noticeably better than uniform sampling). Doing so results in a type of regularization that is conceptually close to entropic regularization, except that the agent has incentive to be as random as possible *in a subset of the states only*. In some cases, including complex scenarios, alternative options might be preferable: having to take control too often could lead to a high cumulative

cost and discourage exploration, unless η is properly tuned. Similarly to residual learning, an interesting substitute is a known, suboptimal policy. In that case, the agent has incentive to take control in states where the base policy is noticeably suboptimal.

Interpretability of lazy policies. Due to the cost of taking control, the lazy policies that are learned should only take control in a handful of states. We hope that this leads to increased interpretability for several reasons: the subset of states (and the corresponding controls) can be assessed against expert knowledge, and the performance of the learned policy can be compared to that of the base policy to ensure that the gains are sufficient and justify the overhead. Specifically, we argue that there are two special cases where lazy actions indeed give information about the *overall* importance (and unimportance) of states:

1. with a uniform random policy as default (and the right penalty), an optimal agent should only pick its actions when selecting among optimal actions brings a substantial advantage over picking the action at random. Therefore, we argue that states where the agent defers its actions are likely to be unimportant in the sense that the agent is content with acting randomly.

2. with a mixture between optimal and uniform random policy as default (and the right penalty), an optimal agent should only pick its actions when selecting among optimal actions brings a substantial advantage over *often* selecting among optimal actions. Therefore, we argue that states where the agent picks its actions are likely to be important in the sense that the agent is *not* content with acting *almost* optimally.

More broadly speaking, lazy actions give information about *the specific importance of states so as to improve on the performance of the default policy.*

We study the interpretability of solutions empirically in the next section.

7.7 Experiments

In this section, we empirically address the following questions: **1)** Do policies from lazy-MDPs learn to take control when it matters? **2)** Are the partitions of states where the agent decides or not to act interpretable? **3)** How does reducing the frequency of agent controls (by increasing the cost η) affect its returns? Details about implementations and the choice of hyperparameters can be found in Appendix D.9.

7.7.1 Taking control when it matters

We first study the behavior of lazy policies on small discrete problems, where the exact solutions can be approached as well as values for $\eta_{\min}$ and $\eta_{\max}$.

Rivers and Bridges. A simple environment that illustrates how lazy-MDPs work is a gridworld involving some dangerous pathways – where the agent needs to provide precise controls, while other states are safe – the agent can rely on the default policy. We implemented a basic scenario in which the agent has to cross three rivers by taking slippery bridges. We call this environment Rivers and Bridges (R&B), which is illustrated in Fig 7.2. Falling in the water is penalized by a strong negative reward ($\mathcal{R} = -100$), while reaching the goal point beyond the rivers results in a small positive reward ($\mathcal{R} = 1$). To simulate the slipperiness of the bridges, we take as default policy the policy that is optimal everywhere but on the bridges where it is uniformly random. That way, the agent should trust the default policy everywhere but on the bridges where it should take the control despite the cost. As there is at least one state where the policy is optimal, applying Theorem 5, we get $\eta_{\min} = 0$. As shown in Fig. 7.2 right, we verify that for any η such that $0 < \eta \leq \eta_{\max}$, the lazy-gap $G^*(s)$ is only positive on the bridges, which means an optimal agent only takes control in those states, and justifies the application of lazy-MDPs to evaluate where and when to trust a default behavior.

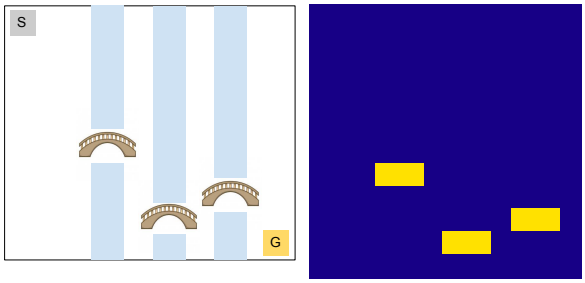
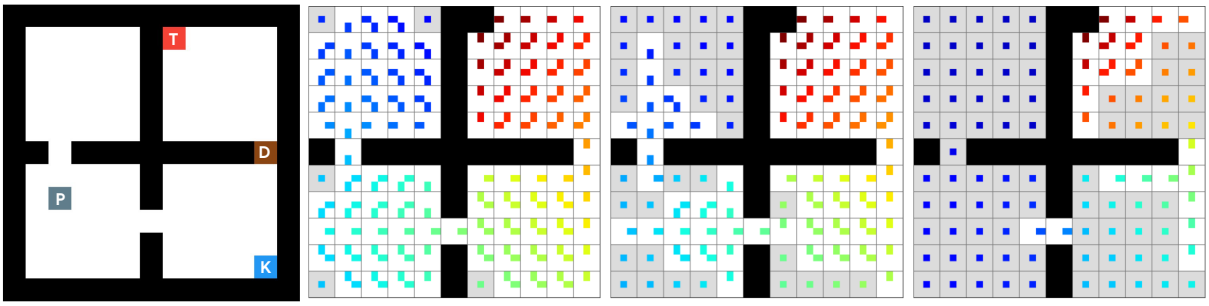


Figure 7.2: **Left:** Rivers and Bridges environment. The agent starts up left (S) and has to cross the rivers through the bridges to reach the goal point (G). Falling in the water is punished by $\mathcal{R} = -100$ and reaching the goal is rewarded by $\mathcal{R} = 1$. The default policy is the optimal π^* everywhere but on the bridges where it is uniformly random. **Right:** Heatmap of the resulting lazy-gap using $\eta = \eta_{\min} = 0$. As expected the lazy-gap is zero everywhere but on the bridges where optimal agents learn to take control. This result is valid for any value of $\eta < \eta_{\max}$.



(a) (b) $\eta = 0.008$ (c) $\eta = 0.02$ (d) $\eta = 0.05$

Figure 7.3: **(a):** Key-Door-Treasure environment. **(b-d):** Policies learned in the lazy-MDP by value iteration, with a uniform random default policy, and increasing η. Centered points in gray cells identify lazy actions, color indicates learned action-values (blue=lowest, red=highest). With η increasing, the policy learns to reduce its controls to states that allow it to progress from one room to another as well as those in the vicinity of the goal state.

Key-Door-Treasure (Oh et al., 2018) (KDT) is a variant of the classic Four Rooms task (Sutton and Barto, 2018) with a harder exploration problem: the agent needs to grab a key, open the door and get to the location of the treasure (in that order) to solve the task. The agent is only rewarded when reaching the treasure. We study the nature of the solutions learned in the lazy-MDP version of KDT under several values of η, with a uniform random default policy. The results are shown in Fig. 7.3. They match our intuition: the higher the value of η, the fewer the states in which the agent takes control; until the agent only acts in the most crucial states (*i.e.* to pass from one room to another or to get to the treasure).

7.7.2 Interpretability

To study interpretability, we use more complex environments where the behaviour of an RL agents is not trivially interpretable. We make the hypothesis that lazy-MDPs can help at explaining which states and what actions are important in order to get high returns. In this study, we opt for a qualitative measure of importance and show (via corresponding frames) the states of importance as the ones where

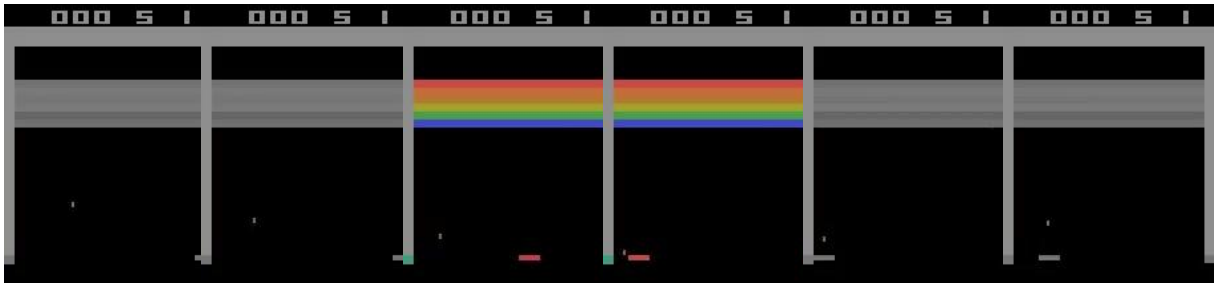


Figure 7.4: Illustration of the policy learned in the lazy-MDP version of Breakout ($\eta = 0.1$, with a uniform random default policy): the agent learns to take control (**colored frames**) only moments before the ball has to be hit in order not to lose.

the agent decides to take control over the uniform random, default policy. We look at lazy policies learned in the lazy-MDP version of Atari 2600 games from the Arcade Learning Environment (Bellemare et al., 2013). We focus on games where timing plays a central role, such as Pong, Breakout and Ms Pacman. We use a standard DQN agent (Mnih et al., 2015), whose implementation we take from the Dopamine framework (Castro et al., 2018). We display a representative portion of a lazy agent trajectory in Breakout in Fig. 7.4, which is well aligned with our intuition of the timing of this task: critical controls happen when the ball gets back to the paddle, while the remaining controls have limited impact on subsequent success. We also display key moments of a lazy agent trajectory in Ms Pacman in Fig. 7.5. The agent alternates between defaulting (most of the time) and taking control (sparsely, either for a single frame or a sequence of frames) in order to escape ghosts, to obtain power-ups and defeat ghosts, or to collect multiple bonuses in a row. Finally, we display a representative portion of episode in Bowling in Fig. 7.6. The agent takes control when aiming with the ball, and defers control to the default policy when the ball is moving towards the pins, during which actions have no effects on the outcome of the throw. All in all, the timing of the agent controls matches our intuitions.

We also compared the importance as quantified by the lazy-gap with usual measures, such as the action-gap (Bellemare et al., 2016b), which measures the difference between the best and the second best action values at a given state ($\max_{\mathcal{A}} Q(s,\cdot) - \max_{\mathcal{A}\setminus a^*} Q(s,\cdot)$), and importance advice (Torrey and Taylor, 2013), which measures the difference between the best and the worst action values at a given state ($\max_{\mathcal{A}} Q(s,\cdot) - \min_{\mathcal{A}} Q(s,\cdot)$). Fig. D.2 in Appendix D.9.1 displays the state importance according to these measures on the KDT environment. As visible, the lazy-gap only attributes importance to states with key actions (picking up the key, passing through doors, reaching the treasure). On the other hand, the action-gap uniformly emphasizes all states along the trajectory of the optimal policy, while the importance advice is dominated by the proximity to the reward and does not discriminate key actions.

7.7.3 Lazy exploration

A side effect of lazy-MDPs with a uniform random default policy is that they push agents to maintain randomness in states where acting randomly is affordable (*i.e.* does not impact future performance too much). This is helpful in hard exploration tasks, where local minima make the exploration more difficult. Actually, encouraging randomness in the policy via lazy-MDPs has two benefits: it regularizes behaviors and avoids determinism, and it rewards smart exploration where the random actions are only taken when it is safe to explore. To study the role of lazy-MDPs for exploration, we add a distractor state (*i.e.* an absorbing state with a small reward) in the upper-left room in KDT so as to introduce a local minimum. Q-learning agents, even when explicitly increasing exploration (e.g. with linearly decayed epsilon-greedy action selection), mostly fail and always go for the distractor. In that situation, augmenting the MDP as

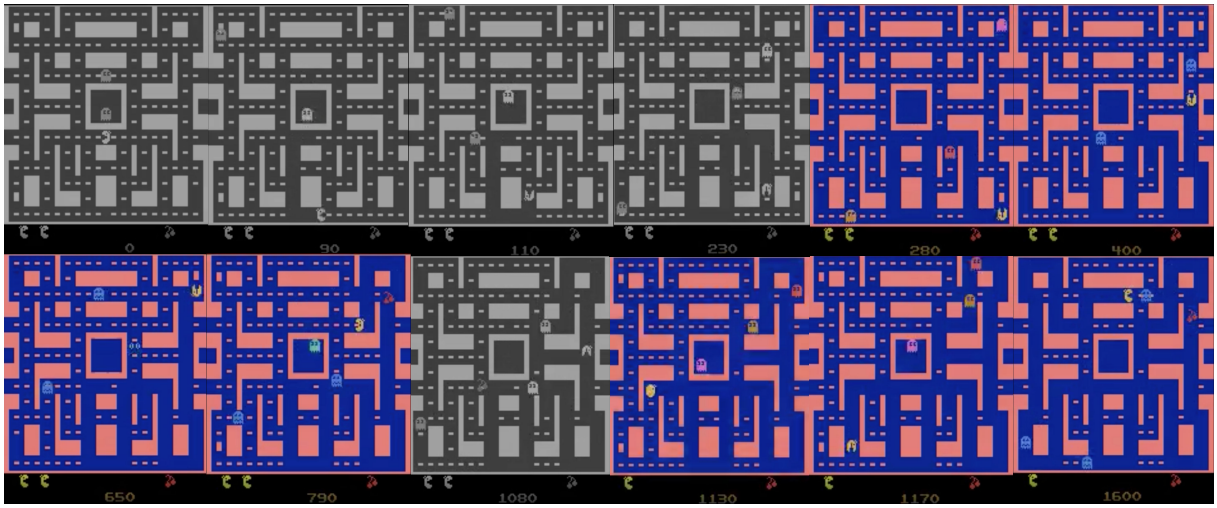


Figure 7.5: Illustration of the policy learned in the lazy-MDP version of Ms Pacman ($\eta = 0.5$, with a uniform random default policy). Frames are ordered from left to right, top to bottom, and correspond to non-consecutive frames from a single episode of interaction. At the beginning of the episode, the agent is immobile for a few frames, during which it defaults its actions to avoid penalties (**frame 1**). Once free to move, it keeps defaulting its actions and collecting nearby bonuses (**frames 2-4**) until ghosts become close enough. The agent then sparsely takes control (colored frames), be it to obtain power-ups (**frames 5 & 7**), and later on eat the ghosts (**frames 6 & 12**), or collect several bonuses in a row (**frames 10-11**).

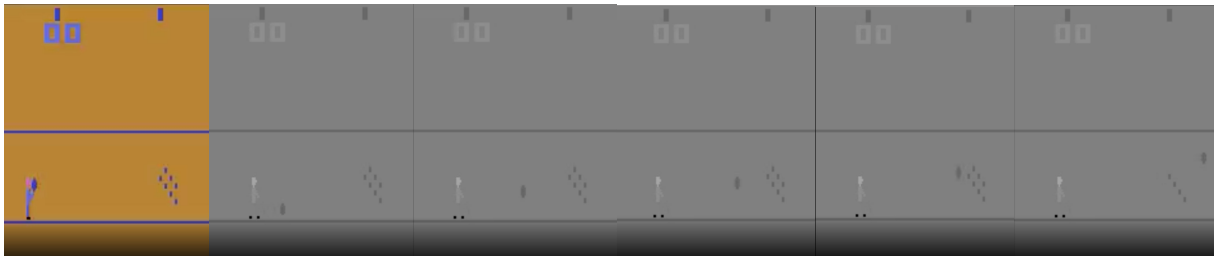
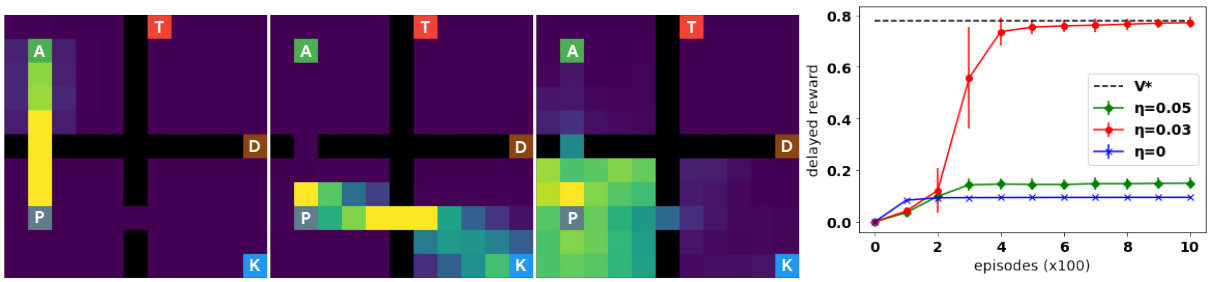


Figure 7.6: Illustration of the policy learned in the lazy-MDP version of Bowling ($\eta = 0.04$, with a uniform random default policy). The agent learns to only take control **(colored frame)** when preparing to throw the ball. This is the only time it can control the orientation of the throw, which is key to knock pins down. In subsequent timesteps, its actions have no effects on the ball, and accordingly the lazy action is picked instead.

a lazy-MDP with a uniform random default policy encourages random behaviors over consecutive steps, which helps exploring and going past the local minimum. We illustrate this effect on Fig. 7.7. For that experiment, we used tabular Q-learning with learning rate $\alpha = 0.5$, epsilon-greedy exploration starting at $\epsilon_0 = 0.1$ and linearly decayed until $\epsilon_\infty = 0$, $\gamma = 0.99$, and a reward $r = 0.1$ for the apple. Episodes that did not end in an absorbing state (*i.e.* treasure or apple) were ended after 1000 steps. Under the right cost ($\eta = 0.03$) lazy policies keep exploring after finding the small reward, and eventually find the key and the treasure, leading to a reward that justifies the cost for taking control. With a cost too high ($\eta = 0.05$), lazy policies never take control.

This motivates investigating if such an effect of taking less control while achieving higher returns



(a) $\eta = 0$ (b) $\eta = 0.03$ (c) $\eta = 0.05$ (d)

Figure 7.7: We add an apple which grants a small reward to KDT. It acts as a distractor for exploration. **(a-c):** Asymptotic state occupancy measure of Q-learning for different values of the cost η, *before the agent has the key*. We can see that Q-learning in the original MDP ($\eta = 0$) is attracted to the local optimum, while Q-learning in the lazy-MDP avoids the local optimum and eventually learns the optimal behavior (when $\eta = 0.03$). **(d):** Scores (excluding the cost) of Q-learning with different values of η. Only the agent solving the lazy-MDP with $\eta = 0.03$ converges to the optimal behavior. Both state occupancy measures and performance curves are averaged over 100 seeds.

can be observed in more complex games, requiring function approximation. Hence we converted hard exploration tasks in Atari to lazy-MDPs, including dense reward tasks (BankHeist, Frostbite, MsPacman, Zaxxon) and sparse reward tasks (Gravitar, PrivateEye), as classified in Bellemare et al. (2016a). As previously, we used uniform random default policies and several values for the cost η (0.005, 0.01, 0.02, 0.05, 0.1, 0.2) and reported the percentage of the score (with respect to a standard DQN agent (Mnih et al., 2015)) as a function of the resulting frequency of control taking (when the lazy policy does not choose the lazy action) in figure 7.8. Reported values are averaged over 3 seeds. We observe that in most cases, reducing the frequency of control taking does not decrease the score too much (up to almost 100% of the score in Gravitar with less than 10% of control, 80% of the score with less than 30% of control in Zaxxon and more than 100% of the score with 20% of control on PrivateEye). Moreover, we even observed in Frostbite that the lazy policy learned with low penalties achieved 200% of the score, confirming that lazy-MDP can also be used for improved exploration.

7.7.4 Learning on top of pretrained agents

The lazy-MDP framework can be viewed as an extension of residual learning to discrete action spaces. Indeed, in lazy-MDPs, agents learn to sparsely take control over the default policy, in a subset of the states. A natural question is: can they be used to reliably improve over the default policy? To verify so, we measure the performance of DQN agents trained in the lazy-MDP version of Atari games, with a suboptimal DQN agent as default policy. To ensure controlled suboptimality of the default policies, we use DQN agents at 50% of their asymptotical performance. Since learning in the lazy-MDPs benefits from the default policy (and the previous interaction it learned from), we compare it to warm-restarted DQN agents. Warm restart consists in starting from pretrained weights instead of randomly initialized ones, and also benefits from past interactions. Here, we use the weights of the agents taken as default policies in lazy-MDPs.

We measure the human-normalized median score (Mnih et al., 2015), which aggregates performance across the 60 Atari 2600 games. Results are reported in Fig. 7.9. We notice that warm restarting is not very efficient: while performance increases faster than agents trained from scratch, it is also less stable and eventually decreases. Using lazy-MDPs with a good penalty ($\eta = 0.05$), performance increases

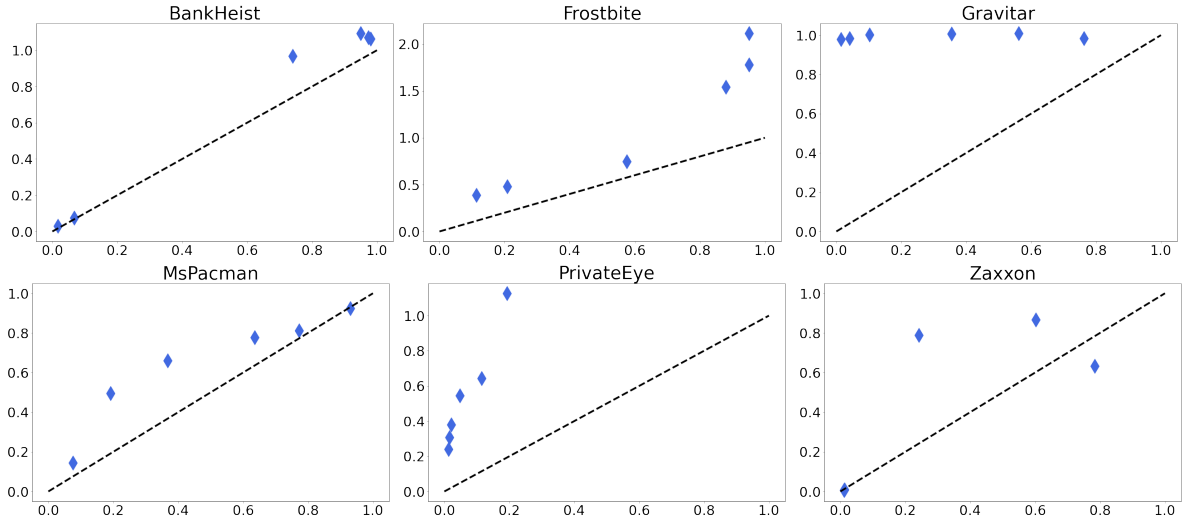


Figure 7.8: Percentage of the score of a standard DQN agent achieved by lazy-policies learned with different values of η and uniformly random default policy on hard exploration tasks of Atari. The x-axis represents the fraction of controls taken and the y-axis the percentage of the DQN baseline score reached. Each value is averaged over 3 seeds. Score 0% correspond to getting no reward and score 100% correspond to getting as much reward as a standard DQN.

faster than DQN from scratch as well, with more stable improvements. There is no asymptotic gain on performance, which is not surprising given that we use the same agent. We observe the same phenomenon as in Chapter 7.7.3: lazy-MDPs lead to better policies in hard exploration games (Frostbite, Zaxxon, Qbert, WizardOfWor, see Fig. D.3). A promising observation is that when selecting the best value of η for each game, we observe a significant performance gain over from scratch DQN. Also, lazy-MDPs can work with any type of policy as default policy (a program or a controller for instance), while warm restarting requires compatible weights. We leave the study of automated η selection to future work. We display the performance curves in all 60 games in Fig. D.3.

7.8 Conclusion

In this chapter, we studied a novel paradigm for decision-making: learning when and how to act. We proposed lazy-MDPs, natural abstractions over MDPs that are well-suited for this learning problem, and showed that RL could still be used to provide solutions in that setting. We studied the theoretical properties of lazy-MDPs, including value functions and optimality. In experiments, we showed that lazy-MDPs present an interesting edge: when converted back to the original MDPs, the policies learned in lazy-MDPs tend to be more interpretable, as they highlight states where taking control is crucial to achieve increased returns, and allow to quantify the action importance with respect to the policy improvement between new and default policies. With uniform random default policies, we show that policies learned in lazy-MDPs via DQN perform close to policies learned in standard MDPs, while only taking control in a fraction of the states; and can even reach higher scores in hard exploration tasks. With pretrained agents as default policies, policies learned in lazy-MDPs tend to outperform policies learned using warm restarts.

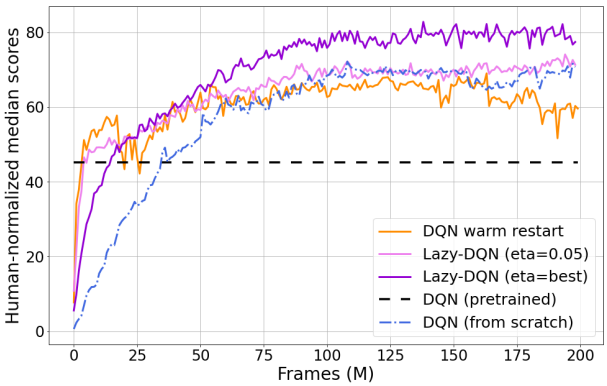


Figure 7.9: Human-normalized median scores in Atari (aggregated over the 60 games).

Chapter 8

Towards Interpretability via Reversibility-Aware RL

Having discussed a form of action importance regarding specific policy improvements, we now wonder how specific actions make behaviors more or less easy to reverse, that is taking a sequence of actions to come back to a chosen starting state. In this chapter, we propose to learn to distinguish reversible from irreversible actions for better informed decision-making in Reinforcement Learning (RL). From theoretical considerations, we show that approximate reversibility can be learned through a simple surrogate task: ranking randomly sampled trajectory events in chronological order. Intuitively, pairs of events that are always observed in the same order are likely to be separated by an irreversible sequence of actions. Conveniently, learning the temporal order of events can be done in a fully self-supervised way, which we use to estimate the reversibility of actions from experience, without any priors. Notably, this allows us to derive a notion of action importance for interpretability, one that quantifies how important an action is to conserve reversible behaviors. We propose two different strategies that incorporate reversibility in RL agents, one strategy for exploration (RAE) and one strategy for control (RAC). We demonstrate the potential of reversibility-aware agents in several environments, including the challenging Sokoban game. In synthetic tasks, we show that we can learn control policies that never fail and reduce to zero the side-effects of interactions, even without access to the reward function. This chapter was published as a full conference paper at NeurIPS 2021 (Grinsztajn et al., 2021).

8.1 Introduction

We address the problem of estimating if and how easily actions can be reversed in the Reinforcement Learning (RL) context. Irreversible outcomes are often not to be taken lightly when making decisions. As humans, we spend more time evaluating the outcomes of our actions when we know they are irreversible (McAllister et al., 1979). As such, irreversibility can be positive (*i.e.* takes risk away for good) or negative (*i.e.* leads to later regret). Also, decision-makers are more likely to anticipate regret for hard-to-reverse decisions (Zeelenberg, 1999). All in all, irreversibility seems to be a good prior to exploit for more principled decision-making. In this work, we explore the option of using irreversibility to guide decision-making and confirm the following assertion: by estimating and factoring reversibility in the action selection process, safer behaviors emerge in environments with intrinsic risk factors. In addition to this, we show that exploiting reversibility leads to more efficient exploration in environments with undesirable irreversible behaviors, including the famously difficult Sokoban puzzle game.

However, estimating the reversibility of actions is no easy feat. It seemingly requires a combination of

planning and causal reasoning in large dimensional spaces. We instead opt for another, simpler approach (see Fig. 8.1): we propose to learn in which direction time flows between two observations, directly from the agents' experience, and then consider *irreversible* the transitions that are assigned a temporal direction with high confidence. *In fine*, we reduce reversibility to a simple classification task that consists in predicting the temporal order of events.

Our contributions are the following: 1) we formalize the link between reversibility and precedence estimation, and show that reversibility can be approximated via temporal order, 2) we propose a practical algorithm to learn temporal order in a self-supervised way, through simple binary classification using sampled pairs of observations from trajectories, 3) we propose two novel exploration and control strategies that incorporate reversibility, and study their practical use for directed exploration and safe RL, illustrating their relative merits in synthetic as well as more involved tasks such as Sokoban puzzles.

8.2 Additional related work

To the best of our knowledge, this work is the first to explicitly model the reversibility of transitions and actions in the context of RL, using temporal ordering to learn from trajectories in a self-supervised way, in order to guide exploration and control. Yet, several aspects of the problem we tackle were studied in different contexts, with other motivations; we review these here.

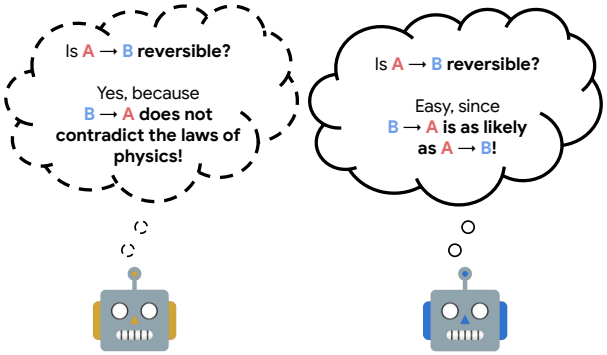


Figure 8.1: High-level illustration of how reversibility can be estimated. **Left:** from an understanding of physics. **Right:** ours, from experience.

Leveraging reversibility in RL. Kruusmaa et al. (2007) estimate the reversibility of state-action couples so that robots avoid performing irreversible actions, since they are more likely to damage the robot itself or its environment. A shortcoming of their approach is that they need to collect explicit state-action pairs and their reversal actions, which makes it hard to scale to large environments. Several works (Savinov et al., 2019; Badia et al., 2020b,a) use reachability as a curiosity bonus for exploration: if the current state has a large estimated distance to previous states, it means that it is novel and the agent should be rewarded. Reachability and reversibility are related, in the sense that irreversible actions lead to states from which previous states are unreachable. Nevertheless, their motivations and ours diverge, and we learn reversibility through a less involved task than that of learning reachability. Nair et al. (2020) learn to reverse trajectories that start from a goal state so as to generate realistic trajectories that reach similar goals. In contrast, we use reversibility to direct exploration and/or control, not for generating learning data. Closest to our work, Rahaman et al. (2020) propose to learn a potential function of the states that increases with time, which can detect irreversibility to some extent. A drawback of the approach is that the potential function is learned using trajectories sampled from a random policy, which is a problem for many tasks where a random agent might fail to cover interesting parts of the state space. In comparison, our method does not use a potential function and learns jointly with the RL agent, which makes it a viable candidate for more complex tasks.

Safe exploration. Safe exploration aims at making sure that the actions of RL agents do not lead to negative or unrecoverable effects that would outweigh the long-term value of exploration (Amodei et al., 2016). Notably, previous works developed distinct approaches to avoid irreversible behavior: by incremental updates to safe policies (Hans et al., 2008; García and Fernández, 2012), which requires knowing such a

policy in advance; by restricting policy search to ergodic policies (Moldovan and Abbeel, 2012) (*i.e.* that can always come back to any state visited), which is costly; and by active exploration (Maillard et al., 2019), where the learner can ask for rollouts instead of exploring potentially unsafe areas of the state space itself.

Self-supervision from the arrow of time. Self-supervision has become a central component of modern machine learning algorithms, be it for computer vision, natural language or signal processing. In particular, using temporal consistency as a source of self-supervision is now ubiquitous, be it to learn representations for downstream tasks (Goroshin et al., 2015; Ramanathan et al., 2015; Dadashi et al., 2020), or to learn to detect temporal inconsistencies (Wei et al., 2018). The closest analogies to our work are methods that specifically estimate some aspects of the arrow of time as self-supervision. Most are to be found in the video processing literature, and self-supervised tasks include predicting which way the time flows (Pickup et al., 2014; Wei et al., 2018), verifying the temporal order of a subset of frames (Misra et al., 2016), predicting which video clip has the wrong temporal order among a subset (Fernando et al., 2017) as well as reordering shuffled frames or clips from the video (Fernando et al., 2015; El-Nouby et al., 2019; Xu et al., 2019). Bai et al. (2020) notably propose to combine several of these pretext tasks along with data augmentation for video classification. Using time as a means of supervision was also explored for image sequencing (Basha et al., 2012), audio (Carr et al., 2021) or EEG processing (Saeed et al., 2020). In RL, self-supervision also gained momentum in recent years (Guo et al., 2020; Srinivas et al., 2020; Yarats et al., 2021), with temporal information being featured (Amiranashvili et al., 2018). Notably, several works (Aytar et al., 2018; Dwibedi et al., 2018; Guo et al., 2018; Sermanet et al., 2018) leverage temporal consistency to learn useful representations, effectively learning to discriminate between observations that are temporally close and observations that are temporally distant. In comparison to all these works, we estimate the arrow of time through temporal order prediction with the explicit goal of finding irreversible transitions or actions.

8.3 Reversibility

Degree of reversibility. We start by introducing formally the notion of reversibility. Intuitively, an action is reversible if it can be undone, meaning that there is a sequence of actions that can bring us back to the original state.

Definition 8. *Given a state s, we call* degree of reversibility within K steps *of an action a*

$$\phi_K(s,a) \coloneqq \sup_{\pi} p_\pi(s \in \tau_{t+1:t+K+1} \mid s_t = s, a_t = a),$$

and the degree of reversibility *of an action is defined as*

$$\phi(s,a) \coloneqq \sup_{\pi} p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s, a_t = a),$$

with $\tau = \{s_i\}_{i=1\dots T} \sim \pi$ corresponding to a trajectory, and $\tau_{t:t'}$ the subset of the trajectory between the timesteps t and t' (excluded). We omit their dependency on π for the sake of conciseness. Given $s \in S$, the action a is reversible *if and only if $\phi(s,a) = 1$, and said* irreversible *if and only if $\phi(s,a) = 0$.*

In deterministic environments, an action is either reversible or irreversible: given a state-action couple (s,a) and the unique resulting state s', $\phi_K(s,a)$ is equal to 1 if there is a sequence of less than K actions which brings the agent from s' to s, and is otherwise equal to zero. In stochastic environments, a given sequence of actions can only reverse a transition up to some probability, hence the need for the notion of degree of reversibility.

Policy-dependent reversibility. In practice, it is useful to quantify the degree of reversibility of an action as the agent acts according to a fixed policy π, for which we extend the notions introduced above. We simply write :

$$\phi_{\pi,K}(s,a) \coloneqq p_\pi(s \in \tau_{t+1:t+K+1} \mid s_t = s, a_t = a) \text{ and } \phi_\pi(s,a) \coloneqq p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s, a_t = a).$$

It immediately follows that $\phi_K(s,a) = \sup_\pi \phi_{\pi,K}(s,a)$ and $\phi(s,a) = \sup_\pi \phi_\pi(s,a)$.

