Learning goal-oriented agents with limited supervision

Lina Mezghani

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Mathématiques et Informatique
Unité de recherche : Laboratoire Jean Kuntzmann

Apprentissage d'agents multi-tâches sous une supervision minimale

Learning goal-oriented agents with limited supervision

Présentée par :

Lina MEZGHANI

Direction de thèse :

Karteek ALAHARI Chargé de recherche HDR, INRIA Centre Grenoble-Rhône-Alpes	Directeur de thèse
Piotr BOJANOWSKI Ingénieur docteur, Meta AI	Co-encadrant de thèse

Rapporteurs :

ABHINAV GUPTA
Full professor, Carnegie Mellon University
OLIVIER SIGAUD
Professeur des Universités, SORBONNE UNIVERSITE

Thèse soutenue publiquement le **3 juillet 2023**, devant le jury composé de :

KARTEEK ALAHARI Chargé de recherche HDR, INRIA CENTRE GRENOBLE-RHONE-ALPES	Directeur de thèse
ABHINAV GUPTA Full professor, Carnegie Mellon University	Président
OLIVIER SIGAUD Professeur des Universités, SORBONNE UNIVERSITE	Rapporteur
PIERRE-YVES OUDEYER Directeur de recherche, INRIA CENTRE BORDEAUX SUD-OUEST	Examinateur
ANNE SPALANZANI Professeur des Universités, UNIVERSITE GRENOBLE ALPES	Examinatrice

Invités :

PIOTR BOJANOWSKI
Ingénieur docteur, Meta AI
SAINBAYAR SUKHBAATAR
Ingénieur docteur, Meta AI

Abstract

The development of intelligent agents has seen significant progress in the last decade, showing impressive capabilities in various tasks, such as video games or robot navigation. These advances were made possible by the advent of deep reinforcement learning (RL), which allows to train neural network-based policies, through interaction of the agent with its environment. However, in practice, the implementation of such agents requires significant human intervention and prior knowledge on the task at hand, which can be seen as forms of *supervision*. In this thesis, we tackle three different aspects of supervision in RL, and propose methods to reduce the amount of human intervention required to train agents.

We first investigate the impact of supervision on the choice of observations seen by the agent. In robot navigation for example, the modalities of the environment observed by the agent are an important design choice that can have a significant impact on the difficulty of the task. To tackle this question, we focus on image-goal navigation in photo-realistic environments, and propose a method for learning to navigate from raw visual inputs, *i.e.*, without relying on depth or position information.

Second, we target the problem of reward supervision in RL. Standard RL algorithms rely on the availability of a well-shaped reward function to solve a specific task. However, the design of such functions is often a difficult and time-consuming process, which requires prior knowledge on the task and environment. This limits the scalability and generalization capabilities of the designed approaches. To address this issue, we tackle the problem of learning state-reaching policies without reward supervision, and design methods that leverage intrinsic reward functions to learn such policies.

Finally, we study the problem of learning agents offline, from pre-collected demonstrations, and question the availability of such data. Collecting expert trajectories is often a difficult and time-consuming process, which can be more difficult than the downstream task itself. Offline algorithms should therefore rely on existing data, and we propose a method for learning goal-conditioned agents from tutorial videos, which contains expert demonstrations aligned with natural language captions.

Contents

Chapter 1

Introduction

1.1 Context and Scope

In the last decade, the development of intelligent agents has seen impressive progress in many tasks, from beating human players in the difficult game of Go [Silver et al., 2016], to achieving complex locomotion behavior in robotics [Chaplot et al., 2020c], to learning to play open-ended video games like MineCraft [Baker et al., 2022]. This progress, in environments and tasks that seem dramatically dissimilar at first sight, is due to the advent of a common framework that formalizes the problem of learning agents. This formalism starts with the definition of an agent, described in the acclaimed book "Artificial Intelligence: A Modern Approach" of Russell and Norvig [1995], as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuator". In the game of Go, the agent sees the current state of the board, and acts by executing a valid move in the game. In the case of robotics, the sensors are cameras or various types of radars, and the robot acts by moving its joints with specific motors.

Given this definition, the interaction between the agent and the environment is formalized in the mathematical and computational framework of reinforcement learning (RL) [Kaelbling et al., 1996]. In this paradigm, the agent can interact with the environment over consecutive episodes, in which the time scale is divided into discrete steps. During each episode, the agent sequentially interacts with the environment by performing an action in the current state, transitioning to the next state, and receiving a reward. This reward, provided by the environment, is a scalar value that indicates to the agent how good the action was (from a certain state) for accomplishing a specific task. Standard RL algorithms optimize the behavior of the agent by maximizing the expected cumulative reward over time. The reward definition critically influences the learned behavior of the agent. For general-purpose navigating robots, computing rewards as a function of the distance to the goal will cause a *goal-reaching* behavior, while rewarding the agent for discovering novel parts of the environment will induce an *exploration* behavior.

However, designing a reward function and training an agent for every single task that we want to solve hinders the scalability of standard RL methods. In many applications, it can indeed be desirable to develop multi-purpose agents, that are prompted to perform various tasks, and are able to execute multiple behaviors. In that respect, *goal-oriented* (or *goal-conditioned*) RL [Kaelbling, 1993] focuses on learning agents that act towards a given goal. At the beginning of each episode, the agent is given a goal information, and needs to perform actions in order to achieve it. The reward function here is defined as a function of the distance to the goal, or as an indicator upon reach. This goal can be given in different ways, such as a textual description [Chevalier-Boisvert et al., 2019], a visual representation [Beattie et al., 2016], or a certain score to reach [Tassa et al., 2018]. In the case of navigating robots, the goal can thus be given as a specific location in the environment [Wijmans et al., 2019], or as a snapshot taken from the target location [Zhu et al., 2017], or even as an instruction to reach a certain object [Chaplot et al., 2020a]. One advantage of goal-oriented RL is that it allows for learning agents that can be prompted to perform different tasks, by simply changing the goal given as input. It requires, however, knowledge about the environment and the task at hand in order to first determine the set of goals on which to train the agent, and then define a metric to measure the distance to the goal, or at least, a goal-reaching function.

More generally, crafting a reward function appropriately is a key step in the development of an agent, and requires a deep understanding of the target task, and the structure of the environment. This is particularly true since, in practice, RL algorithms struggle to learn in cases where the reward function is *sparse*, that is, when nonzero rewards occur rarely [Vecerik et al., 2017, Andrychowicz et al., 2017]. Following this, since reward engineering requires knowledge about the task and human intervention, it can be seen as a form of *supervision*: in the same way that images require class labels in a supervised classification task, state-action transitions require reward labeling in RL. Several works, grouped under the term of *unsupervised* RL, propose to overcome this problem by developing agents that can learn without any reward signal from the environment [Bellemare et al., 2016, Pathak et al., 2017, Andrychowicz et al., 2017, Eysenbach et al., 2018]. They often rely on the idea that the agent should learn to explore the environment in a task-agnostic manner, in order to discover general and re-usable skills.

To sum up, designing a truly intelligent system requires to learn and reason about the world without the intervention of explicit human supervision. Moreover, relying on task-specific characteristics and prior environment knowledge hinders the scalability and generality of the learned agents. Our main focus in this thesis, is therefore to understand the interaction between classical RL algorithms and these forms of supervision, as well as to design methods that do not rely on prior knowledge, a question that we explore from the following three perspectives. We first investigate the technical assumptions behind the design of specific tasks and environments in

RL, in order to question the constraints they involve. We then study the dependence of standard RL algorithms on the different modalities of supervision. Finally, we propose and empirically validate methods for learning RL agents in the absence of diverse forms of supervision.

1.2 Goals and Challenges

The goal of this thesis is to study the impact of supervision on the learning of RL agents. We argue that there exist several forms of supervision, which intervene at different levels of the learning pipeline, and that require various types of human intervention and prior knowledge of the task. We focused in particular, on multi-purpose goal-oriented agents, and on three forms of supervision, that we will detail in this section.

Learning from less modalities. A crucial point when learning agents in the RL framework, is the type of inputs that the agent receives, *i.e.*, the nature of its *observations*. In the case of navigating robots, the most realistic choice for the observations is to use the sensors that are available on the robots, such as cameras, radars, accelerometers, or depth sensors [Zhu et al., 2017, Chaplot et al., 2018, 2020c]. However, in practice, for simulated navigation tasks, the observations are often augmented with additional modalities, such as the absolute position of the agent in the environment [Wijmans et al., 2019, Zhao et al., 2021]. This assumption is unrealistic as the agent does not have access to the entire map of the environment, and constitutes an additional form of hand-crafted supervision. Moreover, we, as humans, are able to navigate using only our visual sensors, and do not require to know our absolute coordinates in the world. Designing truly intelligent navigating robots therefore requires to learn from raw observations alone. More generally, for any task, the nature of modalities on which to rely on is an important design choice when learning autonomous agents.

RL without external rewards. In its original formulation, reinforcement learning is defined as a sequential decision-making problem, where the agent consecutively interacts with the environment by performing actions in the current state, transitioning to the next state, and receiving a reward. The reward function is therefore a key component of the RL framework. Although it is assumed to be given by the environment, it must actually be carefully engineered to capture the desired behavior of the agent for every specific task [Tassa et al., 2018, Laskin et al., 2021]. In many cases however, defining the reward function is impractical, time-consuming and tedious. In goal-oriented RL, a natural choice for the reward is to define it as a function of the distance to the goal, or as an indicator upon reach [Liu et al., 2022]. However, in both cases, there is no easy way of computing it without prior knowledge about the environment and the task at hand. Indeed, computing the distance to the goal

requires to have a notion of distance between states in the environment, which is not always easily computable or defined. In robot navigation for instance, the distance between two locations in the environment might be tricky to implement as the shortest path between them might not be known [Chaplot et al., 2019]. In the second case, where the reward function is defined as an indicator upon reaching the goal, one must assume access to such a function. If this can be assumed in some simple, discrete cases, it is not easily computable in more complex environments, for instance with visual observations. In visual navigation for example, when the goal is given to the agent as an image, naively computing the distance to the goal by calculating the Euclidean distance between images would not be meaningful [Nair et al., 2018, Pong et al., 2020]. Moreover in that case, the rewards would occur only at the end of the episode, which will make the reward distribution very sparse, and thus increasing the difficulty of the problem.

Leveraging fixed datasets. In standard RL, the agent improves its policy while simultaneously interacting with the environment. The training process is therefore *online*, as the agent uses its latest policy to collect training data, and to improve on it. For some applications, this process can be impractical, either because the interaction is expensive, for example in robotics, or for safety reasons, for example in autonomous driving, where deploying a poorly trained agent can be dangerous [Kiran et al., 2021]. In a recent line of study, called *offline* RL [Levine et al., 2020], the agent uses a fixed dataset of trajectories, collected beforehand, to learn its policy without additional online interaction. This field is particularly interesting in tasks where data can be easily collected, such as video games, or where interaction data is already available. A major challenge of learning from offline datasets, is the fact that the pre-collected trajectories must be representative of the environment and the task at hand, that is, the state space must be sufficiently explored, and the behavior of the agent must be diverse enough [Yarats et al., 2022]. The data collection process can therefore be human-guided, by deploying specific exploration strategies, or by directly using human-generated demonstrations. In both these cases, it constitutes a form of supervision, and properly investigating the impact of this supervision on the learning process constitutes an important question.

1.3 Outline and Contributions

We present a detailed background on reinforcement learning, and an overview of the tasks and environments tackled in the manuscript in Chapter 2. We then discuss the different contributions made as part of this PhD to the field of self-supervised learning for learning goal-conditioned agents. A schematic visualization of the structure of the manuscript is shown in Fig. 1.1.

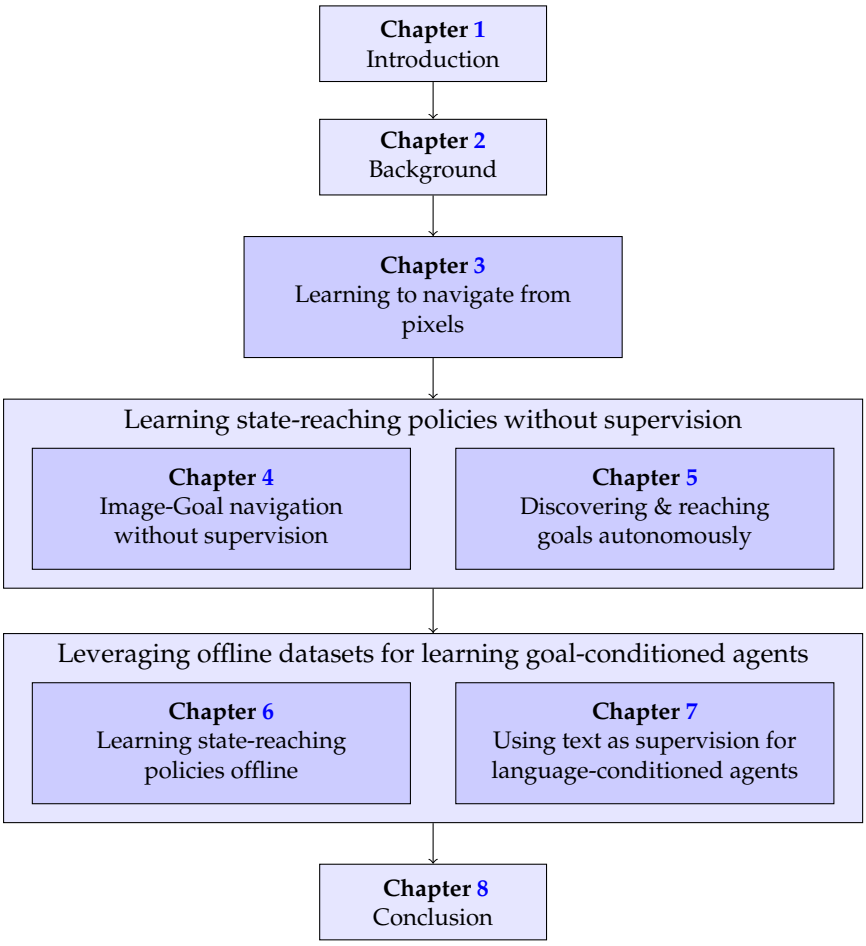


FIGURE 1.1: Structure of the manuscript. The chapters are organized in three main parts, corresponding to the three main contributions of the thesis. Chapter 3 tackles the problem of learning to navigate in photo-realistic environments from visual information alone, without using position information. Then, Chapters 4 and 5 investigate ways of learning state-reaching policies without supervision of any form. Finally, Chapters 6 and 7 propose methods for leveraging datasets of pre-collected trajectories for learning goal-conditioned agents.

1.3.1 Navigation and planning from pixels

Goal-oriented navigation in photo-realistic environment is one of the most widespread task in the field of robotics [Zhu et al., 2017], as it has many applications, and is often a starting point for more complex tasks, such as visual-question answering [Das et al., 2018], or instruction following [Hill et al., 2020]. In the task of image-goal navigation, the agent is given a goal image, and must navigate to the location in the environment where the image was taken from [Savinov et al., 2018a]. It requires the agent to have a good understanding of the environment, as well as decent planning capabilities to move intelligently towards the goal. Most prior works, including RL-based approaches, rely heavily on the availability of the absolute position of the agent in the environment, an assumption which is often unrealistic. Indeed, if simulators allow for accessing such information, it is not the case for real

robots, that are instead only equipped with RGB cameras or depth sensors. In Chapter 3, we investigate the problem of learning to navigate in real-world environments from RGB input only, without access to position information. The main challenge of this task is that, in the absence of such information, the agent must learn an informative representation of the environment from the RGB inputs. Our goal in Chapter 3 is therefore to present a novel method for learning to navigate in real-world environments from RGB input, that uses self-supervised representation learning to plan from images.

Outline. Chapter 3 presents the problem of image-goal navigation with generalization to unseen environments, and introduces a novel memory-augmented approach for learning to visually navigate in real-world environments. The proposed method is based on an attention-based end-to-end model that leverages an episodic memory to learn to navigate. We then validate our approach with extensive evaluations, and show that our model establishes a new state of the art on unseen environments from the challenging Gibson dataset [Xia et al., 2018].

Publication. Chapter 3 is largely based on the publication *"Memory-augmented Reinforcement Learning for Image-Goal Navigation"*, Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, Karteek Alahari, IROS 2022 (see [Mezghani et al., 2022c]). Along with the publication, we released the image-goal navigation dataset, available at `https://github.com/facebookresearch/image-goal-nav-dataset`, that contains training and validation episodes for the task of image-goal navigation.

1.3.2 Learning state-reaching policies without supervision

The problem of image-goal navigation, tackled in Chapter 3, is a particular case of the more general problem of learning to reach a given state in an environment, which amounts to learning state-reaching policies. In its standard formulation, this problem assumes access to a well-shaped reward function that informs the agent about its distance to the goal, and makes it easy for the policy to learn the desired goal-oriented behavior [Wijmans et al., 2019]. For navigation, it requires the ability to compute the length of the shortest path between locations in the environment, which can be tricky in the presence of obstacles or bottlenecks. For other types of tasks, like locomotion or manipulation, defining such a function might not even be feasible [Pong et al., 2020]. The first challenge that arises when learning state-reaching policies without supervision is therefore the need to learn a distance function that can be used to shape a reward signal for the policy. Another important challenge comes from the fact that the set of goals on which the agent is trained might not be known in advance. For a goal-oriented agent to be truly intelligent and autonomous, it should understand what goals are in its scope of capabilities, and which ones would be interesting to pursue. In Chapters 4 and 5, we therefore

investigate the problem of learning state-reaching policies without supervision, and propose methods for learning to reach states without access to well-shaped reward or distance functions.

Outline. Chapter 4 tackles the problem of learning to visually navigate in photo-realistic environments without any supervision, that is, from RGB input only and without access to neither a reward function, nor a notion of distance in the environment. We present a three-stage approach for this problem that consists in first, learning a representation of the environment in a self-supervised fashion, then exploring the environment to build a graph over the states and then learning to navigate by leveraging this graph. In Chapter 5, we extend this method to diverse tasks, such as locomotion and manipulation, and show that our approach can be used to learn to reach states in a variety of environments without supervision. More precisely, we propose to jointly learn the representation of the environment and the state-reaching policy by alternating between goal-oriented trajectories and random exploration. This process allows for fostering exploration at all stages of the training of the goal-conditioned policy, and to further improve the representation of the environment.

Publication. Chapter 4 is based on our technical report "*Learning to Visually Navigate in Photorealistic Environments Without any Supervision*", Lina Mezghani, Sainbayar Sukhbaatar, Arthur Szlam, Armand Joulin and Piotr Bojanowski (see [Mezghani et al., 2020]), and Chapter 5 is based on the paper "*Walk the Random Walk: Learning to Discover and Reach Goals Without Supervision*", Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski and Karteek Alahari, published at the ICLR 2022 workshop on Agent Learning in Open-Endedness (see [Mezghani et al., 2022a]).

1.3.3 Leveraging offline datasets for learning goal-conditioned agents

An important aspect of learning agents with reinforcement learning is the ability to interact with the environment at train time. In some cases, interacting with the environment is not possible, or non desirable, for instance when the environment is not accessible, or when it is unsafe. In such cases, the agent can be trained offline with standard RL methods, using a dataset of already collected data, as shown by Yarats et al. [2022]. However, as in traditional RL problems, the dataset needs to be labeled with rewards for solving a specific task, a process that requires manual engineering and prior knowledge. Chebotar et al. [2021] proposed an offline method for learning a policy that can reach any state in the environment, without any supervision by using hindsight relabeling [Andrychowicz et al., 2017]. However, these methods suffer from the issue of sparsity of the rewards, and therefore fail at long-horizon tasks. In Chapters 6 and 7, we present two methods for tackling this problem by providing dense self-supervision to the agent.

Outline. In Chapter 6, we show how pre-collected trajectories can be used to learn policies that can reach any state in the environment, offline and without supervision. For that, we perform a self-supervised representation learning stage on the environment, and then use the learned representation to shape a dense reward signal for the goal-conditioned policy. In an orthogonal direction, we propose in Chapter 7 to provide self-supervision to the agent by leveraging text-aligned video data, that is, datasets of trajectories augmented with text captions. We show that leveraging this type of offline data allows for learning language-conditioned agents, that can, given a natural language instruction, execute a task in the environment.

Publication. Chapter 6 is largely based on the publication "*Learning Goal-Conditioned Policies Offline with Self-Supervised Reward Shaping*", Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski, Karteek Alahari, published at CoRL 2022 (see [Mezghani et al., 2022b]). Along with the paper, we released the code for the method at `https://github.com/facebookresearch/go-fresh`. Chapter 7, for its part, is based on the paper "*Think Before You Act: Unified Policy for Interleaving Language Reasoning with Actions*", Lina Mezghani, Piotr Bojanowski, Karteek Alahari, and Sainbayar Sukhbaatar, published at the Reincarnating Reinforcement Learning Workshop at ICLR 2023 (see [Mezghani et al., 2023]).

1.4 List of publications and softwares

This manuscript is based on material published in the following papers:

- *Learning to Visually Navigate in Photorealistic Environments Without any Supervision*, Lina Mezghani, Sainbayar Sukhbaatar, Arthur Szlam, Armand Joulin, and Piotr Bojanowski, arXiv preprint arXiv:2004.04954, 2020.

- *Memory-Augmented Reinforcement Learning for Image-Goal Navigation*, Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksander Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari, IROS 2022.

- *Walk the Random Walk: Learning to Discover and Reach Goals Without Supervision*, Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski, and Karteek Alahari, ICLR 2022 Workshop on Agent Learning in Open-Endedness.

- *Learning Goal-Conditioned Policies Offline with Self-Supervised Reward Shaping*, Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski, Alessandro Lazaric, and Karteek Alahari, CoRL 2022.

- *Think Before You Act: Unified Policy for Interleaving Language Reasoning with Actions*, Lina Mezghani, Piotr Bojanowski, Karteek Alahari, and Sainbayar Sukhbaatar, ICLR 2023 Workshop on Reincarnating Reinforcement Learning.

The work conducted in this thesis has led to the following softwares and datasets:

- The image-goal navigation dataset, available at:
  `https://github.com/facebookresearch/image-goal-nav-dataset`

- Go-Fresh, the source code for the implementation of our CoRL 2022 paper [Mezghani et al., 2022b], available at:
  `https://github.com/facebookresearch/go-fresh`

Chapter 2

Background

Recent advances in the problem of learning agents have been driven by the development of a common formalism that comes with its own set of tools and algorithms, namely reinforcement learning (RL). Then, a wide range of research directions that derive from RL have emerged, including unsupervised RL, that aims to learn exploration behavior without any external reward signal, and goal-oriented RL, which purpose is to learn agents that can achieve a variety of goals. In this chapter, we first provide an overview of the RL formalism, and then discuss the specificities of unsupervised and goal-oriented RL. We also present the tasks, environments, and the underlying challenges that are tackled in this manuscript.

2.1 Reinforcement Learning

Reinforcement learning provides a mathematical formalism for learning controllable agents. In its standard formulation, RL addresses the problem of learning a dynamical system, fully-defined by a Markov decision process (MDP). The MDP is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ is the set of states $s \in \mathcal{S}$, $\mathcal{A}$ is the set of actions $a \in \mathcal{A}$, P defines a conditional probability distribution of the form $P(s_{t+1}|s_t, a_t)$ that describes the dynamics of the system, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines a reward function, and $\gamma \in (0, 1]$ is a discount factor.

In this work, we focus on the partially observed setup, where the agent does not have access to the full state of the environment, and instead receives an observation. Formally, the partially observed Markov decision process (POMDP) is therefore defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{O}$ is the set of observations $o \in \mathcal{O}$. The ultimate goal of reinforcement learning is to learn a policy π, that defines a conditional distribution over actions, which maximizes the discounted cumulative reward: $R = \sum_{t=0}^{\infty} \gamma^t r_t$. In the fully-observed formulation, the policy is conditioned on the state s_t, while in the partially-observed formulation, it can either be conditioned on the observation o_t, or on the history of previous observations $o_{0:t}$ [Parr and Russell, 1995].

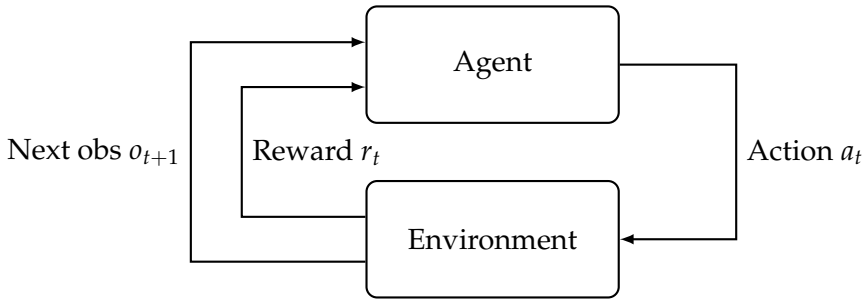


FIGURE 2.1: Diagram of the interaction between the agent and the environment. At timestep t, the agent receives an observation o_t from the environment, selects an action a_t according to its policy, and receives a reward r_t and the next observation o_{t+1}. Inspired from Sutton [1988].