8.4 Reversibility estimation via classification

Quantifying the exact degree of reversibility of actions is generally hard. In this section, we show that reversibility can be approximated efficiently using simple binary classification.

8.4.1 Precedence estimation

Supposing that a trajectory contains the states s and s', we want to be able to establish *precedence*, that is predicting whether s or s' comes first *on average*. It is a binary classification problem, which consists in estimating the quantity $\mathbb{E}_{s_t=s,s_{t'}=s'}\left[\mathbb{1}_{t'>t}\right]$. Accordingly, we introduce the precedence estimator which, using a set of trajectories, learns to predict which state of an arbitrary pair is most likely to come first.

Definition 9. *Given a fixed policy π, we define the* finite-horizon precedence estimator *between two states as follows:*

$$\psi_{\pi,T}(s,s') = \mathbb{E}_{\tau\sim\pi}\,\mathbb{E}_{\substack{s_t=s,s_{t'}=s' \\ t,t'<T}}\left[\mathbb{1}_{t'>t}\right].$$

Conceptually, given two states s and s', the precedence estimator gives an approximate probability of s' being visited after s, given that both s and s' are observed in a trajectory. The indices are sampled uniformly within the specified horizon $T \in \mathbb{N}$, so that this quantity is well-defined even for infinite trajectories. Additional properties of ψ, regarding transitivity for instance, can be found in Appx. E.1.2.

Remark 1. *The quantity $\psi_{\pi,T}(s,s')$ is only defined for pairs of states which can be found in the same trajectory, and is otherwise irrelevant. In what follows, we implicitly impose this condition when considering state pairs.*

Theorem 6. *For every policy π and $s, s' \in S$, $\psi_{\pi,T}(s,s')$ converges when T goes to infinity. We refer to the limit as the* precedence estimator*, written $\psi_\pi(s,s')$.*

The proof of this theorem is developed in Appendix E.1.3. This result is key to ground theoretically the notion of empirical reversibility $\bar{\phi}$, which we introduce in the next definition. It simply consists in extending the notion of precedence to a state-action pair.

Definition 10. *We finally define the* empirical reversibility *using the precedence estimator:*

$$\bar{\phi}_\pi(s,a) = \mathbb{E}_{s'\sim P(s,a)}\left[\psi_\pi(s',s)\right].$$

In a nutshell, given that we start in s and take the action a, the empirical reversibility $\bar{\phi}_\pi(s,a)$ measures the probability that we go back to s, starting from a state s' that follows (s,a). We now show that our empirical reversibility is linked with the notion of reversibility defined in the previous section, and can behave as a useful proxy.

8.4.2 Estimating reversibility from precedence

We present here our main theoretical result which relates reversibility and empirical reversibility:

Theorem 7. *Given a policy π, a state s and an action a, we have: $\bar{\phi}_\pi(s,a) \geq \frac{\phi_\pi(s,a)}{2}$.*

The full proof of the theorem is given in Appendix E.1.3.

This result theoretically justifies the name of empirical reversibility. From a practical perspective, it provides a way of using $\bar{\phi}$ to detect actions which are irreversible or hardly reversible: $\bar{\phi}_\pi(s,a) \ll 1$ implies $\phi_\pi(s,a) \ll 1$ and thus provides a sufficient condition to detect actions with low degrees of reversibility. This result gives a way to detect actions that are irreversible given a specific policy followed by the agent. Nevertheless, we are generally interested in knowing if these actions are irreversible for any policy, meaning $\phi(s,a) \ll 1$ with the definition of Section 8.3. The next proposition makes an explicit connection between $\bar{\phi}_\pi$ and ϕ, under the assumption that the policy π is stochastic.

Proposition 1. *We suppose that we are given a state s, an action a such that a is reversible in K steps, and a policy π. Under the assumption that π is stochastic enough, meaning that there exists $\rho > 0$ such that for every state and action s, a, $\pi(a \mid s) > \rho$, we have: $\bar{\phi}_\pi(s,a) \geq \frac{\rho^K}{2}$. Moreover, we have for all $K \in \mathbb{N}$: $\bar{\phi}_\pi(s,a) \geq \frac{\rho^K}{2}\phi_K(s,a)$.*

The proof is given in Appendix E.1.4. As before, this proposition gives a practical way of detecting irreversible moves. If for example $\bar{\phi}_\pi(s,a) < \rho^k/2$ for some $k \in \mathbb{N}$, we can be sure that action a is not reversible in k steps. The quantity ρ can be understood as a minimal probability of taking any action in any state. This condition is not very restrictive: ϵ-greedy strategies for example satisfy this hypothesis with $\rho = \frac{\epsilon}{|A|}$.

In practice, it can also be useful to limit the maximum number of time steps between two sampled states. That is why we also define the windowed precedence estimator as follows:

Definition 11. *Given a fixed policy π, we define the* windowed precedence estimator *between two states as follows:*

$$\psi_{\pi,T,w}(s,s') = \mathbb{E}_{\tau\sim\pi}\mathbb{E}_{\substack{s_t=s, s_{t'}=s' \\ t,t'<T \\ |t-t'|\leq w}}\left[\not\Vdash_{t'>t}\right].$$

Intuitively, compared to previous precedence estimators, $\psi_{\pi,T,w}$ is restricted to short-term dynamics, which is a desirable property in tasks where distinguishing the far future from the present is either trivial or impossible.

8.5 Reversibility-Aware Reinforcement Learning

Leveraging the theoretically-grounded bridge between precedence and reversibility established in the previous section, we now explain how reversibility can be learned from the agent's experience and used in a practical setting.

Learning to rank events chronologically. Learning which observation comes first in a trajectory is achieved by binary supervised classification, from pairs of observations sampled uniformly in a sliding window on observed trajectories. This can be done fully offline, *i.e.* using a previously collected dataset of trajectories for instance, or fully online, *i.e.* jointly with the learning of the RL agent; but also anywhere on the spectrum by leveraging variable amounts of offline and online data.

This procedure is not without caveats. In particular, we want to avoid overfitting to the particularities of the behavior of the agent, so that we can learn meaningful, generalizable statistics about the order

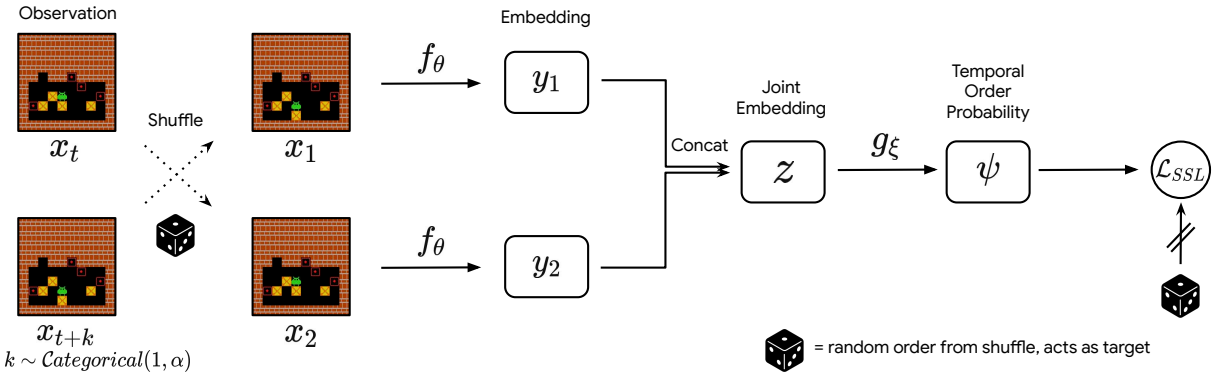


Figure 8.2: The proposed self-supervised procedure for precedence estimation.

of events in the task at hand. Indeed, if an agent always visits the state s_a before s_b, the classifier will probably assign a close-to-one probability that s_a precedes s_b. This might not be accurate with other agents equipped with different policies, unless transitioning from s_b to s_a is hard due to the dynamics of the environment, which is in fact exactly the cases we want to uncover. We make several assumptions about the agents we apply our method to: 1) agents are learning and thus, have a policy that changes through interactions in the environment, 2) agents have an incentive not to be too deterministic. For this second assumption, we typically use an entropic regularization in the chosen RL loss, which is a common design choice in modern RL methods. These assumptions, when put together, alleviate the risk of overfitting to the idiosyncrasies of a single, non-representative policy.

We illustrate the precedence classification procedure in Fig. 8.2. A temporally-ordered pair of observations, distant of no more than w timesteps, is sampled from a trajectory and uniformly shuffled. The result of the shuffling operation is memorized and used as a target for the binary classification task. A Siamese network creates separate embeddings for the pair of observations, which are concatenated and fed to a separate feed-forward network, whose output is passed through a sigmoid to obtain a probability of precedence. This probability is updated via negative log-likelihood against the result of the shuffle, so that it matches the actual temporal order.

Then, a transition (and its implicit sequence of actions) represented by a starting observation x and a resulting observation x' is deemed irreversible if the estimated precedence probability $\psi(x, x')$ is superior to a chosen threshold β. Note that we do not have to take into account the temporal proximity of these two observations here, which is a by-product of sampling observations uniformly in a window in trajectories. Also, depending on the threshold β, we cover a wide range of scenarios, from pure irreversibility (β close to 1) to soft irreversibility ($\beta > 0.5$, the bigger β, the harder the transition is to reverse). This is useful because different tasks call for different levels of tolerance for irreversible behavior: while a robot getting stuck and leading to an early experiment failure is to be avoided when possible, tasks involving human safety might call for absolute zero tolerance for irreversible decision-making. We elaborate on these aspects in Chapter 8.6.

Reversibility-Aware Exploration and Control. We propose two different algorithms based on reversibility estimation: Reversibility-Aware Exploration (RAE) and Reversibility-Aware Control (RAC). We give a high-level representation of how the two methods operate in Fig. 8.3.

In a nutshell, RAE consists in using the estimated reversibility of a pair of consecutive observations to create an auxiliary reward function. In our experiments, the reward function is a piecewise linear function of the estimated reversibility and a fixed threshold, as in Fig. 8.3: it grants the agent a negative reward if the transition is deemed too hard to reverse. The agent optimizes the sum of the extrinsic and auxiliary

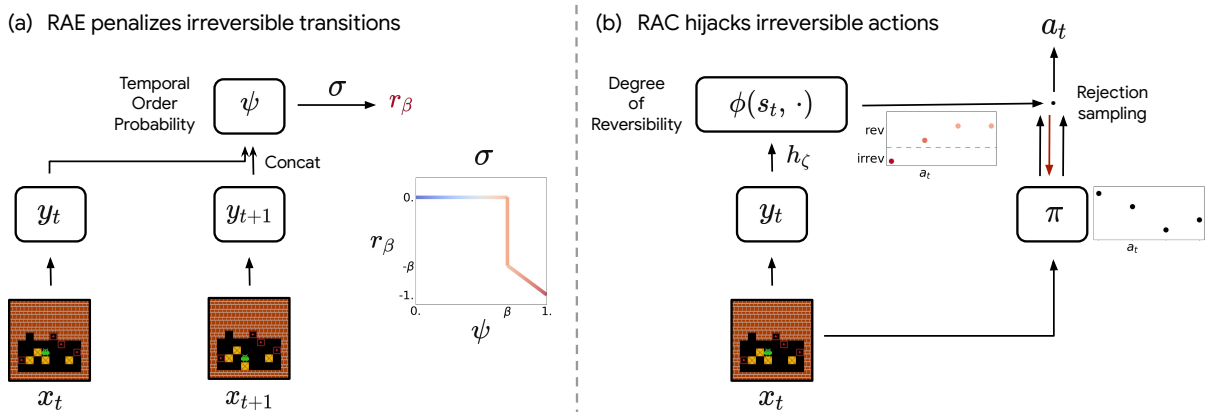


Figure 8.3: Our proposed methods for reversibility-aware RL. **(a):** RAE encourages reversible behavior via auxiliary rewards. **(b):** RAC avoids irreversible behavior by rejecting actions whose estimated reversibility is inferior to a threshold.

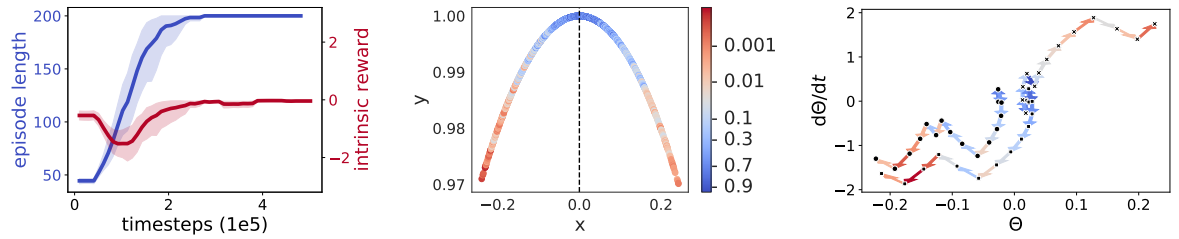
rewards. Note that the specific function we use penalizes irreversible transitions but could encourage such transitions instead, if the task calls for it.

RAC can be seen as the action-conditioned counterpart of RAE. From a single observation, RAC estimates the degree of reversibility of all available actions, and "takes control" if the action sampled from the policy is not reversible enough (*i.e.* has a reversibility inferior to a threshold β). "Taking control" can have many forms. In practice, we opt for rejection sampling: we sample from the policy until an action that is reversible enough is sampled. This strategy has the advantage of avoiding irreversible actions entirely, while trading-off pure reversibility for performance when possible. RAC is more involved than RAE, since the action-conditioned reversibility is learned from the supervision of a standard, also learned precedence estimator. Nevertheless, our experiments show that it is possible to learn both estimators jointly, at the cost of little overhead.

We now discuss the relative merits of the two methods. In terms of applications, we argue that RAE is more suitable for directed exploration, as it only encourages reversible behavior. As a result, irreversible behavior is permitted if the benefits (*i.e.* rewards) outweigh the costs (*i.e.* irreversibility penalties). In contrast, RAC shines in safety-first, real-world scenarios, where irreversible behavior is to be banned entirely. With an optimal precedence estimator and task-dependent threshold, RAC will indeed hijack all irreversible sampled actions. RAC can be especially effective when pre-trained on offline trajectories: it is then possible to generate fully-reversible, safe behavior from the very first online interaction in the environment. We explore these possibilities experimentally in Chapter 8.6.2.

Both algorithms can be used online or offline with small modifications to their overall logic. The pseudo-code for the online version of RAE and RAC can be found in Appendix E.2.2.

The self-supervised precedence classification task could have applications beyond estimating the reversibility of actions: it could be used as a means of getting additional learning signal or representational priors for the RL algorithm. Nevertheless, we opt for a clear separation between the reversibility and the RL components so that we can precisely attribute improvements to the former, and leave aforementioned studies for future work.



(a) Training curves (b) Relative pole coordinates (c) Random trajectories

Figure 8.4: **(a):** Training curves of a PPO+RAE agent in reward-free Cartpole. Blue: episode length. Red: intrinsic reward. A 95% confidence interval over 10 random seeds is shown. **(b):** The x and y axes are the coordinates of the end of the pole relatively to the cart position. The color denotes the online reversibility estimation between two consecutive states (logit scale). **(c):** The representation of three random trajectories according to θ (angle of the pole) and $\frac{d\theta}{dt}$. Arrows are colored according to the learned reversibility of the transitions they correspond to.

8.6 Experiments

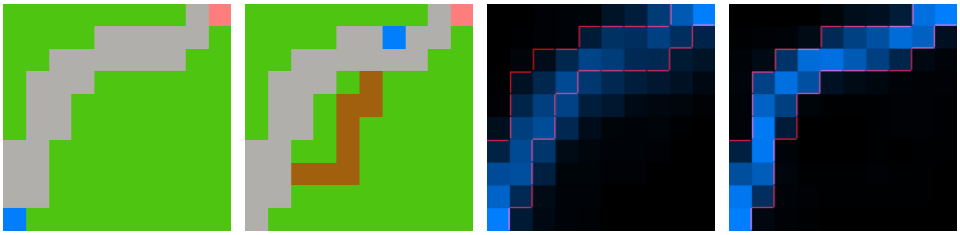
The following experiments aim at demonstrating that the estimated precedence ψ is a good proxy for reversibility, and at illustrating how beneficial reversibility can be in various practical cases. We benchmark RAE and RAC on a diverse set of environments, with various types of observations (tabular, pixel-based), using neural networks for function approximation. See Appendix E.3 for details.

8.6.1 Reward-free Reinforcement Learning

We illustrate the ability of RAE to learn sensible policies without access to rewards. We use the classic pole balancing task Cartpole (Barto et al., 1983), using the OpenAI Gym (Brockman et al., 2016b) implementation. In the usual setting, the agent gets a reward of 1 at every time step, such that the total undiscounted episode reward is equal to the episode length, and incentivizes the agent to learn a policy that stabilizes the pole. Here, instead, we remove this reward signal and give a PPO agent (Schulman et al., 2017) an intrinsic reward based on the estimated reversibility, which is learned online from agent trajectories. The reward function penalizes irreversibility, as shown in Fig. 8.3. Note that creating insightful rewards is quite difficult: too frequent negative rewards could lead the agent to try and terminate the episode as soon as possible.

We display our results in Fig. 8.4. Fig. 8.4a confirms the claim that RAE can be used to learn meaningful rewards. Looking at the intrinsic reward, we discern three phases. Initially, both the policy and the reversibility classifier are untrained (and intrinsic rewards are 0). In the second phase, the classifier is fully trained but the agent still explores randomly (intrinsic rewards become negative). Finally, the agent adapts its behavior to avoid penalties (intrinsic rewards go to 0, and the length of trajectories increases). Our reward-free agent reaches the score of 200, which is the highest possible score.

To further assess the quality of the learned reversibility, we freeze the classifier after 300k timesteps and display its predicted probabilities according to the relative coordinates of the end of the pole (Fig. 8.4b) and the dynamics of the angle of the pole θ (Fig. 8.4c). In both cases, the empirical reversibility matches our intuition: the reversibility should decrease as the angle or angular momentum increase, since these coincide with an increasing difficulty to go back to the equilibrium.



(a) Initial state (b) A trajectory (c) PPO (500k) (d) PPO+RAE (500k)

Figure 8.5: **(a):** The Turf environment. The agent can walk on grass, but the grass then turns brown. **(b):** An illustrative trajectory where the agent stepped on grass pixels. **(c):** State visitation heatmap for PPO. **(d):** State visitation heatmap for PPO+RAE. It coincides with the stone path (red).

8.6.2 Learning reversible policies

In this section, we investigate how RAE can be used to learn reversible policies. When we train an agent to achieve a goal, we usually want it to achieve that goal following implicit safety constraints. Handcrafting such safety constraints would be time-consuming, difficult to scale for complex problems, and might lead to reward hacking; so a reasonable proxy consists in limiting irreversible side-effects in the environment (Leike et al., 2017).

To quantify side-effects, we propose Turf, a new synthetic environment. As depicted in Fig. 8.5a,8.5b, the agent (blue) is rewarded when reaching the goal (pink). Stepping on grass (green) will spoil it, causing it to turn brown. Stepping on the stone path (grey) does not induce any side-effect.

In Fig. 8.5c,8.5d, we compare the behaviors of a trained PPO agent with and without RAE. The baseline agent is indifferent to the path to the goal, while the agent benefitting from RAE learns to follow the road, avoiding irreversible consequences.

8.6.3 Sokoban

Sokoban is a popular puzzle game where a warehouse keeper (controlled by the player) must move boxes around and place them in dedicated places. Each level is unique and involves planning, since there are many ways to get stuck. For instance, pushing a box against a wall is often un-undoable, and prevents the completion of the level unless actually required to place the box on a specific target. Sokoban is a challenge to current model-free RL algorithms, as advanced agents require millions of interactions to reliably solve a fraction of levels (Weber et al., 2017; Guez et al., 2019). One of the reasons for this is tied to exploration: since agents learn from scratch, there is a long preliminary phase where they act randomly in order to explore the different levels. During this phase, the agent will lock itself in unrecoverable states many times, and further exploration is wasted. It is worth recalling that contrary to human players, the agent does not have the option to reset the game when stuck. In these regards, Sokoban is a great testbed for reversibility-aware approaches, as we expect them to make the exploration phase more efficient, by incorporating the prior that irreversible transitions are to be avoided if possible, and by providing tools to identify such transitions.

We benchmark performance on a set of 1k levels. Results are displayed in Fig. 8.6. Equipping an IMPALA agent (Espeholt et al., 2018) with RAE leads to a visible performance increase, and the resulting agent consistently solves all levels from the set. We take a closer look at the reversibility estimates and show that they match the ground truth with high accuracy, despite the high imbalance of the distribution (*i.e.* few irreversible transitions, see Fig. 8.6c) and complex reversibility estim ation (see Fig. 8.6a).

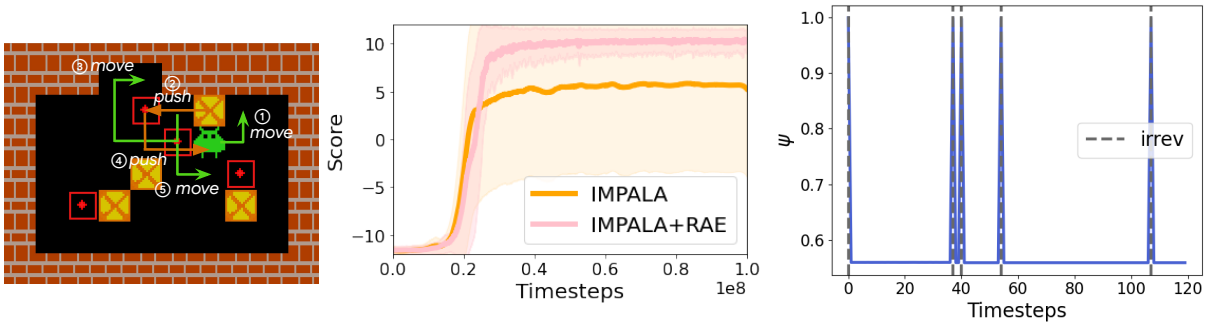


Figure 8.6: **(a):** Non-trivial reversibility: pushing the box against the wall can be reversed by pushing it to the left, going around, pushing it down and going back to start. A minimum of 17 moves is required to go back to the starting state. **(b):** Performances of IMPALA and IMPALA+RAE on 1k levels of Sokoban (5 seeds average). **(c):** Evolution of the estimated reversibility along one episode.

8.6.4 Safe control

In this section, we put an emphasis on RAC, which is particularly suited for safety related tasks.

Cartpole+. We use the standard Cartpole environment, except that we change the maximum number of steps from 200 to 50k to study long-term policy stability. We name this new environment Cartpole+. It is substantially more difficult than the initial setting. We learn reversibility offline, using trajectories collected from a random policy. Fig. 8.7a shows that a random policy augmented with RAC achieves seemingly infinite scores. For the sake of comparison, we indicate that a DQN (Mnih et al., 2015) and the state-of-the-art M-DQN (Vieillard et al., 2020b) achieve a maximum score of respectively 1152 and 2801 under a standard training procedure, described in Appendix E.3.5. This can be surprising, since RAC was only trained on random thus short trajectories (mean length of 20). We illustrate the predictions of our learned estimator in Fig. 8.7b,8.7c. When the pole leans to the left ($x < 0$), we can see that moving the cart to the left is perceived as more reversible than moving it to the right. This is key to the good performance of RAC and perfectly on par with our understanding of physics: when the pole is leaning in a direction, agents must move the cart in the same direction to stabilize it.

Turf. We now illustrate how RAC can be used for safe online learning: the implicitly safe constraints provided by RAC prevent policies from deviating from safe trajectories. This ensures for example that agents stay in recoverable zones during exploration.

We learn the reversibility estimator offline, using the trajectories of a random policy. We reject actions whose reversibility is deemed inferior to $\beta = 0.2$, and train a PPO agent with RAC. As displayed in Fig. 8.8, PPO with RAC learns to reach the goal without causing any irreversible side-effect (*i.e.* stepping on grass) during the whole training process.

The threshold β is a very important parameter of the algorithm. Too low a threshold could lead to overlooking some irreversible actions, while a high threshold could prevent the agent from learning the new task at hand. We discuss this performance/safety trade-off

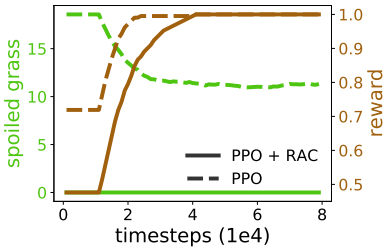
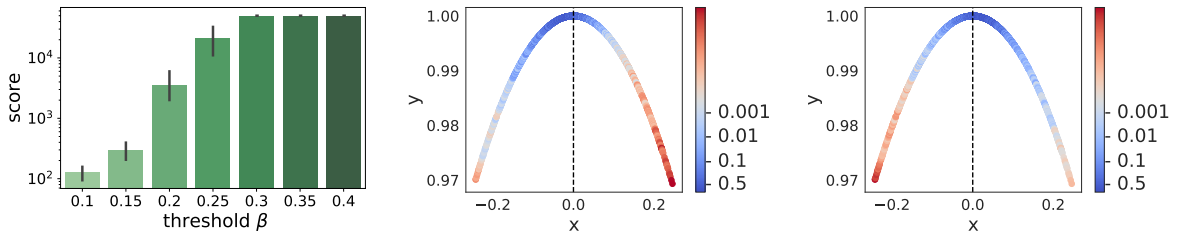


Figure 8.8: PPO and RAC (solid lines) vs PPO (dashed lines). At the cost of slower learning (brown), our approach prevents the agent from producing a single irreversible side-effect (green) during the learning phase. Curves are averaged over 10 runs.



(a) Trajectory lengths (b) Coordinates: action 0 (c) Coordinates: action 1

Figure 8.7: **(a):** Mean score of a random policy augmented with RAC on Cartpole+ for several threshold values, with 95% confidence intervals over 10 random seeds (log scale). **(b) and (c):** The x and y axes are the coordinates of the end of the pole relatively to the cart position. The color indicates the estimated reversibility values.

in more details in Appendix. E.3.7.

8.7 Conclusion

In this chapter, we formalized the link between the reversibility of transitions and their temporal order, which inspired a self-supervised procedure to learn how important actions are to stay reversible, from experience. In combination with two novel reversibility-aware exploration strategies, RAE for directed exploration and RAC for directed control, we showed the empirical benefits of our approach in various scenarios, ranging from safe RL to risk-averse exploration. Notably, we demonstrated increased performance in procedurally-generated Sokoban puzzles, which we tied to more efficient exploration. This is a promising avenue for interpretability since knowing about the reversibility of actions gives information about the task and which of the agent decisions might matter.

Chapter 9

Conclusion

In this work, we studied how explicit and implicit forms of action importance could serve the distinct purposes of credit assignment and interpretability in RL. An interesting finding is that these apparently disconnected, open problems in RL share ideas, methods and algorithms: at the same time, important actions should get more credit, and important actions give information about the structure of the task, the soundness of learned decision-making strategies and why agents improve at the task. Regarding credit assignment, which is a major object of study of this thesis, we showed that it is generally useful to distinguish between implicit and explicit forms of credit assignment, and that the distinction leads to a useful framework we can express most of existing methods in. Implicit forms of credit assignment are everywhere in the RL literature, value and action-value estimation being prominent examples that answer (at least partially for action-values) the question of action importance with respect to the performance of policies. We showed that careful modifications of these implicit forms could lead to significant improvement in terms of asymptotical performance and sample-efficiency, notably in hard exploration problems, and to better action importance estimation by increasing the action-gap. Explicit forms (either context-free or grounded) are less represented despite apparent promises regarding the collection of desiderata we proposed for principled credit assignment. In that spirit, we proposed an explicit algorithm for offline credit assignment that can be leveraged to derive augmented reward functions that identify crucial actions and do not change optimal policies, and also generalize and transfer well to tasks unseen during offline training. Compared to the previously mentioned, explicit algorithm, it estimates a different form of action importance that is contextualized with the reward accumulated within the input trajectory. Interpretability is another object of study of this thesis. We focused on two aspects: gaining understanding at the task level, and gaining understanding of specific policy improvements. At the task level, we proposed a self-supervised mechanism to approximate the reversibility of actions. This is useful in order to explore more efficiently, for instance by avoiding deadlocks, and this can also be useful to gain understanding of which states and corresponding decisions cannot be reversed. We also proposed an alternative formalism based on MDPs which encourages laziness in decision-making, essentially amounting to learning when to take control over a default policy. Under the right default policy, analyzing in which states the agent takes control (*i.e.* is not lazy) gives us information about which decisions are crucial to get right in order to succeed at the task. More generally, under an arbitrary default policy, a similar approach can be taken in order to make sense of the performance improvement (or decrease) between the current policy and the default policy. All in all, the problem of action importance estimation has many facets. This is not surprising since the notion of importance calls for a goal for this importance to be specified against. Accordingly, we managed to propose different algorithms, some sharing ideas, that tackle specific forms of importance, either for credit assignment or interpretability. We find that in most cases, actions that are important (to reach high performance, to obtain specific policy improvements, or to keep a reversible

behavior) can be rewarded for improved sample-efficiency of RL agents learning, which go in the direction of the assertion that one should exploit in states with important actions. To a certain extent, we even show that such actions can be rewarded for improved exploration (in a sense, exploiting to explore).

9.1 Perspectives

There are many exciting perspectives in the research directions we investigated during the course of this thesis. We review the most interesting ones in the next paragraphs.

Credit assignment There is still much to be done regarding the temporal credit assignment problem in RL. To being with, we think that we desperately need to formalize better what credit assignment is about if we want to really understand how to address it. One promising way to start would be to characterize, both empirically and theoretically, what the limitations of existing agents are and why. Current RL algorithms struggle in tasks with temporal delay, high stochasticity, partial observability and misspecified rewards, but we only partly why. For that, a benchmark with an established evaluation protocol and standardized environments should prove necessary. A good benchmark would provide tasks featuring challenging credit assignment problems, tools to evaluate the quality of the credit assignment assigned, and an empirical comparison of existing methods to address the credit assignment issue. Being ultimately interested in finding actions that are the root causes of later effects, we are sure that as causality tools progress they will prove useful to mitigate credit assignment problems. All in all, we think that credit assignment is key to obtain RL algorithms that will learn robustly and efficiently in complex tasks, and that a lot of future research will tackle this difficult question.

Interpretability There are many directions of interest for interpretable RL. Given the recent successes in zero-shot and few-shot learning with large language models (LLMs), we think that exploring how pretrained LLMs can learn to summarize or explain agent behavior (and transfer this ability) is a promising research avenue. Another success of LLMs is in code generation. Code generating LLMs could be leveraged to produce programmatic policies that are interpretable by construction, which would be a nice follow-up to the works on programmatic learning (Verma et al., 2019b,a). Another direction that emerges from the work presented in this thesis is interpretable Dynamic Programming, using the formalism of Lazy-MDPs. Indeed, lazy-MDPs encourage laziness on the basis of individual states, so the subsets of states where agents take control can be taken a look at to make sense of the policy improvement between the default and newly learned policy. We think that studying the evolution of these subsets as part of the Dynamic Programming scheme could make it more interpretable.

Explicit and interpretable credit assignment Explicit credit assignment methods would bridge the gap between credit assignment and interpretability, by quantifying precisely how much each outcome depends on each past action. Notably, there would be much merit in finding a simple, general method that satisfies all the explicit credit assignment criteria we listed, which would have applications to both credit assignment and interpretability.

Learnable sequential priors In our work on reversibility-aware RL, we found a simple yet powerful way to estimate the reversibility of actions in a self-supervised way. One could argue that most truly important decisions are irreversible, so that could be used as a prior for credit assignment, which is yet to be explored. Also, similar techniques could be used to estimate other sequential quantities of interest (i.e. not only the reversibility but the commutativity of actions for instance) which could then be used as general, learnable sequential priors.

Agents that learn their own rewards As humans, we possess an internal drive that defines the biological rewards to seek and influences our decision-making process. There exist similar ideas in the RL space, notably inverse RL algorithms (Ng et al., 2000), which construct a reward function for which observed behavior is optimal. Such algorithms have the drawback that they involve solving a problem that is as hard as solving the original task via RL. Learning the reward function also happens in a separate process to the RL learning. We think it would be worthwhile to have agents learn rewards and control end-to-end. Having agents infer their own reward function as they learn might be useful when rewards are particularly sparse, misspecified or simply absent like in the case of unsupervised RL. For that, it is possible that explicit credit assignment tools prove useful, including those proposed in this thesis, by inferring how much actions contribute to delayed effects.

General action importance In this work, we presented several forms of action importance and methods to estimate them. An interesting idea to pursue would be to parameterize families of forms of action importance (an example could be the importance of actions to reach specific goal states, and parameters could be those of a goal generator) and derive a method that can estimate action importance over the whole parameter space. Such a method could be of empirical utility in many scenarios.

Bibliography

Abbeel, P., Quigley, M., and Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*.

Agarwal, R., Liang, C., Schuurmans, D., and Norouzi, M. (2019). Learning to generalize from sparse and underspecified rewards. *International Conference on Machine Learning*.

Ahmed, Z., Le Roux, N., Norouzi, M., and Schuurmans, D. (2019). Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning*.

Alipov, V., Simmons-Edler, R., Putintsev, N., Kalinin, P., and Vetrov, D. (2021). Towards practical credit assignment for deep reinforcement learning. *arXiv preprint arXiv:2106.04499*.

Amir, D. and Amir, O. (2018). Highlights: Summarizing agent behavior to people. In *International Conference on Autonomous Agents and Multiagent Systems*.

Amiranashvili, A., Dosovitskiy, A., Koltun, V., and Brox, T. (2018). Motion perception in reinforcement learning with dynamic objects. In *Conference on Robot Learning*.

Amitai, Y. and Amir, O. (2021). "I don't think so": Disagreement-based policy summaries for comparing agents. *arXiv preprint arXiv:2102.03064*.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P. F., Schulman, J., and Mané, D. (2016). Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., and Welinder, Peter, e. a. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058.

Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., and Hochreiter, S. (2019). Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*.

Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*.

Arora, S. and Doshi, P. (2021). A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*.

Arumugam, D., Henderson, P., and Bacon, P.-L. (2021). An information-theoretic perspective on credit assignment in reinforcement learning. *arXiv preprint arXiv:2103.06224*.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*.

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *AAAI Conference on Artificial Intelligence*.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020a). Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., et al. (2020b). Never give up: Learning directed exploration strategies. *International Conference on Learning Representations*.

Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., and Müller, K.-R. (2010). How to explain individual classification decisions. *Journal of Machine Learning Research*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Bahdanau, D., Hill, F., Leike, J., Hughes, E., Hosseini, A., Kohli, P., and Grefenstette, E. (2019). Learning to understand goal specifications by modelling reward. In *International Conference on Learning Representations*.

Bai, Y., Fan, H., Misra, I., Venkatesh, G., Lu, Y., Zhou, Y., Yu, Q., Chandra, V., and Yuille, A. (2020). Can temporal information help with contrastive self-supervised learning? *arXiv preprint arXiv:2011.13046*.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning*.

Baird, L. C. (1999). Reinforcement learning through gradient descent. Technical report, Carnegie Mellon University.

Bakker, B. (2007). Reinforcement learning by backpropagation through an lstm model/critic. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*.

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D. K., Hunt, J. J., Mourad, S., Silver, D., et al. (2019). The option keyboard: Combining skills in reinforcement learning. In *Advances in Neural Information Processing Systems*.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*.

Basha, T., Moses, Y., and Avidan, S. (2012). Photo sequencing. In *European Conference on Computer Vision*.

Bastani, O., Pu, Y., and Solar-Lezama, A. (2018). Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016a). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P., and Munos, R. (2016b). Increasing the action gap: New operators for reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Bellman, R. (1957). *Dynamic Programming*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1995). Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*.

Bica, I., Jarrett, D., Hüyük, A., and van der Schaar, M. (2021). Learning "what-if" explanations for sequential decision-making. In *International Conference on Learning Representations*.

Biedenkapp, A., Rajan, R., Hutter, F., and Lindauer, M. (2021). Towards TempoRL: learning when to act. *International Conference on Machine Learning, BIG workshop*.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brafman, R. I. and Tennenholtz, M. (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016a). Openai gym.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016b). Openai gym.

Buesing, L., Weber, T., Zwols, Y., Racaniere, S., Guez, A., Lespiau, J.-B., and Heess, N. (2018). Woulda, coulda, shoulda: Counterfactually-guided policy search. *arXiv preprint arXiv:1811.06272*.

Buesing, L., Weber, T., Zwols, Y., Racaniere, S., Guez, A., Lespiau, J.-B., and Heess, N. (2019). Woulda, coulda, shoulda: Counterfactually-guided policy search. In *International Conference on Learning Representations*.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by random network distillation. *International Conference on Learning Representations*.

Camacho, E. F. and Alba, C. B. (2013). *Model predictive control*.

Campero, A., Raileanu, R., Küttler, H., Tenenbaum, J. B., Rocktäschel, T., and Grefenstette, E. (2021). Learning with amigo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*.

Carr, A. N., Berthet, Q., Blondel, M., Teboul, O., and Zeghidour, N. (2021). Self-supervised learning of audio representations from permutations with differentiable ranking. In *IEEE Signal Processing Letters*.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. (2018). Dopamine: A Research Framework for Deep Reinforcement Learning. *arXiv preprint arXiv:1812.06110*.

Cauchy, A. et al. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comptes Rendus Scientifiques*.

Chapelle, O., Scholkopf, B., and Zien, A. (2009). Semi-supervised learning. *IEEE Transactions on Neural Networks*.

Chelu, V., Precup, D., and van Hasselt, H. (2020). Forethought and hindsight in credit assignment. In *Advances in Neural Information Processing Systems*.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*.

Chevalier-Boisvert, M., Willems, L., and Pal, S. (2018). Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*.

Co-Reyes, J. D., Gupta, A., Sanjeev, S., Altieri, N., DeNero, J., Abbeel, P., and Levine, S. (2019). Meta-learning language-guided policy learning. In *International Conference on Learning Representations*.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*.

Coppens, Y., Efthymiadis, K., Lenaerts, T., Nowé, A., Miller, T., Weber, R., and Magazzeni, D. (2019). Distilling deep reinforcement learning policies in soft decision trees. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*.

Dadashi, R., Hussenot, L., Geist, M., and Pietquin, O. (2020). Primal wasserstein imitation learning. *arXiv preprint arXiv:2006.04678*.

Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in neural information processing systems*.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2018). Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*.

Deb, K. (2011). Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the Annual Meeting of North American chapter of ACL (NAACL 2019)*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*.

Dosovitskiy, A. and Koltun, V. (2016). Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. In *Proceedings of the International Conference on Learning Representations*.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Dwibedi, D., Tompson, J., Lynch, C., and Sermanet, P. (2018). Learning actionable representations from visual observations. In *International Conference on Intelligent Robots and Systems*.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Edwards, A. D., Downs, L., and Davidson, J. C. (2018). Forward-backward reinforcement learning. *arXiv preprint arXiv:1803.10227*.

El-Nouby, A., Zhai, S., Taylor, G. W., and Susskind, J. M. (2019). Skip-clip: Self-supervised spatiotemporal representation learning by future clip order ranking. *arXiv preprint arXiv:1910.12770*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*.

Eysenbach, B., Salakhutdinov, R., and Levine, S. (2019). Search on the replay buffer: Bridging planning and reinforcement learning. *arXiv preprint arXiv:1906.05253*.

Farahmand, A.-m. (2011). Action-gap phenomenon in reinforcement learning. In *Advances in Neural Information Processing Systems*.

Farebrother, J., Machado, M. C., and Bowling, M. (2018). Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*.

Fernando, B., Bilen, H., Gavves, E., and Gould, S. (2017). Self-supervised video representation learning with odd-one-out networks. In *Conference on Computer Vision and Pattern Recognition*.

Fernando, B., Gavves, E., Oramas, J. M., Ghodrati, A., and Tuytelaars, T. (2015). Modeling video evolution for action recognition. In *Conference on Computer Vision and Pattern Recognition*.

Ferret, J., Marinier, R., Geist, M., and Pietquin, O. (2020). Self-attentional credit assignment for transfer in reinforcement learning. In *International Joint Conference on Artificial Intelligence*.

Ferret, J., Pietquin, O., and Geist, M. (2021). Self-imitation advantage learning. In *International Conference on Autonomous Agents and Multiagent Systems*.

Finn, C., Abbeel, P., and Levine, S. (2017a). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*.

Finn, C., Abbeel, P., and Levine, S. (2017b). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*.

Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P., and Geist, M. (2021a). Adversarially guided actor-critic. In *International Conference on Learning Representations*.

Flet-Berliac, Y., Ouhamma, R., Maillard, O.-A., and Preux, P. (2021b). Learning value functions in deep policy gradients using residual variance. In *International Conference on Learning Representations*.

Flet-Berliac, Y. and Preux, P. (2020). Only relevant information matters: Filtering out noisy samples to boost rl. In *International Joint Conference on Artificial Intelligence*.

Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2018). Noisy networks for exploration. *International Conference on Learning Representations*.

Gangwani, T., Zhou, Y., and Peng, J. (2020). Learning guidance rewards with trajectory-space smoothing. In *Advances in Neural Information Processing Systems*.

García, J. and Fernández, F. (2012). Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*.

Geist, M., Pérolat, J., Laurière, M., Elie, R., Perrin, S., Bachem, O., Munos, R., and Pietquin, O. (2021). Concave utility reinforcement learning: the mean-field game viewpoint. *arXiv preprint arXiv:2106.03787*.

Geist, M., Scherrer, B., and Pietquin, O. (2019). A theory of regularized markov decision processes. In *International Conference on Machine Learning*.

Ghasemipour, S. K. S., Zemel, R., and Gu, S. (2020). A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*.

Goertzel, B., Iklé, M., and Wigmore, J. (2012). The architecture of human-like general intelligence. In *Theoretical Foundations of Artificial General Intelligence*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and LeCun, Y. (2015). Unsupervised learning of spatiotemporally coherent metrics. In *International Conference on Computer Vision*.

Goyal, A., Brakel, P., Fedus, W., Singhal, S., Lillicrap, T., Levine, S., Larochelle, H., and Bengio, Y. (2019). Recall traces: Backtracking models for efficient reinforcement learning. In *International Conference on Learning Representations*.

Greydanus, S., Koul, A., Dodge, J., and Fern, A. (2018). Visualizing and understanding atari agents. In *International Conference on Machine Learning*.

Grinsztajn, N., Ferret, J., Pietquin, O., Preux, P., and Geist, M. (2021). There is no turning back: A self-supervised approach for reversibility-aware rl. In *Advances in Neural Information Processing Systems*.

Gruslys, A., Dabney, W., Azar, M. G., Piot, B., Bellemare, M., and Munos, R. (2017). The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. *arXiv preprint arXiv:1704.04651*.

Guez, A., Mirza, M., Gregor, K., et al. (2019). An investigation of model-free planning. In *International Conference on Machine Learning*.

Guez, A., Viola, F., Weber, T., Buesing, L., Kapturowski, S., Precup, D., Silver, D., and Heess, N. (2020). Value-driven hindsight modelling. In *Advances in Neural Information Processing Systems*.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. (2018). Learning to search with mctsnets. *arXiv preprint arXiv:1802.04697*.

Guo, Y., Choi, J., Moczulski, M., Bengio, S., Norouzi, M., and Lee, H. (2019). Self-imitation learning via trajectory-conditioned policy for hard-exploration tasks. *arXiv*.

Guo, Y., Oh, J., Singh, S., and Lee, H. (2018). Generative adversarial self-imitation learning. *arXiv preprint arXiv:1812.00950*.

Guo, Z. D., Pires, B. A., Piot, B., Grill, J.-B., Altché, F., Munos, R., and Azar, M. G. (2020). Bootstrap latent-predictive representations for multitask reinforcement learning. In *International Conference on Machine Learning*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*.

Han, S. and Sung, Y. (2020). Diversity actor-critic: Sample-aware entropy regularization for sample-efficient exploration. In *International Conference on Machine Learning*.

Hans, A., Schneegaß, D., Schäfer, A. M., and Udluft, S. (2008). Safe exploration for reinforcement learning. In *European Symposium on Artificial Neural Networks*.

Hansen, S., Pritzel, A., Sprechmann, P., Barreto, A., and Blundell, C. (2018). Fast deep reinforcement learning using online adjustments from the past. In *Advances in Neural Information Processing Systems*.

Harutyunyan, A., Dabney, W., Mesnard, T., Gheshlaghi Azar, M., Piot, B., Heess, N., van Hasselt, H. P., Wayne, G., Singh, S., Precup, D., and Munos, R. (2019). Hindsight credit assignment. In *Advances in Neural Information Processing Systems*.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*.

He, F. S., Liu, Y., Schwing, A. G., and Peng, J. (2016). Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *Advances in Neural Information Processing Systems*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018a). Rainbow: Combining improvements in deep reinforcement learning. *Association for the Advancement of Artificial Intelligence*.

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. (2018b). Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., et al. (2018). Deep q-learning from demonstrations. *Association for the Advancement of Artificial Intelligence*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., et al. (2020). Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Ho, O. J., and Abbeel, P. (2018). Evolved policy gradients. In *Advances in Neural Information Processing Systems*.

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *ACL*.

Howard, R. A. (1960). *Dynamic programming and markov processes.*

Huang, S. H., Bhatia, K., Abbeel, P., and Dragan, A. D. (2018). Establishing appropriate trust via critical states. In *International Conference on Intelligent Robots and Systems*.

Huang, S. H., Held, D., Abbeel, P., and Dragan, A. D. (2019a). Enabling robots to communicate their objectives. *Autonomous Robots*.

Huang, Y., Kavitha, V., and Zhu, Q. (2019b). Continuous-time markov decision processes with controlled observations. In *Allerton Conference on Communication, Control, and Computing*.

Hull, C. L. (1943). *Principles of behavior*.

Hung, C.-C., Lillicrap, T., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. (2019). Optimizing agent behavior over long time scales by transporting value. *Nature Communications*.

Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S., and Hofmann, K. (2019). Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*.

Jacq, A. D., Ferret, J., Pietquin, O., and Geist, M. (2022). Lazy-mdp: Towards interpretable reinforcement learning by learning when to act. In *Autonomous Agents and Multi-Agent Systems*.

Janner, M., Li, Q., and Levine, S. (2021). Reinforcement learning as one big sequence modeling problem. *arXiv preprint arXiv:2106.02039*.

Jiang, N., Kulesza, A., Singh, S., and Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*.

Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. (2019). Residual reinforcement learning for robot control. In *International Conference on Robotics and Automation*.

Johnstone, T., Grinsztajn, N., Ferret, J., and Preux, P. (2021). More efficient exploration with symbolic priors on action sequence equivalences. *arXiv preprint arXiv:2110.10632*.

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive science*.

Juozapaitis, Z., Koul, A., Fern, A., Erwig, M., and Doshi-Velez, F. (2019). Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*.

Justesen, N., Rodriguez Torrado, R., Bontrager, P., Khalifa, A., Togelius, J., and Risi, S. (2018). Illuminating generalization in deep reinforcement learning through procedural level generation. In *NeurIPS Workshop on Deep Reinforcement Learning*.

Kakade, P. D. S. (2001). Explaining away in weight space. In *Advances in Neural Information Processing Systems*.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2018). Recurrent experience replay in distributed reinforcement learning. In *International Conference on Machine Learning*.

Ke, N. R., Goyal, A., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. (2018). Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in Neural Information Processing Systems*.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*.

Khan, O., Poupart, P., and Black, J. (2009). Minimal sufficient explanations for factored markov decision processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Kiefer, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*.

Kim, B., Khanna, R., and Koyejo, O. O. (2016). Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in neural information processing systems*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kipf, T., Li, Y., Dai, H., Zambaldi, V., Grefenstette, E., Kohli, P., and Battaglia, P. (2018). Compositional imitation learning: Explaining and executing one task at a time. *arXiv preprint arXiv:1812.01483*.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (2017)*.

Klissarov, M. and Precup, D. (2018). Diffusion-based approximate value functions. *OpenReview preprint OR:BkgkoToZZ7*.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*.

Kruusmaa, M., Gavshin, Y., and Eppendahl, A. (2007). Don't do things you can't undo: Reversibility models for generating safe behaviours. In *International Conference on Robotics and Automation*.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*.

Kumar, A., Peng, X. B., and Levine, S. (2019). Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial machine learning at scale. In *International Conference on Learning Representations*.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of machine learning research*.

Lakshminarayanan, A., Sharma, S., and Ravindran, B. (2017). Dynamic action repetition for deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Lattimore, T. and Szepesvári, C. (2020). *Bandit algorithms*.

Lee, K., Lee, K., Shin, J., and Lee, H. (2020a). Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*.

Lee, K.-H., Fischer, I., Liu, A., Guo, Y., Lee, H., Canny, J., and Guadarrama, S. (2020b). Predictive information accelerates learning in RL. In *Advances in Neural Information Processing Systems*.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. (2019). Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*.

Lee, S. Y., Choi, S., and Chung, S.-Y. (2018). Sample-efficient deep reinforcement learning via episodic backward update. *arXiv preprint arXiv:1805.12375*.

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. (2017). AI safety gridworlds. *arXiv preprint arXiv:1711.09883*.

Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Levy, A., Platt, R., and Saenko, K. (2019). Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*.

Lin, Z., Feng, M., dos Santos, C. N., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. In *Proceedings of the International Conference on Learning Representations*.

Liu, G., Schulte, O., Zhu, W., and Li, Q. (2018a). Toward interpretable deep reinforcement learning with linear model u-trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Liu, S., Kailkhura, B., Chen, P.-Y., Ting, P., Chang, S., and Amini, L. (2018b). Zeroth-order stochastic variance reduction for nonconvex optimization. *Advances in Neural Information Processing Systems*.

Liu, Y., Luo, Y., Zhong, Y., Chen, X., Liu, Q., and Peng, J. (2019). Sequence modeling of temporal credit assignment for episodic reinforcement learning. *arXiv preprint arXiv:1905.13420*.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*.

Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*.

Mahadevan, S. and Maggioni, M. (2007). Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*.

Maillard, O.-A., Mann, T., Ortner, R., and Mannor, S. (2019). Active Rollouts in MDP with Irreversible Dynamics. *Hal preprint hal-02177808*.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Machine Learning Proceedings*.

McAllister, D. W., Mitchell, T. R., and Beach, L. R. (1979). The contingency model for the selection of decision strategies: An empirical test of the effects of significance, accountability, and reversibility. In *Organizational Behavior and Human Performance*.

McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Hassabis, D., Kim, B., Paquet, U., and Kramnik, V. (2021). Acquisition of chess knowledge in alphazero. *arXiv preprint arXiv:2111.09259*.

Meresht, V. B., De, A., Singla, A., and Gomez-Rodriguez, M. (2020). Learning to switch between machines and humans. *arXiv preprint arXiv:2002.04258*.

Mesnard, T., Weber, T., Viola, F., Thakoor, S., Saade, A., Harutyunyan, A., Dabney, W., Stepleton, T., Heess, N., Guez, A., et al. (2021). Counterfactual credit assignment in model-free reinforcement learning. In *International Conference on Machine Learning*.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*.

Minsky, M. (1961). Steps toward artificial intelligence. In *Computers and Thought*.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. In *International Conference on Learning Representations*.

Misra, I., Zitnick, C. L., and Hebert, M. (2016). Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*.

Miyato, T., Maeda, S.-i., Koyama, M., Nakae, K., and Ishii, S. (2016). Distributional smoothing with virtual adversarial training. In *International Conference on Learning Representations*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., et al. (2015). Human-level control through deep reinforcement learning. *Nature*.

Moldovan, T. M. and Abbeel, P. (2012). Safe exploration in markov decision processes. In *International Conference on Machine Learning*.

Molnar, C. (2019). *Interpretable machine learning*.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*.

Mott, A., Zoran, D., Chrzanowski, M., Wierstra, D., and Rezende, D. J. (2019). Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems*.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in neural information processing systems*.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*.

Nair, S., Babaeizadeh, M., Finn, C., Levine, S., and Kumar, V. (2020). Time reversal as self-supervision. In *International Conference on Robotics and Automation*.

Neu, G., Jonsson, A., and Gómez, V. (2017). A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*.

Norris, J. R. (1998). *Markov chains*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press.

Oh, J., Guo, Y., Singh, S., and Lee, H. (2018). Self-imitation learning. In *International Conference on Machine Learning*.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*.

Orsini, M., Raichuk, A., Hussenot, L., Vincent, D., Dadashi, R., Girgin, S., Geist, M., Bachem, O., Pietquin, O., and Andrychowicz, M. (2021). What matters for adversarial imitation learning? *arXiv preprint arXiv:2106.00672*.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvari, C., Singh, S., et al. (2020). Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*.

Paine, T. L., Gulcehre, C., Shahriari, B., Denil, M., Hoffman, M., Soyer, H., Tanburn, R., Kapturowski, S., Rabinowitz, N., Williams, D., et al. (2019). Making efficient use of demonstrations to solve hard exploration problems. In *International Conference on Learning Representations*.

Parisotto, E., Ba, J., and Salakhutdinov, R. (2016). Actor-mimic: Deep multitask and transfer reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR 2016)*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*.

Pearl, J. (2009). *Causality*.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*.

Peters, J., Janzing, D., and Schölkopf, B. (2017). *Elements of causal inference: foundations and learning algorithms*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the Annual Meeting of North American chapter of ACL (NAACL 2018)*.

Pfau, D. and Vinyals, O. (2016). Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*.

Pickup, L. C., Pan, Z., Wei, D., Shih, Y. C., Zhang, C., Zisserman, A., Scholkopf, B., and Freeman, W. T. (2014). Seeing the arrow of time. In *Conference on Computer Vision and Pattern Recognition*.

Piot, B., Geist, M., and Pietquin, O. (2014). Boosted bellman residual minimization handling expert demonstrations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst.

Puiutta, E. and Veith, E. (2020). Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*.

Puri, N., Verma, S., Gupta, P., Kayastha, D., Deshmukh, S., Krishnamurthy, B., and Singh, S. (2019). Explain your move: Understanding agent actions using specific and relevant feature attribution. In *International Conference on Learning Representations*.

Puterman, M. L. (1994). *Markov Decision Processes*.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. (2019). Stable baselines3. https://github.com/DLR-RM/stable-baselines3.

Rahaman, N., Wolf, S., Goyal, A., Remme, R., and Bengio, Y. (2020). Learning the arrow of time for problems in reinforcement learning. In *International Conference on Learning Representations*.

Raileanu, R. and Rocktäschel, T. (2019). Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*.

Ramanathan, V., Tang, K., Mori, G., and Fei-Fei, L. (2015). Learning temporal embeddings for complex video analysis. In *International Conference on Computer Vision*.

Raposo, D., Ritter, S., Santoro, A., Wayne, G., Weber, T., Botvinick, M., van Hasselt, H., and Song, F. (2021). Synthetic returns for long-term credit assignment. *arXiv preprint arXiv:2102.12425*.

Rauber, P., Ummadisingu, A., Mutz, F., and Schmidhuber, J. (2017). Hindsight policy gradients. *arXiv preprint arXiv:1711.06006*.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. In *International Conference on Machine Learning*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*.

Rummery, G. A. and Niranjan, M. (1994). *Online Q-learning using connectionist systems*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*.

Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016a). Policy distillation. In *International Conference on Learning Representations*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016b). Progressive neural networks. *CoRR*.

Saeed, A., Grangier, D., Pietquin, O., and Zeghidour, N. (2020). Learning from heterogeneous eeg signals with differentiable channel reordering. In *International Conference on Acoustics, Speech and Signal Processing*.

Savinov, N., Dosovitskiy, A., and Koltun, V. (2018). Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*.

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. (2019). Episodic curiosity through reachability. *International Conference on Learning Representations*.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015a). Universal value function approximators. In *International Conference on Machine Learning*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized experience replay. *International Conference on Learning Representations*.

Schmidhuber, J. (2019). Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*.

Schmitt, S., Hudson, J. J., Zídek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Küttler, H., Zisserman, A., Simonyan, K., and Eslami, S. M. A. (2018). Kickstarting deep reinforcement learning. *CoRR*.

Schrader, M.-P. B. (2018). gym-sokoban. https://github.com/mpSchrader/gym-sokoban.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019). Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seo, M., Vecchietti, L. F., Lee, S., and Har, D. (2019). Rewards prediction-based credit assignment for reinforcement learning with sparse binary rewards. *IEEE Access*.

Sequeira, P. and Gervasio, M. (2020). Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. *Artificial Intelligence*.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. (2018). Time-contrastive networks: Self-supervised learning from video. In *International Conference on Robotics and Automation*.

Sethi, S. P. (2019). What is optimal control theory? In *Optimal Control Theory*.

Shani, L., Efroni, Y., and Mannor, S. (2019). Exploration conscious reinforcement learning revisited. In *International Conference on Machine Learning*.

Sharma, S., Lakshminarayanan, A. S., and Ravindran, B. (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *International Conference on Learning Representations*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017a). Mastering the game of go without human knowledge. *Nature*.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. (2017b). The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

Silver, T., Allen, K., Tenenbaum, J., and Kaelbling, L. (2018). Residual policy learning. *arXiv preprint arXiv:1812.06298*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable markovian decision processes. In *Machine Learning Proceedings 1994*.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*.

Skinner, B. F. (1951). How to teach animals. *Scientific American*.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR.

Song, X., Jiang, Y., Du, Y., and Neyshabur, B. (2020). Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*.

Srinivas, A., Laskin, M., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*.

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., and Schmidhuber, J. (2019). Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International Conference on Machine Learning*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*.

Sutton, R. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*.

Sutton, R. S., Mahmood, A. R., and White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research*.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*.

Taïga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. (2019). Benchmarking bonus-based exploration methods on the arcade learning environment. *arXiv preprint arXiv:1908.02388*.

Talvitie, E. (2014). Model regularization for stable sample rollouts. In *Uncertainty in Artificial Intelligence*.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value iteration networks. In *Advances in Neural Information Processing Systems*.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017). # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762.

Tang, Y. (2020). Self-imitation learning via generalized lower bound q-learning. *arXiv preprint arXiv:2006.07442*.

Tang, Y., Nguyen, D., and Ha, D. (2020). Neuroevolution of self-interpretable agents. In *Genetic and Evolutionary Computation Conference*.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*.

Tieleman, O. 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*.

Tomašev, N., Paquet, U., Hassabis, D., and Kramnik, V. (2020). Assessing game balance with alphazero: Exploring alternative rule sets in chess. *arXiv preprint arXiv:2009.04374*.

Topin, N. and Veloso, M. (2019). Generation of policy-level explanations for reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Torrey, L. and Taylor, M. (2013). Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*.

Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In *Advances in Neural Information Processing Systems*.

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *Association for the Advancement of Artificial Intelligence*.

Van Seijen, H., Fatemi, M., and Tavakoli, A. (2019). Using a logarithmic mapping to enable lower discount factors in reinforcement learning. In *Advances in Neural Information Processing Systems*.

Van Seijen, H., Van Hasselt, H., Whiteson, S., and Wiering, M. (2009). A theoretical and empirical analysis of expected sarsa. In *IEEE symposium on adaptive dynamic programming and reinforcement learning*.

Vasic, M., Petrovic, A., Wang, K., Nikolic, M., Singh, R., and Khurshid, S. (2019). Moet: Mixture of expert trees and its application to verifiable reinforcement learning. *arXiv preprint arXiv:1906.06717*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*.

Venuto, D., Lau, E., Precup, D., and Nachum, O. (2021). Policy gradients incorporating the future. *arXiv preprint arXiv:2108.02096*.

Verma, A., Le, H. M., Yue, Y., and Chaudhuri, S. (2019a). Imitation-projected programmatic reinforcement learning. In *Advances in Neural Information Processing Systems*.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. (2019b). Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*.

Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., et al. (2016). Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

Vieillard, N., Kozuno, T., Scherrer, B., Pietquin, O., Munos, R., and Geist, M. (2020a). Leverage the average: an analysis of kl regularization in rl. In *Advances in Neural Information Processing Systems*.

Vieillard, N., Pietquin, O., and Geist, M. (2020b). Munchausen reinforcement learning. In *Advances in Neural Information Processing Systems*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nature*.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016a). Learning to reinforcement learn. *CoRR*, abs/1611.05763.

Wang, N., Pynadath, D. V., and Hill, S. G. (2016b). The impact of pomdp-generated explanations on trust and performance in human-robot teams. In *International Conference on Autonomous Agents and Multiagent Systems*.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*.

Wayne, G., Hung, C.-C., Amos, D., Mirza, M., Ahuja, A., Grabska-Barwinska, A., Rae, J., Mirowski, P., Leibo, J. Z., Santoro, A., et al. (2018). Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*.

Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. (2017). Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*.

Wei, D., Lim, J. J., Zisserman, A., and Freeman, W. T. (2018). Learning and using the arrow of time. In *Conference on Computer Vision and Pattern Recognition*.

Wiewiora, E., Cottrell, G. W., and Elkan, C. (2003). Principled methods for advising reinforcement learning agents. In *International Conference on Machine Learning*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems*.

Xu, D., Xiao, J., Zhao, Z., Shao, J., Xie, D., and Zhuang, Y. (2019). Self-supervised spatiotemporal learning via video clip order prediction. In *Conference on Computer Vision and Pattern Recognition*.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021). Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*.

Young, K. (2019). Variance reduced advantage estimation with δ hindsight credit assignment. *arXiv preprint arXiv:1911.08362*.

Zahavy, T., Ben-Zrihem, N., and Mannor, S. (2016). Graying the black box: Understanding dqns. In *International Conference on Machine Learning*.

Zeelenberg, M. (1999). Anticipated regret, expected feedback and behavioral decision making. In *Journal of Behavioral Decision Making*.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. (2018). A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.

Zheng, Z., Vuorio, R., Lewis, R., and Singh, S. (2021). Pairwise weights for temporal credit assignment. *arXiv preprint arXiv:2102.04999*.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. (2008). Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Zoph, B., Yuret, D., May, J., and Knight, K. (2016). Transfer learning for low-resource neural machine translation. *ACL*.

Zou, H., Ren, T., Yan, D., Su, H., and Zhu, J. (2019). Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*.

Chapter 10

Acknowledgements

"It is understated how important being a team member who brings fun, joy, courage or optimism to stressful moments [...] is."
- Suhail Doshi

I am forever grateful to my two PhD advisors, Olivier and Philippe, who gave me the opportunity to work alongside them, and took the time to educate me on what being a researcher means, and what good science is. To Philippe, without you noticing my email to the Scool team (at the time SequeL!), my life would be quite different now (no need for fancy credit assignment to infer that!). To Olivier, thanks for taking a chance on me, and leaving me lots of room for exploration. There would be many things to say and far from enough space for that. Thanks again.

I want to offer special thanks to my third, non-official PhD advisor: Matthieu. You really tried to convey to me what the essence of Reinforcement Learning was and I think some of it actually got through.

Next are my co-PhD students, my partners in crime, Nino and Léonard. Thanks for the infinite discussions, debates, conference parties (well, at least for the first part of this PhD!)... during those three years and some together. I hope we (and I am sure we will) finally write a research paper together someday!

I want to thank all my co-authors and close collaborators: Nathan, Yannis, Raphaël, Johan, Toby and the others. It was really a pleasure to work with all of you, and I learned tremendously in the process, both as a researcher and a human being. Special thanks to Yannis for kindly allowing me to use his front page template for this thesis.

I am of course grateful to my family members, who supported me all along the PhD adventure, and who genuinely made the effort to try to understand what it was I was doing during these three years.

To Lucie, among so many essential things, you remind me everyday of "optimism in the face of uncertainty". This is invaluable to me.

To my friends, who are always here for me, I am eternally grateful to each and every one of you.

I also want to give a heartfelt thanks to my Google collaborators. Google Brain has been an awesome environment to conduct research and the team in Paris is a set of amazing people.

Next are my Inria teammates. Though I did not get to spend enough time with all of you, I did enjoy tremendously my visits to the lab and collaborations. Thanks so much!

I would like to thank all the artists whose music accompanied me during writing and coding times as a background: Tyler the Creator, Loscil, Autechre, Rafael Anton Irisarri, Lone, Between the Buried and Me and many many others.

Appendix A

Adversarially-Guided Actor-Critic

We now present an algorithmic-driven mechanism for improved exploration in RL. We think that this complements nicely the contributions of this thesis on the exploration side (in Chapters 5 and 8). This annex was submitted as a conference paper to ICLR 2021 (Flet-Berliac et al., 2021a).

Despite definite success in deep reinforcement learning problems, actor-critic algorithms are still confronted with sample inefficiency in complex environments, particularly in tasks where efficient exploration is a bottleneck. These methods consider a policy (the actor) and a value function (the critic) whose respective losses are built using different motivations and approaches. This section introduces a third protagonist: the adversary. While the adversary mimics the actor by minimizing the KL-divergence between their respective action distributions, the actor, in addition to learning to solve the task, tries to differentiate itself from the adversary predictions. This novel objective stimulates the actor to follow strategies that could not have been correctly predicted from previous trajectories, making its behavior innovative in tasks where the reward is extremely rare. Our experimental analysis shows that the resulting Adversarially Guided Actor-Critic (`AGAC`) algorithm leads to more exhaustive exploration. Notably, `AGAC` outperforms current state-of-the-art methods on a set of various hard-exploration and procedurally-generated tasks.

A.1 Introduction

Research in deep reinforcement learning (RL) has proven to be successful across a wide range of problems (Silver et al., 2014; Schulman et al., 2016; Lillicrap et al., 2016; Mnih et al., 2015). Nevertheless, generalization and exploration in RL still represent key challenges that leave most current methods ineffective. First, a battery of recent studies (Farebrother et al., 2018; Zhang et al., 2018; Song et al., 2020; Cobbe et al., 2020) indicates that current RL methods fail to generalize correctly even when agents have been trained in a diverse set of environments. Second, exploration has been extensively studied in RL; however, most hard-exploration problems use the same environment for training and evaluation. Hence, since a well-designed exploration strategy should maximize the information received from a trajectory about an environment, the exploration capabilities may not be appropriately assessed if that information is memorized. In this line of research, we choose to study the exploration capabilities of our method and its ability to generalize to new scenarios. Our evaluation domains will, therefore, be tasks with sparse reward in procedurally-generated environments.

In this work, we propose Adversarially Guided Actor-Critic (`AGAC`), which reconsiders the actor-critic framework by introducing a third protagonist: the adversary. Its role is to predict the actor's actions correctly. Meanwhile, the actor must not only find the optimal actions to maximize the sum of expected returns, but also counteract the predictions of the adversary. This formulation is lightly inspired by adversarial methods, specifically generative adversarial networks (GANs) (Goodfellow et al., 2014). Such

a link between GANs and actor-critic methods has been formalized by Pfau and Vinyals (2016); however, in the context of a third protagonist, we draw a different analogy. The adversary can be interpreted as playing the role of a discriminator that must predict the actions of the actor, and the actor can be considered as playing the role of a generator that behaves to deceive the predictions of the adversary. This approach has the advantage, as with GANs, that the optimization procedure generates a diversity of meaningful data, corresponding to sequences of actions in `AGAC`.

This paper analyses and explores how `AGAC` explicitly drives diversity in the behaviors of the agent while remaining reward-focused, and to which extent this approach allows to adapt to the evolving state space of procedurally-generated environments where the map is constructed differently with each new episode. Moreover, because stability is a legitimate concern since specific instances of adversarial networks were shown to be prone to hyperparameter sensitivity issues (Arjovsky and Bottou, 2017), we also examine this aspect in our experiments.

The contributions of this work are as follow: (i) we propose a novel actor-critic formulation inspired from adversarial learning (`AGAC`), (ii) we analyse empirically `AGAC` on key reinforcement learning aspects such as diversity, exploration and stability, (iii) we demonstrate significant gains in performance on several sparse-reward hard-exploration tasks including procedurally-generated tasks.