2.1.1 Online, Off-Policy, and Offline RL

Most RL algorithms are based on the following training loop, depicted in Fig. 2.1: the agent interacts with the MDP by using its policy, observes the current observation o_t, selects an action a_t according to its policy, and receives a reward r_t, as well as the next observation o_{t+1}. The policy is commonly represented as a neural network [Arulkumaran et al., 2017] with parameters θ, which is updated over the environment interactions. There exist several ways of updating the policy given the agent's interactions in the environment, which make up the family of RL algorithms. These can be divided into three main categories, as highlighted in Fig. 2.2.

Online RL. In online reinforcement learning, the policy is updated with streaming data collected by itself, and the collected data is not reused. Common online reinforcement learning algorithms include the vanilla policy-gradient method [Williams, 1992] (also called REINFORCE), actor-critic methods [Mnih et al., 2016], and Proximal Policy Optimization (PPO) [Schulman et al., 2017]. Learning the policy with data collected using the policy itself makes the training relatively stable, but also highly data inefficient, as it requires a large number of interactions with the environment to learn the policy.

Off-Policy RL. In contrast to online RL, off-policy RL algorithms are more efficient as they store a buffer of past experience and can learn from it. Indeed, in the classic off-policy setting (Fig. 2.2 (b)), the agent's experience is appended to a data buffer $\mathcal{D}$ (also called a *replay buffer*), and each new policy π_k collects additional data, such that $\mathcal{D}$ is composed of samples from $\pi_0, \pi_1, ..., \pi_k$, and all of this data is used to train an updated new policy π_{k+1}. These methods therefore have better data efficiency properties than online RL algorithms. The most common off-policy RL algorithms are temporal difference (TD) methods [Sutton, 1988], Deep Q-Networks (DQN) [Mnih et al., 2013], Deep Deterministic Policy Gradients (DDPG) [Lillicrap et al., 2015], and Soft Actor-Critic (SAC) [Haarnoja et al., 2018].

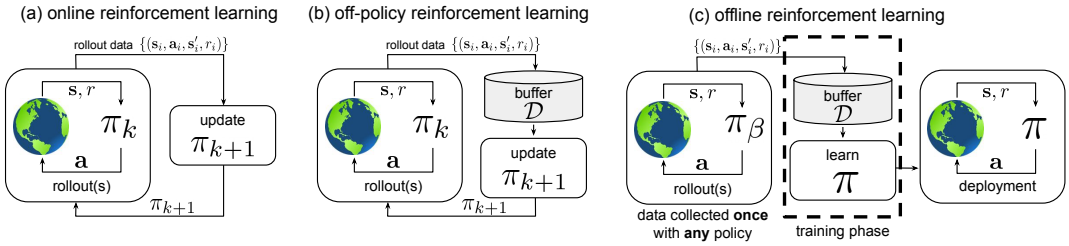


FIGURE 2.2: Pictorial illustration of online reinforcement learning (a), off-policy reinforcement learning (b), and offline reinforcement learning (c). In online reinforcement learning (a), the policy π_k is updated with streaming data collected by π_k itself. In the classic off-policy setting (b), the agent's experience is appended to a data buffer $\mathcal{D}$ (also called a *replay buffer*), and each new policy π_k collects additional data, such that $\mathcal{D}$ is composed of samples from $\pi_0, \pi_1, ..., \pi_k$, and all of this data is used to train an updated new policy π_{k+1}. In contrast, offline reinforcement learning employs a dataset $\mathcal{D}$ collected by some (potentially unknown) behavior policy π_β. The dataset is collected once, and is not altered during training, which makes it feasible to use large previous collected datasets. The training process does not interact with the MDP at all, and the policy is only deployed after being fully trained. Source: Levine et al. [2020].

Offline RL. Finally, a limit case of off-policy RL is offline RL, where the agent does not interact with the environment at all, and learns from a fixed dataset. More precisely, offline reinforcement learning employs a dataset $\mathcal{D}$ fixed, and collected by some – potentially unknown – behavior policy. The dataset is collected once, and is not altered during training, which makes it feasible to use large previously-collected datasets. Offline RL algorithms include Behavioral Cloning [Torabi et al., 2018], Imitation Learning [Ho and Ermon, 2016], and more recently, Decision Transformer [Chen et al., 2021].

In this manuscript, we will present methods that use algorithms for all of these three categories: DD-PPO [Wijmans et al., 2019], a Decentralized and Distributed variant of Proximal Policy Optimization (PPO) [Schulman et al., 2017], Soft-Actor Critic (SAC) [Haarnoja et al., 2018], and Decision Transformer (DT) [Chen et al., 2021].

2.1.2 Unsupervised RL

Standard reinforcement learning methods, as mentioned so far, heavily rely on the availability of a reward function, that associates the behavior of the agent to a numerical value. In several tasks, like continuous control, crafting such a reward function is not trivial, and can be very tedious and time-consuming. In other cases, the reward function might be too sparse in the state space, which makes it difficult for the agent to learn a good policy. This can be the case for long-horizon tasks, when the reward is computed as a positive value upon achieving the goal. To tackle these problems, a new paradigm has emerged to learn policies without the need for an external reward function, called unsupervised reinforcement learning, as it does not rely on reward supervision. For classic RL algorithms to work in this context, the

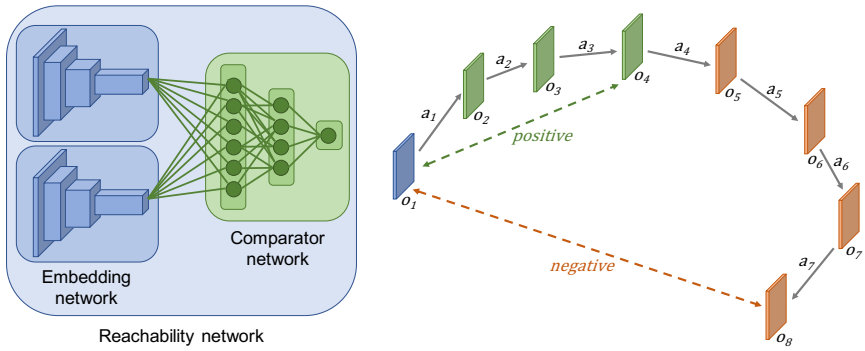


FIGURE 2.3: **Left:** siamese architecture of the reachability network. **Right:** the reachability network is trained based on a sequence of observations that the agent encounters while acting. The temporally close (within threshold) pairs of observations are positive examples, while temporally far ones — negatives. Source: Savinov et al. [2018b].

agent must craft a reward function for itself, an *intrinsic* reward, that is used to train the policy. Such approaches are task-agnostic, and consist in encouraging the agent to explore the environment in different ways.

The first methods for implementing intrinsic rewards were based on the idea of state-visitation counts [Bellemare et al., 2016, Ostrovski et al., 2017], which consists in counting the number of times the agent has visited a given state, and encouraging the less-visited ones. However, this approach is limited to finite state spaces, and does not scale well to large state spaces. To cope with this problem, some approaches rely on the idea of state-visitation entropy maximization [Hazan et al., 2019, Mutti et al., 2021, Liu and Abbeel, 2021], which encourages the agent to visit states that are less likely to be visited. Other methods rely on the idea of curiosity [Pathak et al., 2017], which consists in fostering the agent to explore state-action pairs for which it does not have a good prediction of the outcome. Additionally, a line of research [Eysenbach et al., 2018, Lee et al., 2019, Liu and Abbeel, 2021] focuses on learning competence-based agents, that learn a diverse set of skills. Finally, other approaches include, among others, distilling random networks [Burda et al., 2018] and ensemble disagreement [Sekar et al., 2020].

In a major part of this manuscript, we drew inspiration from a particular method of unsupervised RL, presented in the paper "Episodic Curiosity Through Reachability" [Savinov et al., 2018b], in which the authors propose an exploration method based on a network trained with a self-supervised learning objective, namely, the Reachability Network. This method proposes to reward the agent for reaching states that are novel with respect to states it has previously visited, and measures novelty with the help of the reachability network. The general idea of the reachability network is to approximate the spatial distance between locations as the number of time steps taken by an agent with a random policy to reach one location starting from the other. Indeed, the expected distance covered by a random walk is the square root of the number of time steps. Savinov et al. [2018b] use the temporal distance between

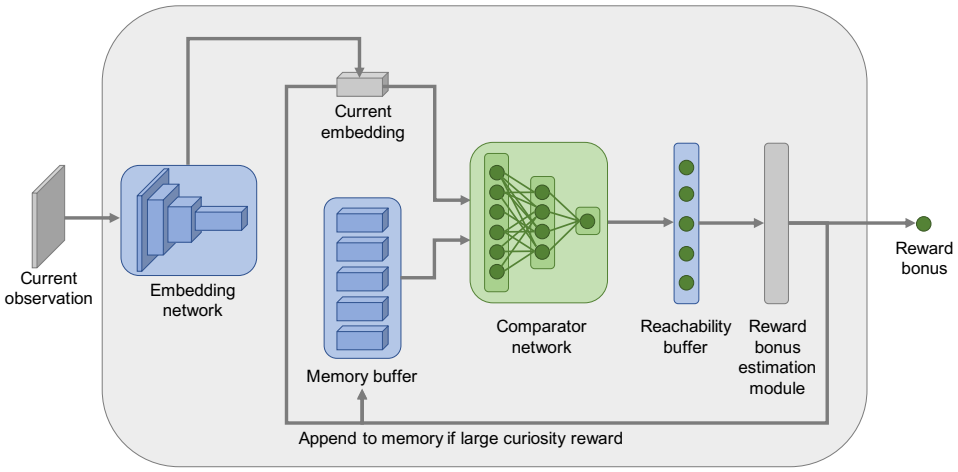


FIGURE 2.4: The use of episodic curiosity (EC) module for reward bonus computation. The module takes a current observation as input and computes a reward bonus which is higher for novel observations. Source: Savinov et al. [2018b].

observations as a surrogate similarity measure, and craft an intrinsic reward signal based on this measure.

More formally, the reachability network, depicted in Fig. 2.3 (Left) is represented by a siamese neural network $R : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}$ where $\mathcal{O}$ is the observation space. It takes as input two observations, and outputs a scalar value which represents the *reachability score* between these two observations. This network is trained by forming pairs of positive and negative samples from random trajectories, as depicted in Fig. 2.3 (Right). More precisely, we let a random agent interact with the environment for T time steps, and denote by $(x_1, ..., x_T)$ the sequence of observations. We then define a reachability label $y_{i,j}$ for each pair of observations (x_i, x_j) based on their distance in the sequence, *i.e.*, the label $y_{i,j}$ is equal to 1 if $|i - j| \leq \tau$, and 0 otherwise, where τ, the reachability threshold, is a hyperparameter. The network is then trained to predict the reachability label $y_{i,j}$ from the input pair (x_i, x_j) in a supervised fashion, with a logistic regression loss.

Once the reachability network is trained, the agent uses it to compute an intrinsic reward signal for each observation, as shown in Fig. 2.4. To do so, the agent stores a set of previously encountered observations in a memory buffer. For efficiency reasons, the memory buffer is limited in size, and the agent filters out observations by only keeping the ones that are novel enough (as measured by the reachability network). At every timestep, the agent compares the current observation with those already in the memory buffer. It then aggregates this result with a given function (such as argmax) to obtain a single reachability score, which is used to compute the intrinsic reward signal. In this manuscript, we used several components of this exploration strategy in our experiments.

2.1.3 Goal-conditioned RL

A specific branch of reinforcement learning studies the problem of learning a policy that is able to achieve a given goal [Kaelbling, 1993]. The formalism of goal-conditioned RL is slightly different from the one of standard RL. The agent is given a goal g in a goal space $\mathcal{G}$ at the beginning of the episode, and must learn to achieve it. Both the policy and the reward function are therefore conditioned on the goal g. The main challenge of goal-conditioned RL is to learn policies that are able to generalize to goals unseen during training. A subclass of goal-conditioned policies, called *state-reaching policies*, focuses on the specific case where the goals are represented by observations (or states in the fully-observed case), *i.e.*, $\mathcal{G} \subseteq \mathcal{O}$ (resp. $\mathcal{G} \subseteq \mathcal{S}$). In this case, the goal-conditioned policy is conditioned on an observation, and the goal of the agent is to reach this observation.

Learning state-reaching policies. Several works studies the problem of learning state-reaching policies in the supervised context [Kaelbling, 1993, Schaul et al., 2015, Nasiriany et al., 2019], that is, assuming that the goal space is known at train time, and that a reward function is available to guide the agent towards the goal. However, these assumptions might not hold for every task: as pointed out in Section 2.1.2, crafting such a reward function can be difficult, and the targeted goal space might be unavailable at train time. Some approaches propose solutions for generating goals automatically when training goal-conditioned policies, including self-play [Sukhbaatar et al., 2018b,a, OpenAI et al., 2021], where an agent learns to reach goals with an adversarial objective and a second agent proposes goals of increasing difficulty. In the same spirit, Campero et al. [2020] presents a student-teacher policy trained in a single module with an adversarial learning loss. These methods assume access to a hand-crafted goal achievement function and therefore require supervision.

Unsupervised goal-conditioned RL. Another line of research [Warde-Farley et al., 2018, Nair et al., 2018, Péré et al., 2018, Ecoffet et al., 2019, Colas et al., 2022, Mendonca et al., 2021] focuses on learning goal-conditioned policies in a fully unsupervised setting, where the goal space is unknown at train time, and no reward function is available. There are two challenges that these methods have to overcome: first, how to generate goals at train time, and second how to assess whether a goal was reached or not. To tackle these challenges, Warde-Farley et al. [2018] propose to learn a goal achievement reward function jointly with the goal-conditioned policy using a mutual information objective. In an orthogonal research direction, Nair et al. [2018] train a variational auto-encoder and generate goals in its latent space, while using the Euclidean distance in this space to compute a dense reward function. Pong et al. [2020] improves this approach by "skewing" the set of goals to encourage exploration. More recently, Mendonca et al. [2021] train a model-based agent that learns to discover novel goals with an explorer model, and reach them

FIGURE 2.5: The U-shaped **UMaze** environment.

with an achiever policy via imagined rollouts. Finally, Venkattaramanujam et al. [2019] and Hartikainen et al. [2019] learn a distance function in the state space jointly with the goal-conditioned policy. This distance function is not only used to compute rewards for the policy, but also to sample goals in clever ways: by encouraging far-away goals [Hartikainen et al., 2019] or by sampling goals of intermediate difficulty [Venkattaramanujam et al., 2019, Colas et al., 2019].

RL with language instructions. Goal-oriented RL is not limited to the case of state-reaching tasks, and also comprises other forms of goal specifications. In fact, defining the goal as a state can be restrictive in some cases, since the goal configuration may require more than one observation to be specified. To illustrate this point, consider a humanoid agent, whose goal is to perform a back-flip. The final observation of the agent – corresponding to the agent standing up – is not enough to specify the desired behavior. In that respect, language instructions are an interesting form of goal specification [Luketina et al., 2019, Akakzia et al., 2021], as they are more descriptive than state-reaching goals, and can be used to specify more complex and abstract behaviors. Several works [Hermann et al., 2017, Misra et al., 2017, Chevalier-Boisvert et al., 2019, Colas et al., 2020] study this problem for diverse tasks in a variety of environments. Moreover, working with textual information allows for exploiting recent advances in the field of natural language processing, especially regarding large language models (LLM) [Radford et al., 2018]. Indeed, a very recent line of study proposes to use large language models on offline datasets for decision-making tasks: Zhang et al. [2022] show that task instructions can be effectively used to pre-train offline policies and Carta et al. [2023] train an adaptive LLM-based policy in language-grounded tasks. In an orthogonal direction, Li et al. [2022b] show the effectiveness of LLMs on decision-making tasks, specifically on the BabyAI [Chevalier-Boisvert et al., 2019] environment. This research direction suggests that there exists an interesting synergy between reinforcement learning and natural language processing, that can be intuitively explained by their sequential nature.

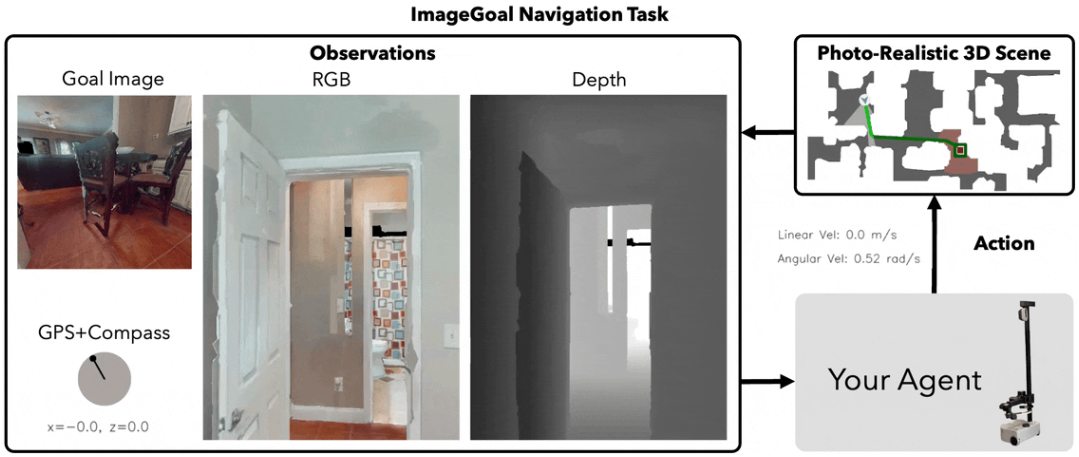


FIGURE 2.6: Visualization of the ImageGoal Navigation task in Habitat [Savva et al., 2019]. In the original formulation, the agent gets as input an RGB image of the goal, as well as its current observation, composed of its xy coordinates in the environment, as well as the RGB and depth sensors. The action space is discrete, and composed of 3 actions: move forward, turn left and turn right.

2.2 Tasks and Environments

The experimental setup studied in this thesis can be divided into two categories: maze-based environments on which we perform navigation-like tasks, and continuous control environments for manipulation tasks.

2.2.1 Navigation in Maze-based Environments

We studied three distinct navigation tasks: **UMaze** [Kanagawa, 2021], a continuous agent in a U-shaped maze, navigation in **Habitat** [Savva et al., 2019], a simulator in photo-realistic houses, and **BabyAI** [Chevalier-Boisvert et al., 2019], a multi-modal agent for learning language-conditioned agents.

UMaze. Introduced by Kanagawa [2021], **UMaze** is a two-dimensional U-shaped maze with four rooms, as shown in Fig. 2.5. The point agent starts always at the same position, in the top left corner, and can move in the maze by performing actions in a continuous space. Here, the observations are state-based: they contain the agent's position, direction and velocity. The goals are defined as states, as in the state-reaching formulation defined in Sec. 2.1.3. We generate an evaluation set of 500 goals sampled at random in the environment, and we assess the performance of the agent by measuring the distance between its final position and the goal position. The main challenge of this task is the fact that the exploration of the state space is made difficult by the fixed initial position, and the presence of the three doors, that represent bottlenecks.

Navigation in Habitat. Habitat [Savva et al., 2019] is a simulation platform for photo-realistic environments, with a large variety of embodied tasks, including

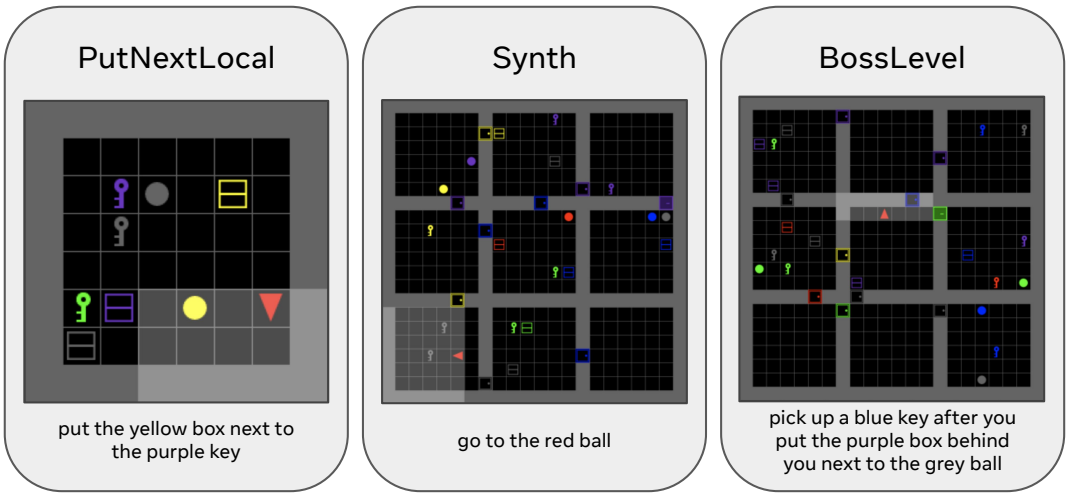


FIGURE 2.7: Three example levels for the BabyAI [Chevalier-Boisvert et al., 2019] environment.

navigation, instruction following, and visual question answering. It includes a high-performance physics-enabled 3D simulator with support for three dimensional scans of indoor and outdoor spaces. In this manuscript, we focused on scans from Gibson [Xia et al., 2018], a dataset of 3D scans of indoor houses made for learning a real-world perception for active agents. Navigation tasks can be broken down in three categories, depending on how the goal is specified: (i) **PointGoal** navigation, where the goal is specified by an xy-coordinate in the environment, (ii) **ImageGoal** navigation, where the goal is specified by an image taken from the target location, and (iii) **ObjectGoal** navigation, where the goal is specified by an object (with text) and the agent must find an instance of it. We focused on the task of ImageGoal navigation, depicted in Fig. 2.6. In our work, we make an important assumption that the agent does not have access to its xy coordinates in the environment, neither to the depth sensor. In other words, the agent gets as input its current view of the environment, through an RGB observation, as well as the image of the goal it has to navigate to. This extra assumption aims at making the task more challenging and more realistic, since knowing the exact position of the agent in the environment is unfeasible in many scenarios, and the depth sensor is not always available on embodied robots.

In the standard setting of image-goal navigation, the initial position of the agent is randomized, making random exploration of the environment possible. Moreover, the action space is discrete, and composed of three actions: move forward (of 0.25 meters), turn left and turn right (10 degrees). We studied two distinct setups: in the first one, the agent is trained and evaluated in the same house and in the second one the agent is trained on a set of train scenes, and evaluated on a test set of unseen scenes. For the latter setup, a standard split of the Gibson dataset [Xia et al., 2018] consists of 72 train scenes and 14 test scenes, as used in prior works [Chaplot et al., 2020c, 2018].

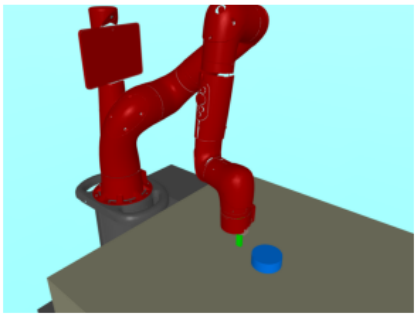


FIGURE 2.8: The Pusher environment.

BabyAI. As a third navigation task, we studied the BabyAI environment [Chevalier-Boisvert et al., 2019], a multi-modal environment for learning language-conditioned agents. BabyAI is based on the **MiniGrid** environment [Chevalier-Boisvert et al., 2018], a grid-based environment with a large variety of tasks, that involve navigating in several rooms, and interacting with objects, like doors, keys and boxes. The environment is partially observable, and the agent can see a 7x7 grid of the environment around it. The action space is discrete, and contains a few actions, such as moving forward, turning left or right, picking up an object and opening a door. One particularity of BabyAI is that goals are specified to the agent as natural language instructions, as shown in Fig. 2.7. It contains levels of various difficulty, ranging from PutNextLocal, where the agent has to put an object next to another one, to the BossLevel, the hardest level, where the agent has to execute a sequence of instructions, requiring planning and long-term reasoning.

2.2.2 Continuous control tasks

In addition to navigation environments, we also studied two continuous control tasks: the **Pusher** environment, a manipulation task involving a robotic arm and a moveable puck, as well as the **RoboYoga Walker** task, a locomotion task in which a humanoid agent has to hold yoga poses.