A.2 Additional related work

Actor-critic methods (Barto et al., 1983; Sutton, 1984) have been extended to the deep learning setting by Mnih et al. (2015), who combined deep neural networks and multiple distributed actors with an actor-critic setting, with strong results on Atari. Since then, many additions have been proposed, be it architectural improvements (Vinyals et al., 2019), better advantage or value estimation (Schulman et al., 2016; Flet-Berliac et al., 2021b), or the incorporation of off-policy elements (Wang et al., 2016a; Oh et al., 2018; Flet-Berliac and Preux, 2020). Regularization was shown to improve actor-critic methods, either by enforcing trust regions (Schulman et al., 2016, 2017; Wu et al., 2017), or by correcting for off-policiness (Munos et al., 2016; Gruslys et al., 2017); and recent works analyzed its impact from a theoretical standpoint (Geist et al., 2019; Ahmed et al., 2019; Vieillard et al., 2020b,a). Related to our work, Han and Sung (2020) use the entropy of the mixture between the policy induced from a replay buffer and the current policy as a regularizer. To the best of our knowledge, none of these methods explored the use of an adversarial objective to drive exploration.

While introduced in supervised learning, adversarial learning (Goodfellow et al., 2015; Miyato et al., 2016; Kurakin et al., 2017) was leveraged in several RL works. Ho and Ermon (2016) propose an imitation learning method that uses a discriminator whose task is to distinguish between expert trajectories and those of the agent while the agent tries to match expert behavior to fool the discriminator. Bahdanau et al. (2019) use a discriminator to distinguish goal states from non-goal states based on a textual instruction, and use the resulting model as a reward function. Florensa et al. (2018) use a GAN to produce sub-goals at the right level of difficulty for the current agent, inducing a form of curriculum. Additionally, Pfau and Vinyals (2016) provide a parallel between GANs and the actor-critic framework.

While exploration is driven in part by the core RL algorithms (Fortunato et al., 2018; Han and Sung, 2020; Ferret et al., 2021), it is often necessary to resort to exploration-specific techniques. For instance, intrinsic motivation encourages exploratory behavior from the agent. Some works use state-visitation counts or pseudo-counts to promote exhaustive exploration (Bellemare et al., 2016a), while others use curiosity rewards, expressed in the magnitude of prediction error from the agent, to push it towards unfamiliar areas of the state space (Burda et al., 2019). Ecoffet et al. (2019) propose a technique akin to tree traversal to explore while learning to come back to promising areas. Eysenbach et al. (2018) show that encouraging diversity helps with exploration, even in the absence of reward.

Last but not least, generalization is a key challenge in RL. Zhang et al. (2018) showed that, even when

the environment is not deterministic, agents can overfit to their training distribution and that it is difficult to distinguish agents likely to generalize to new environments from those that will not. In the same vein, recent work has advocated using procedurally-generated environments, in which a new instance of the environment is sampled when a new episode starts, to assess generalization capabilities better (Justesen et al., 2018; Cobbe et al., 2020). Finally, methods based on network randomization (Igl et al., 2019), noise injection (Lee et al., 2020a), and credit assignment (Ferret et al., 2020) have been proposed to reduce the generalization gap for RL agents.

A.3 Additional background and notations

We place ourselves in the infinite-horizon setting, *i.e.*, we seek a policy that optimizes $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right)]$. The entropy $\mathcal{H}^\pi$ of a policy is calculated as: $\mathcal{H}^\pi(s) = \mathbb{E}_{\pi(\cdot|s)}\left[-\log \pi(\cdot|s)\right]$.

Actor-critic and deep policy gradients. We recall that an actor-critic algorithm is composed of two main components: a policy and a value predictor. In deep RL, both the policy and the value function are obtained via parametric estimators; we denote θ and ϕ their respective parameters. The policy is updated via policy gradient, while the value is usually updated via temporal difference or Monte Carlo rollouts. In practice, for a sequence of transitions $\{s_t, a_t, r_t, s_{t+1}\}_{t\in[0,N]}$, we use the following policy gradient loss (including the commonly used entropic penalty):

$$\mathcal{L}_{PG} = -\frac{1}{N}\sum_{t'=t}^{t+N}(A_{t'} \log \pi\left(a_{t'}|s_{t'}, \theta\right) + \alpha\mathcal{H}^\pi(s_{t'}, \theta)),$$

where α is the entropy coefficient and A_t is the generalized advantage estimator (Schulman et al., 2016) defined as: $A_t = \sum_{t'=t}^{t+N}(\gamma\lambda)^{t'-t}(r_{t'} + \gamma V_{\phi_{\text{old}}}(s_{t'+1}) - V_{\phi_{\text{old}}}(s_{t'}))$, with λ a fixed hyperparameter and $V_{\phi_{\text{old}}}$ the value function estimator at the previous optimization iteration. To estimate the value function, we solve the non-linear regression problem $\text{minimize}_\phi \sum_{t'=t}^{t+N}(V_\phi(s_{t'}) - \hat{V}_{t'})^2$ where $\hat{V}_t = A_t + V_{\phi_{\text{old}}}(s_{t'})$.

A.4 Adversarially Guided Actor-Critic

To foster diversified behavior in its trajectories, `AGAC` introduces a third protagonist to the actor-critic framework: the adversary. The role of the adversary is to accurately predict the actor's actions, by *minimizing* the discrepancy between its action distribution π_{adv} and the distribution induced by the policy π. Meanwhile, in addition to finding the optimal actions to *maximize* the sum of expected returns, the actor must also counteract the adversary's predictions by *maximizing* the discrepancy between π and π_{adv} (see Appendix A.9 for an illustration). This discrepancy, used as a form of exploration bonus, is defined as the difference of action log-probabilities (see Equ. A.4), whose expectation is the Kullback–Leibler divergence:

$$D_{\text{KL}}(\pi(\cdot|s)\|\pi_{\text{adv}}(\cdot|s)) = \mathbb{E}_{\pi(\cdot|s)}\left[\log \pi(\cdot|s) - \log \pi_{\text{adv}}(\cdot|s)\right].$$

Formally, for each state-action pair (s_t, a_t) in a trajectory, an action-dependent bonus $\log \pi(a_t|s_t) - \log \pi_{\text{adv}}(a_t|s_t)$ is added to the advantage. In addition, the value target of the critic is modified to include the action-independent equivalent, which is the KL-divergence $D_{\text{KL}}(\pi(\cdot|s_t)\|\pi_{\text{adv}}(\cdot|s_t))$. We discuss the role of these mirrored terms below, and the implications of `AGAC`'s modified objective from a more theoretical standpoint in the next section. In addition to the parameters θ (resp. θ_{old} the parameter of the policy at the previous iteration) and ϕ defined above (resp. ϕ_{old} that of the critic), we denote ψ (resp. ψ_{old}) that of the adversary.

AGAC minimizes the following loss:

$$\mathcal{L}_{\texttt{AGAC}} = \mathcal{L}_{\text{PG}} + \beta_V \mathcal{L}_V + \beta_{\text{adv}} \mathcal{L}_{\text{adv}}.$$

In the new objective $\mathcal{L}_{PG} = -\frac{1}{N}\sum_{t=0}^{N}(A_t^{\texttt{AGAC}} \log \pi\left(a_t|s_t,\theta\right) + \alpha\mathcal{H}^{\pi}(s_t,\theta))$, AGAC modifies A_t as:

$$A_t^{\texttt{AGAC}} = A_t + c\left(\log \pi(a_t|s_t,\theta_{\text{old}}) - \log \pi_{\text{adv}}(a_t|s_t,\psi_{\text{old}})\right),$$

with c a varying hyperparameter that controls the dependence on the action log-probability difference. To encourage exploration without preventing asymptotic stability, c is linearly annealed during the course of training. $\mathcal{L}_V$ is the objective function of the critic defined as:

$$\mathcal{L}_V = \frac{1}{N}\sum_{t=0}^{N}\left(V_\phi(s_t) - \left(\hat{V}_t + c\, D_{\text{KL}}(\pi(\cdot|s_t,\theta_{\text{old}})\|\pi_{\text{adv}}(\cdot|s_t,\psi_{\text{old}}))\right)\right)^2.$$

Finally, $\mathcal{L}_{\text{adv}}$ is the objective function of the adversary:

$$\mathcal{L}_{\text{adv}} = \frac{1}{N}\sum_{t=0}^{N} D_{\text{KL}}(\pi(\cdot|s_t,\theta_{\text{old}})\|\pi_{\text{adv}}(\cdot|s_t,\psi)).$$

Equ. A.4, Equ. A.4 and Equ. A.4 are the three equations that our method modifies (we color in blue the specific parts) in the traditional actor-critic framework. The terms β_V and β_{adv} are fixed hyperparameters.

Under the proposed actor-critic formulation, the probability of sampling an action is increased if the modified advantage is positive, *i.e.* (i) the corresponding return is larger than the predicted value and/or (ii) the action log-probability difference is large. More precisely, our method favors transitions whose actions were less accurately predicted than the average action, *i.e.* $\log \pi(a|s) - \log \pi_{\text{adv}}(a|s) \geq D_{\text{KL}}(\pi(\cdot|s)\|\pi_{\text{adv}}(\cdot|s))$. This is particularly visible for $\lambda \to 1$, in which case the generalized advantage is $A_t = G_t - V_{\phi_{\text{old}}}(s_t)$, resulting in the appearance of both aforementioned mirrored terms in the modified advantage:

$$A_t^{\texttt{AGAC}} = G_t - \hat{V}_t^{\phi_{\text{old}}} + c\left(\log \pi(a_t|s_t) - \log \pi_{\text{adv}}(a_t|s_t) - \hat{D}_{\text{KL}}^{\phi_{\text{old}}}(\pi(\cdot|s_t)\|\pi_{\text{adv}}(\cdot|s_t))\right),$$

with G_t the observed return, $\hat{V}_t^{\phi_{\text{old}}}$ the estimated return and $\hat{D}_{\text{KL}}^{\phi_{\text{old}}}(\pi(\cdot|s_t)\|\pi_{\text{adv}}(\cdot|s_t))$ the estimated KL-divergence (estimated components of $V_{\phi_{\text{old}}}(s_t)$ from Eq. A.4).

To avoid instability, in practice the adversary is a separate estimator, updated with a smaller learning rate than the actor. This way, it represents a delayed and more steady version of the actor's policy, which prevents the agent from having to constantly adapt or focus solely on fooling the adversary.

A.4.1 Building motivation

In the following, we provide an interpretation of AGAC by studying the dynamics of attraction and repulsion between the actor and the adversary. To simplify, we study the equivalent of AGAC in a policy iteration (PI) scheme. PI being the dynamic programming scheme underlying the standard actor-critic, we have reasons to think that some of our findings translate to the original AGAC algorithm. In PI, the quantity of interest is the action-value, which AGAC would modify as:

$$Q_{\pi_k}^{\texttt{AGAC}} = Q_{\pi_k} + c\,(\log \pi_k - \log \pi_{\text{adv}}),$$

with π_k the policy at iteration k. Incorporating the entropic penalty, the new policy π_{k+1} verifies:

$$\pi_{k+1} = \arg\max_{\pi} \mathcal{J}_{\text{PI}}(\pi) = \arg\max_{\pi} \mathbb{E}_s \mathbb{E}_{a\sim\pi(\cdot|s)}[Q_{\pi_k}^{\texttt{AGAC}}(s,a) - \alpha \log \pi(a|s)].$$

We can rewrite this objective:

$$\begin{aligned}
\mathcal{J}_{\mathrm{PI}}(\pi) &= \mathbb{E}_s \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\texttt{AGAC}}_{\pi_k}(s,a) - \alpha \log \pi(a|s)] \\
&= \mathbb{E}_s \mathbb{E}_{a\sim\pi(\cdot|s)}[Q_{\pi_k}(s,a) + c\,(\log \pi_k(a|s) - \log \pi_{\mathrm{adv}}(a|s)) - \alpha \log \pi(a|s)] \\
&= \mathbb{E}_s \mathbb{E}_{a\sim\pi(\cdot|s)}[Q_{\pi_k}(s,a) + c\,(\log \pi_k(a|s) - \log \pi(a|s) + \log \pi(a|s) - \log \pi_{\mathrm{adv}}(a|s)) - \alpha \log \pi(a|s)] \\
&= \mathbb{E}_s \Big[\mathbb{E}_{a\sim\pi(\cdot|s)}[Q_{\pi_k}(s,a)] \underbrace{- c\, D_{\mathrm{KL}}(\pi(\cdot|s)||\pi_k(\cdot|s))}_{\pi_k \text{ is attractive}} \underbrace{+ c\, D_{\mathrm{KL}}(\pi(\cdot|s)||\pi_{\mathrm{adv}}(\cdot|s))}_{\pi_{\mathrm{adv}} \text{ is repulsive}} \underbrace{+ \alpha\, \mathcal{H}(\pi(\cdot|s))}_{\text{enforces stochastic policies}} \Big].
\end{aligned}$$

Thus, in the PI scheme, AGAC finds a policy that maximizes Q-values, while at the same time remaining close to the current policy and far from a mixture of the previous policies (*i.e.*, π_{k-1}, π_{k-2}, π_{k-3}, ...). Note that we experimentally observe (see Section A.5.2) that our method performs better with a smaller learning rate for the adversarial network than that of the other networks, which could imply that a stable repulsive term is beneficial.

This optimization problem is strongly concave in π (thanks to the entropy term), and is state-wise a Legendre-Fenchel transform. Its solution is given by (see Appendix A.12 for the full derivation):

$$\pi_{k+1} \propto \left(\frac{\pi_k}{\pi_{\mathrm{adv}}}\right)^{\frac{c}{\alpha}} \exp \frac{Q_{\pi_k}}{\alpha}.$$

This result gives us some insight into the behavior of the objective function. Notably, in our example, if π_{adv} is fixed and $c = \alpha$, we recover a KL-regularized PI scheme (Geist et al., 2019) with the modified reward $r - c \log \pi_{\mathrm{adv}}$.

A.4.2 Implementation

In all of the experiments, we use PPO (Schulman et al., 2017) as the base algorithm and build on it to incorporate our method. Hence,

$$\mathcal{L}_{PG} = -\frac{1}{N} \sum_{t'=t}^{t+N} \min \left(\frac{\pi(a_{t'}|s_{t'},\theta)}{\pi(a_{t'}|s_{t'},\theta_{\mathrm{old}})} A^{\texttt{AGAC}}_{t'}, \mathrm{clip}\Big(\frac{\pi(a_{t'}|s_{t'},\theta)}{\pi(a_{t'}|s_{t'},\theta_{\mathrm{old}})}, 1-\epsilon, 1+\epsilon\Big) A^{\texttt{AGAC}}_{t'} \right),$$

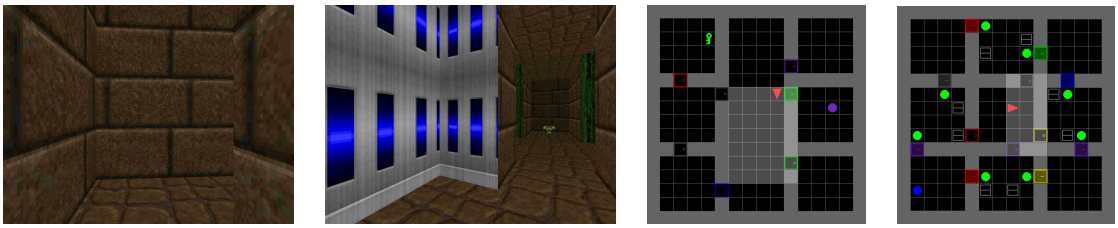
with $A^{\texttt{AGAC}}_{t'}$ given in Equ. A.4, N the temporal length considered for one update of parameters and ϵ the clipping parameter. Similar to RIDE (Raileanu and Rocktäschel, 2019), we also discount PPO by episodic state visitation counts, except for VizDoom (*cf.* Section A.5.1). The actor, critic and adversary use the convolutional architecture of the Nature paper of DQN (Mnih et al., 2015) with different hidden sizes (see Appendix A.11 for architecture details). The three neural networks are optimized using Adam (Kingma and Ba, 2014). Our method does not use RNNs in its architecture; instead, in all our experiments, we use frame stacking. Indeed, Hausknecht and Stone (2015) interestingly demonstrate that although recurrence is a reliable method for processing state observation, it does not confer any systematic advantage over stacking observations in the input layer of a CNN. Note that the parameters are not shared between the policy, the critic and the adversary and that we did not observe any noticeable difference in computational complexity when using AGAC compared to PPO. We direct the reader to Appendix A.10 for a list of hyperparameters. In particular, the c coefficient of the adversarial bonus is linearly annealed.

At each training step, we perform a stochastic optimization step to minimize $\mathcal{L}_{\texttt{AGAC}}$ using stop-gradient:

$$\theta \leftarrow \mathrm{Adam}\left(\theta, \nabla_\theta \mathcal{L}_{PG}, \eta_1\right), \qquad \phi \leftarrow \mathrm{Adam}\left(\phi, \nabla_\phi \mathcal{L}_V, \eta_1\right), \qquad \psi \leftarrow \mathrm{Adam}\left(\psi, \nabla_\psi \mathcal{L}_{\mathrm{adv}}, \eta_2\right).$$

A.5 Experiments

In this section, we describe our experimental study in which we investigate: (i) whether the adversarial bonus alone (*e.g.* without episodic state visitation count) is sufficient to outperform other methods in VizDoom, a sparse-reward task with high-dimensional observations, (ii) whether `AGAC` succeeds in partially-observable and procedurally-generated environments with high sparsity in the rewards, compared to other methods, (iii) how well `AGAC` is capable of exploring in environments without extrinsic reward, (iv) the training stability of our method. In all of the experiments, lines are average performances and shaded areas represent one standard deviation. The code for our method is released at github.com/yfletberliac/adversarially-guided-actor-critic.



(a) (b) (c) (d)

Figure A.1: (a,b) Frames from the 3-D navigation task VizdoomMyWayHome. (c) MiniGrid-KeyCorridorS6R3. (d) MiniGrid-ObstructedMazeFull.

Environments. To carefully evaluate the performance of our method, its ability to develop robust exploration strategies and its generalization to unseen states, we choose tasks that have been used in prior work, which are tasks with high-dimensional observations, sparse reward and procedurally-generated environments. In **VizDoom** (Kempka et al., 2016), the agent must learn to move along corridors and through rooms without any reward feedback from the 3-D environment. The **MiniGrid** environments (Chevalier-Boisvert et al., 2018) are a set of challenging partially-observable and sparse-reward gridworlds. In this type of procedurally-generated environments, memorization is impossible due to the huge size of the state space, so the agent must learn to generalize across the different layouts of the environment. Each gridworld has different characteristics: in the MultiRoom tasks, the agent is placed in the first room and should reach a goal placed in the most distant room. In the KeyCorridor tasks, the agent must navigate to pick up an object placed in a room locked by a door whose key is in another room. Finally, in the ObstructedMaze tasks, the agent must pick up a box that is placed in a corner of a 3x3 maze in which the doors are also locked, the keys are hidden in boxes and balls obstruct the doors. All considered environments (see Fig. A.1 for some examples) are available as part of OpenAI Gym (Brockman et al., 2016a).

Baselines. For a fair assessment of our method, we compare to some of the most prominent methods specialized in hard-exploration tasks: **RIDE** (Raileanu and Rocktäschel, 2019), based on an intrinsic reward associated with the magnitude of change between two consecutive state representations and state visitation, **Count** as Count-Based Exploration (Bellemare et al., 2016a), which we couple with IMPALA (Espeholt et al., 2018), **RND** (Burda et al., 2019) in which an exploration bonus is positively correlated to the error of predicting features from the observations and **ICM** (Pathak et al., 2017), where a module only predicts the changes in the environment that are produced by the actions of the agent. Finally, we compare to most the recent and best performing method at the time of writing in procedurally-generated environments: **AMIGo** (Campero et al., 2021) in which a goal-generating teacher provides count-based intrinsic goals.

A.5.1 Adversarially-based exploration (no episodic count)

Table A.1: Average return in VizDoom at different timesteps.

Nb. of Timesteps	2M	4M	6M	8M	10M
AGAC	**0.74** $\pm$ 0.05	**0.96** $\pm$ 0.001	**0.96** $\pm$ 0.001	**0.97** $\pm$ 0.001	**0.97** $\pm$ 0.001
RIDE	0.	0.	0.95 $\pm$ 0.001	**0.97** $\pm$ 0.001	**0.97** $\pm$ 0.001
ICM	0.	0.	0.95 $\pm$ 0.001	**0.97** $\pm$ 0.001	**0.97** $\pm$ 0.001
AMIGo	0.	0.	0.	0.	0.
RND	0.	0.	0.	0.	0.
Count	0.	0.	0.	0.	0.

In this section, we assess the benefits of using an adversarially-based exploration bonus and examine how `AGAC` performs without the help of count-based exploration. In order to provide a comparison to state-of-the-art methods, we choose VizDoom, a hard-exploration problem used in prior work. In this game, the map consists of 9 rooms connected by corridors where 270 steps separate the initial position of the agent and the goal under an optimal policy. Episodes are terminated either when the agent finds the goal or if the episode exceeds 2100 timesteps. Importantly, while other algorithms (Raileanu and Rocktäschel, 2019; Campero et al., 2021) benefit from count-based exploration, this study has been conducted with our method not benefiting from episodic count whatsoever. Results in Table A.1 indicate that `AGAC` clearly outperforms other methods in sample-efficiency. Only the methods ICM and RIDE succeed in matching the score of `AGAC`, and with about twice as much transitions ($\sim$ 3M vs. 6M). Interestingly, AMIGo performs similarly to Count and RND. We find this result surprising because AMIGo has proven to perform well in the MiniGrid environments. Nevertheless, it appears that concurrent works to ours experienced similar issues with the accompanying implementation[1]. The results of `AGAC` support the capabilities of the adversarial bonus and show that it can, on its own, achieve significant gains in performance. However, the VizDoom task is not procedurally-generated; hence we have not evaluated the generalization to new states yet. In the following section, we use MiniGrid to investigate this.

A.5.2 Hard-exploration tasks with partially-observable environments

We now evaluate our method on multiple hard-exploration procedurally-generated tasks from MiniGrid. Details about MiniGrid can be found in Appendix A.10.1. Fig. A.2 indicates that `AGAC` significantly outperforms other methods on these tasks in sample-efficiency and performance. `AGAC` also outperforms the current state-of-the-art method, AMIGo, despite the fact that it uses the fully-observable version of MiniGrid. Note that we find the same poor performance results when training AMIGo in MiniGrid, similar to Vizdoom results. For completeness, we also report in Table A.2 of Appendix A.7.1 the performance results with the scores reported in the original papers Raileanu and Rocktäschel (2019) and Campero et al. (2021). We draw similar conclusions: `AGAC` clearly outperforms the state-of-the-art RIDE, AMIGo, Count, RND and ICM.

In all the considered tasks, the agent must learn to generalize across a very large state space because the layouts are generated procedurally. We consider three main arguments to explain why our method is successful: (i) our method makes use of partial observations: in this context, the adversary has a harder time predicting the actor's actions; nevertheless, the mistakes of the former benefit the latter in the form of an exploration bonus, which pushes the agent to explore further in order to deceive the adversary, (ii) the exploration bonus (*i.e.* intrinsic reward) does not dissipate compared to most other methods, as observed in Fig. A.9 in Appendix A.8, (iii) our method does not make assumptions about the environment dynamics (*e.g.*, changes in the environment produced by an action as in Raileanu and Rocktäschel (2019))

[1] AMIGo implementation GitHub Issue.

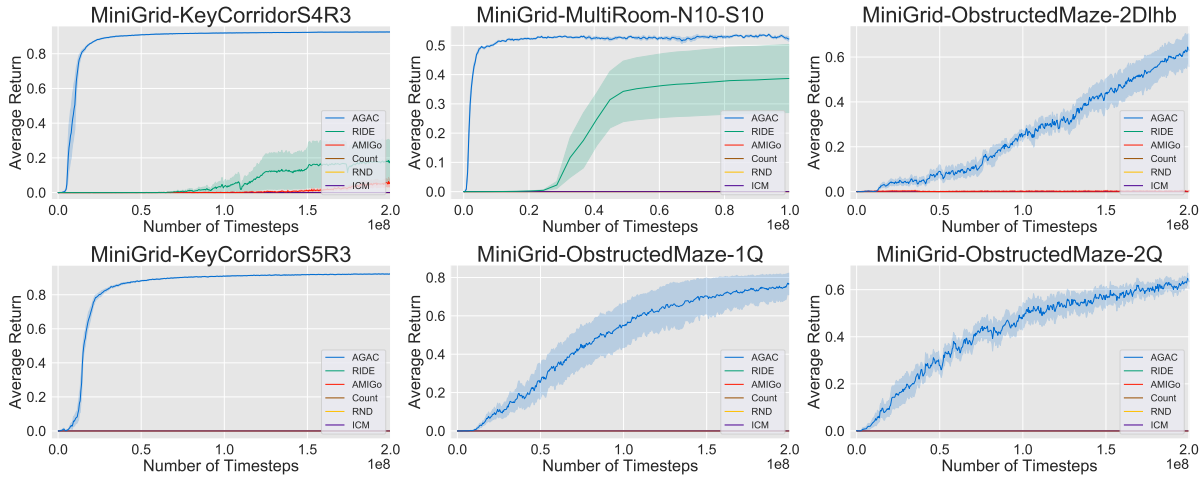


Figure A.2: Performance evaluation of AGAC.

since this can hinder learning when the space of state changes induced by an action is too large (such as the action of moving a block in ObstructedMaze).

In Appendix A.7.3, we also include experiments in two environments with extremely sparse reward signals: KeyCorridorS8R3 and ObstructedMazeFull. Despite the challenge, AGAC still manages to find rewards and can perform well by taking advantage of the diversified behaviour induced by our method. To the best of our knowledge, no other method ever succeeded to perform well (> 0 average return) in those tasks. We think that given more computing time, AGAC's score could go higher.

Training stability

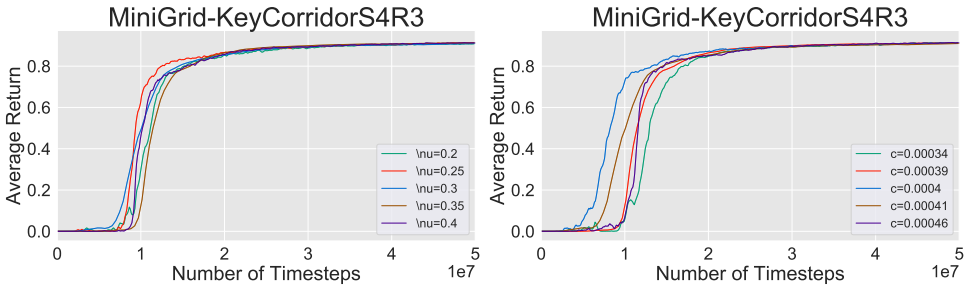


Figure A.3: Sensitivity analysis of AGAC in KeyCorridorS4R3.

Here we want to analyse the stability of the method when changing hyperparameters. The most important parameters in AGAC are c, the coefficient for the adversarial bonus, and the learning rates ratio $\nu = \frac{\eta_2}{\eta_1}$. We choose KeyCorridorS4R3 as the evaluation task because among all the tasks considered, its difficulty is at a medium level. Fig. A.3 shows the learning curves. For readability, we plot the average return only; the standard deviation is sensibly the same for all curves. We observe that deviating from the hyperparameter values found using grid search results in a slower training. Moreover, although reasonable, c appears to have more sensitivity than ν.

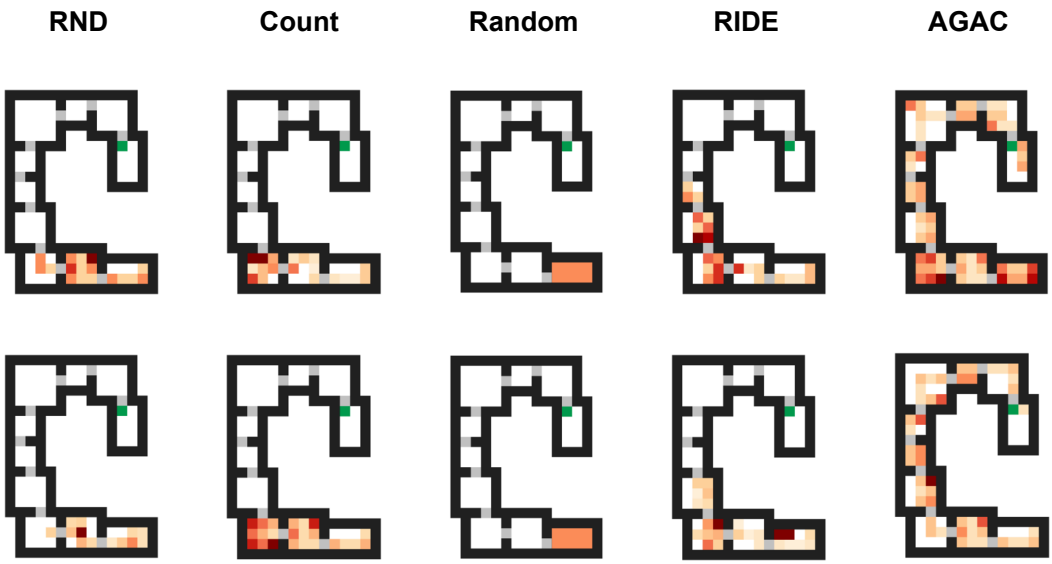


Figure A.5: State visitation heatmaps for RND, Count, a random uniform policy, RIDE, and `AGAC` trained in a singleton environment (top row) and procedurally-generated environments (bottom row) without extrinsic reward for 10M timesteps in the MultiRoomN10S6 task.

A.5.3 Exploration in reward-free environments

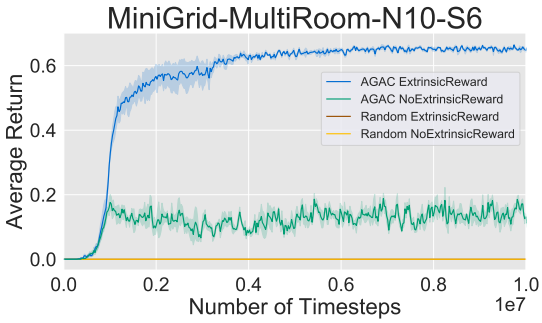


Figure A.4: Average return on N10S6 with and without extrinsic reward.

To better understand the effectiveness of our method and inspect how the agent collects rewards that would not otherwise be achievable by simple exploration heuristics or other methods, we analyze the performance of `AGAC` in another (procedurally-generated) challenging environment, MultiRoomN10S6, when there is no reward signal, *i.e.* no extrinsic reward. Beyond the good performance of our method when extrinsic rewards are given to the agent, Fig. A.4 indicates that the exploration induced by our method makes the agent succeed in a significant proportion of the episodes: in the configuration "NoExtrinsicReward" the reward signal is not given (the goal is invisible to the agent) and the performance of `AGAC` stabilizes around an average return of ~ 0.15. Since the return of an episode is either 0 or 1 (depending on whether the agent reached the goal state or not), and because this value is aggregated across several episodes, the results indicate that reward-free `AGAC` succeeds in $\sim 15\%$ of the tasks. Comparatively, random agents have a zero average return. This poor performance is in accordance with the results in Raileanu and Rocktäschel (2019) and reflects the complexity of the task: in order to go from one room to another, an agent must perform a specific action to open a door and cross it within the time limit of 200 timesteps. In the following, we visually investigate how different methods explore the environments.

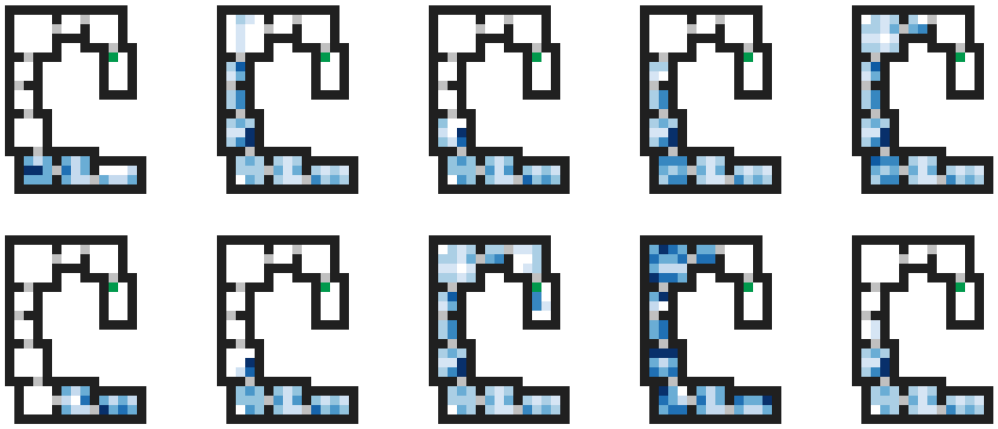


Figure A.6: State visitation heatmaps of the last ten episodes of an agent trained in *procedurally-generated* environments without extrinsic reward for 10M timesteps in the MultiRoomN10S6 task. The agent is continuously engaging in new strategies.

A.5.4 Visualizing coverage and diversity

In this section, we first investigate how different methods explore environments without being guided by extrinsic rewards (the green goal is invisible to the agent) on both *procedurally-generated* and *singleton* environments. In singleton environments, an agent has to solve the same task in the same environment/maze in every episode. Fig. A.5 shows the state visitation heatmaps (darker areas correspond to more visits) after a training of 10M timesteps. We observe that most of the methods explore inefficiently in a singleton environment and that only RIDE succeeds in reaching the fifth room while `AGAC` reaches the last (tenth) room. After training the agents in procedurally-generated environments, the methods explore even less efficiently while `AGAC` succeeds in exploring all rooms.

We now qualitatively study the diversity of an agent's behavior when trained with `AGAC`. Fig. A.6 presents the state visitation heatmaps of the last ten episodes for an agent trained in *procedurally-generated* environments in the MultiRoomN10S6 task without extrinsic reward. The heatmaps correspond to the behavior of the resulting policy, which is still learning from the `AGAC` objective. Looking at the figure, we can see that the strategies vary at each update with, for example, back-and-forth and back-to-start behaviors. Although there are no extrinsic reward, the strategies seem to diversify from one update to the next. Finally, Fig. A.7 in Appendix A.7.2 shows the state visitation heatmaps in a different configuration: when the agent has been trained on a *singleton* environment in the MultiRoomN10S6 task without extrinsic reward. Same as previously, the agent is updated between each episode. Looking at the figure, we can make essentially the same observations as previously, with a noteworthy behavior in the fourth heatmap of the bottom row where it appears the agent went to the fourth room to remain inside it. Those episodes indicate that, although the agent sees the same environment repeatedly, the successive adversarial updates force it to continuously adapt its behavior and try new strategies.