Pusher. The Pusher environment [Nair et al., 2018] is a simulated task in which a robotic arm with four degrees of freedom has to push a puck on a table to a target position. The action space is therefore continuous, as the agent controls the different joints of the arm. For this task, we study two distinct setups: the state-based one, in which the observation contains the position of the end-effector and the puck, and the RGB setup, where the observation is a snapshot showing the position of the puck and the arm. The goal is specified by the position of the puck, and the agent has to push it to the goal position. The initial position of both the puck and the arm are fixed, which makes it challenging for the agent to explore the whole state space.

RoboYoga Walker. The DeepMind control suite [Tassa et al., 2018] is a widespread collection of continuous control tasks, that are used to benchmark reinforcement

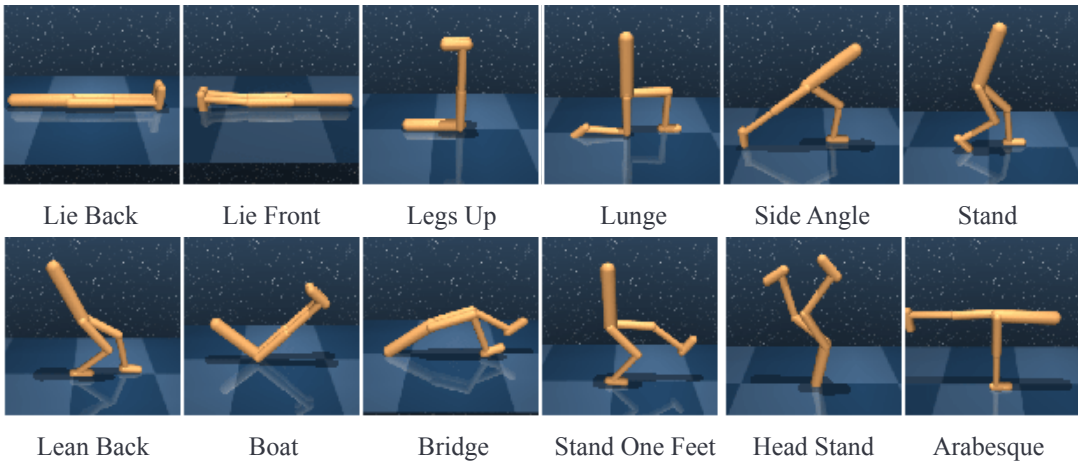


FIGURE 2.9: Goals for the RoboYoga Walker task. The agent has to hold the target pose until the end of the episode.

learning algorithms [Fu et al., 2020, Laskin et al., 2021, Yarats et al., 2022]. Although very challenging, most of the tasks are designed for standard RL algorithms, with a single objective and a well-shaped reward function. Among the tasks, the Walker domain presents a humanoid agent, whose objective is either to stand up, walk or run. Based on this humanoid, Mendonca et al. [2021] introduced the RoboYoga Walker task, which consists of a goal-conditioned version of the Walker domain, where the agent has to perform a set of yoga poses. The set of 12 goals that correspond to body poses inspired from yoga (*e.g.*, lying down, raising one leg or balancing) is shown in Fig. 2.9. Both state and action spaces are continuous, as the observation contains information about the relative position of its different joints, and the action is a multi-dimensional vector containing the force that should be applied to the joints. The starting position of the agent is randomized, which makes the coverage of the state space relatively uniform, even with a random policy. The difficulty of the task lies in the fact that the agent has to hold the pose until the end of the episode, which requires delicate balancing skills.

Chapter 3

Learning to Navigate from Pixels

Goal-oriented navigation in photo-realistic environments is one of the most widespread task in the field of robotics, as it has many applications, and is often a starting point for more complex problems, such as embodied question answering [Das et al., 2018], or instruction following [Hill et al., 2020]. In the task of image-goal navigation, the agent is given a goal image, and must navigate to the location in the environment where the image was taken from. It requires for the agent to have a good understanding of the environment, as well as decent planning capabilities to move intelligently towards the goal. Most prior works, including RL-based [Zhu et al., 2017, Fang et al., 2019] approaches, rely heavily on the availability of the absolute position of the agent in the environment, an assumption which is often unrealistic. Indeed, simulators may allow access to such information, but it is not the case for real robots, that are instead only equipped with RGB cameras or depth sensors.

In this chapter, we investigate the problem of learning to navigate from RGB input only, without access to position information. The main challenge of this task is that, in the absence of such information, the agent must learn an informative representation of the environment from the visual inputs. Our method relies on an attention-based end-to-end model that leverages an episodic memory to learn to navigate. First, we train a state-embedding network in a self-supervised fashion, and then use it to embed previously-visited states into the agent's memory. Our navigation policy takes advantage of this information through an attention mechanism. We validate our approach with extensive evaluations, and show that our model establishes a new state of the art on the challenging Gibson dataset.

3.1 Introduction

The challenges of addressing navigation tasks go beyond the classical computer-vision setup of learning from pre-defined fixed datasets. They consist of problems such as low-level control point-goal navigation [Wijmans et al., 2019], object-goal navigation [Batra et al., 2020] or even tasks requiring natural language understanding, e.g., embodied question answering [Das et al., 2018].

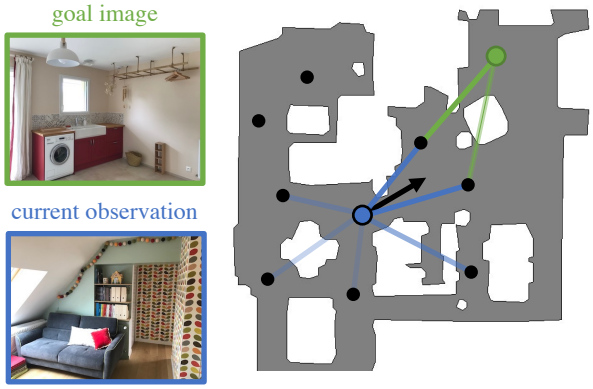


FIGURE 3.1: We tackle the problem of image-goal navigation. The agent (shown as the blue dot) is given an image from a goal location (green dot) which it must navigate to. To address this task, our agent stores an episodic memory of visited states (black dots), and uses a navigation policy that puts attention (lines) on this memory (best viewed in pdf).

We focus on one such critical problem: image-goal navigation [Zhu et al., 2017], wherein an agent has to learn to navigate to a location specified by visual observations taken from there. Consider the agent in Fig. 3.1, which is spawned at the location in blue on the map, where it observes a sofa. The agent's task is to find the location in the house where it would see the washing machine shown in the goal image. In terms of difficulty, this task lies in between point-goal and object-goal navigation. Indeed, it does not require learning the association between visual inputs and manual labels (as in object-goal navigation), but it needs a higher-level understanding of scenes for navigating through them. There are several facets to this task, which make it challenging.

The first challenge is to design methods that are completely end-to-end. This allows for approaches that require less expert knowledge and are easily transferable to new simulators and tasks. Classical methods for agent navigation, based on simultaneous localization and mapping [Thrun, 2002], comprise multiple hand-crafted modules, and require a large amount of annotated data. Reinforcement learning (RL) is a popular framework for tackling navigation problems in an end-to-end manner [Zhu et al., 2017, Fang et al., 2019]. However, in the context of photorealistic data, existing methods have either shown results in a limited setting with synthetic data [Fang et al., 2019] or suffer from poor RL-based performance [Chaplot et al., 2020c].

The second challenge in image-goal navigation is the need for a high-level understanding of the surrounding scene. In photorealistic environments, agents trained with RL are subject to overfitting due to the high dimensionality of the data and the limited number of environments available for training [Xia et al., 2018]. Learning to navigate in such visually complex environments, therefore, requires learning a more informative representation than pixels, which captures the visual diversity of the scene and is generalizable to unseen environments. Previous methods use the position of the agent and/or a depth map [Chaplot et al., 2020c, Fang et al., 2019]

to improve the generalization of the model. Approaches that learn from RGB input only have not shown generalization to unseen environments [Savinov et al., 2018a, Mezghani et al., 2020].

Finally, a requirement for navigating agents is to build and exploit a representation of the states visited. Indeed, the agent should be able to remember the places it has already visited within an episode for efficient exploration [Mezghani et al., 2020, Fang et al., 2019]. This "memory" can take the form of a buffer [Fang et al., 2019], or a metric [Thrun, 2002, Chaplot et al., 2018] or a topological map [Savinov et al., 2018a, Chaplot et al., 2020c]. In most methods, the agent exploits this information with an explicit planner [Savinov et al., 2018a, Chaplot et al., 2020c], but some work [Fang et al., 2019] has also leveraged it with an implicit attention. However, the generalization properties of these schemes remain unclear: the approach by [Savinov et al., 2018a] requires exploration videos collected by a human when navigating to an unseen environment, while [Fang et al., 2019] only shows limited generalization results on synthetic data.

We tackle these three challenges with a memory-augmented reinforcement learning approach. First, the agent learns representations of the environment in a self-supervised fashion, and acquires a high-level understanding of the surrounding scene. It then learns a policy for image-goal navigation in an end-to-end manner. In order to explore and navigate effectively, the policy is conditioned on an external memory module that remembers useful information from the current episode. With this approach we establish the new state of the art on the challenging image-goal navigation task on the Gibson dataset [Xia et al., 2018] by a large margin. We achieve this performance from RGB input alone, without access to position information, which is a major improvement compared to previous works like [Chaplot et al., 2020c].

3.2 Related Work

SLAM-based Methods The task of navigation has been studied in the context of simultaneous localization and mapping (SLAM) in robotics [Thrun, 2002]. Several SLAM methods comprise multiple hand-crafted modules to address strictly-defined problems in specific environments [Tomatis et al., 2001, Mur-Artal et al., 2015, Choset and Nagatani, 2001]. These modules have been progressively replaced with learning-based functions: some approaches [Chaplot et al., 2018] implement the localization module with a neural network, while others [Gupta et al., 2017a] replace the metric map with a latent mapping of the environment. Variants of latent mapping also include a topological map whose nodes contain geometric and semantic information about the environment, as well as a global planner that relies on it [Chaplot et al., 2020c]. Other works replace SLAM entirely by deep models without explicit planning, and instead rely on a map or memory structure [Parisotto

and Salakhutdinov, 2018, Zhang et al., 2017, Avraham et al., 2019]. The major drawback of such methods is that they contain multiple modules that are often trained in a supervised fashion, requiring a large amount of annotated data. Moreover, a variety of methods inspired by SLAM rely on the availability of position [Zhang et al., 2017, Parisotto and Salakhutdinov, 2018] and depth [Taketomi et al., 2017, Avraham et al., 2019, Chaplot et al., 2020c] sensors. In our work, we focus on the realistic and challenging case with access to RGB input only.

RL-based Navigation Another popular class of methods involves training deep models with reinforcement learning to solve navigation tasks without an explicit world representation. They use end-to-end frameworks with modules that are less hand-crafted than SLAM-based methods, and have shown good performance on synthetic mazes [Mirowski et al., 2017] as well as real-world data [Mirowski et al., 2018, Chancán and Milford, 2020, Kadian et al., 2020, Wijmans et al., 2019]. Such methods have also been explored on indoor-scenes datasets, similar to our setup on image-goal [Zhu et al., 2017] and object-goal [Yang et al., 2018, Maksymets et al., 2021, Chaplot et al., 2020a] navigation tasks. They use an actor-critic model whose policy is a function of both the target and the current state. While episodic RL is a simple and elegant framework for navigation, it ignores the fact that useful information comes from previous episodes. In our work, we propose to augment the policy used in RL frameworks with an external memory and to attend on it with a novel, dedicated module.

Combining RL and Planning A few recent works have augmented RL-based methods with topological structures, like graphs [Wu et al., 2019, Beeching et al., 2020b, Savinov et al., 2018a, Chen et al., 2019, Mezghani et al., 2020] or memory buffers [Fang et al., 2019, Beeching et al., 2020a, Kumar et al., 2018]. They store representations of the visited locations and exploit them at navigation time. The process of building these representations can be done offline [Savinov et al., 2018a, Beeching et al., 2020b, Mezghani et al., 2020], and requires human-generated data in some cases [Savinov et al., 2018a]. For example, the test phase in [Savinov et al., 2018a] contains a warm-up stage where the agent builds a graph memory from human trajectories. Alternatives to this manual annotation do exist, such as building a graph directly with reinforcement learning, using the value function of a goal-conditioned policy as edges weights [Eysenbach et al., 2019], or buffer of past observations [Fang et al., 2019]. These methods were evaluated on synthetic datasets, and have not been scalable to high-dimensional visually-realistic setups. In particular, [Fang et al., 2019] proposes a method related to our model, with a policy that puts attention on an observation buffer. In contrast, we learn a representation in a self-supervised fashion to store memory efficiently. We also present results on a large-scale photorealistic dataset, while [Fang et al., 2019] is limited to synthetic setups.

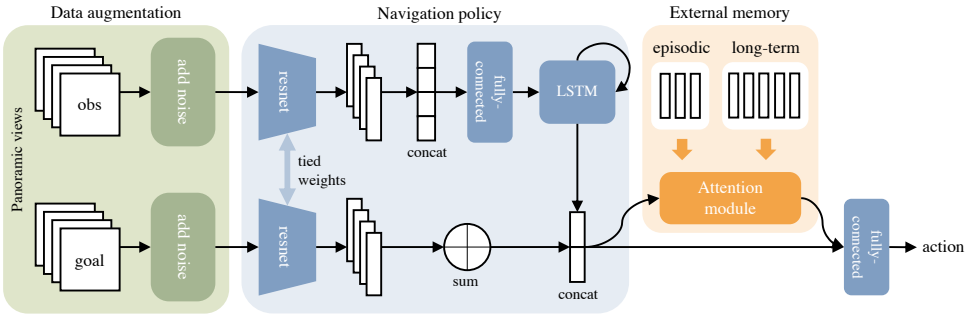


FIGURE 3.2: An overview of our model that consists of three parts: a data augmentation module (green) for better generalization, a navigation policy (blue) for picking actions, and an external memory (orange) for conditioning on previous observations.

Exploration and Representation Learning Closely related to navigation, the task of exploration has also been extensively studied and has led to interesting breakthroughs in representation learning. In particular, learning to explore unseen environments has been seen through the spectrum of computer vision [Jayaraman and Grauman, 2018, Chaplot et al., 2020b], SLAM-based [Chaplot et al., 2019], and RL-based [Chen et al., 2018, Devo et al., 2020] approaches. Methods such as [Savinov et al., 2018b, Mezghani et al., 2020] leverage a self-supervised representation learning stage to prepare the exploration phase. Our work extends this line of study by first showing that a self-supervised pre-training phase allows to learn useful information that generalizes to unseen environments, as well as proposing a novel attention-based navigation policy that takes advantage of this information. Posterior to our work, there have been extensions of our method, including [Sang et al., 2022] that exploits a similar approach for object-goal navigation in unseen environments. This work uses the same components than our method: both short-term and long-term memories, as well as an attention mechanism to leverage past experience. The main difference with our work relies on the task being tackled: contrary to our work, [Sang et al., 2022] focuses on object-goal navigation, which consists in navigating to an instance of a target object.

3.3 Image-Goal Navigation

We consider the classical formulation of episodic image-goal navigation as defined in [Chaplot et al., 2020c]. At the beginning of a navigation episode, an agent is given a target observation x^*, composed of an RGB image from the target location. At each timestep t, the agent performs an action a_t and receives the next observation x_{t+1} as well as a reward r_t from the environment. The objective is to learn a navigation policy function $\pi(a_t|x_t, x^*)$ that brings the agent closer to the target location. We complete the definition of our setup with the following details.

Action Space It comprises four actions: `MOVE_FORWARD`, `TURN_LEFT`, `TURN_RIGHT` and `STOP`. Please refer to Sec. 3.5.1 for numerical details.

Success Criterion An episode is considered successful if the agent performs the stop action within a range of l from the target location. In cases where the agent performs the stop action outside of this range, or if the maximum number of steps is exceeded before the agent performs the stop action, the episode is considered a failure.

Observation Space The observation of the agent x_t as well as the goal observation x^* are the RGB images of the first-person view at those locations. Each RGB image is a panoramic sensor of size $v \times 3 \times 128 \times 128$. We compute this panoramic input by gathering observations from v successive rotations of angle $(360/v)^\circ$ from our agent's location. Note that we do not have access to neither the agent's position nor any depth sensor information.

Reward We follow the classic setup for image-goal navigation [Chaplot et al., 2020c] where the reward is split into three components: (i) *sparse success reward*: that rewards the agent for performing the stop action within the success range around the target location, (ii) *dense shaping reward*: that is equal to the decrease in distance to the goal, (iii) *dense slack reward*: that penalizes the agent for being alive at each step (through a constant negative value), and encourages shorter trajectories.

3.4 Memory-Augmented Navigation Policy

As shown in Fig. 3.2, our agent model has three parts: a data augmentation part for improving generalization, a navigation policy that learns to pick appropriate actions, and an external memory for leveraging past experiences.

3.4.1 Data Augmentation

To improve the generalization capacity of our agent to unseen environments, we apply random transformations on the observations of the simulator by using classic data augmentation techniques. We use two kinds of data transformations: (i) **random cropping** that increases the input image size and takes a random crop of the original size in it, and (ii) **color jitter** that randomly changes the brightness, contrast, saturation and hue levels of the image. An illustration of these transformations is shown in Fig. 3.3. At navigation time, the agent receives the current and the goal observations from the simulator at each timestep. We apply both transformations sequentially to each of the v views of the current and goal observations independently, producing x_t and x^* respectively. This process allows for more visual diversity in the training data.

FIGURE 3.3: Illustration of data augmentation that we use to train our model. We consider both color jittering (left) and random crops (right). For a panoramic observation with v views, the parameters of the augmentation are sampled independently.

3.4.2 Navigation Policy

Once the current and goal observations pass through the data augmentation phase, we use them in the navigation policy module, which computes the probability distribution over all possible actions: $\pi(a_t|x_t, x^*)$.

First, the policy encodes each observation separately, as shown in Fig. 3.2. We encode the current observation by feeding each of the v views separately to the same convolutional neural network. The v vectors resulting from this operation are concatenated and passed into a fully-connected network. This dimension-reduced output is then fed into a 2-layer Long Short-Term Memory (LSTM) along with a representation of previous actions, and the resulting vector w_t^{obs} represents the embedding of the current observation at step t.

To encode the goal observation x^*, we process it through the same convolutional neural network as the current observation. However, the outputs corresponding to the different views are added together instead of being concatenated, so as to make the representation of the goal rotation-invariant. We denote by w_t^{goal} the resulting feature vector at step t.

Next, we make a joint representation by concatenating the current and goal feature vectors, before passing it through a fully-connected network to output an action a_t as follows:

$$w_t^{\text{joint}} = \texttt{cat}(w_t^{\text{obs}}, w_t^{\text{goal}}), \tag{3.1}$$

$$\pi(a_t|x_t, x^*) = \texttt{FC}(w_t^{\text{joint}}). \tag{3.2}$$

3.4.3 External Memory

To add an external memory mechanism to the navigation policy, we first train a state-embedding network in a self-supervised fashion. This network, trained to detect nearby locations, allows us to build an external memory containing representations

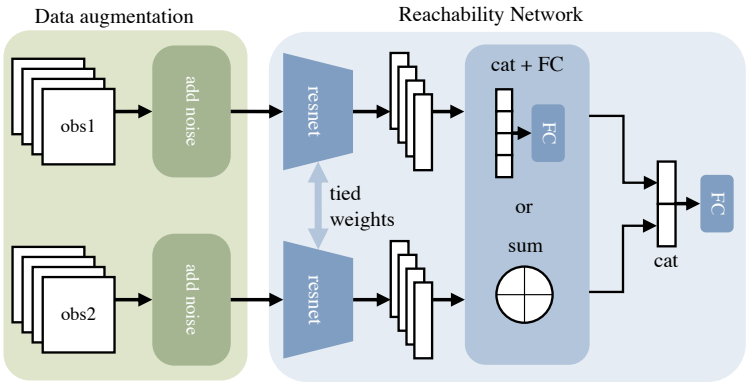


FIGURE 3.4: Architecture of the Reachability Network. We adapted the architecture from Savinov et al. [2018b] by using a data augmentation module and a layer that handles panoramic observations. The output of the last fully-connected module is the similarity score between the two observations.

of past observations. To leverage this memory, we add an attention module to the navigation policy.

Training a State-Embedding Network. Before learning the navigation policy, we train a state-embedding network to learn representations of the environment's locations. The motivation for introducing this network is to encourage nearby locations in the environment to have similar representations, while ensuring distant locations to have different ones. However, since we do not have access to the agent's position, the notion of distance between locations in the environment cannot be computed directly. As in Savinov et al. [2018b], we will use the number of steps taken by an agent with a random policy to approximate this distance.

We let an agent with random policy explore the environment for T steps and denote by $(x_1, ..., x_T)$ the corresponding sequence of observations.[1] We then define a reachability label y_{ij} for each pair of observations (x_i, x_j) that depends on their distance in the sequence. More precisely,

$$y_{ij} = \begin{cases} 1 & \text{if} \quad |i-j| \leq k, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } 1 \leq i, j \leq T \tag{3.3}$$

where k is a hyperparameter.

We train a siamese neural network R, to predict the reachability label y_{ij} from a pair of observations (x_i, x_j). The architecture of R is shown in Fig. 3.4. R is defined by a convolutional network g to embed the observations, and a fully-connected network f to compare the embeddings, i.e.,

$$R(x_i, x_j) = f(g(x_i), g(x_j)). \tag{3.4}$$

[1] We ensure that the length of the computed sequence is T by removing the `STOP` action from the action space.

We apply the same data augmentation techniques to observations during this reachability network training phase.

Episodic Memory. Once we have the reachability network that can distinguish observations from nearby and distant locations, the agent can collect a compact memory of previously visited states.

We follow a process similar to Savinov et al. [2018b] for building episodic memory. At timestep t, the agent has a memory buffer M_{t-1} with embeddings from observations seen at previous timesteps. Since storing every observation seen by the agent would be inefficient, we store only observations that are considered novel, i.e., distant from the current memory vectors. In other words, at each timestep, we use the network R to compute a reachability score between the current observation x_t and the memory buffer M_{t-1}. This score corresponds to the maximum value obtained when comparing the current observation to every vector in the memory, i.e.,

$$r(x_t, M_{t-1}) = \max\{\, f(g(x_t), m) \mid m \in M_{t-1}\}. \tag{3.5}$$

We then add the embedding of current observation to the memory if the reachability score is lower than a threshold, i.e., distant enough from the memory vectors. In other words,

$$M_t = \begin{cases} M_{t-1} \cup g(x_t) & \text{if } r(x_t, M_{t-1}) < \tau, \\ M_{t-1} & \text{otherwise,} \end{cases} \tag{3.6}$$

where τ is the *reachability threshold* hyperparameter. The episodic memory is reset after each episode.

This process for building episodic memory can be extended so that it persists across episodes within the same scene. The impact of using such a cross-episodic memory is studied in Sec. 3.5.4

Attention Module. The navigation policy can leverage the episodic memory to move towards the target observation. Rather than using an explicit planner on the memory, we take advantage of the information stored in them implicitly, using an *attention module*.

Our attention module has a multi-layer architecture similar to transformers [Vaswani et al., 2017]. Each layer consists of a multi-head attention sublayer (`Attn`), followed by a feedforward sublayer (`FF`). See Vaswani et al. [2017] for more details about these sublayers. However, unlike transformers our attention module attends over a fixed set of vectors. It comprises N layers,

$$z_t^l = \texttt{FF}\left(\texttt{Attn}(z_t^{l-1}, M_t)\right), \quad \text{for } l \leq N. \tag{3.7}$$

Here, z_t^l is the output from the l-th layer, but the initial input z_t^0 is obtained from a linear transformation of the joint representation computed in Equation 3.1, w_t^{joint}.

The output from the attention module is then concatenated with the joint representation in Equation 3.2, so the final action is now computed as:

$$\pi(a_t|x_t, x^*) = \texttt{FC}(\texttt{cat}(w_t^{\text{joint}}, z_t^N)). \tag{3.8}$$

3.5 Experimental Results

3.5.1 Implementation Details

Task Setup. We conducted all of our experiments on the Habitat [Savva et al., 2019] simulator with the Gibson [Xia et al., 2018] dataset, which contains a set of visually-realistic indoor scenes. We used the standard 72/14 train/test scene split for this dataset. As stated earlier, **we do not use the agent's pose or depth sensor information**. The forward step range and turn angle are set to their standard values $(0.25\text{m}, 10^\circ)$ for navigation episodes and $(1\text{m}, 30^\circ)$ when training the reachability network. The maximum number of steps in an episode is 500, and the success distance l is 1m. In addition to the success rate, we also use success weighted by inverse path length (SPL) [Anderson et al., 2018] as an evaluation metric. SPL takes into account the length of the path that the agent has taken to the goal.