A.6 Discussion

This paper introduced `AGAC`, a modification to the traditional actor-critic framework: an adversary network is added as a third protagonist. The mechanics of `AGAC` have been discussed from a policy iteration point of view, and we provided theoretical insight into the inner workings of the proposed algorithm: the

adversary forces the agent to remain close to the current policy while moving away from the previous ones. In a nutshell, the influence of the adversary makes the actor *conservatively diversified*.

In the experimental study, we have evaluated the adversarially-based bonus in VizDoom and empirically demonstrated its effectiveness and superiority compared to other relevant methods (some benefiting from count-based exploration). Then, we have conducted several performance experiments using `AGAC` and have shown a significant performance improvement over some of the most popular exploration methods (RIDE, AMIGo, Count, RND and ICM) on a set of various challenging tasks from MiniGrid. These procedurally-generated environments have served another purpose which is to validate the capacity of our method to generalize to unseen scenarios. In addition, the training stability of our method has been studied, showing a greater but acceptable sensitivity for c, the adversarial bonus coefficient. Finally, we have investigated the exploration capabilities of `AGAC` in a reward-free setting where the agent demonstrated exhaustive exploration through various strategic choices, confirming that the adversary successfully drives diversity in the behavior of the actor.

A.7 Additional experiments

A.7.1 Performance on MiniGrid

In this section, we report the final performance of all methods considered in the MiniGrid experiments of Fig. A.2 with the scores reported in Raileanu and Rocktäschel (2019) and Campero et al. (2021). All methods have a budget of 200M frames.

Table A.2: Final average performance of all methods on several MiniGrid environments.

Task	KC-S4R3	KC-S5R3	MR-N10S10	OM-2Dlhb	OM-1Q	OM-2Q
AGAC	**0.95**	**0.93**	**0.52**	**0.64**	**0.78**	**0.63**
RIDE	0.19	0.	0.40	0.	0.	0.
AMIGo	0.54	0.	0.	0.20	0.	0.
RND	0.	0.	0.	0.03	0.	0.
Count	0.	0.	0.	0.	0.	0.
ICM	0.	0.	0.	0.	0.	0.

A.7.2 State visitation heatmaps in singleton environments with no extrinsic reward

In this section, we provide additional state visitation heatmaps. The agent has been trained on a singleton environment from the MultiRoomN10S6 task without extrinsic reward. The last ten episodes of the training suggest that although the agent experiences the same maze over and over again, the updates force it to change behavior and try new strategies.

A.7.3 (Extremely) Hard-Exploration Tasks with Partially-Observable Environments

In this section, we include additional experiments on one of the hardest tasks available in MiniGrid. The first is KeyCorridorS8R3, where the size of the rooms has been increased. In it, the agent has to pick up an object which is behind a locked door: the key is hidden in another room and the agent has to explore the environment to find it. The second, ObstructedMazeFull, is similar to ObstructedMaze4Q, where the agent has to pick up a box which is placed in one of the four corners of a 3x3 maze: the doors are

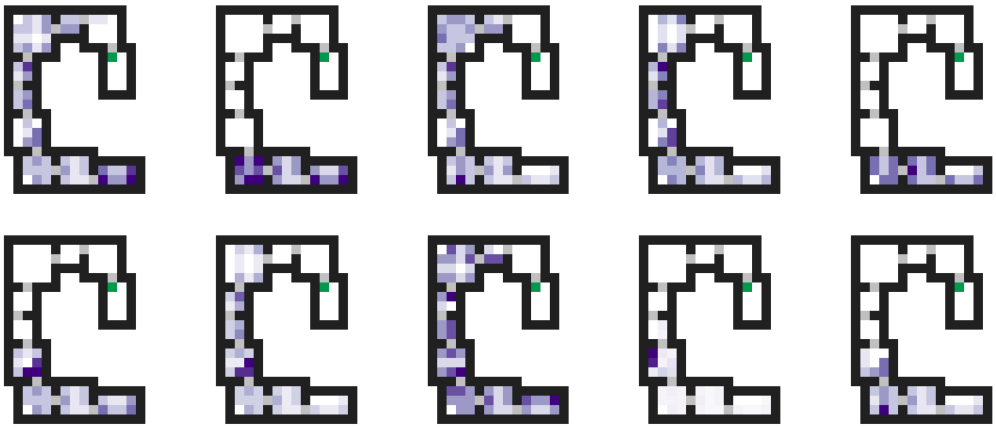


Figure A.7: State visitation heatmaps of the last ten episodes of an agent trained in a *singleton* environment with no extrinsic reward 10M timesteps in the MultiRoomN10S6 task. The agent is continuously engaging into new strategies.

locked, the keys are hidden in boxes and the doors are obstructed by balls. In those difficult tasks, only our method succeeds in exploring well enough to find rewards.

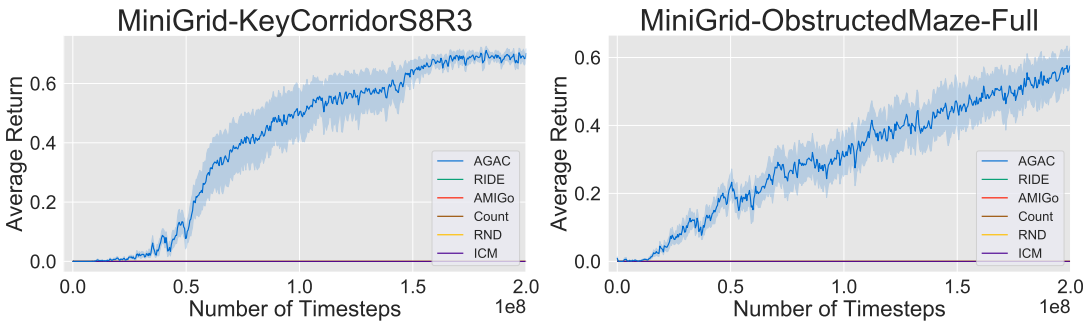


Figure A.8: Performance evaluation of `AGAC` compared to RIDE, AMIGo, Count, RND and ICM on extremely hard-exploration problems.

A.8 Distribution of intrinsic rewards

In this section, we report the mean intrinsic reward computed for an agent trained in MultiRoomN12S10 to conveniently compare our results with that of Raileanu and Rocktäschel (2019). We observe in Fig. A.9 that the intrinsic reward is consistently larger for our method and that, contrary to other methods, does not converge to low values. Please note that, in all considered experiments, the adversarial bonus coefficient c in Eq. A.4 and A.4 is linearly annealed throughout the training since it is mainly useful at the beginning of learning when the rewards have not yet been met. In the long run, this coefficient may

prevent the agent from solving the task by forcing it to always favour exploration over exploitation.

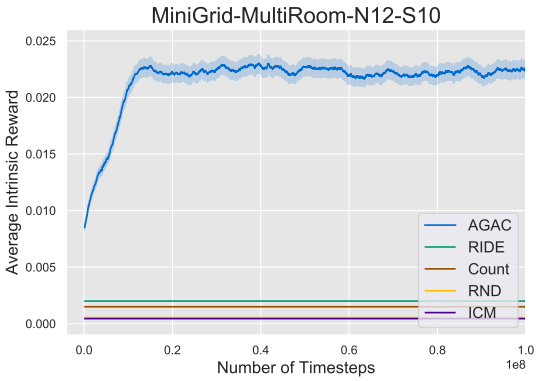


Figure A.9: Average intrinsic reward for different methods trained in MultiRoomN12S10.

A.9 Illustration of AGAC

We illustrate how AGAC works from a high-level perspective in Fig. A.9.

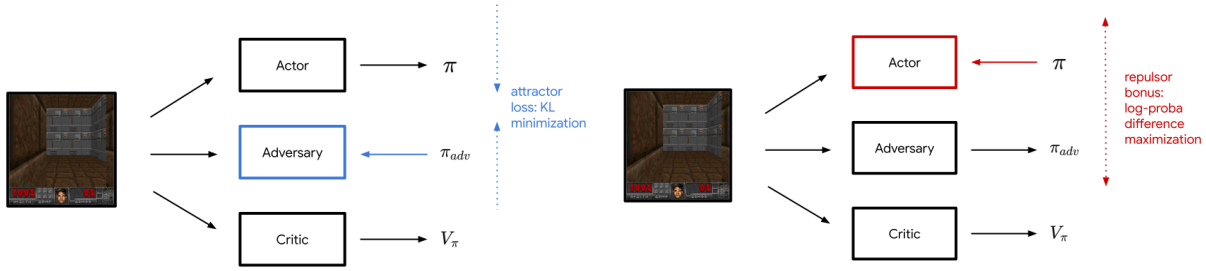


Figure A.10: A simple schematic illustration of AGAC. **Left:** the adversary minimizes the KL-divergence with respect to the action probability distribution of the actor. **Right:** the actor receives a bonus when counteracting the predictions of the adversary.

A.10 Experimental details and hyperparameters

A.10.1 MiniGrid setup

Here, we describe in more details the experimental setup we used in our MiniGrid experiments.

There are several different MiniGrid scenarios that we consider in this paper. MultiRoom corresponds to a set of navigation tasks, where the goal is to go from a starting state to a goal state. The notation MultiRoom-N2S4 means that there are 2 rooms in total, and that each room has a maximal side of 4. In order to go from one room to another, the agent must perform a specific action to open a door. Episodes are terminated with zero reward after a maximum of $20 \times N$ steps with N the number of rooms. In KeyCorridor, the agent also has to pick up a key, since the goal state is behind a door that only lets it in with the key. The notation KeyCorridor-S3R4 means that there are 4 side corridors, leading to

rooms that have a maximal side of 3. The maximum number of steps is 270. In ObstructedMaze, keys are hidden in boxes, and doors are obstructed by balls the agent has to get out of its way. The notation ObstructedMaze-1Dl means that there are two connected rooms of maximal side 6 and 1 door (versus a 3x3 matrix and 2 doors if the leading characters are $2D$), adding h as a suffix places keys in boxes, and adding b as a suffix adds balls in front of doors. Using Q as a suffix is equivalent to using lhb (that is, both hiding keys and placing balls to be moved). The maximum number of steps is 576. ObstructedMazeFull is the hardest configuration for this scenario, since it has the maximal number of keys, balls to move, and doors possible.

In each scenario, the agent has access to a partial view of the environment, a 7x7 square that includes itself and points in the direction of its previous movement.

A.10.2 Hyperparameters

In all experiments, we train six different instances of our algorithm with different random seeds. In Table A.3, we report the list of hyperparameters.

Table A.3: Hyperparameters used in `AGAC`.

Parameter	Value
Horizon T	2048
Nb. epochs	4
Nb. minibatches	8
Nb. frames stacked	4
Nonlinearity	ELU (Clevert et al., 2016)
Discount γ	0.99
GAE parameter λ	0.95
PPO clipping parameter ϵ	0.2
β_V	0.5
c	$4 \cdot 10^{-4}$ ($4 \cdot 10^{-5}$ in VizDoom)
c anneal schedule	linear
β_{adv}	$4 \cdot 10^{-5}$
Adam stepsize η_1	$3 \cdot 10^{-4}$
Adam stepsize η_2	$9 \cdot 10^{-5} = 0.3 \cdot \eta_1$

A.11 Implementation details

In Fig. A.11 is depicted the architecture of our method.

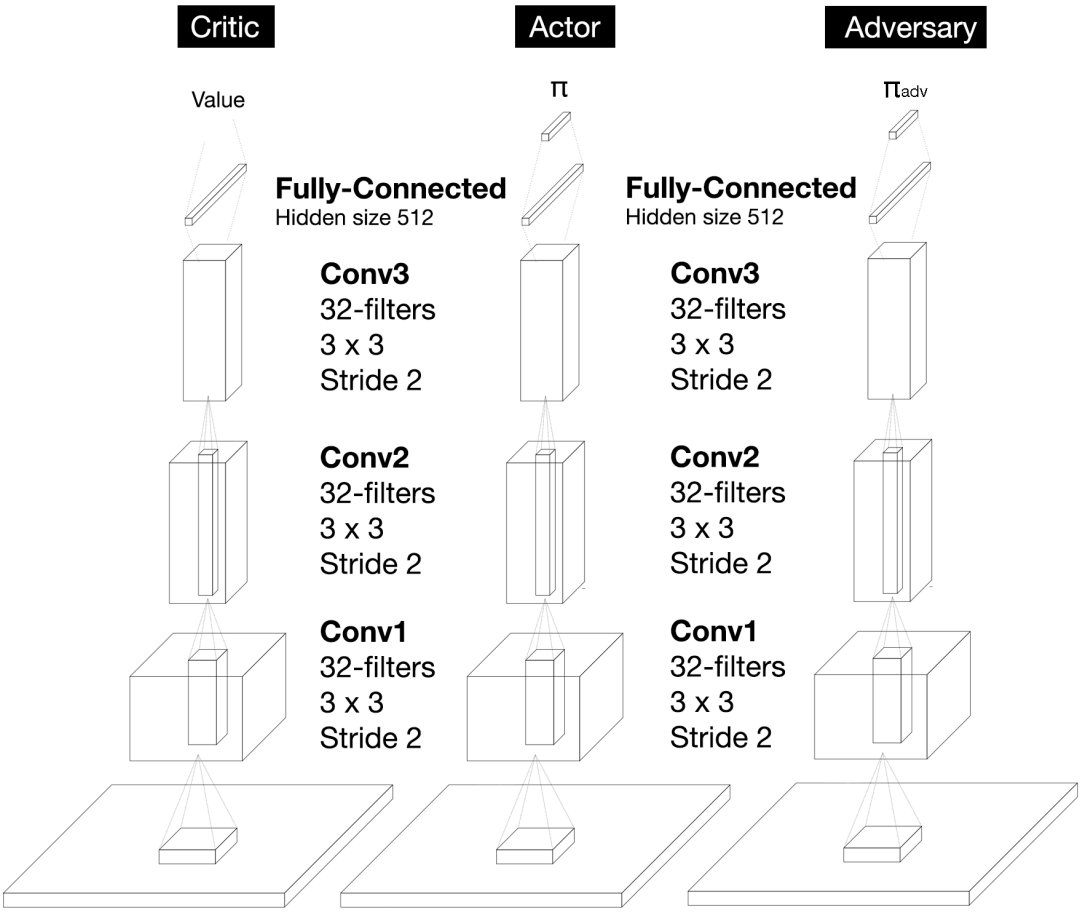


Figure A.11: Artificial neural architecture of the critic, the actor and the adversary.

A.12 Proof of Section A.4.1 results

In this section, we provide a short proof for the result of the optimization problem in Section A.4.1. We recall the result here:

$$\pi_{k+1} = \arg\max_{\pi} \mathcal{J}_{\mathrm{PI}}(\pi) \propto \left(\frac{\pi_k}{\pi_{\mathrm{adv}}}\right)^{\frac{c}{\alpha}} \exp \frac{Q_{\pi_k}}{\alpha},$$

with the objective function:

$$\mathcal{J}_{\mathrm{PI}}(\pi) = \mathbb{E}_s \mathbb{E}_{a\sim\pi(\cdot|s)}[Q_{\pi_k}(s,a) + c\,(\log \pi_k(a|s) - \log \pi_{\mathrm{adv}}(a|s)) - \alpha \log \pi(a|s)].$$

Proof. We first consider a simpler optimization problem: $\arg\max_{\pi} \langle \pi, Q_{\pi_k} \rangle + \alpha \mathcal{H}(\pi)$, whose solution is

known (Vieillard et al., 2020b, Appendix A). The expression for the maximizer is the α-scaled softmax:

$$\pi^* = \frac{\exp(\frac{Q_{\pi_k}}{\alpha})}{\langle 1,\, \exp(\frac{Q_{\pi_k}}{\alpha})\rangle}.$$

We now turn towards the optimization problem of interest, which we can rewrite as:

$$\arg\max_{\pi} \langle \pi, Q_{\pi_k} + c\,(\log \pi_k - \log \pi_{\text{adv}})\rangle + \alpha \mathcal{H}(\pi).$$

By the simple change of variable $\tilde{Q}_{\pi_k} = Q_{\pi_k} + c\,(\log \pi_k - \log \pi_{\text{adv}})$, we can reuse the previous solution (replacing Q_{π_k} by $\tilde{Q}_{\pi_k}$). With the simplification:

$$\exp \frac{Q_{\pi_k} + c\,(\log \pi_k - \log \pi_{\text{adv}})}{\alpha} = \left(\frac{\pi_k}{\pi_{\text{adv}}}\right)^{\frac{c}{\alpha}} \exp \frac{Q_{\pi_k}}{\alpha},$$

we obtain the result and conclude the proof. □

Appendix B

Annex of Chapter 4

In this annex we provide additional elements about the framework presented in Chapter 4.

B.1 Multi-step credit assignment

We illustrate a case of multi-step credit assignment in Fig. B.1.

B.2 On the assessment of existing methods against the proposed criteria

B.2.1 Examples of similarity functions

We give examples of similarity functions for the objects discussed in 4.3.2. These examples have the purpose of showing that such similarity functions exist, even if these might not be ideal in all cases. We start with standard types of outcomes:

- return: a natural similarity function is the negative squared distance between returns.
- individual rewards: a natural similarity function is the negative squared distance between rewards.
- future state: a candidate similarity function between future states is the negative squared distance between the respective backward values of the states. The backward value of a state $B_\pi(s)$ quantifies the expected discounted cumulative sum of rewards when reaching such a state: $B_\pi(s) = \mathbb{E}_\pi\big[\sum_{t=0}^{T} \mathbb{1}\{S_t = s\} \sum_{t'=0}^{t} \gamma^{t'} r(S_{t'}, A_{t'})\big]$.

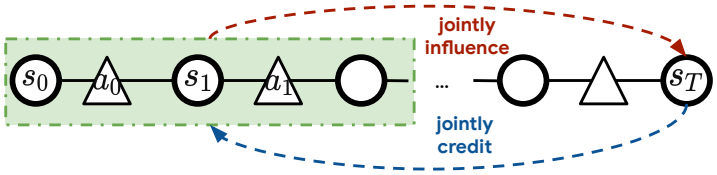


Figure B.1: Multi-step credit assignment. We suppose that the granularity of the actions of the task makes it uninformative to credit actions individually (*e.g.* muscular contractions for a tennis game) so actions are credited in sequences instead (*e.g.* swings).

We now give similarity functions for other elements:

- state distribution: a natural similarity function is the negative Kullback-Leibler divergence between the state distributions. This requires either an MDP with a finite state space, or approximate state distributions.
- policies: a candidate similarity function for policies is the negative Kullback-Leibler divergence between the respective state occupancy measures induced by the policies.
- trajectories: a candidate similarity function is the Wasserstein metric, using a state distance as the cost function and the normalized number of visits as the mass for each state.
- assignments: a candidate similarity function is the Wasserstein metric, using a state distance as the cost function and the cumulative assigned credit as the mass for each state.

B.3 Hard credit assignment tasks

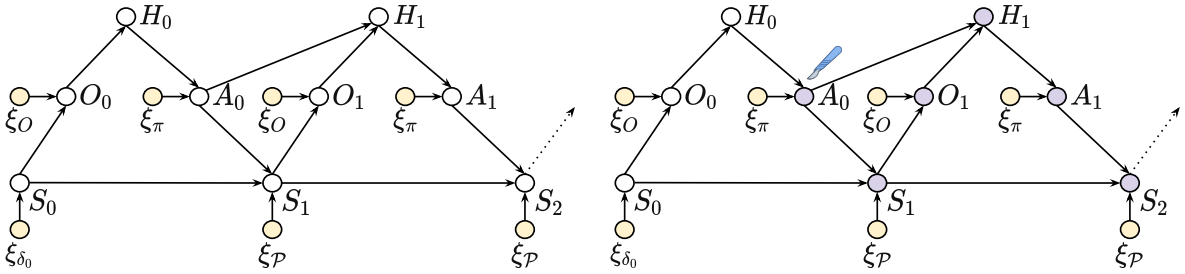
We characterize more precisely the notion of hard credit assignment task we introduced in the main text. Hard credit assignment tasks are RL tasks where a subset of all possible actions have great influence over outcomes that characterize optimality, and such actions are hard to map to these outcomes. Hard credit assignment differs from hard exploration and sparse reward RL. Hard exploration (Bellemare et al., 2016a) groups together tasks which require precise decision-making to experience rewards. Sparse reward RL (Riedmiller et al., 2018) groups tasks where the reward function is sparse. Instead, hard credit assignment groups tasks where important actions are hard to reinforce, and is compatible with easy exploration and dense rewards: an example that features both aspects is the Key-to-Door task from Mesnard et al. (2021). Other examples include the Triggers environment from Ferret et al. (2021), or the delayed Catch from Raposo et al. (2021).

B.4 Supplement on causality

B.4.1 Causality elements.

We introduce elements related to causality (Pearl, 2009) here. A Structural Causal Model (SCM) (Peters et al., 2017) is a directed acyclic graph where nodes correspond to random variables, some of which are observed, some of which are not. Edges indicate parenthood. Each node X is computed via a deterministic function of its parents: $X = f(pa(X), \xi_f)$, where ξ_f is an independent random variable we call *stochastic factor*. In particular, any MDP (or POMDP) can be represented as an SCM: we provide an illustration of the SCM that corresponds to a standard POMDP in Fig. B.2a. We note ξ_{δ_0}, ξ_π, $\xi_\mathcal{P}$, ξ_O and ξ_R the respective stochastic factors of the initial state distribution, the behavior policy, the transition kernel, the observation kernel and the reward function; these are the random variables such that we have the following deterministic functions: $\delta_0(\xi_{\delta_0}) \in \mathcal{S}$, $\pi(s_t, \xi_\pi) \in \mathcal{A}$, $\mathcal{P}(s_t, a_t, \xi_\mathcal{P}) \in \mathcal{S}$, $\mathcal{P}_\mathcal{O}(s_t, \xi_O) \in \mathcal{O}$ and $R(s_t, a_t, \xi_R) \in [r_{min}, r_{max}]$. Given a policy π and the set of stochastic factors $\xi = \{\xi_{\delta_0}\} \cup \{\xi_{O,i}, \xi_{\pi,i}, \xi_{R,i}, \xi_{\mathcal{P},i}\}_{i=0}^{T-1}$ the unique corresponding trajectory is expressed as the result of a deterministic function: $\tau = f(\pi, \xi)$.

An intervention I modifies a subset of the functions of the SCM, whose typical node becomes: $X_I = f_I(pa(X_I), \xi_f)$. *Stochastic factors are left unchanged.* Other nodes whose functions are unchanged can have different distributions due to the updated distributions of their parent nodes. We note an atomic intervention as $I(f \to \bar{f}, t)$ to signify that we intervene on the random variable at timestep t, replacing its function f by $\bar{f}$. We illustrate an atomic intervention in Fig. B.2b.



(a) (b)

Figure B.2: **a)** The SCM for a generic POMDP. Yellow nodes correspond to unobserved variables. We do not represent rewards for clarity. **b)** An atomic intervention in a SCM. The scalpel indicates which random variable is intervened on. The purple nodes are the nodes affected by the intervention (whether directly or indirectly affected).

In what follows, we take a special interest in interventions on action variables, as they allow to simulate counterfactual futures (i.e. what would have happened under a different action) and ultimately credit actions. Given a trajectory τ, the corresponding stochastic factors ξ, a behavior policy π and an intervention policy $\bar{\pi}$, we note the trajectory resulting from the intervention on the action at timestep t: $\bar{\tau} = f_{I(\pi \to \bar{\pi}, t)}(\pi, \xi)$. Note that this is, on average, equivalent to taking as behavior policy the non-stationary policy that acts according to $\bar{\pi}$ at timestep t and otherwise according to π. Still, when modifying a single existing trajectory, we will express it under the SCM formalism: it handles well stochastic factors, which will be kept identical between the original and the modified trajectory. On the other hand, when dealing with probabilities (resp. expectations), we will opt for the notation $p_{\bar{a} \sim \bar{\pi}, \pi}$ (resp. $\mathbb{E}_{\bar{a} \sim \bar{\pi}, \pi}$) for simplicity, which means that an immediate action is sampled according to $\bar{\pi}$ and the remaining actions according to π.

B.4.2 The do operator

We do not make use of the do operator (Pearl, 2009) in the main text. The do operator is a central component in the causality literature so we explicit how it connects to our work here. The do operator is used to distinguish interventional distributions (distributions of random variables given the intervention, *i.e.* $p(Y|\,\mathrm{do}(X = x))$) from standard, observational distributions (*i.e.* $p(Y|X = x)$). Interventions lead to different distributions when Y shares parents with X in the corresponding SCM. The interventions we discuss only target action choices, *i.e.* we only locally modify the policy. In that regard, we intervene on an action, conditionally to a state, and due to the Markov property there is no shared parents between the action and the outcome, so we can show that for any policy π and any outcome $Z = f(\mathcal{T})$, we have $d_\pi(Z \,|\, S_t = s, A_t = \mathrm{do}(a_t)) = d_\pi(Z \,|\, S_t = s, A_t = a_t)$. Evaluating an outcome under an action intervention is thus equivalent to evaluating the outcome under the distribution induced by either a non-stationary policy (if the intervention is atomic) or a different policy (if the intervention is persistent).

B.4.3 Nature of stochastic factors

From the main text, one could get the wrong impression that stochastic factors are the distinct noises that are used to sample actions, rewards and transitions based on corresponding distributions. In other words, one might get that stochastic factors equate random number generation. In the partially observable setting, stochastic factors can also include other sources of stochasticity. Those are covered by the stochastic factor of the state-to-observation kernel, but can correspond to latent variables that are not necessarily

intuitive. For instance, in the Umbrella problem (Osband et al., 2020) (see Fig. 4.2b), the weather is a random variable that conditions the optimal policy (*i.e.* take an umbrella or not) and could have to be deduced from observations.

B.4.4 Persistent interventions

In contrast, we note a persistent intervention $I(f \to \bar{f}, \geq t)$, which means that we do a similar atomic intervention on all random variables that are computed via f at any timestep $t' \geq t$.

B.5 Oracle credit assignment via counterfactual simulation

To the best of our knowledge, there is no known method for ground truth credit assignment in the existing literature. Given the informal definition of credit assignment from the introduction, optimality is difficult to specify. Instead, we argue in favor of a quantity that satisfies the desiderata listed in the previous section. In what follows, we provide an oracle method to establish ground truth credit assignment for an observed outcome and the corresponding trajectory, based on counterfactual simulation. It consists in modifying a single action at a time from the original trajectory, and in simulating the corresponding counterfactual outcome to assess the impact of the action change.

Simulation-based influence of actions over outcomes. The criteria **1**,**2**,**3** from the previous section all depend on a notion of influence over the observed outcome (in the context of a given trajectory). We thus present candidate notions of influence that can be computed via simulation, given a trajectory, the corresponding outcome, and the timestep of the specific action to assess (noted a_t). Note that the trajectory implicitly defines the value of the outcome: $z = f(\tau)$ (with the corresponding random variables Z for the outcome and $\mathcal{T}$ for the trajectory). We start by introducing a *probabilistic* notion of influence.

- Forward probabilistic influence quantifies *how much more likely the outcome is given the action*: $c_\pi(s_t, a_t \,|\, z) = p_\pi(Z = z \,|\, S_t = s_t, A_t = a_t) - p_\pi(Z = z \,|\, S_t = s_t)$ (which requires a discrete outcome).

- Backward probabilistic influence quantifies *how much more likely the action is given the outcome*: $c_\pi(s_t, a_t \,|\, z) = p_\pi(A_t = a_t \,|\, S_t = s_t, Z = z) - \pi(a_t \,|\, s_t)$.

Note that $p_\pi(A_t = a_t \,|\, S_t = s_t, Z = z)$ is the probability that the action taken at timestep t in s_t is a_t, given that the outcome is z. An alternative formulation for the backward probabilistic influence is explored in Harutyunyan et al. (2019), using a ratio instead of a difference.

Importantly, we might want to establish influence in a relative way, by comparing the results of distinct actions taken in the same state. We thus define *counterfactual influence*, which takes two forms: *probabilistic* counterfactual influence (forward and backward) and *metric* counterfactual influence.

- Forward probabilistic counterfactual influence quantifies *how much more likely the outcome is than under the action change*: $c_\pi(s_t, a_t \,|\, z) = p_\pi(Z = z \,|\, S_t = s_t, A_t = a_t) - p_{\bar{a}\sim\bar{\pi},\pi}(Z = z \,|\, S_t = s_t, A_t = \bar{a})$.

- Backward probabilistic counterfactual influence quantifies *how much more likely the action is than counterfactual actions, given the outcome*: $c_\pi(s_t, a_t \,|\, z) = p_\pi(A_t = a_t \,|\, S_t = s_t, Z = z) - p_{\bar{a}\sim\bar{\pi},\pi}(A_t = \bar{a} \,|\, S_t = s_t, Z = z)$.

- Metric counterfactual influence quantifies *how far from the outcome the counterfactual outcome is*: $c_\pi(s_t, a_t) = dist(\mathbb{E}_\pi[Z \,|\, S_t = s_t, A_t = a_t], \mathbb{E}_{\bar{a}\sim\bar{\pi},\pi}[Z \,|\, S_t = s_t, A_t = \bar{a}])$.

In practice, such influence functions can be approximated empirically via simulation, by averaging over a set of sampled trajectories $\mathcal{T}$. In more details, given the ability to reset the simulator to an arbitrary state, approximations of above quantities are obtained by replacing probabilities by empirical probabilities, and expectations by empirical averages. Depending on the term to be approximated, trajectories are either sampled from the behavior policy or from the non-stationary policy that takes a counterfactual action and samples actions according to the behavior policy afterwards. Given the influence $c_\pi(\dots)$, we note its empirical counterpart $\hat{c}_\pi(\dots, \mathcal{T})$.

Note that the metric counterfactual influence is an implicit function according to our definitions, but still depends on an outcome Z, which we calculate the expectation of. Also, an estimate of it is obtained as $c_\pi(s_t, a_t \mid z) = dist(z, \mathbb{E}_{\bar{a}\sim\bar{\pi},\pi}[Z \mid S_t = s_t, A_t = \bar{a}])$, which is less costly to compute via simulations, since it does not require to simulate outcomes under the behavior policy.

Counterfactual actions. In what comes next, we take an interest in the counterfactual notion of influence, which requires to take counterfactual actions. We call the policy $\bar{\pi}$ from which such actions are sampled the *counterfactual policy*. The observed action that is modified is noted a_t (with t arbitrary in $[0, T]$). While counterfactual policies can take many forms, two stand out naturally (see Fig. B.5):

- deterministically modifying the behavior policy, *i.e.* choosing $\bar{a} \neq a_t$ such that $\bar{\pi}(\bar{a}|s_t) = 1$.
- pruning the observed action from the behavior policy, *i.e.* having $\bar{\pi}(a_t|s_t) = 0$ and $\bar{\pi}(\bar{a}|s_t) = \pi(\bar{a}|s_t)/(1 - \pi(a_t|s_t))$ for all $\bar{a} \neq a_t$.

Counterfactual consistency. Action interventions generate counterfactual scenarios, that give quantitative answers to *"what if I had acted differently?"* questions. For consistency, *except from the action that is modified, every element of the trajectory should be left unchanged.* Nevertheless, the agent should not conserve its exact behavior after the action modification: for proper consistency, the simulation should *keep the stochastic factors of the original trajectory*, which means that all actions after the intervention have the potential to differ in the modified trajectory. We illustrate the difference it makes in Fig. B.4. Consistency is key to isolating the contribution of the modified action, and to assigning accurate credit. We provide a visual justification for that assertion in Fig. B.3 right. Another argument in favor of consistency is the Invariance criterion (criterion **4**), which implies that actions that lead to identical immediate states should be assigned identical credit. If we do not enforce consistency (*i.e.* we sample new stochastic factors to get transitions and actions), taking a counterfactual action that leads to the same immediate state can give different outcomes because the rest of the trajectory can diverge (due to randomness).

Grounded credit assignment oracle. We thus propose a novel way to determine ground truth credit assignment, given a behavior policy. It consists in *evaluating the influence of counterfactually modifying the selected action via simulation.* It is a grounded, explicit credit assignment function, and is compatible with any type of counterfactual influence. We provide the pseudo-code for the counterfactually consistent version of the credit assignment oracle in Algorithm 3. Note that we use a simple form of counterfactual inference here, namely Monte-Carlo simulation, though more advanced methods could be used (Buesing et al., 2019). We qualitatively assess the oracle against the proposed criteria in Appendix B.6.1 and we give additional details about implementation in Appendix B.6.2.

B.6 Additional details about the credit assignment oracle

In this section, we complete our presentation of the credit assignment oracle (see Algorithm 3).

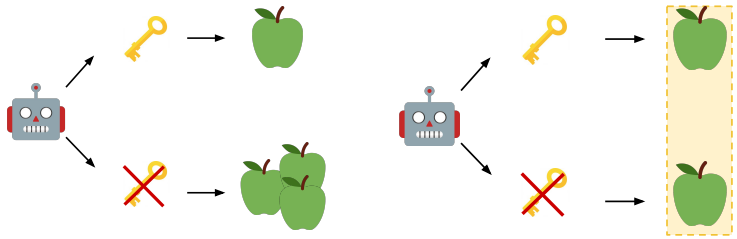


Figure B.3: **Left:** Standard counterfactual inference in the Key-to-Door task (see Fig. 4.2). Due to randomness, the agent collects more apples in the counterfactual trajectory where it does not take the key, which can lead to bias, *i.e.* mapping the action of taking the key to a lesser outcome. **Right:** Counterfactually consistent inference in the Key-to-Door task. Consistency enforces equal dynamics *after intervention* in the counterfactual scenario, so the resulting difference in outcome precisely measures the contribution of taking the key.