Training the Reachability Network. We generate one trajectory per train scene from an agent with a random policy. We allow 5k steps for each trajectory and remove the stop action from the action space. This results in a total of 360k steps from 72 scenes to train the reachability network. From each trajectory, we sample 1k positive pairs (within 10 timesteps) and 1k negative pairs, yielding a dataset of 144k image pairs. We implement the reachability network as a siamese network with ResNet18 for the embedding function g. Each of the v views from the RGB observation is passed through the ResNet separately. We sum the resulting outputs to form the embedding vector of a panoramic observation. The comparison function f is composed of two hidden layers of dimension 512 with ReLU activations. We train this network using SGD for 30 epochs with a batch size of 256, a learning rate of 0.01, a momentum of 0.9, a weight decay of $1\text{e} - 7$, and no dropout.

Training Data for the Navigation Policy. We generated 9k navigation episodes in each training scene, following the protocol of [Chaplot et al., 2020c]. [2] We split our navigation episodes into three levels of difficulty, based on the distance between the start and the goal locations: *easy* (1.5 - 3m), *medium* (3 - 5m), and *hard* (5 - 10m). For each scene, we sample 3k start-goal location pairs per level of difficulty, resulting in

[2] We obtained the generation procedure from the authors of [Chaplot et al., 2020c] by e-mail. The dataset is available at: https://github.com/facebookresearch/image-goal-nav-dataset

	Observation Type			Easy		Medium		Hard		Overall	
Model	RGB	Pose	Depth	Succ	SPL	Succ	SPL	Succ	SPL	Succ	SPL
NTS-D	✗	✗	✗	**0.87**	**0.65**	0.58	0.38	0.43	0.26	0.63	0.43
ResNet + GRU + IL	✗	✗	-	0.57	0.23	0.14	0.06	0.04	0.02	0.25	0.10
Target-Driven RL				0.56	0.22	0.17	0.06	0.06	0.02	0.26	0.10
Active Neural SLAM				0.63	0.45	0.31	0.18	0.12	0.07	0.35	0.23
NTS				0.80	0.60	0.47	0.31	0.37	0.22	0.55	0.38
SPTM	✗	-	-	0.64	0.35	0.52	0.27	0.36	0.19	0.51	0.27
Ours	✗	-	-	0.78	0.63	**0.70**	**0.57**	**0.60**	**0.48**	**0.69**	**0.56**

TABLE 3.1: Comparison of our proposed model with several baselines and state-of-the-art approaches. The "observation type" column shows the type of observation for each method: raw pixel observations (RGB), depth map (Depth), and position information (Pose). We report success rate and SPL, over three levels of difficulty. Our method establishes a new state of the art, e.g., doubling the SPL on *hard* episodes, while not requiring any pose or depth information.

648k train episodes. Similarly, we sample 100 episodes per test scene and per level of difficulty, resulting in 4.2k test episodes.

Navigation Policy Implementation. At the beginning of each episode, the simulator generates the observation from the goal location as a $v \times 3 \times 128 \times 128$ panoramic RGB image and gives it to the agent as target observation. We used ResNet18 with shared weights for encoding the current and target observations in the policy. The size of the embedding space is 512. We concatenate the encoder's outputs for the v views of the observation and feed them into an LSTM with two recurrent layers. Our attention module consists of 4 stacked layers of a 4-headed attention network. We set the buffer's capacity to 20 for the episodic memory. We train the policy using DD-PPO [Wijmans et al., 2019] for 50k updates, with 2 PPO epochs, a forward of 64 steps, an entropy coefficient of 0.01, and a clipping of 0.2. We used the Adam optimizer with a learning rate of $9e-5$.

Data Augmentation. For the training stages of both the reachability network and the navigation policy, we used random cropping with a minimum scale of 0.8 and color jittering with value 0.2 for brightness, contrast, saturation, and hue levels. These transformations are applied at two different levels when training the navigation policy: (i) when the agent samples the action a_t from the policy, and (ii) during the forward-backward in PPO. Note that, for these two steps, the transformation applied to the images is independent and results in different input images.

3.5.2 Comparison with the state of the art

We consider several methods in this comparison: an adaptation of SPTM [Savinov et al., 2018a] to Habitat, as well as four approaches taken from [Chaplot et al., 2020c].

- **ResNet + GRU + IL** [Chaplot et al., 2020c]: A simple baseline consisting of ResNet18 image encoder and a GRU-based policy trained with imitation learning.
- **Target-Driven RL** [Zhu et al., 2017]: A vanilla baseline trained with reinforcement learning, similar to our ablated variant.
- **Active Neural SLAM** [Chaplot et al., 2019]: An exploration model based on metric maps, adapted to navigation.
- **Neural Topological SLAM (NTS)** [Chaplot et al., 2020c]: The previous state-of-the-art method for navigation in unseen environments, based on neural SLAM and endowed with a topological graph for long-term planning. We compare against two versions of this method: **NTS**, that uses as input the RGB observation and the pose sensor, and **NTS-D** that uses the depth map in addition to the two other modalities.
- **SPTM** [Savinov et al., 2018a]: A navigation model based on explicit construction of a graph over past experience. The method requires an exploration episode given *a priori*. To run this baseline, we adapt the available code[3] to Habitat. We train the reachability and locomotion networks on trajectories from the training set. For evaluation we used a random exploration trajectory of 10k steps, and build a graph with 100k top-scoring edges. As this method does not implement a `STOP` action, we give it advantage and automatically end the episode when the agent gets within 1m of the goal.

The first four methods are designed for episodic navigation, i.e. the validation episodes are performed on completely unseen scenes, without pre-exploration. Conversely, **SPTM** is intended for setups where a pre-exploration phase is allowed on the test scenes before starting the actual navigation phase, and therefore uses more privileged information.

We trained our model on the *easy*, *medium*, *hard* combined dataset for 500M steps for three random seeds, and evaluated it on the corresponding test set. As shown in Table 3.1, the performance obtained with our memory-augmented model is superior to that of previous work by a significant margin: +13% SPL on average compared to **NTS-D** and +29% SPL vs. **SPTM**. We obtain this strong performance while, unlike **NTS-D**, not using position nor depth information. We note that the success rate on *easy* episodes is lower for our method than **NTS-D** (−9%). This is potentially due to the lack of position and depth features or the use of discretized panoramic observations, both of which are particularly useful for nearby goals. Note that the four methods from [Chaplot et al., 2020c] require 25M steps of training until convergence in comparison to our 500M steps. This difference in training horizon is due

[3] https://github.com/nsavinov/SPTM

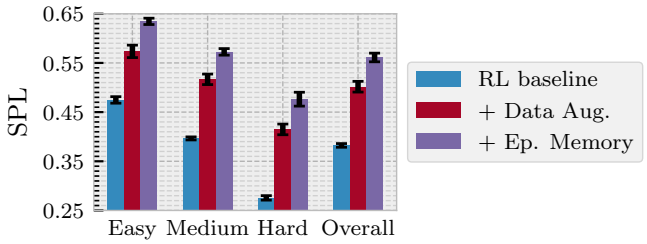


FIGURE 3.5: Ablation study. We present the SPL for three variants of our model. All models are trained for 500M steps on a combination of *easy*, *medium*, and *hard* episodes.

to several factors, including the absence of position information in the observations for our setup, and the absence of an explicit planner in our method. Rather than comparing all the methods at a fixed-step budget, when they may not be trained optimally, we made the fair choice to compare all the methods after the convergence of their respective optimization processes.

3.5.3 Ablation Study and Analysis

Ablation. We perform an ablation study to empirically validate the design choices of our model. We evaluate the performance of the simple target-driven RL baseline, as well as the improvements brought by the data augmentation module and the episodic memory. To this end, we train three variants of our model: (i) **RL baseline**, the vanilla target-driven RL baseline to which we consecutively add (ii) **Data Aug.**, the data augmentation module, and (iii) **Ep. Memory**, the episodic memory module. We train all models on this dataset for 500M steps for three random seeds and report the average SPL for each level of difficulty in Fig. 3.5.

First, we see that using data augmentation when training a RL-based navigation policy in this context improves SPL significantly: the gap with the vanilla baseline is +12% *overall*. Second, we observe that the episodic memory-based policy improve over the very competitive data-augmented baseline (+6% *overall*). Finally, we note that the data-augmented baseline significantly outperforms the state of the art method **NTS** (+12% *overall*), highlighting the power of this simple end-to-end model.

Number of views (v)	1	3	4	6
SPL	0.08	0.31	0.36	0.36
Frames per sec.	1890	2000	2080	2340

TABLE 3.2: Analysis of SPL obtained with the **RL baseline** for various panoramic view configurations. We report average SPL and the number of frames processed per second for each configuration.

Panoramic Observations. As described in section 3.3, we generate panoramic observations by equally spaced planar observations around the agent. In this experiment, we compare the performance of the vanilla **RL baseline** model trained with

1, 3, 4 and 6 views around the agent. We let the model train for 500M steps for three random seeds and report the average SPL obtained by this agent in Table 3.2.

We note that an agent trained with a single view, i.e., without panoramic observations, completely fails to learn a successful policy, obtaining only 0.08 SPL. This result is quite intuitive, as the relative localization with respect to the goal is made easier by multiple views. Better performance is obtained with either four or six views, with an SPL of 0.36. We also see that there is a tradeoff between performance and additional runtime required with more views. Thus, we run all the variants of the models with four views.

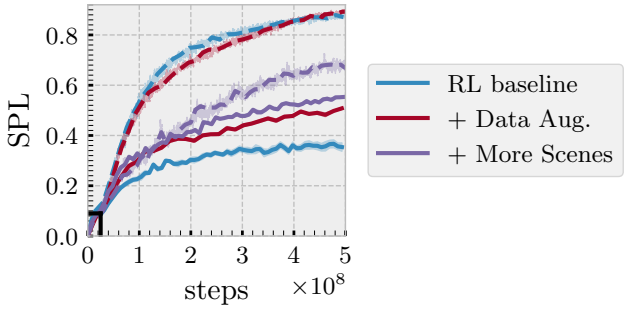


FIGURE 3.6: Performance measured in SPL as a function of training steps taken in the environment for the **RL baseline**, with data augmentation (+ Data Aug.) and by adding more training scenes (+ More Scenes). We report both the training (dashed line) and test (solid line) SPL on the same figure. The generalization gap is large and can be reduced by using data augmentation. It can be reduced significantly further by training the model on a larger set of scenes. The black mark at 25M steps corresponds to the **Target-Driven RL** baseline as reported in [Chaplot et al., 2020c].

Data Augmentation and Overfitting. We investigate how data augmentation and adding more training scenes allow us to bridge the train / test gap observed when navigating an unseen environment. To this end, we generate an extended training set by considering 150 additional scans, which are usually rated as being of poor quality. We perform this experiment for the vanilla **RL Baseline**, as well as its data-augmented version trained on the standard dataset (**Data Aug.**), and the extended one (**More Scenes**).

As seen from the train and test SPL for the three methods in Figure 3.6, the generalization gap is large (almost 65% for the **RL baseline**). The use of data augmentation allows to remedy this problem reducing this gap to about 40%. Training the model on more scenes reduces this discrepancy even further to only 15%. We also observe that data augmentation not only improves the test performance but also helps faster convergence (+2% on the train SPL), as opposed to what is usually observed in supervised learning.

Notice that in the figure, we indicate the 25M step mark – this is the number of environment steps used to train the **Target-Driven RL** baseline reported in [Chaplot et al., 2020c]. We see that the performance of a reinforcement-based model with

Model	Easy	Med.	Hard	Extra	Overall
Ep. Memory	0.560	0.505	0.428	0.138	0.407
LT Memory	**0.569**	**0.528**	**0.455**	**0.161**	**0.428**

TABLE 3.3: Evaluating the impact of a memory that persists across episode. We train both models for 500M steps on a combination of *easy*, *medium*, *hard*, *extra* episodes. The long-term memory improves the performance of the model, especially on the *hard* and *extra* splits.

such a limited number of steps is quite poor, and that convergence is far from being achieved at this point. Though sample efficiency is an important challenge in robotics applications, the **RL baseline** should be trained sufficiently to leverage its full potential. Indeed, after 500M steps of training, it reaches 0.37 SPL *overall*, which is almost on par (−1%) with the performance of the method proposed in [Chaplot et al., 2020c].