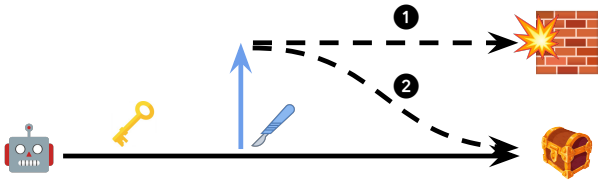


Figure B.4: Modifying one action and keeping others identical (1) leads to poor counterfactual inference: the modified action seems to impact a lot performance, while the expected resulting trajectory when resampling subsequent actions (2) leads to an unchanged outcome.

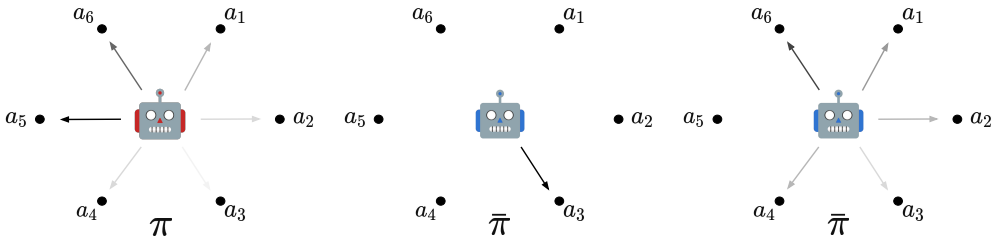


Figure B.5: **Left:** original policy. **Middle:** deterministic counterfactual policy. **Right:** pruned counterfactual policy.

Algorithm 2: Consistent simulation of a counterfactual trajectory.

Input: An MDP $\mathcal{M}$, an outcome z, a trajectory τ, stochastic factors $\xi = \{\xi_{\delta_0}\} \cup \{\xi_{\pi,i}, \xi_{R,i}, \xi_{\mathcal{P},i}\}_{i=0}^{T-1}$, a timestep t, a counterfactual policy $\bar{\pi}$.
Output: The counterfactual trajectory $\bar{\tau}$.
Reset $\mathcal{M}$ to s_t;
Set $\bar{s} \leftarrow s_t$;
Sample $\bar{\xi}_{\pi,t}$;
Set $\bar{a} \leftarrow \bar{\pi}(\bar{s}, \bar{\xi}_{\pi,t})$;
Initialize the counterfactual trajectory $\bar{\tau} \leftarrow \tau_{1:t-1} \cup \{\bar{s}, \bar{a}, R(\bar{s}, \bar{a}, \xi_{R,t})\}$;
`/* Sample the rest of the counterfactual trajectory. */`
$j \leftarrow t$;
while $j < T$ **do**
  Compute the state $\bar{s} \leftarrow \mathcal{P}(\bar{s}, \bar{a}, \xi_{P,j})$;
  Compute the action $\bar{a} \leftarrow \pi(\bar{s}, \xi_{\pi,j+1})$;
  Update the trajectory $\bar{\tau} \leftarrow \bar{\tau} \cup \{\bar{s}, \bar{a}, R(\bar{s}, \bar{a}, \xi_{R,j+1})\}$;
  $j \leftarrow j + 1$;
end

Algorithm 3: Counterfactually consistent credit assignment oracle.

Input: An MDP $\mathcal{M}$, an outcome z, a trajectory τ, stochastic factors $\xi = \{\xi_{\delta_0}\} \cup \{\xi_{\pi,i}, \xi_{R,i}, \xi_{\mathcal{P},i}\}_{i=0}^{T-1}$, a timestep t, a counterfactual policy $\bar{\pi}$, a number of trajectories to sample k.
Output: The oracle credit assignment $c_\pi(t \,|\, \tau)$.
Initialize the counterfactual trajectory buffer $\bar{\mathcal{T}} \leftarrow []$;
for *i in* $[1, k]$ **do**
  Simulate a counterfactual trajectory $\bar{\tau} = f_{I(\pi \to \bar{\pi}, t)}(\tau, \xi)$ (Algorithm 2);
  Save to buffer $\bar{\mathcal{T}} \leftarrow \bar{\mathcal{T}} \cup \bar{\tau}$;
end
Compute the credit $c_\pi(t \,|\, \tau) = \hat{c}_\pi(t \,|\, \tau, \bar{\mathcal{T}})$;

B.6.1 Criteria satisfied by the counterfactual oracle

We now show that the proposed oracle satisfies all the desiderata expressed in the previous section, which we back by informal arguments when obvious, and through experimentation when not. Choosing the counterfactual metric influence as the desired notion of influence, criteria **1-3** are satisfied since the oracle makes use of this very notion of influence. Invariance (criterion **4**) is the by-product of counterfactual consistency: a counterfactual action that leads to the same immediate state will reproduce the exact same trajectory as the original in the counterfactual simulation. Stability (criterion **5**) and Generalizability (criterion **6**) depend on the similarity functions used, so we provide experiments to assess if these criteria are satisfied in a simple task. Expressivity (criterion **7**) comes from the fact that the proposed oracle is quite general, and can be adapted to many outcomes and tasks. We provide additional elements on Scalability (criterion **8**) in Appendix B.1.

B.6.2 Implementing the counterfactual oracle

To make the computation of the oracle feasible, one should have control over both 1) the randomness of the action selection (*i.e.* to reproduce actions from the original trajectory when states match at identical timesteps), and 2) the randomness of the environment (*i.e.* both the transition kernel and reward function, to reproduce transitions and rewards from the original trajectory when states match at identical timesteps). To achieve 1) and 2), a solution consists in storing the sequences of states of the random number generators (RNGs) controlling randomness. We call the corresponding data structure a *keychain*. The keychain keeps lists of RNG states for the different stochastic elements (i.e. the policy, transition kernel, state-to-observation kernel, reward function).

We now describe how to implement the proposed counterfactual oracle. The oracle is simulation-based, so one needs to have access to a simulator of the MDP at hand, which we will call the environment. We describe how to sample a counterfactual trajectory in a consistent way, the rest of the process being described in Algorithm 3. We use a Python-like syntax overall and a Jax-like (Bradbury et al., 2018) syntax for the RNG. The implementation is based on two main objects: a *keychain* and a *wrapper*. The keychain is implemented as a key-value store that keeps track of the current key of the RNG in each state seen so far in the current episode:

```
class Keychain:

  def __init__(self, keychain=None):
    if keychain is None:
      self.keychain = collections.defaultdict(lambda: None)
    else:
      self.keychain = keychain

  def get_key(self, state):
    if self.keychain[state] is None:
      self.keychain[state] = jax.random.PRNGKey(hash(state))
    return self.keychain[state]

  def set_key(self, state, key):
    self.keychain[state] = key

  def gen_key(self, state):
    new_key, subkey = jax.random.split(self.get_key(state))
    self.set_key(state, new_key)
    return subkey

  def deepcopy(self):
    return Keychain(copy.deepcopy(self.keychain))
```

The wrapper adds a keychain as an attribute of the environment (resp. policy) and uses it as the source of RNG when sampling. Here is an example of such a wrapper for the environment:

```
class StatewiseRNGEnvWrapper:

  def __init__(self, env):
    self.env = env
    self.reset()

  @property
  def state(self):
    return self.env.state

  def reset(self):
    self.keychain = Keychain()
    return self.env.reset()

  def step(self, action, return_keychain=False):
    subkey = self.keychain.gen_key(self.state)
    if return_keychain:
      return self.env.step(action, subkey), self.keychain.deepcopy()
    return self.env.step(action, subkey)

  def set_state(self, state, keychain):
    self.env.state = state
    self.keychain = keychain.deepcopy()
```

Here is an example of a policy wrapper:

```
class StatewiseRNGPolicyWrapper:

  def __init__(self, policy):
    self.policy = policy
    self.reset()

  def reset(self):
    self.keychain = Keychain()

  def sample(self, state, return_keychain=False):
    subkey = self.keychain.gen_key(state)
    if return_keychain:
      return self.policy.sample(state, subkey), self.keychain.deepcopy()
    return self.policy.sample(state, subkey)

  def set_rng(self, keychain):
    self.keychain = keychain.deepcopy()
```

B.7 A discussion on lucky outcomes

A criticism that might be addressed to explicit credit assignment methods is that they might encourage lucky outcomes by crediting the responsible actions. We illustrate that aspect on a variant of the Key-to-Door problem that we call the Gambler's Key-to-Door. It is identical to the standard Key-to-Door task (see Fig. 4.2a) except that the reward for opening the door is drawn from a given distribution that is independent from the actions of the agent. In particular, we suppose that the expected reward for opening the door is negative, which means optimal policies never do so. In that scenario, and for a policy that will always go for the door, accurate credit assignment will map lucky outcomes to the action of taking the key, which in turns might lead to mistakenly reinforcing the action of taking the key. We argue

that this can and should be mitigated by a good incorporation strategy. A clever incorporation strategy could leverage optimism to kickstart learning but should asymptotically take into account the likelihood of the outcome to decide what to reinforce. Another way to view this is to say that credit assignment pinpoints actions that are responsible for the observed outcomes, but it is the role of the RL algorithm to provide with reinforcements that do not suffer from to much variance.

Appendix C

Annex of Chapter 6

In this annex we provide additional details about our experimental setup and the hyperparameters we use in Chapter 6.

C.1 Time limits

In Triggers, episodes stop after 50 timesteps in 8x8 grids and 100 timesteps in 12x12 grids. Those values were chosen large enough such that the policies used to train the reward predictor have a chance to gather informative rewards. In DMLab, episodes stop after 900 timesteps, as defined by the standard settings.

C.2 Reward prediction model

We use the same set of hyperparameters in all our experiments with few variation. In Triggers experiments, we use 128 units per dense layer, 32 convolutional filters and a single convolutional layer to process partial states and actions. We apply dropout (Srivastava et al., 2014) at several places in the model as a regularizer. We use a dropout rate of 0.1 after dense layers, a dropout rate of 0.2 in the self-attention mechanism, and a dropout rate of 0.2 in the normalization blocks of the Transformer architecture. Since Transformers create representations via pooling, they need additional information so as to take into account the relative positions of sequence elements. Hence, we use the same positional encoding scheme as in Vaswani et al. (2017), that is we add sine and cosine signals of varying frequencies to the sequence of input embeddings X_{emb} before the self-attention layer:

$$PE_{t,2i} = sin\Big(\frac{t}{\nu^{2i/d_i}}\Big),$$

$$PE_{t,2i+1} = cos\Big(\frac{t}{\nu^{2i/d_i}}\Big),$$

$$X = X_{emb} + PE.$$

In these equations, t refers to the timestep in the sequence, i to the dimension of the embedding considered, and ν has a constant value of 10000.

In DMLab experiments, we use two convolutional layers, 16 filters for each, and otherwise identical hyperparameters.

Typically, we set the class weights in the loss function to $w(1) = w(-1) = 0.499$, $w(0) = 0.02$. Fig. C.1 gives an overview of the whole architecture.

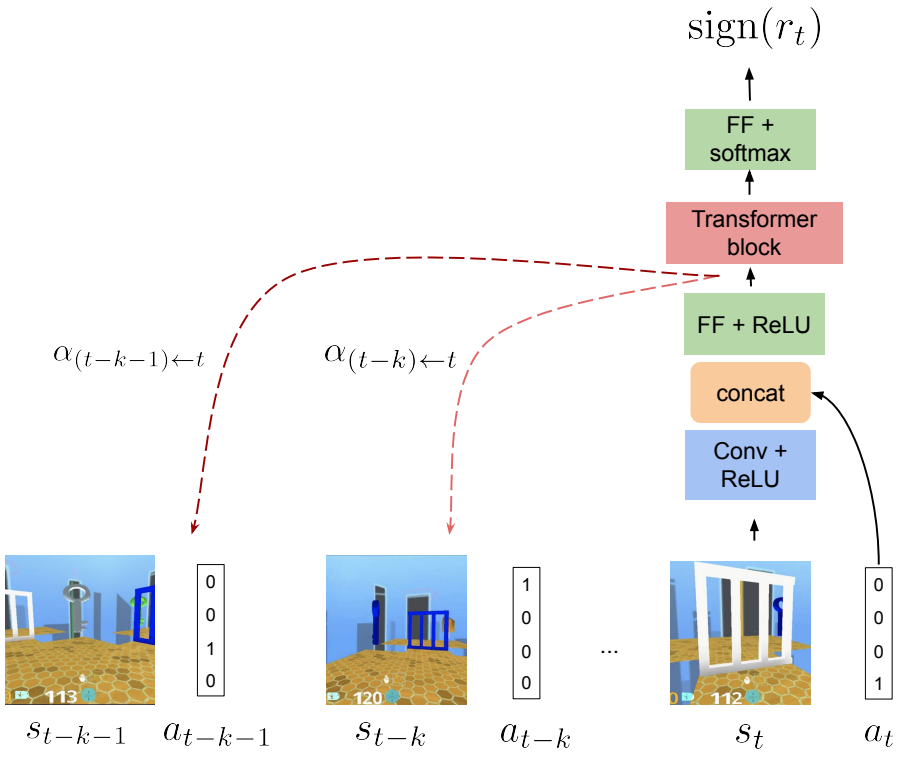


Figure C.1: The architecture used for SECRET. $\alpha_{.\leftarrow t}$ is the vector containing the attention weights of the model for its prediction at step t.

C.3 Heuristic for precision-recall analysis

In Chapter 6.3.1, we compare the attention vectors we get as outputs from the reward prediction model to ideal credit assignment with binary metrics. The ground truth we use is a binary vector of the size of the attention vector. Its values are 0 everywhere and 1 for timesteps that correspond to the activation of a Triggers. To do so, we introduce a simple heuristic to binarize the attention scalars: we consider all values above a threshold α to correspond to events to be credited. Then, we can measure precision and recall as in a binary classification paradigm. The precision and recall reported are the average precision and recall over 4 scenarios : Triggers with a 8x8 grid, 1 trigger and 1 reward; Triggers with a 8x8 grid, 1 trigger and 2 rewards; Triggers with a 8x8 grid, 2 triggers and 2 rewards; and Triggers with a 8x8 grid, 3 triggers and 1 reward. In each scenario, we train the model over a set of 40000 trajectories, each of which is drawn from a randomly sampled maze. Then, we apply it on 5000 trajectories from held-out environments and collect the attention weights corresponding to predictions on timesteps where the agent experiences positive reward. We use a fixed α of 0.2.

C.4 Transfer in Triggers

We collect 40000 trajectories sampled from random policies to train the prediction reward model in all distributions of environments.

In experiments involving tabular Q-learning we use online Q-learning with a learning rate of 0.1 and a constant greediness factor ϵ also equal to 0.1.

For the out-of-domain transfer experiment with modified dynamics, we use a smaller version of the DQN architecture in Mnih et al. (2013). The first convolutional layer has 8 filters, a 3x3 kernel size and a stride of 2. The second and the third convolutional layers have both 16 filters, a 3x3 kernel size and a stride of 1. Those are completed by a feed-forward layer with 64 units followed by another feedforward layer with as many units as the number of available actions. The greediness factor ϵ is decayed linearly from 1 to 0.01 over 250000 steps in the environment at train time and has a constant value of 0.001 at test time. We use RMSProp (Tieleman and Hinton, 2012) as an optimizer with a base learning rate of 0.00025. We update the target network every 2000 steps and initially fill the replay buffer with 5000 transitions sampled following a random policy. The replay buffer has a maximum size of 1000000.

C.5 In-domain transfer in DMLab

We provide additional details about this setup: we train SECRET using 10000 trajectories sampled from a distribution of mazes that are generated randomly. These trajectories are sampled using an agent trained over the same distribution. We do so to increase the proportion of trajectories where rewards are experienced. Indeed, we found that using random policies yielded very few of these. Once the model is trained, we use it to compute the attentional potential function over a fixed maze. 1000 trajectories are sampled on the fixed maze using the same policy as the one that generated the trajectories used to train the reward prediction model. Since consecutive frames can be very similar, we consider a positive reward prediction to be correct (and thus use the corresponding attention weights when estimating the potential) if it happens within 5 frames of a reward actually experienced in the environment. We then compare the performance of agents trained with the original reward function to those trained with the shaped reward.

An important point is that we use the knowledge of the position and the keys the agent possesses to build the state used to compute the potential function. This information is not given to the agent. We create an approximate state as such: it is the concatenation of the discretized position and the identifier of the key possessed. The discretized position is the result of the euclidean division by a cell size integer c_s and is necessary since the DMLab position is continuous. c_s is fixed and has the value of 50 in our

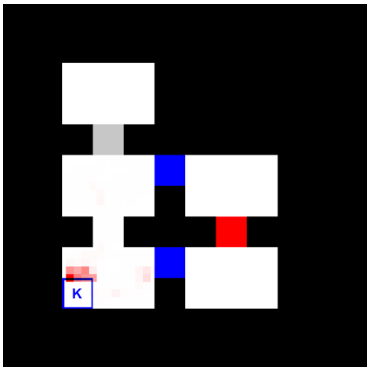


Figure C.2: Attention heatmap in DMLab. Attention concentrates around the key which is necessary to unlock reward.

experiments. We acknowledge that relying on a manually constructed state limits the generality of our approach, but we are confident that this limitation can be addressed in future work by using an estimate of the true state.

All agents mentioned in that section are PPO learners with a learning rate of 0.00019, an entropy coefficient of 0.0011, 12 actors, a discount factor $\gamma = 0.99$. They use generalized advantage estimation (Schulman et al., 2016) with $\lambda = 0.95$. All those hyperparameter values are taken as is from the experiment section in Savinov et al. (2019), whose code is open-source.

C.6 Attention heatmap in DMLab

In Fig. C.2, we display the per-position average attention weight along trajectories starting in the middle-left room that led to opening the blue doors. Attention (shown in shades of red) concentrates around the key position in the lower-left corner.

C.7 Additional experiments

In this section we provide the results of additional experiments that are useful to understand several aspects of Secret.

C.7.1 Influence of the state transformation in Triggers

We study the influence of the size of the window in the transformation used to turn states into observations in Triggers. The goal of this experiment is notably to study the effect of the degree of partial observability to the assigned credit, and also the effect of transferring a possibly badly assigned credit to an agent, through reward shaping.

We consider the following window types: 3x3, 5x5 and 7x7 (odd numbers because the window is centered on the agent), as well as the full state. We place ourselves in the in-domain setting and evaluate Secret on held-out environments from the same distribution as the source. Environments are 8x8 mazes with 3 triggers and 1 prize. We display reward prediction and credit assignment metrics for each window size in Table C.1, and the average discounted return of tabular Q-learning agents for each window size in Fig C.3a. Notice that the considered reward prediction accuracy (Table C.1) is weighted, such that all classes have the same importance.

Window size	Reward prediction accuracy	Credit precision	Credit recall
3x3	0.67 ± 0.06	$1.0 \pm 0.$	$1.0 \pm 0.$
5x5	0.92 ± 0.02	0.57 ± 0.04	0.38 ± 0.03
7x7	0.75 ± 0.12	0.07 ± 0.04	0.03 ± 0.02
Full state	0.76 ± 0.14	0.10 ± 0.05	0.02 ± 0.02

Table C.1: Influence of the window size on the reward prediction and credit assignment quality.

$w(0)$	Reward prediction accuracy	Credit precision	Credit recall
1.0	0.72 ± 0.15	0.83 ± 0.34	0.59 ± 0.27
0.1	0.72 ± 0.09	$1.0 \pm 0.$	0.89 ± 0.05
0.01	0.68 ± 0.09	$1.0 \pm 0.$	0.93 ± 0.05
0.001	0.65 ± 0.04	$1.0 \pm 0.$	0.95 ± 0.03

Table C.2: Influence of the class weighting on the reward prediction and credit assignment quality.

In this setting, we can observe that bigger window sizes are detrimental to the quality of the credit assigned, even if it helps to predict reward (Table C.1). This was to be expected: the more "observable" is the sequence of observations used to train the reward predictor, the easier is the reward prediction (in the full state case, it can be predicted solely from the current state-action couple), but also the less likely the assigned credit will be centered on the trigger. However, even with a low credit precision, the shaped reward still accelerates the learning process (Fig. C.3a).

C.7.2 Influence of the class weights in the reward prediction loss

We stated that class weighting was important to keep the variance of prediction metrics across a variety of datasets of sampled trajectories low, the classes (sign of the reward) being quite imbalanced. Here, we study the influence of the class weights considered in the loss of the reward prediction model in Triggers.

We consider the following class weights for the class corresponding to zero rewards: $w(0) = 1$, $w(0) = 0.1$, $w(0) = 0.01$ and $w(0) = 0.001$. The other class weights are fixed and have the value $w(1) = w(-1) = 1$. We place ourselves in the in-domain setting and evaluate SECRET on held-out environments from the same distribution as the source. Environments are 8x8 mazes with 3 triggers and 1 prize.

We display reward prediction and credit assignment metrics for each window size in Table C.2, and the average discounted return of tabular Q-learning agents for each window size in Fig C.3b. In that setting, putting heavier misclassification penalties for under-represented classes (-1 and 1) results in improved credit and RL performance.

C.7.3 Influence of the amount of available data

We study the influence of the amount of data used by SECRET in a Triggers task. The objective is to assess how data-demanding is the proposed approach, and the effect of a lack of data on the transfer. There are two types of data: the data from the source domain (used to train its reward prediction component), and the data from the target domain (used to build the potential function).

Source distribution We consider the following number of trajectories to train the self-attentive reward predictor: 500, 1000, 5000, 10000, 25000 and 50000. We place ourselves in the in-domain setting and

Number of source episodes	Reward prediction accuracy	Credit precision	Credit recall
500	0.53 ± 0.11	0.23 ± 0.37	0.09 ± 0.2
1000	0.65 ± 0.07	0.49 ± 0.33	0.52 ± 0.42
5000	0.79 ± 0.08	0.9 ± 0.30	0.9 ± 0.30
10000	0.84 ± 0.07	0.9 ± 0.30	0.9 ± 0.30
25000	0.88 ± 0.09	0.9 ± 0.30	0.9 ± 0.30
50000	0.83 ± 0.11	0.8 ± 0.4	0.8 ± 0.4

Table C.3: Influence of the number of episodes in the source domain on the reward prediction and credit assignment quality.

evaluate SECRET on held-out environments from the same distribution as the source. Environments are 8x8 mazes with 3 triggers and 1 prize.

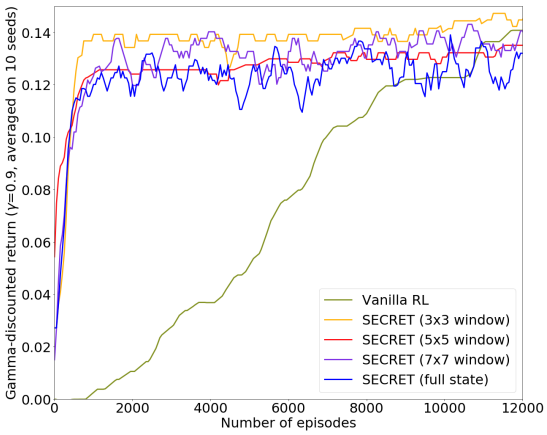
We display reward prediction and credit assignment metrics for each window size in Table C.3, and the average discounted return of tabular Q-learning agents for each number of trajectories in Fig. C.3c. We observe that increasing the size of the dataset improves the results, both for the efficiency of the reward predictor (Table C.3) and for the transfer (Fig. C.3c). Having too few data can slow down learning compared to a vanilla agent (Fig. C.3c), but it does not prevent from learning to solve the task.

Target distribution We consider the following number of trajectories sampled from the target environment to build the potential function: 100, 500, 1000, 2000, 5000, 10000. We place ourselves in the in-domain setting and evaluate SECRET on held-out environments from the same distribution as the source. Environments are 8x8 mazes with 3 triggers and 1 prize.

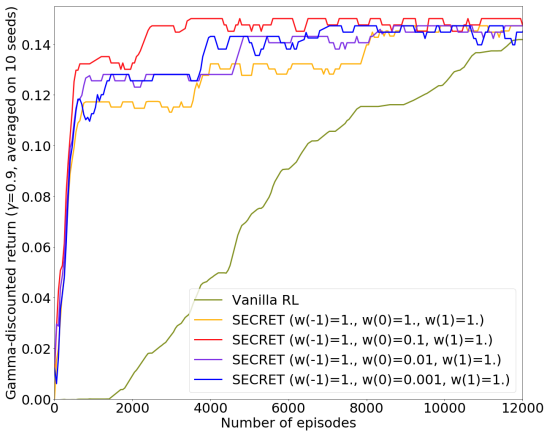
We display the average discounted return of tabular Q-learning agents for each number of trajectories in the source domain in Fig.C.3d. Here again, we observe that the number of episodes in the target distribution is an important parameter for SECRET, but it has less impact than the number of episodes in the source domain.

C.7.4 Attention distributions in out-of-domain scenarios

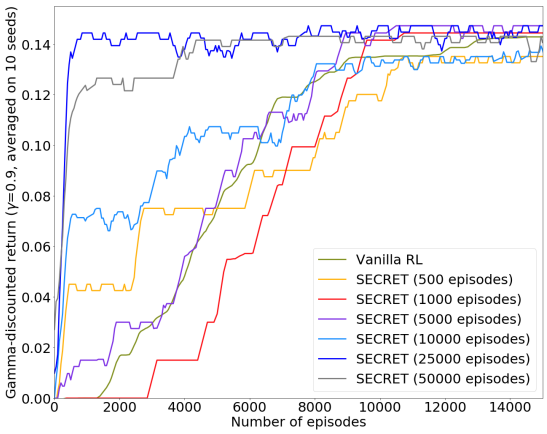
In the main paper, we have only shown the distribution of attention for an in-domain scenario. Here, following the same protocol as in Chapter 6.3.1, we measure the distribution of attention weights in the two out-of-domain scenarios considered in our experiments: bigger mazes and inverted dynamics. The results are reported in Fig. C.4. We observe a similar distribution of attention, peaked on the triggers.



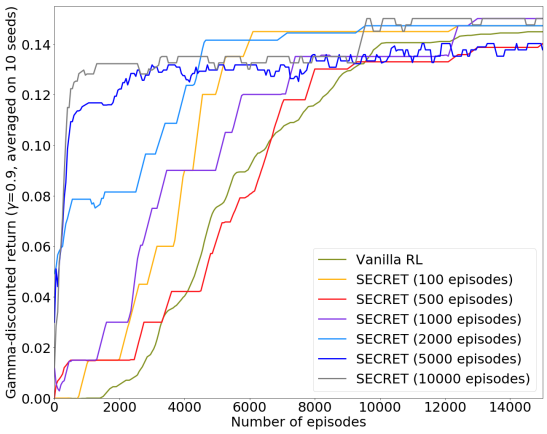
(a) Effect of the window size on SECRET.



(b) Effect of the class weighting on SECRET.



(c) Effect of source data on SECRET.



(d) Effect of target data on SECRET.

Figure C.3: Effect of various parameters on SECRET.

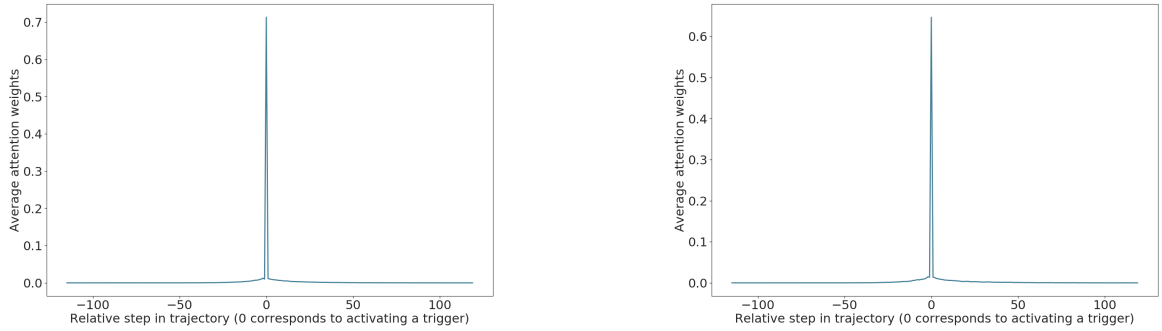


Figure C.4: On the **left**, the distribution of attention weights around triggers for correct positive reward predictions in a Triggers maze from an out-of-domain distribution (bigger mazes). The x-axis denotes the signed number of steps between the state-action couple receiving attention and the closest actual moment the agent activated a switch. On the **right**, same figure for another out-of-domain distribution (inverted dynamics).

Appendix D

Annex of Chapter 7

In this annex we provide additional proofs and details about our experimental setup and the hyperparameters we use in Chapter 7.

D.1 Proof of Thm. 2

Theorem. *The cost function satisfies the following Bellman equation:*

$$C^{\pi_+}(s) = -\eta(1-\pi_+(\bar{a}|s)) + \gamma \mathbb{E}_{a\sim\pi, s'\sim\mathcal{P}(\cdot|s,a)} C^{\pi_+}(s').$$

Proof.

$$\begin{aligned}
V_+^{\pi_+}(s) &:= \sum_{a\in\mathcal{A}_+} \pi_+(a|s)\Big(r_+(s,a) + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s')\Big]\Big) \\
&= \sum_{a\in\mathbb{A}} \Big(\pi(a|s) - \pi_+(\bar{a}|s)\bar{\pi}(a|s)\Big)\Big(r(s,a) - \eta + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s)\Big]\Big) \\
&\quad + \pi_+(\bar{a}|s)\mathbb{E}_{\bar{\pi}}\Big[r(s,a) + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s)\Big]\Big] \\
&= \sum_{a\in\mathbb{A}} \pi(a|s)\Big(r(s,a) - \eta + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s)\Big]\Big) \\
&\quad - \pi_+(\bar{a}|s)\sum_{a\in\mathbb{A}} \bar{\pi}(a|s)\Big(r(s,a) - \eta + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s)\Big]\Big) \\
&\quad + \pi_+(\bar{a}|s)\mathbb{E}_{\bar{\pi}}\Big[r(s,a) + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s)\Big]\Big] \\
&= \sum_{a\in\mathbb{A}} \pi(a|s)\Big(r(s,a) + \gamma \mathbb{E}_{s'}\Big[V_+^{\pi_+}(s)\Big]\Big) - \eta + \pi_+(\bar{a}|s)\eta.
\end{aligned}$$

So,

$$\begin{aligned}
V^{\pi}(s) + C^{\pi_+}(s) = \mathbb{E}_{a\sim\pi, s'\sim s,a}\Big(r(s,a) + \gamma \mathbb{E}_{s'}\Big[V^{\pi}(s') + C^{\pi_+}(s')\Big]\Big) \\
- \eta(1-\pi_+(\bar{a}|s)),
\end{aligned}$$

$\Leftrightarrow$

$$C^{\pi_+}(s) = -\eta(1-\pi_+(\bar{a}|s)) + \gamma \mathbb{E}_{a\sim\pi, s'\sim s,a}\Big[C^{\pi_+}(s')\Big],$$

as

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi, s'\sim s,a}\Big(r(s,a) + \gamma \mathbb{E}_{s'}\Big[V^{\pi}(s')\Big]\Big).$$

□

D.2 Proof of Prop. 1

Property 4.

$$\begin{aligned} V_+^{\pi_+}(s) =& (1-\pi_+(\bar{a}|s))\mathbb{E}_{a\sim\pi_{\setminus\bar{a}}}\Big[Q^{\pi_+}_{\setminus\bar{a}}(s,a)\Big] \\ &+ \pi_+(\bar{a}|s)\Big(\mathbb{E}_{a\sim\bar{\pi}}\Big[Q^{\pi_+}_{\setminus\bar{a}}(s,a)\Big] + \eta\Big). \end{aligned}$$

Proof. We separately compute V^{π} and C^{π_+}, the two components of $V_+^{\pi_+}$. First we have:

$$\begin{aligned} V^{\pi}(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) Q^{\pi}(s,a) \\ &= \sum_{a\in\mathcal{A}} \pi_+(a|s) Q^{\pi}(s,a) + \pi_+(\bar{a}|s) \sum_{a\in\mathcal{A}} \bar{\pi}(a|s) Q^{\pi}(s,a) \\ &= (1-\pi_+(\bar{a}|s))\mathbb{E}_{a\sim\pi_{\setminus\bar{a}}}\Big[Q^{\pi}(s,a)\Big] + \pi_+(\bar{a}|s)\mathbb{E}_{a\sim\bar{\pi}}\Big[Q^{\pi}(s,a)\Big]. \end{aligned}$$

Second, we have:

$$\begin{aligned} C^{\pi_+}(s) &= -(1-\pi_+(\bar{a}|s))\eta + \gamma \sum_{a\in\mathcal{A}} \pi(a|s)\mathbb{E}_{s'}\Big[C^{\pi_+}(s')\Big] \\ &= -(1-\pi_+(\bar{a}|s))\eta \\ &\quad + \gamma \sum_{a\in\mathcal{A}} \Big(\pi_+(a|s) + \pi_+(\bar{a}|s)\bar{\pi}(a|s)\Big)\mathbb{E}_{s'}\Big[C^{\pi_+}(s')\Big] \\ &= (1-\pi_+(\bar{a}|s))\Big(-\eta + \gamma\mathbb{E}_{a\sim\pi_{\setminus\bar{a}}}\Big[C^{\pi_+}(s')\Big]\Big) \\ &\quad + \pi_+(\bar{a}|s)\gamma\mathbb{E}_{a\sim\bar{\pi}}\Big[C^{\pi_+}(s')\Big]. \end{aligned}$$

Consequently,

$$\begin{aligned} V_+^{\pi_+}(s) &= V^{\pi}(s) + C^{\pi_+}(s) \ \textit{(from Theorem 2)} \\ &= (1-\pi_+(\bar{a}|s))\mathbb{E}_{a\sim\pi_{\setminus\bar{a}}}\Big[Q^{\pi_+}_{\setminus\bar{a}}(s,a)\Big] \\ &\quad + \pi_+(\bar{a}|s)\Big(\mathbb{E}_{a\sim\bar{\pi}}\Big[Q^{\pi_+}_{\setminus\bar{a}}(s,a)\Big] + \eta\Big). \end{aligned}$$

□

D.3 Proof of Prop. 2

Property 5.

$$Q_+^{\pi_+}(s,a) = \begin{cases} Q_{\setminus\bar{a}}^{\pi_+}(s,a) & \text{if } a \neq \bar{a}, \\ \mathbb{E}_{a\sim\bar{\pi}}\left[Q_{\setminus\bar{a}}^{\pi_+}(s,a)\right] + \eta & \text{if } a = \bar{a}. \end{cases}$$

Proof.

$$\begin{aligned} Q_+^{\pi_+}(s,a) &:= r_+(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}_+(.|s,a)}\left[V_+^{\pi_+}(s')\right] \\ &= \begin{cases} r(s,a) - \eta + \gamma \mathbb{E}_{s'\sim\mathcal{P}(.|s,a)}\left[V_+^{\pi_+}(s')\right] & \text{if } a \neq \bar{a} \\ \mathbb{E}_{a\sim\bar{\pi}}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(.|s,a)}\left[V_+^{\pi_+}(s')\right]\right] & \text{if } a = \bar{a} \end{cases} \\ &= \begin{cases} Q_{\setminus\bar{a}}^{\pi_+}(s,a) & \text{if } a \neq \bar{a} \\ \mathbb{E}_{a\sim\bar{\pi}}\left[Q_{\setminus\bar{a}}^{\pi_+}(s,a)\right] + \eta & \text{if } a = \bar{a} \end{cases}. \end{aligned}$$

□

D.4 Proof of Prop. 3

Property 6. *The following policy π_+ is greedy with respect to Q_+:*

$$\pi_+(\cdot|s) = \begin{cases} \frac{\mathbb{I}\left\{a \in \arg\max_{\mathcal{A}} Q_{+\setminus\bar{a}}(s,a)\right\}}{\left|\arg\max_{\mathcal{A}} Q_{+\setminus\bar{a}}(s,a)\right|} & \text{if } G_{Q_{+\setminus\bar{a}}}(s) > \eta, \\ \mathbb{I}\left\{a = \bar{a}\right\} & \text{otherwise.} \end{cases}$$

Proof.

$$\begin{aligned} \max_{\mathcal{A}} Q_+(s,\cdot) > Q_+(s,\bar{a}) &\Leftrightarrow \max_{\mathcal{A}} Q_{+\setminus\bar{a}}(s,a) > \mathbb{E}_{\bar{\pi}}\left[Q_{+\setminus\bar{a}}(s,a)\right] + \eta \\ &\Leftrightarrow G_{Q_{+\setminus\bar{a}}}(s) > \eta. \end{aligned}$$

□

D.5 Proof of Thm. 3

Theorem. *$\mathcal{T}$ is a γ-contraction, and converges to $Q_{\setminus\bar{a}}^{\pi_+^*}$ where π_+^* is the optimal policy in the lazy-MDP.*

Proof. First, we define the operator in the augmented MDP

$$\mathcal{T}_+ : Q_+(s,a) \to r_+(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}_+(.|s,a)}\left[\max_{\mathcal{A}_+} Q_+(s',.)\right]$$

which is known to be a contraction, and its successive applications converges to the unique, optimal fixed point, $Q_+^*(s, a \in \mathcal{A}_+)$:

$$Q_+^*(s,a) = r_+(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}_+(.|s,a)}\left[\max_{\mathcal{A}_+} Q_+^*(s',.)\right].$$

Now, let $Q_{\setminus \bar{a}_1}(s, a \in \mathcal{A})$ and $Q_{\setminus \bar{a}_2}(s, a \in \mathcal{A})$ be arbitrary Q-function of the base MDP, let s, a be any state-action pair in the base MDP, and assume without loss of generality that $TQ_{\setminus \bar{a}_1}(s, a) \geq TQ_{\setminus \bar{a}_2}(s, a)$. We have

$$\begin{aligned}
0 &\leq |\mathcal{T}Q_{\setminus \bar{a}_1}(s, a) - \mathcal{T}Q_{\setminus \bar{a}_2}(s, a)| \\
&= \mathcal{T}Q_{\setminus \bar{a}_1}(s, a) - \mathcal{T}Q_{\setminus \bar{a}_2}(s, a) \\
&= \mathcal{T}_+ Q_{1+}(s, a) - \mathcal{T}_+ Q_{2+}(s, a) \\
&= \gamma \mathbb{E}_{s'|s,a}[Q_{1+}(s', a_1^*) - \underbrace{Q_{2+}(s', a_2^*)}_{\geq Q_{2+}(s', a_1^*)}] \text{ with } a_i^* \in \arg\max_{\mathcal{A}^+} Q_i(s', \cdot) \\
&\leq \gamma \mathbb{E}_{s'|s,a}[Q_{1+}(s', a_1^*) - Q_{2+}(s', a_1^*)].
\end{aligned}$$

If $a_1^* \neq \bar{a}$, we have

$$\begin{aligned}
&\gamma \mathbb{E}_{s'|s,a}\Big[Q_{1+}(s', a_1^*) - Q_{2+}(s', a_1^*)\Big] \\
&\qquad = \gamma \mathbb{E}_{s'|s,a}\Big[Q_{\setminus \bar{a}_1}(s', a_1^*) - Q_{\setminus \bar{a}_{2+}}(s', a_1^*)\Big] \\
&\qquad \leq \Big\| Q_{\setminus \bar{a}_1} - Q_{\setminus \bar{a}_2} \Big\|_\infty .
\end{aligned}$$

If $a_1^* = \bar{a}$, we have

$$\begin{aligned}
&\gamma \mathbb{E}_{s'|s,a}\Big[Q_{1+}(s', a_1^*) - Q_{2+}(s', a_1^*)\Big] \\
&\quad = \gamma \mathbb{E}_{s'|s,a}\Big[\mathbb{E}_{\bar{\pi}}\big[\hat{Q}_1(s', a') + \eta\big] - \mathbb{E}_{\bar{\pi}}\big[\hat{Q}_2(s', a') + \eta\big]\Big] \\
&\quad \leq \Big\| Q_{\setminus \bar{a}_1} - Q_{\setminus \bar{a}_2} \Big\|_\infty .
\end{aligned}$$

Thus, we can conclude that $\|\mathcal{T}Q_{\setminus \bar{a}_1} - \mathcal{T}Q_{\setminus \bar{a}_2}\|_\infty \leq \gamma \|Q_{\setminus \bar{a}_1} - Q_{\setminus \bar{a}_2}\|_\infty$, making $\mathcal{T}$ a γ-contraction.

Now we show that successively applied, it converges to $Q_{\setminus \bar{a}}^{\pi_+^*}$.

Let $\mathcal{T}_+^N(q_+) = \underbrace{\mathcal{T}_+ \circ \ldots \circ \mathcal{T}_+}_{N \text{times}}(Q_+)$. As $\mathcal{T}_+^N(Q_+)$ converges to Q_+^* with N, we have:

$$\forall \epsilon > 0, \quad \exists N_\epsilon, \quad \Big\| \mathcal{T}_+^{N_\epsilon}(Q_+) - Q_+^* \Big\|_\infty < \epsilon$$

Besides, for $a \neq \bar{a}$, we have:

$$\begin{aligned}
\mathcal{T}_+(Q_+)(s, a) &= \mathcal{T}(Q_{\setminus \bar{a}})(s, a), \\
Q_+^*(s, a) &= Q_{\setminus \bar{a}}^*(s, a),
\end{aligned}$$

so,

$$\forall s \in \mathcal{S}, a \in \mathcal{A} \quad \Big| \mathcal{T}^{N_\epsilon}(Q_{\setminus \bar{a}})(s, a) - Q_{\setminus \bar{a}}^*(s, a) \Big| = \Big| \mathcal{T}_+^{N_\epsilon}(Q_+)(s, a) - Q_+^*(s, a) \Big|,$$

and

$$\begin{aligned}\left\|\mathcal{T}^{N_\epsilon}(Q_{\setminus\bar{a}}) - Q^*_{\setminus\bar{a}}\right\|_\infty &= \max_{s\in\mathcal{S},a\in\mathcal{A}}\left|\mathcal{T}^{N_\epsilon}(Q_{\setminus\bar{a}})(s,a) - Q^*_{\setminus\bar{a}}(s,a)\right| \\ &= \max_{s\in\mathcal{S},a\in\mathcal{A}}\left|\mathcal{T}^{N_\epsilon}_+(Q_+)(s,a) - Q^*_+(s,a)\right| \\ &\leq \max_{s\in\mathcal{S},a\in\mathcal{A}_+}\left|\mathcal{T}^{N_\epsilon}_+(Q_+)(s,a) - Q^*_+(s,a)\right| \\ &= \left\|\mathcal{T}^{N_\epsilon}_+(Q_+) - Q^*_+\right\|_\infty < \epsilon\end{aligned}$$

□

D.6 Proof of Thm. 4

Theorem 8. *Let $Q^{\bar{\pi}}(s, a \in \mathcal{A})$ be the Q-function of the default policy in the base MDP. Then: $\eta_{max} = \max_s G_{Q^{\bar{\pi}}}(s)$.*

Proof. Since the lazy-gap does not depend on η when the agent always follows the default policy:

$$\eta_{\max} = \sup\left\{\eta > 0 \,|\; \exists s,\;\; G^*(s) = \eta\right\} = \max_s G^*(s).$$

□

D.7 Proof of Thm. 5

Theorem. *Let π^* be the optimal policy in the base MDP, and $Q^*(s, a \in \mathcal{A})$ the associated Q-function. Then:*

$$\eta_{min} = \min_s \max_a \frac{Q^*(s,a) - \mathbb{E}_{\bar{\pi}}\Big[Q^*(s,\cdot)\Big]}{1 + \left(\mathbb{E}_{\pi^*}\Big[Z^{\pi^*}(s,\cdot)\Big] - \mathbb{E}_{\bar{\pi}}\Big[Z^{\pi^*}(s,\cdot)\Big]\right)},$$

with $\eta_{min} \geq 0$.

Proof.

Corollary 2. *Let $x(t) = \frac{u(t)}{1+v(t)}$ for all t where $x(t) \geq 0$, and $x_{min} = \min_t x(t)$, $x_{max} = \max_t x(t)$ Then:*

$$x_{min} = \min_t \Big(u(t) - x_{min}v(t)\Big)$$
$$x_{max} = \max_t \Big(u(t) - x_{max}v(t)\Big)$$

Proof of corollary:
We have, for all t, $x_{\min} = x(t^*) \leq x(t)$. Since $x(t) > 0$, we have for all t,

$$u(t) - x(t^*)v(t) \geq u(t) - x(t)v(t) = x(t)$$

And since $x(t) > x(t^*)$, for all t,

$$u(t) - x(t^*)v(t) \geq x(t^*) = u(t^*) - x(t^*)v(t^*),$$

which means that

$$x(t^*) = \min_t \Big(u(t) - x(t^*)v(t)\Big)$$

Similarly, we have, for all t, $x_{\max} = x(T^*) \geq x(t)$. Since $x(t) > 0$, we have for all t,

$$u(t) - x(T^*)v(t) \leq u(t) - x(t)v(t) = x(t)$$

And since $x(t) \leq x(T^*)$, for all t,

$$u(t) - x(T^*)v(t) \leq x(T^*) = u(T^*) - x(T^*)v(T^*),$$

which means that

$$x(T^*) = \max_t \Big(u(t) - x(T^*)v(t)\Big)$$

So we prove both qualities of the corollary. An immediate consequence of this corollary is that if $x(s,t) = \frac{u(s,t)}{1+v(s,t)}$ with $v(s,t) \geq 0$, then $x_{\text{minimax}} = \min_s \max_t x(s,t)$ verifies:

$$x_{\text{minimax}} = \min_s \max_t \Big(u(s,t) - x_{\text{minimax}}v(s,t)\Big)$$

Now, we will use $u(s,a) = Q^*(s,a) - \mathbb{E}_{\bar{\pi}}\Big[Q^*(s,.)\Big]$ and $v(s,a) = \mathbb{E}_{\pi^*}\Big[Z^{\pi^*}(s,.)\Big] - \mathbb{E}_{\bar{\pi}}\Big[Z^{\pi^*}(s,.)\Big]$. Note that $u(s,a) > 0$. If $v(s,a) \leq -1$ then whatever is η, it is always better to follow π^* (indeed, the lazy-gap is always larger than η):

$$\begin{aligned} G_{Q^*_{\setminus \bar{a}}}(s) &= \max_a Q^*_{\setminus \bar{a}}(s,a) - \mathbb{E}_{\bar{\pi}}\Big[Q^*_{\setminus \bar{a}}(s,.)\Big] \\ &= \max_a Q^{\pi^*}(s,a) - \eta\mathbb{E}_{\pi^*}\Big[Z^{\pi^*}(s,.)\Big] - \mathbb{E}_{\bar{\pi}}\Big[Q^{\pi^*}(s,.) - \eta Z^{\pi^*}(s,.)\Big] \\ &= \max_a u(s,a) - \eta v(s,a) > \eta \end{aligned}$$

Hence we only consider $v(s,a) > -1$. In that case, $x(s,a) = \frac{u(s,a)}{1+v(s,a)} > 0$ and we can apply equ. D.7 to $\eta^* = x_{\text{minimax}}$:

$$\begin{aligned} \eta^* &= \min_s \max_a \Big(u(s,a) - \eta^* v(s,a)\Big) \\ &= \min_s G_{Q^*_{\setminus \bar{a}}}(s) \end{aligned}$$

□

D.8 Effect of the cost

In order to empirically validate the cost boundaries we established in Chapter 7.5, we display in Fig. D.1 the frequency of lazy actions played by an optimal agent as a function of η in the R&B and KDT environments, under different default policies. As expected, when $\eta < \eta_{\min}$ no lazy actions are ever selected, and when $\eta > \eta_{\max}$, the agent always chooses the lazy action.

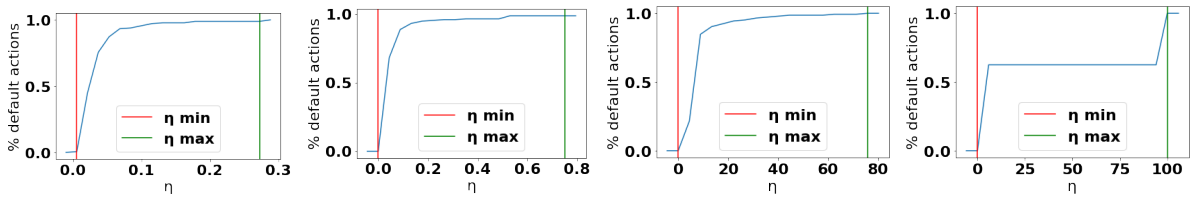


Figure D.1: Frequency of the lazy action as a function of η in various scenarios. With $\eta < \eta_{\min}$, the agent never uses the lazy action, and with $\eta > \eta_{\max}$, the agent always uses the lazy action. Note that the empirical values for $\eta_{\min}$ and $\eta_{\max}$ match the values calculated with the theoretical expressions from Chapter 7.5. **(a)** KDT with a uniform random default policy. **(b)** KDT with a default policy that always takes second-best actions. **(c)** R&B with a uniform random default policy. **(d)** R&B with a default policy that always takes second-best actions.

D.9 Implementation details

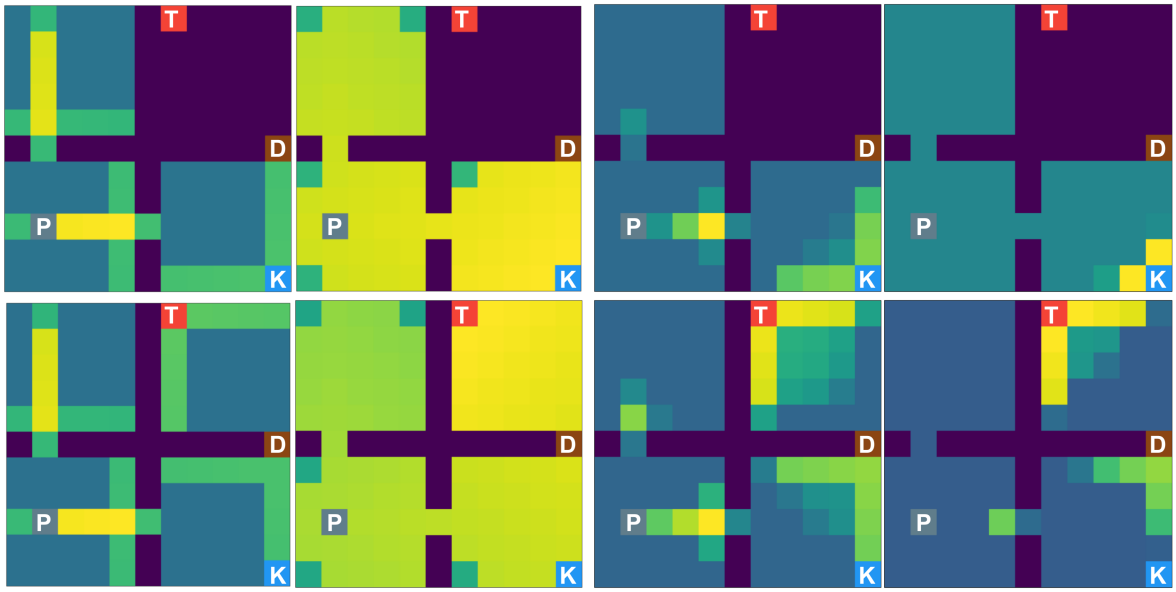
For the Atari experiments, we used the default architectures and hyperparameters in the standard implementations from Dopamine. Details of the networks architecture, learning procedure and hyperparameter selection are described in Castro et al. (2018). For the discrete environments (KDT and R&B) we could compute exact value functions via Value Iteration up to convergence. In the exploration experiment, we assumed that the agent had no knowledge of the dynamics of the environment and used tabular Q-learning with learning rate $\alpha = 0.5$, epsilon-greedy exploration starting at $\epsilon_0 = 0.1$ and linearly decayed until $\epsilon_\infty = 0$. The training is performed over 100 steps containing 1000 episodes of maximum length 1000.

D.9.1 Lazy-gap as a measure of state importance

Fig. D.2 displays states importance according to these measures on the KDT environment. As visible, the lazy-gap only attributes importance to the states that lead to key actions (picking up the keys, passing through the doors, reaching the treasure). On the other hand, the action-gap uniformly emphasizes all the states along the trajectory of the optimal policy, while the importance advice is dominated by temporal proximity to the rewards and does not discriminate key actions.

D.9.2 Atari curves

We display the in-game scores of DQN, DQN with warm restarts, and DQN in Lazy-MDPs in all Atari 2600 games that are mentioned in Chapter 7.7.4 in Fig. D.3. For each game, two figures are provided: the left figure contains the scores, the right figure contains the fraction of controls (between 0 and 1, 0 being maximally lazy, and 1 taking control in all states) by the agent in the lazy-MDP.



(a) action gap (b) important advice (c) lazy-gap $\eta = 0.03$ (d) lazy-gap $\eta = 0.05$

Figure D.2: State importance according to different metrics. Top row: KDT without the key. Bottom row: KDT with the key. (a) Importance according to the action-gap. (b) Importance according to importance advice. (c) Importance according to the lazy-gap with $\eta = 0.03$. (d) Importance according to the lazy-gap with $\eta = 0.05$. Brighter colors indicate higher importance.

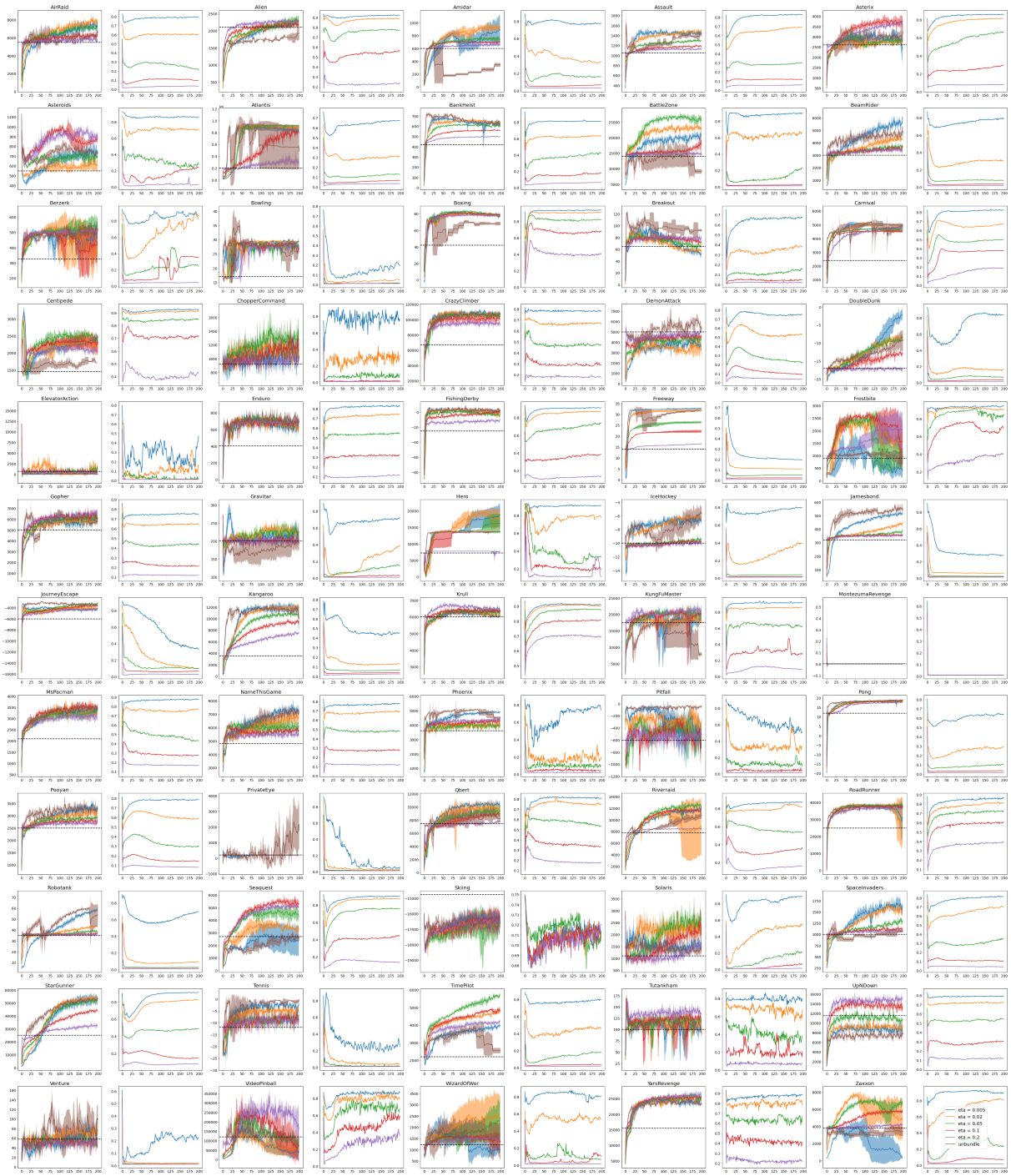


Figure D.3: Performances on all 60 Atari 2600 games.

Appendix E

Annex of Chapter 8

We organize the supplementary material for Chapter 8 as follows: in Appendix E.1, we include the proofs of results from the main text, as well as additional formalism; in Appendix E.2, we provide additional details about the proposed algorithms, including pseudo-code and figures that did not fit in the main text; and in Appendix E.3, we detail our experimental procedure, including hyperparameters for all methods.

E.1 Mathematical elements and proofs

E.1.1 Possible definitions of reversibility

In this section, we present several intuitive definitions of reversibility in MDPs. We chose the third definition as our reference, which we argue presents several advantages over the others, although they can be interesting in specific contexts. Indeed, Equ. E.1.1 is simpler than Equ. E.1.1, as it does not depends on the discount factor, and more general than Equ. E.1.1, as it does not enforce a fixed number of timesteps for going back to the starting state.

Discounted reward.

$$\phi_{\pi,K}(s,a) \coloneqq \sum_{k>t}^{K} \gamma^{k-t} p_\pi(s_{t+k} = s \mid s_t = s, a_t = a)\,,$$
$$\phi_\pi(s,a) \coloneqq \sum_{k>t}^{\infty} \gamma^{k-t} p_\pi(s_{t+k} = s \mid s_t = s, a_t = a).$$

Fixed time step.

$$\phi_{\pi,K}(s,a) \coloneqq \sup_{k \leq K} p_\pi(s_{t+k} = s \mid s_t = s, a_t = a)\,,$$
$$\phi_\pi(s,a) \coloneqq \sup_{k \in \mathbb{N}} p_\pi(s_{t+k} = s \mid s_t = s, a_t = a).$$

Undiscounted reward.

$$\begin{aligned}\phi_{\pi,K}(s,a) &\coloneqq \sum_{k=1}^{K} p_\pi(s_{t+k}=s, s_{t+k-1}\neq s, \dots, s_{t+1}\neq s \mid s_t=s, a_t=a)\,,\\ &= p_\pi(s\in\tau_{t+1:t+K+1} \mid s_t=s, a_t=a)\,.\\ \phi_{\pi}(s,a) &\coloneqq \sum_{k=1}^{\infty} p_\pi(s_{t+k}=s, s_{t+k-1}\neq s, \dots, s_{t+1}\neq s \mid s_t=s, a_t=a)\,,\\ &= p_\pi(s\in\tau_{t+1:\infty} \mid s_t=s, a_t=a).\end{aligned}$$

E.1.2 Additional properties

We write $s \to s'$ if $\psi_\pi(s,s') \geq 0.5$ ("it is more likely to go from s to s' than to go from s' to s") and $s \Rightarrow s'$ if $\psi_\pi(s,s') = 1$ ("it is possible to go from s to s', but it is not possible to come back to s from s'").

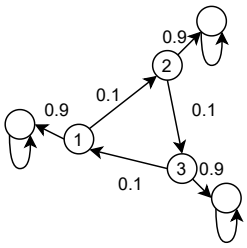


Figure E.1: Counter-example for proposition 4. The initial state is sampled uniformly amongst $\{0,1,2\}$.

1. $\psi_\pi(s,s') + \psi_\pi(s',s) = 1$
2. if $s_0 \Rightarrow s_1 \Rightarrow s_2$ then $s_0 \Rightarrow s_2$ (transitivity for $\Rightarrow$)
3. if $s_0 \to s_1 \to \cdots \to s_i \Rightarrow s_{i+1} \to \cdots \to s_t$ then $s_0 \Rightarrow s_t$
4. in general $s_1 \to s_2$ and $s_2 \to s_3$ doesn't imply $s_1 \to s_3$

Proofs:
(1) $\psi_\pi(s,s') + \psi_\pi(s',s) = \mathbb{E}_{\tau\sim\pi}\mathbb{E}_{t\neq t'|s_t=s,s_{t'}=s'}\left[\mathbb{1}_{t'>t} + \mathbb{1}_{t'<t}\right] = \mathbb{E}_{\tau\sim\pi}\mathbb{E}_{t\neq t'|s_t=s,s_{t'}=s'}\left[1\right] = 1$.
(2) and (3): As (3) is stronger than (2), we only prove (3). If it is possible to have s_0 after s_t in a trajectory, then it is possible to have s_i after s_t. As we have a positive probability of seeing s_t after s_{i+1}, we have a positive probability of seeing s_i after s_{i+1}, which contradicts $s_i \Rightarrow s_{i+1}$.
(4) A counter example can be found in Fig. E.1. In this case we clearly have $s_1 \to s_2$, $s_2 \to s_3$ and $s_3 \to s_1$.

E.1.3 Proofs of Theorem 1 and Theorem 2

In the following, we prove simultaneously Theorem 6 and Theorem 7. We begin by two lemmas.

Lemma 1. *Given a trajectory τ, we denote by $\#_T(s \to s')$ the number of pairs (s,s') in $\tau_{1:T}$ such that s appears before s'. We present a simple formula for $\psi(s',s)$ according to the structure of the state trajectory:*

$$\psi_{\pi,T}(s,s') = \frac{\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s')\right]}{\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s') + \#_T(s'\to s)\right]}\,.$$

Proof. In order to simplify the notations, we leave implicit the fact that indices are always sampled within $[0,T]$.

$$\begin{aligned}\psi_{\pi,T}(s,s') &= \mathbb{E}_\pi\mathbb{E}_{t\neq t'|s_t=s,s_{t'}=s'}\left[\mathbb{1}_{t'>t}\right]\,,\\ &= \frac{\mathbb{E}_\pi\mathbb{E}_{t\neq t'}\left[\mathbb{1}_{t'>t}\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}{\mathbb{E}_\pi\mathbb{E}_{t\neq t'}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}\,.\end{aligned}$$

Similarly, we have:

$$\mathbb{E}_\pi \mathbb{E}_{t'>t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right] = \frac{\mathbb{E}_\pi \mathbb{E}_{t\neq t'}\left[\mathbb{1}_{t'>t}\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}{\mathbb{E}_{t\neq t'}\left[\mathbb{1}_{t'>t}\right]}\,.$$

Combining it with our previous equation:

$$\begin{aligned}\psi_{\pi,T}(s,s') &= \frac{\mathbb{E}_\pi \mathbb{E}_{t'>t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]\mathbb{E}_{t\neq t'}\left[\mathbb{1}_{t'>t}\right]}{\mathbb{E}_\pi \mathbb{E}_{t\neq t'},\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}\,,\\ &= \frac{1}{2}\frac{\mathbb{E}_\pi \mathbb{E}_{t'>t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}{\mathbb{E}_\pi \mathbb{E}_{t\neq t'}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}\,.\end{aligned}$$

Looking at the denominator, we can notice:

$$\begin{aligned}\mathbb{E}_\pi \mathbb{E}_{t\neq t'}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right] &= \frac{1}{2}\mathbb{E}_\pi \mathbb{E}_{t<t'}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right] + \frac{1}{2}\mathbb{E}_\pi \mathbb{E}_{t'<t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]\,,\\ &= \frac{1}{2}\mathbb{E}_\pi \mathbb{E}_{t<t'}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'} + \mathbb{1}_{s_t=s'}\mathbb{1}_{s_{t'}=s}\right]\,,\end{aligned}$$

which comes from the fact that t and t' play a symmetrical role. Thus,

$$\psi_{\pi,T}(s,s') = \frac{\mathbb{E}_{\tau\sim\pi}\mathbb{E}_t \mathbb{E}_{t'>t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]}{\mathbb{E}_{\tau\sim\pi}\mathbb{E}_t \mathbb{E}_{t'>t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'} + \mathbb{1}_{s_t=s'}\mathbb{1}_{s_{t'}=s}\right]}\,.$$

Since

$$\begin{aligned}\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s')\right] &= \sum_{i<j\leq T}\mathbb{1}_{s_i=s}\mathbb{1}_{s_j=s'}\,,\\ &= \binom{T}{2}\sum_{i<j\leq T}\frac{1}{\binom{T}{2}}\mathbb{1}_{s_i=s}\mathbb{1}_{s_j=s'}\,,\\ &= \binom{T}{2}\mathbb{E}_{\tau\sim\pi}\mathbb{E}_t \mathbb{E}_{t'>t}\left[\mathbb{1}_{s_t=s}\mathbb{1}_{s_{t'}=s'}\right]\,,\end{aligned}$$

we get:

$$\begin{aligned}\psi_{\pi,T}(s,s') &= \frac{\binom{T}{2}\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s')\right]}{\binom{T}{2}\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s') + \#_T(s'\to s)\right]}\,,\\ \psi_{\pi,T}(s,s') &= \frac{\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s')\right]}{\mathbb{E}_{\tau\sim\pi}\left[\#_T(s\to s') + \#_T(s'\to s)\right]}\,.\end{aligned}$$

□

Lemma 2. *Assume that we are given a fixed trajectory where s appears $k\in\mathbb{N}$ times, in the form of :*

$$s_0 \underbrace{\longrightarrow}_{n_0(s')} s \underbrace{\longrightarrow}_{n_1(s')} s \underbrace{\longrightarrow}_{n_2(s')} s \underbrace{\longrightarrow}_{n_3(s')} s \cdots \underbrace{\longrightarrow}_{n_{k-1}(s')} s \underbrace{\longrightarrow}_{n_k(s')}\,,$$

where $n_i(s')$ denotes the number of times s' appears between the i^{th} and the $(i+1)^{th}$ occurrence of s. In this case,

$$\#_T(s \to s') = \sum_{i=0}^{k} i \times n_i(s')\,.$$

If we suppose that $n_1(s') = n_2(s') = \cdots = n_{k-1}(s')$, we also have

$$\#_T(s \to s') - \#_T(s' \to s) = k\left(n_k(s') - n_0(s')\right).$$

Proof. Equ. 2 comes directly from $\#_T(s \to s') = \sum_{i=1}^{k}\sum_{j=i}^{k} n_j(s') = \sum_{i=0}^{k} i \times n_i(s')$. To prove Equ. 2, we first notice that $\#_T(s \to s') + \#_T(s' \to s) = k \times \sum_{i=0}^{k} n_i(s')$. Thus

$$\begin{aligned}
\#_T(s \to s') - \#_T(s' \to s) &= 2 \times \#_T(s \to s') - \left(\#_T(s \to s') + \#_T(s' \to s)\right), \\
&= 2\left(k\,n_k(s') + n_1(s')\sum_{i=0}^{k-1} i\right) - \left(k\,n_k(s') + k\,n_0(s') + k\,(k-1)\,n_1(s')\right), \\
&= k\,n_k(s') - k\,n_0(s')\,.
\end{aligned}$$

□

Theorem 1. *For every policy π and $s, s' \in S$, $\psi_{\pi,T}(s, s')$ converges when T goes to infinity.*

Theorem 2. *Given a policy π, a state s, and an action a, we can link reversibility and empirical reversibility with the inequality: $\bar{\phi}_\pi(s,a) \geq \frac{\phi_\pi(s,a)}{2}$.*

Proof. For a policy π and $s, s' \in S$, we define $\hat{\phi}_\pi(s, s')$ the quantity $p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s')$ such that $\phi_\pi(s,a) = \mathbb{E}_{s' \sim P(s,a)}\left[\hat{\phi}_\pi(s, s')\right]$ In order to prove the theorem, we first prove that $\psi_T(s', s)$ converges to a quantity denoted by $\psi(s', s)$, and that:

$$\forall s, s' \in S, \frac{\hat{\phi}^\pi(s, s')}{2} \leq \psi(s', s)\,.$$

We subdivide our problem into four cases, depending on whether s and s' are recurrent or transient.

Case 1: $p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s) < 1$ and $p_\pi(s' \in \tau_{t+1:\infty} \mid s_t = s') = 1$ (s is transient and s' is recurrent for the Markov chain induced by π). Informally, this means that if a trajectory contains the state s' we tend to see s' an infinite number of times, and we only see s a finite number of times in a given trajectory.

This implies $\hat{\phi}_\pi(s, s') = p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s') = 0$, as recurrent states can only be linked to other recurrent states (Norris, 1998). It is not possible to find trajectories where s appears after s', thus $\psi_T(s', s) = 0 = \psi(s', s)$. Equ. E.1.3 becomes "$0 \leq 0$".

Case 2: $p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s) = 1$ and $p_\pi(s' \in \tau_{t+1:\infty} \mid s_t = s') < 1$ (s is recurrent and s' is transient for the Markov chain induced by π).

As before, this implies $\hat{\phi}_\pi(s', s) = p_\pi(s' \in \tau_{t+1:\infty} \mid s_t = s) = 0$, and thus it is not possible to see in a trajectory s after s'. It implies $\psi_T(s', s) = 1 = \psi(s', s)$, so Equ. E.1.3 is verified.

Case 3: $p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s) = 1$ and $p_\pi(s' \in \tau_{t+1:\infty} \mid s_t = s') = 1$ (s is recurrent and s' is recurrent for the Markov chain induced by π). We denote by T_k the random variable corresponding to the time of the k^{th} visit to s. A trajectory can be represented as follows:

$$s_0 \underbrace{\longrightarrow}_{n_1(s')} s \underbrace{\longrightarrow}_{n_2(s')} s \underbrace{\longrightarrow}_{n_3(s')} s \underbrace{\longrightarrow}_{n_4(s')} s \cdots \underbrace{\longrightarrow}_{n_k(s')} s = s_{T_k} \underbrace{\longrightarrow}_{n_{k+1}(s')} ,$$

where, writing $\sim$ the equality in distribution, $n_2(s') \sim n_3(s') \sim \cdots \sim n_k(s')$ and $\mathbb{E}_\tau n_2(s') = \mathbb{E}_\tau n_3(s') = \cdots = \mathbb{E}_\tau n_k(s')$ using the strong Markov property. From Lemma 1 we get:

$$\begin{aligned}
\psi_{\pi,T}(s,s') &= \frac{\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s')\big]}{\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s') + \#_T(s' \to s)\big]}\,, \\
&= \frac{1}{2}\frac{\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s') + \#_T(s' \to s) + \#_T(s \to s') - \#_T(s' \to s)\big]}{\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s') + \#_T(s' \to s)\big]}\,, \\
&= \frac{1}{2} + \frac{\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s') - \#_T(s' \to s)\big]}{2\,\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s') + \#_T(s' \to s)\big]}\,.
\end{aligned}$$

We can see from Lemma 2 :

$$\mathbb{E}_\tau\big[\#_{T_k}(s \to s') - \#_{T_k}(s' \to s)\big] = -k\,\mathbb{E}_\tau n_1(s')\,.$$

Thus,

$$\begin{aligned}
\frac{\mathbb{E}_\tau\big[\#_{T_k}(s \to s') - \#_{T_k}(s' \to s)\big]}{\mathbb{E}_{\tau\sim\pi}\big[\#_{T_k}(s \to s') + \#_{T_k}(s' \to s)\big]} &= \frac{-k\,\mathbb{E}_\tau n_1(s')}{k\,\mathbb{E}_\tau n_1(s') + k^2\,\mathbb{E}_\tau n_2(s')} \\
&\xrightarrow[k\to\infty]{} 0.
\end{aligned}$$

Given $t \in \mathbb{N}$ and a trajectory τ, we denote $\#_T(s)$ the random variable corresponding to the number of times when s appear before t, such that a trajectory has the following structure :

$$s_0 \underbrace{\longrightarrow}_{n_1(s')} s \underbrace{\longrightarrow}_{n_2(s')} s \underbrace{\longrightarrow}_{n_3(s')} s \underbrace{\longrightarrow}_{n_4(s')} s \cdots \underbrace{\longrightarrow}_{n_k(s')} s = s_{T_{\#_T(s)}} \underbrace{\longrightarrow s_t \longrightarrow}_{n_{k+1}(s')} s = s_{T_{\#_T(s)+1}}\,.$$

$$\begin{aligned}
\frac{\mathbb{E}_\tau\big[\#_T(s \to s') - \#_T(s' \to s)\big]}{\mathbb{E}_\tau\big[\#_T(s \to s') + \#_T(s' \to s)\big]} &\leq \frac{\mathbb{E}_\tau\big[\#_{\#_T(s)}(s \to s') - \#_{\#_T(s)}(s' \to s)\big] + \mathbb{E}_\tau \#_T(s) n_{k+1}(s')}{\mathbb{E}_\tau \#_{\#_T(s)}(s \to s') + \mathbb{E}_\tau \#_{\#_T(s)}(s' \to s)}\,, \\
&\xrightarrow[T\to\infty]{} 0 \text{ as in Equ. E.1.3.}
\end{aligned}$$

And,

$$\begin{aligned}
\frac{\mathbb{E}_\tau\big[\#_T(s \to s') - \#_T(s' \to s)\big]}{\mathbb{E}_\tau\big[\#_T(s \to s') + \#_T(s' \to s)\big]} &\geq \frac{\mathbb{E}_\tau\big[\#_{\#_T(s)}(s \to s') - \#_{\#_T(s)}(s' \to s)\big] - \mathbb{E}_\tau \sum_{i=1}^{\#_T(s)+1} n_i(s')}{\mathbb{E}_\tau\big[\#_{\#_T(s)}(s \to s') + \#_{\#_T(s)}(s' \to s)\big]}\,, \\
&\xrightarrow[T\to\infty]{} 0
\end{aligned}$$

Therefore,

$$\frac{\mathbb{E}_\tau\big[\#_T(s \to s') - \#_T(s' \to s)\big]}{\mathbb{E}_\tau\big[\#_T(s \to s') + \#_T(s' \to s)\big]} \xrightarrow[T\to\infty]{} 0 \text{ , and finally,}$$

$$\psi_{\pi,T}(s,s') = \frac{1}{2} + \frac{\mathbb{E}_\tau\big[\#_T(s \to s') - \#_T(s' \to s)\big]}{2\mathbb{E}_{\tau\sim\pi}\big[\#_T(s \to s') + \#_T(s' \to s)\big]} \xrightarrow[T\to\infty]{} \frac{1}{2}\,.$$

As $\hat{\phi}_\pi(s,s') = 1$ here, we immediately have $\frac{\hat{\phi}_\pi(s,s')}{2} = \psi(s',s)$. We can notice that the inequality is tight in this case.

Case 4: $p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s) < 1$ and $p_\pi(s' \in \tau_{t+1:\infty} \mid s_t = s') < 1$ (s is transient and s' is transient for the Markov chain induced by π). To simplify the following formulas, we will write $\alpha = p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s')$. Here, we denote by $\#(s)$ the random variable corresponding to the total number of visits of the state s, and $\#(s \to s')$ the number of pairs such that s appears before s'. $\#(s)$ follows the geometric distribution $G\left(1 - p_\pi(s \in \tau_{t+1:\infty} \mid s_t = s)\right)$.

$\#_T(s \to s')$ converges almost surely to $\#(s \to s')$, and we have $\#_T(s \to s') \leq \#(s \to s')$. Therefore, using the dominated convergence theorem, $\mathbb{E}_\tau\big[\#_T(s \to s')\big] \xrightarrow[T\to\infty]{} \mathbb{E}_\tau\big[\#(s \to s')\,\big]$, and thus:

$$\psi_{\pi,T}(s',s) = \frac{\mathbb{E}_\tau\big[\#_T(s' \to s)\big]}{\mathbb{E}_\tau\big[\#_T(s \to s') + \#_T(s' \to s)\big]} \xrightarrow[T\to\infty]{} \frac{\mathbb{E}_\tau \#(s' \to s)}{\mathbb{E}_\tau\big[\#(s \to s') + \#(s' \to s)\big]} = \psi^\pi(s',s)\,.$$

This time, we consider a trajectory τ where s appears k times after s', such that it is of the form:

$$\underbrace{s' \dots s'}_{n_0(s')\geq 0} \longrightarrow \underbrace{s \dots s}_{n_1(s)>0} \longrightarrow \underbrace{s' \dots s'}_{n_1(s')>0} \longrightarrow \underbrace{s \dots s}_{n_2(s)>0} \longrightarrow \cdots \longrightarrow \underbrace{s' \dots s'}_{n_{k-1}(s')>0} \longrightarrow \underbrace{s \dots s}_{n_k(s)>0} \longrightarrow \underbrace{s' \dots s'}_{n_k(s')\geq 0} \longrightarrow$$

Here, $n_0(s')$ is the number of times when s' appears in the trajectory before the first appearance of s', $n_i(s)$ is the number of times when s appears between two occurrences of s', and $n_k(s')$ the number of times when s' appears after the last appearance of s. From the strong Markov property, $n_1(s') \sim n_2(s') \sim \cdots \sim n_{k-1}(s')$ and $n_1(s) \sim n_2(s) \sim \cdots \sim n_k(s)$. Note also that these variables are all independent. Here k is a random variable following the geometric distribution $G(\alpha)$ where $\alpha = p(s \in \tau_{t:\infty} \mid s_t = s')$. Notice that when $n_k(s') > 0$, we have $n_k(s) \sim n_1(s)$ and $n_k(s') \sim n_1(s')$.

Using these two simplifications, one can write:

$$\begin{aligned}
\mathbb{E}_\tau\Big[\#(s' \to s) - \#(s \to s')\Big|k\Big] &\geq \mathbb{E}_\tau\Big[\#(s' \to s) - \#(s \to s')\Big|k, n_k(s') > 0\Big]\,,\\
&\geq \mathbb{E}_\tau\Big[n_0(s')\big[n_1(s) + (k-1)n_1(s) + n_k(s)\big] - n_1(s)\big[kn_1(s') + n_k(s')\big] +\\
&\quad n_k(s)\big[kn_1(s') - n_k(s')\big] - n_k(s')(k-1)n_1(s)\Big|k, n_k(s') > 0\Big]\,,\\
&\geq -k\mathbb{E}_\tau\Big[n_1(s)\Big|k\Big]\mathbb{E}_\tau\Big[n_k(s')\Big|k, n_k(s') > 0\Big] \text{ as in Lemma 2}\,,\\
&\geq -k\mathbb{E}_\tau(n_1(s))\mathbb{E}_\tau(n_1(s'))\,.
\end{aligned}$$

Likewise,

$$\mathbb{E}_\tau\big[\#(s' \to s) + \#(s \to s') \,\big|\, k\big] = \mathbb{E}_\tau\Big[k\, n_1(s) n_k(s') + k\, n_0(s') n_1(s) + k\,(k-1) n_1(s) n_1(s') \Big| k\Big]\,,$$
$$= k\Big[\mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big] + \mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_0(s')\big]\Big] + k(k-1)\mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big]\,.$$

Thus,

$$\frac{\mathbb{E}_\tau\big[\#(s' \to s) - \#(s \to s')\big]}{\mathbb{E}_\tau\big[\#(s \to s') + \#(s' \to s)\big]} = \frac{\sum_{i=1}^\infty p(k=i)\mathbb{E}_\tau\left[\#(s' \to s) - \#(s \to s') \,|\, k=i\right]}{\sum_{i=1}^\infty p(k=i)\mathbb{E}_\tau\left[\#(s \to s') + \#(s' \to s) \,|\, k=i\right]}\,,$$
$$\geq -\frac{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\, i\, \mathbb{E}_\tau\big[n_1(s)\big]\, \mathbb{E}_\tau\big[n_1(s')\big]}{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\Big[i\left(\mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big] + \mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_0(s')\big]\right) + i\,(i-1)\mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big]\Big]}\,,$$

$$\geq -\frac{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha) i \mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big]}{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\Big[i\,\mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big] + i\,(i-1)\mathbb{E}_\tau\big[n_1(s)\big]\mathbb{E}_\tau\big[n_1(s')\big]\Big]}\,,$$
$$\geq -\frac{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\, i}{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\big[i + i\,(i-1)\big]}\,,$$
$$\geq -\frac{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\, i}{\sum_{i=1}^\infty \alpha^{i-1}(1-\alpha)\, i^2}\,,$$
$$\geq -\frac{\frac{1}{1-\alpha}}{\frac{1+\alpha}{(1-\alpha)^2}}\,,$$
$$\geq -\frac{1-\alpha}{1+\alpha}\,.$$

From Lemma 1,

$$\psi^\pi(s', s) = \frac{1}{2}\left(1 + \frac{\mathbb{E}_\tau\big[\#(s' \to s) - \#(s \to s')\big]}{\mathbb{E}_{\tau\sim\pi}\big[\#(s \to s') + \#(s' \to s)\big]}\right)\,,$$
$$\geq \frac{1}{2}\left(1 - \frac{1-\alpha}{1+\alpha}\right)\,,$$
$$\geq \frac{\alpha}{1+\alpha}\,,$$
$$\geq \frac{\alpha}{2} = \frac{\hat{\phi}_\pi(s, s')}{2}\,.$$

As a quick summary, we divided our problem in 4 cases, and proved that in each case, for every pair of states s, s', we have $\psi^\pi(s', s) \geq \frac{\hat{\phi}_\pi(s,s')}{2}$.

To end the proof, we simply take the expectation over the distribution of the next states:

$$\mathbb{E}_{s'\sim P(s,a)}\psi_\pi(s', s) \geq \frac{1}{2}\mathbb{E}_{s'\sim P(s,a)}\hat{\phi}_\pi(s, s')\,,$$
$$\bar{\phi}_\pi(s, a) \geq \frac{\phi_\pi(s, a)}{2}\,.$$

□

E.1.4 Proof of Proposition 1

Proposition 1. *We suppose that we are given a state s, an action a such that a is reversible in K steps, a policy π and $\rho > 0$. Then, $\bar{\phi}_\pi(s,a) \geq \frac{\rho^K}{2}$, where A denotes the number of actions. Moreover, we have for all $K \in \mathbb{N}$: $\bar{\phi}_\pi(s,a) \geq \frac{\rho^K}{2}\phi_K(s,a)$.*

Proof. We first prove the second part of the proposition, which is more general. From Definition 8, and as the set of policies is closed, there is a policy π^* such that $\phi_K(s,a) = p_{\pi^*}(s \in \tau_{t+1:t+K+1} \mid s_t = s, a_t = a)$. We begin by noticing that π has a probability at least equal to ρ to copy the policy π^* in every state.

It can be stated more formally:

$$\forall s \in S, \mathbb{E}_{a\sim\pi(s),a^*\sim\pi^*(s)}(\mathbb{1}_{a=a'}) = \sum_{a\in A} p_\pi(a \mid s) p_{\pi^*}(a \mid s) \geq \rho\Big(\sum_{a\in A} p_{\pi^*}(a \mid s)\Big) = \rho\,.$$

Then, we have:

$$\begin{aligned}
&\phi_{\pi,K}(s,a) = p_\pi(s \in \tau_{t+1:t+K+1} \mid s_t = s, a_t = a)\,,\\
&= \mathbb{E}_\pi\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a\big]\,,\\
&= \mathbb{E}_{s_{t+2},\dots,s_{t+K+1}\sim\pi}\mathbb{E}_{s_{t+1}\sim p(s_t,a_t)}\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a\big]\,,\\
&= \mathbb{E}_{s_{t+3},\dots,s_{t+K+1}\sim\pi}\mathbb{E}_{\substack{a_{t+1}\sim\pi(s_{t+1})\\ s_{t+2}\sim p(s_{t+1},a_{t+1})}}\mathbb{E}_{s_{t+1}\sim p(s_t,a_t)}\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a\big]\,,\\
&= \mathbb{E}_{s_{t+3},\dots,s_{t+K+1}\sim\pi}\mathbb{E}_{\substack{a_{t+1}\sim\pi(s_{t+1}),a'_{t+1}\sim\pi^*(s_{t+1})\\ s_{t+2}\sim p(s_{t+1},a_{t+1})}}\mathbb{E}_{s_{t+1}\sim p(s_t,a_t)}\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a\big]\,,\\
&\geq \mathbb{E}_{s_{t+3},\dots,s_{t+K+1}\sim\pi}\mathbb{E}_{\substack{a_{t+1}\sim\pi(s_{t+1}),a^*_{t+1}\sim\pi^*(s_{t+1})\\ s_{t+2}\sim p(s_{t+1},a_{t+1})\\ s_{t+1}\sim p(s_t,a_t)}}\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a, a_{t+1} = a^*_{t+1}\big]\mathbb{1}_{a_{t+1}=a^*_{t+1}}\,,\\
&\geq \rho\,\mathbb{E}_{s_{t+3},\dots,s_{t+K+1}\sim\pi}\mathbb{E}_{s_{t+1},s_{t+2}\sim\pi^*}\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a\big]\,\text{, and iterating the same process,}\,,\\
&\geq \rho^K\mathbb{E}_{s_{t+1},s_{t+2},\dots,s_{t+K+1}\sim\pi^*}\big[\mathbb{1}_{s\in\tau_{t+1:t+K+1}} \mid s_t = s, a_t = a\big]\,,\\
&\geq \rho^K\phi_K(s,a)\,.
\end{aligned}$$

We can conclude using Theorem 7: $\bar{\phi}_\pi(s,a) \geq \frac{\phi_\pi(s,a)}{2} \geq \frac{\phi_{\pi,K}(s,a)}{2} \geq \frac{\rho^K}{2}\phi_K(s,a)$.

□

E.2 Additional details about Reversibility-Aware RL

E.2.1 Learning a reversibility estimator

We illustrate how the reversibility estimator is trained in Fig. E.2. We remind the reader that it is a component that is specific to RAC. See Algorithm 5 for the detailed procedure of how to train it jointly with the standard precedence estimator and the RL agent.

E.2.2 Pseudo-code for RAE and RAC

We give the pseudo-code for the online versions of RAE (Algorithm 4) and RAC (Algorithm 5). The rejection sampling policy $\bar{\pi}$ under approximate reversibility ϕ and threshold β is expressed as follows:

$$\bar{\pi}(a|x) = \begin{cases} 0 & \text{if } \phi(x,a) < \beta \\ \pi(a|x)/Z & \text{otherwise, with } Z = \sum_{a'\in\mathcal{A}} \mathbb{1}\{\phi(x,a') \geq \beta\}\pi(a'|x) \end{cases}.$$

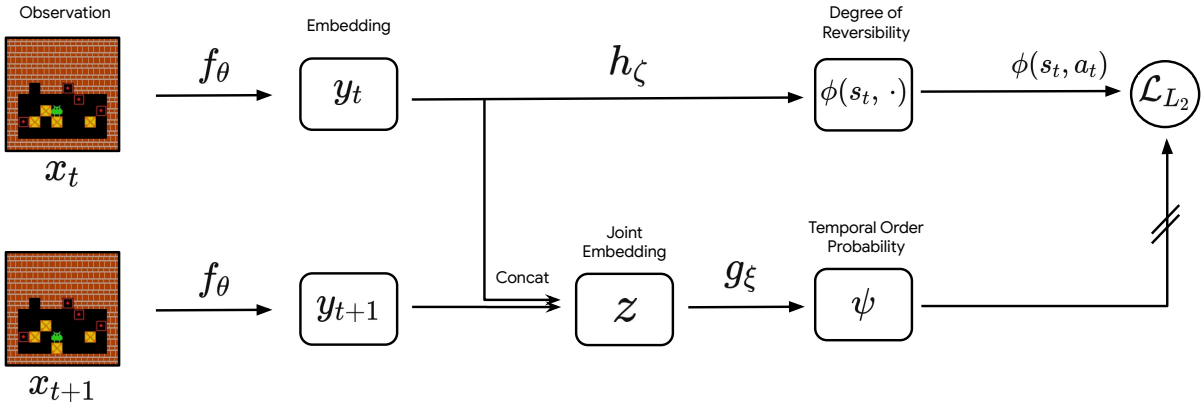


Figure E.2: The training procedure for the reversibility estimator used in RAC.

Algorithm 4: RAE: Reversibility-Aware Exploration (online)

Initialize the agent weights Θ and number of RL updates per trajectory k;
Initialize the precedence classifier weights θ, ξ, window size w, threshold β and learning rate η;
Initialize the replay buffer $\mathcal{B}$;
for *each iteration* **do**
  `/* Collect interaction data and train the agent. */`
  Sample a trajectory $\tau = \{x_i, a_i, r_i\}_{i=1...T}$ with the current policy;
  Incorporate irreversibility penalties $\tau' = \left\{x_i, a_i, r_i + r_\beta\left(\psi_{\theta,\xi}(x_i, x_{i+1})\right)\right\}_{i=1...T}$;
  Store the trajectory in the replay buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \tau$;
  Do k RL steps and update Θ;
  `/* Update the precedence classifier. */`
  for *each training step* **do**
    Sample a minibatch $\mathcal{D}_{batch}$ from $\mathcal{B}$;
    `/* Self-supervised precedence classification, loss in Equ. E.2.2. */`
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{SSL}(\mathcal{D}_{batch})$;
    $\xi \leftarrow \xi - \eta \nabla_\xi \mathcal{L}_{SSL}(\mathcal{D}_{batch})$;
  end
end

Algorithm 5: RAC: Reversibility-Aware Control (online)

Initialize the agent weights Θ and number of RL updates per trajectory k;
Initialize the precedence classifier weights θ, ξ, window size w, threshold β and learning rate η;
Initialize the reversibility estimator weights ζ;
Initialize the replay buffer $\mathcal{B}$;
for *each iteration* **do**
  `/* Collect interaction data with the modified control policy and train the agent. */`
  Sample a trajectory τ under the rejection sampling policy $\bar{\pi}$ from Equ. E.2.2 ;
  Store the trajectory in the replay buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \tau$;
  Do k RL steps and update Θ;
  `/* Update the precedence classifier. */`
  for *each training step* **do**
    Sample a minibatch $\mathcal{D}_{batch}$ from $\mathcal{B}$;
    `/* Self-supervised precedence classification, loss in Equ. E.2.2. */`
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{SSL}(\mathcal{D}_{batch})$;
    $\xi \leftarrow \xi - \eta \nabla_\xi \mathcal{L}_{SSL}(\mathcal{D}_{batch})$;
  end
  `/* Update the reversibility estimator, loss in Equ. E.2.2. */`
  for *each training step* **do**
    Sample a minibatch $\mathcal{D}_{batch}$ from $\mathcal{B}$;
    `/* Regression of the precedence classifier probabilities. */`
    $\zeta \leftarrow \zeta - \eta \nabla_\zeta \mathcal{L}_{L2}(\mathcal{D}_{batch}, \psi_{\theta,\xi})$;
  end
end

This is equivalent, on average, to sampling from the policy π until an action that is reversible enough is found.

The loss we use to train the precedence estimator has the expression:

$$\mathcal{L}_{SSL}(\mathcal{D}_{batch}) = \frac{1}{|\mathcal{D}_{batch}|} \sum_{(x,x',y)\in\mathcal{D}_{batch}} -y \log\big(\psi_{\theta,\xi}(x,x')\big) + (1-y)\log\big(1-\psi_{\theta,\xi}(x,x')\big),$$

where y is the binary result of the shuffle, with value 1 if observations were not shuffled (thus in the correct temporal order), and 0 otherwise. Pairs of observations (x, x') can be separated by up to w timesteps.

The loss we use to train the reversibility estimator (in RAC only) has the expression:

$$\mathcal{L}_{L2}(\mathcal{D}_{batch}, \psi_{\theta,\xi}) = \frac{1}{2\,|\mathcal{D}_{batch}|} \sum_{(x,a,x')\in\mathcal{D}_{batch}} \big(\psi_{\theta,\xi}(x,x') - \phi_\zeta(x,a)\big)^2,$$

where (x, a, x') are triples of state, action and next state sampled from the collected trajectories.

The offline versions of both RAE and RAC can be derived by separating each online algorithm into two parts: 1) training the precedence classifier (and the reversibility estimator for RAC), which is achieved by removing the data collection and RL steps and by using a fixed replay buffer; and 2) training the RL agent, which is the standard RL procedure augmented with modified rewards for RAE, and modified control for RAC, using the classifiers learned in the first part without fine-tuning.

E.3 Experimental details

E.3.1 Reward-free Reinforcement Learning

Cartpole. The observation space is a tabular 4-dimensional vector: (cart position x, cart velocity $\dot{x}$, pole angle θ, pole velocity $\dot{\theta}$). The discrete action space consists of applying a force left or right. The episode terminates if the pole angle is more than $\pm 12°$ ($|\theta| \leq 0.209$ radians), if the cart position is more than ± 2.4, or after 200 time-steps. It is considered solved when the average return is greater than or equal to 195.0 over 100 consecutive trials.

Architecture and hyperparameters. The reversibility network inputs a pair of observations and produces an embedding by passing each one into 2 fully connected layers of size 64 followed by ReLU. The two embeddings are concatenated, and projected into a scalar followed by a sigmoid activation. We trained this network doing 1 gradient step every 500 time steps, using the Adam optimizer (Kingma and Ba, 2014) and a learning rate of 0.01. We used batches of 128 samples, that we collected from a replay buffer of size 1 million. The penalization threshold β was fine-tuned over the set $[0.5, 0.6, 0.7, 0.8, 0.9]$ and eventually set to 0.7. We notice informally that it was an important parameter. A low threshold could lead to over penalizing the agent leading the agent to terminate the episode as soon as possible, whereas a high threshold could slow down the learning.

Regarding PPO, both the policy network and the value network are composed of two hidden layers of size 64. Training was done using Adam and a learning rate of 0.01. Other PPO hyperparameters were defaults in Raffin et al. (2019), except that we add an entropy cost of 0.05.

E.3.2 Learning reversible policies

Environment. The environment consists of a 10 x 10 pixel grid. It contains an agent, represented by a single blue pixel, which can move in four directions: up, down, left, right. The pink pixel represents the goal, green pixels grass and grey pixels a stone path. Stepping on grass spoils it and the corresponding pixel turns brown, as shown in Fig. 8.5b. A level terminates after getting to the goal, or after 120 timesteps.

Upon reaching the goal, the agent receives a reward of +1, every other action being associated with 0 reward.

Architecture and hyperparameters. The reversibility network takes a pair of observations as input and produces an embedding by passing each observation through 3 identical convolutional layers of kernel size 3, with respectively 32, 64 and 64 channels. The convolutional outputs are flattened, linearly projected onto 64 dimensional vectors and concatenated. The resulting vector is projected into a scalar, which goes through a final sigmoid activation.

As done for Cartpole, we trained this network doing 1 gradient step every 500 time steps, using the Adam optimizer with a learning rate of 0.01. We used minibatches of 128 samples, that we collected from a replay buffer of size 1M. The penalization threshold β was set to 0.8, and the intrinsic reward was weighted by 0.1, such that the intrinsic reward was equal to $-0.1\,\mathbb{1}_{\psi(s_t,s_{t+1})>0.8}\,\psi(s_t,s_{t+1})$.

For PPO, both the policy network and the value network are composed of 3 convolutional layers of size 32, 64, 64. The output is flattened and passed through a hidden layer of size 512. Each layers are followed by a ReLU activation. Policy logits (size 4) and baseline function (size 1) were produced by a linear projection. Other PPO hyperparameters were defaults in Raffin et al. (2019), except that we add an entropy cost of 0.05.

E.3.3 Sokoban

We use the Sokoban implementation from Schrader (2018). The environment is a 10x10 grid with a unique layout for each level. The agent receives a -0.1 reward at each timestep, a +1 reward when placing a box on a target, a -1 reward when removing a box from a target, and a +10 reward when completing a level. Observations are of size $(10, 10, 3)$. Episodes have a maximal length of 120, and terminate upon placing the last box on the remaining target. At the beginning of each episode, a level is sampled uniformly from a set of 1000 levels, which prevents agents from memorizing puzzle solutions. The set is obtained by applying random permutations to the positions of the boxes and the position of the agent, and is pre-computed for efficiency. All levels feature four boxes and targets.

We use the distributed IMPALA implementation from the Acme framework (Hoffman et al., 2020) as our baseline agent in these experiments. The architecture and hyperparameters were obtained by optimizing for sample-efficiency on a single held-out level. Specifically, the agent network consists of three 3x3 convolutional layers with 8, 16 and 16 filters and strides 2, 1, and 1 respectively; each followed by a ReLU nonlinearity except the last one. The outputs are flattened and fed to a 2-layer feed-forward network with 64 hidden units and ReLU nonlinearities. The policy and the value network share all previous layers, and each have a separate final one-layer feed-forward network with 64 hidden units and ReLU nonlinearities as well. Regarding agent hyperparameters, we use 64 actors running in parallel, a batch size of 256, an unroll length of 20, and a maximum gradient norm of 40. The coefficient of the loss on the value is 0.5, and that of the entropic regularization 0.01. We use the Adam optimizer with a learning rate of 0.0005, a momentum decay of 0 and a variance decay of 0.99.

The precedence estimator network is quite similar: it consists of two 3x3 convolutional layers with 8 filters each and strides 2 and 1 respectively; each followed by a ReLU nonlinearity except the last one. The outputs are flattened and fed to a 3-layer feed-forward network with 64 hidden units and ReLU nonlinearities, and a final layer with a single neuron. We use dropout in the feed-forward network, with a probability of 0.1. Precedence probabilities are obtained by applying the sigmoid function to the outputs of the last feed-forward layer. The precedence estimator is trained offline on 100k trajectories collected from a random agent. It is trained on a total of 20M pairs of observations sampled with a window of size 15, although we observed identical performance with larger sizes (up to 120, which is the maximal window size). We use the Adam optimizer with a learning rate of 0.0005, a momentum decay of 0.9, a variance

decay of 0.999. We also use weight decay, with a coefficient of 0.0001. We use a threshold β of 0.9. We selected hyperparameters based on performance on validation data.

E.3.4 Reversibility-Aware Control in Cartpole+

Learning ψ. The model architecture is the same as described in Appendix E.3.1. The training is done offline using a buffer of 100k trajectories collected using a random policy. State pairs are fed to the classifier in batches of size 128, for a total of 3M pairs. We use the Adam optimizer with a learning rate of 0.01. We use a window w equal to 200, which is the maximum number of timesteps in our environment.

Learning ϕ. We use a shallow feed-forward network with a single hidden layer of size 64 followed by a ReLU activation. From the same buffer of trajectories used for ψ, we sample 100k transitions and feed them to ϕ in batches of size 128. As before, training is done using Adam and a learning rate of 0.01.

E.3.5 DQN and M-DQN in Cartpole+

We use the same architecture for DQN and M-DQN. The network is a feed-forward network composed of two hidden layers of size 512 followed by ReLU activation. In both cases, we update the online network every 4 timesteps, and the target network every 400 timesteps. We use a replay buffer of size 50k, and sample batches of size 128. We use the Adam optimizer with a learning rate of 0.001.

We train both algorithms for 2M timesteps. We run an evaluation episode every 1000 timesteps, and report the maximum performance encountered during the training process. We perform a grid search for the discount factor $\gamma \in [0.99, 0.999, 0.9997]$, and for M-DQN parameters $\alpha \in [0.7, 0.9, 0.99]$ and $\tau \in [0.008, 0.03, 0.1]$. The best performances were obtained for $\alpha = 0.9, \tau = 0.03$, and $\gamma = 0.99$.

E.3.6 Reversibility-Aware Control in Turf

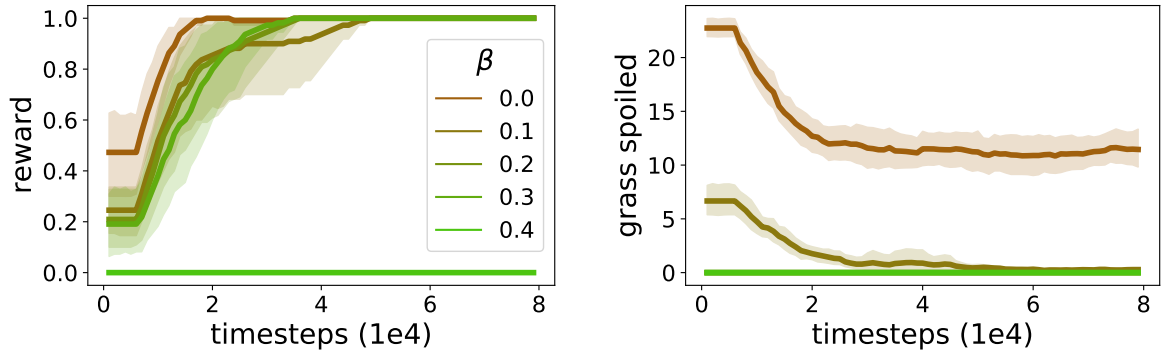
Learning ψ. We use the same model architecture as in RAE (Appendix E.3.2), and the same offline training procedure that was used for Cartpole+ (Appendix E.3.4). The window w was set to 120, which is the maximum number of steps in Turf.

Learning ϕ. The architecture is similar to ψ, except for the last linear layers: the output of the convolutional layers is flattened and fed to a feed-forward network with one hidden layer of size 64 followed by a ReLU. Again, we used the exact same training procedure as in the case of Cartpole+ (Appendix E.3.4).

E.3.7 Safety and Performance Trade-off in Turf

We investigate the performance-to-safety trade-off induced by reversibility-awareness in Turf. In Fig. E.3a, we see that the agent is not able to reach the goal when the threshold is greater than 0.4: with too high a threshold, every action leading to the goal could be rejected. We also see that it solves the task under lower threshold values, and that lowering β results in faster learning. On the other hand, Fig. E.3b shows that achieving zero irreversible side-effects during the learning is only possible when β is greater than 0.2. In this setting, the optimal thresholds are thus between 0.2 and 0.3, allowing the agent to learn the new task while eradicating every side-effect.

This experiment gives some insights on how to tune β in new environments. It should be initialized at 0.5 and decreased progressively, until the desired agent behaviour is reached. This would ensure that the chosen threshold is the maximal threshold such that the environment can be solved, while having the greatest safety guarantees.



(a) (b)

Figure E.3: **(a):** Reward learning curve for PPO+RAC and several thresholds β (average over 10 random seeds). A threshold of 0 means actions are never rejected, and corresponds to the standard PPO. **(b):** Number of irreversible side-effects (grass pixels stepped on). For β between 0.2 and 0.4, 0 side-effects are induced during the whole learning.