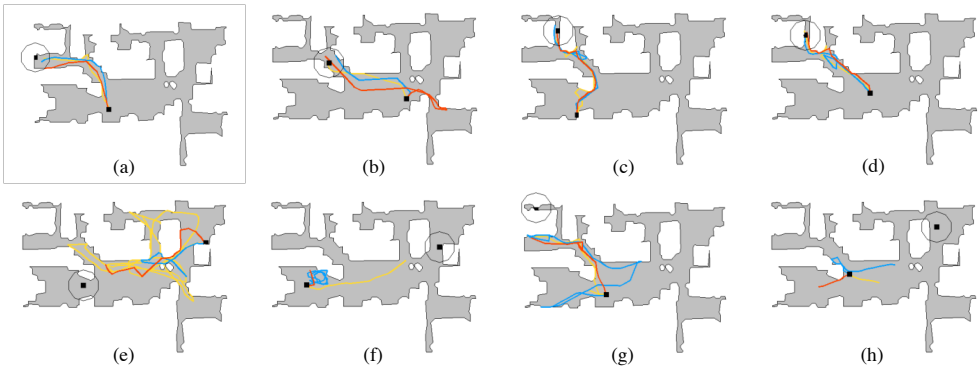


FIGURE 3.7: The top (resp. bottom) row shows trajectories from test episodes with the highest (resp. lowest) SPL on the Eastville scene. Start and goal locations are shown in black, with the goal being circled by a line showing the success area. Results are shown for our model with memory trained on the standard dataset, for 3 seeds (corresponding to the 3 colors) on the *hard* validation split.

3.5.4 Impact of a Long-Term Memory

We also studied the impact of a memory that persists across episodes in the same scene, motivated by the observation that embodied agents, once deployed, do not simply cease to exist after an episode has ended: they persist, and so should their memories. We therefore added an additional memory to the episodic one, that remains for 100 episodes in the same scene. We compare this model with long-term memory (**LT Memory**) to the one with episodic memory only (**Ep. Memory**) in Table 3.3. For this experiment, we generated an additional split to the *easy*, *medium*, and *hard* episodes, named "*extra*," for which the distance between start and goal locations is 10 - 15m. This dataset, voluntarily more challenging, is intended to highlight the importance of a memory that persists across episodes. The long-term memory improves the performance of the model over the episodic one (+2.1% *overall*), and the difference is more important on *hard* and *extra* episodes (+2.7% and +2.3% respectively) than on *easy* ones (+0.9%).

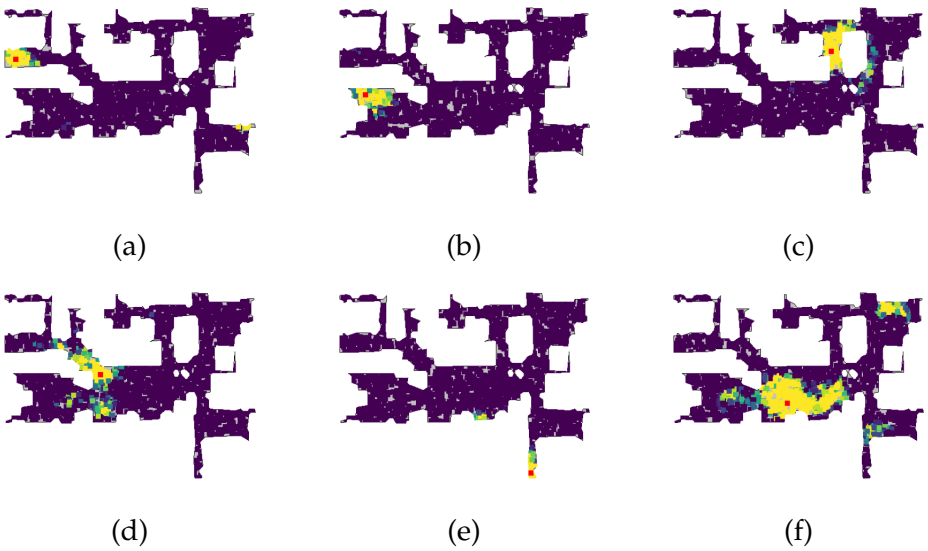


FIGURE 3.8: Heat-maps of the similarity score between the observation (the red point) and the observations at $N = 2000$ points sampled randomly in the environment. The color at a location corresponds to the similarity score at that location: low values, close to 0, are in dark violet and high values, close to 1, in yellow. These visualizations were performed on Eastville: an environment of the validation set.

3.5.5 Qualitative Visualizations

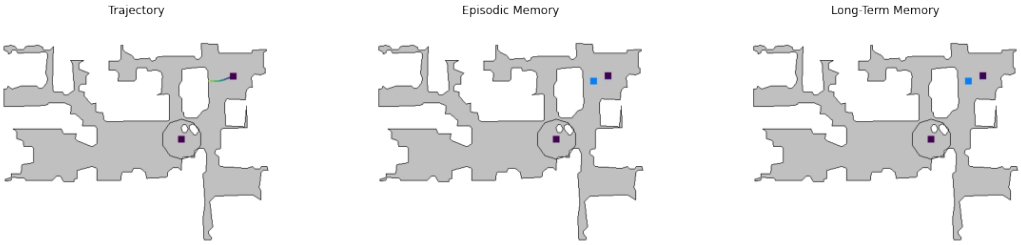
Trajectories. Figure 3.7 shows example success and failure cases from episodes of the test dataset. We see that our agent successfully learns to navigate to challenging locations, which are distant from the start location (3.7-a) and/or located at extremities of the scenes (3.7-c, 3.7-d). Moreover, our agent shows interesting skills, like bypassing obstacles (3.7-c) or looking around in a room (3.7-b). From the failure cases (bottom row in the figure), we see that our agent has some undesired behavior. For example, it can get stuck in a loop (3.7-f), stop too early (3.7-h), or fail to reach some extremely challenging goals (3.7-g).

Reachability Network. We visualize the quality of the Reachability Network with the following experiment. First, we put the agent at a random location in the environment and sample an observation x from there. Then, we randomly sample N observations in the environment and for each of these observations, we compute their similarity score with observation x, using the Reachability Network. We present these results on a heat-map, where the color at a location represents the corresponding similarity score, in Figure 3.8. We see that the high similarity scores are at locations that are around the comparison observation, which implies that the Reachability Network performs well at learning representations that are similar for nearby locations, and dissimilar for representations that are far away. Since these experiments are shown on a validation environment, we note that the Reachability Network generalizes well to unseen environments.

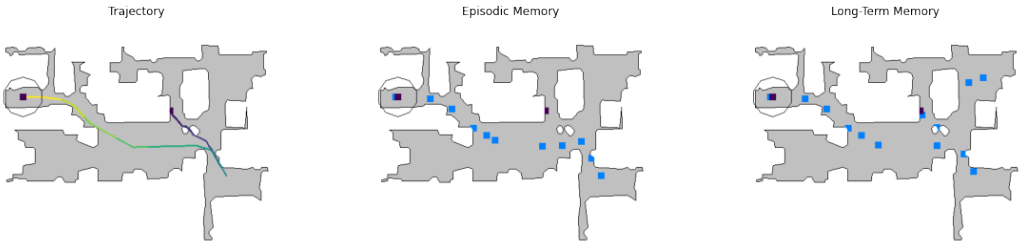
Episodic and Long-term Memories. Finally, we visualize the state of the episodic and long-term memories for consecutive validation episodes in Fig. 3.9. From this, we observe how these memories are filled through consecutive validation episodes. After 100 navigation episodes (3.9-d), the long-term memory is well filled and covers most of the environment. This allows the agent to reach challenging goals.

3.6 Conclusion

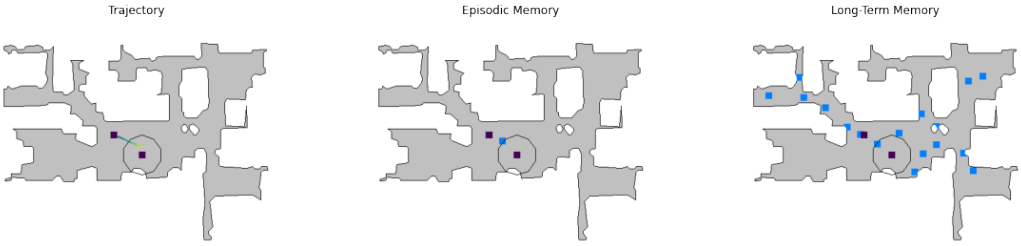
In this chapter, we have presented a memory-endowed agent that we train end-to-end with reinforcement learning. This memory is accessed in the navigation policy using a transformer-inspired neural network with attention modules. We evaluated our agent on the challenging task of image-goal navigation, and have shown that it surpasses previous work by a large margin. This impressive performance is obtained from RGB observations alone, i.e., without using any position information or depth sensors.



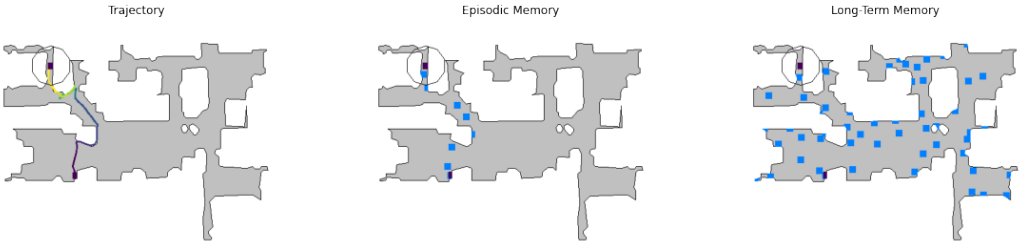
(a) Step 1



(b) Step 2



(c) Step 5



(d) Step 100

FIGURE 3.9: Visualization of the agent's trajectory, episodic and long-term memories for first, second, fifth and 100th episode in the Eastville environment. The start and goal locations are shown in black, goal location being circled by a line showing the success area. The blue points represent the location of the episodic and long-term memory vectors. The episodic memory is reset after each episode, while the long-term memory remains for 100 episodes in the same scene.

Chapter 4

Image-Goal Navigation without Supervision

In Chapter 3, we studied the problem of image-goal navigation in the absence of an important supervision modality, namely the position of the agent in the environment. In that sense, the proposed method encourages learning agents that do not rely on human-engineered data, and instead leverage what is available on real robots – inputs from RGB cameras. However, one limitation of the method presented in the previous chapter is that it relies on external rewards, provided by the environment. Indeed, in its original formulation, the task of image-goal navigation assumes the availability of a dense reward function that recompenses the agent for getting closer to the goal. In practice, this function is computed by calculating the shortest-path from the agent to the goal, which can be both tricky and expensive to compute for certain tasks.

In this chapter, we study the problem of navigation from image inputs without external supervision or reward, in photorealistic environments. We present a three-stage approach, where the agent first learns good representations of the environment from visual inputs, then explores using a memory-based mechanism, and finally learns to navigate by setting its own goals. The navigation agent is trained using only intrinsic rewards, computed using the learned representations, thus not relying on external hand-crafted rewards. We show the benefits of our approach by training an agent to navigate challenging photo-realistic environments from the Gibson dataset [Xia et al., 2018] with RGB inputs only.

4.1 Introduction

As mentioned in the previous chapter, the problem of learning to navigate is challenging, especially in settings where it is necessary to do without accurate depth or position information; or more generally, with as little supervision as possible. Furthermore, if the goal location is specified as an image, the agent needs to learn a

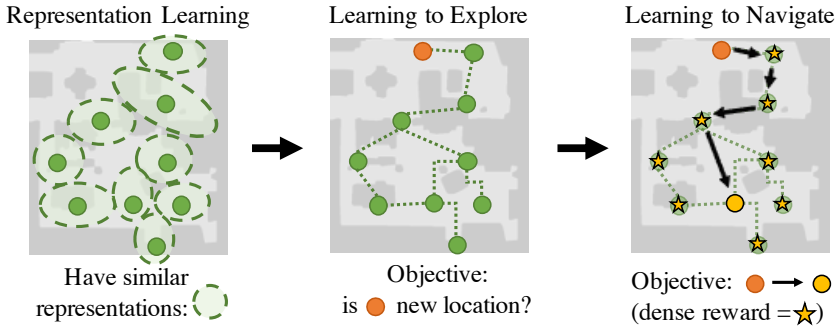


FIGURE 4.1: **Three stages of training:** the agent learns to distinguish locations from its visual inputs, then it explores the environment and builds a map of the environment, finally it uses this map to learn how to navigate the environment. Each step requires no external supervision or reward and the agent has only access to a visual RGB input, and has no information about its position.

good visual representation and an efficient exploration strategy in addition to the navigation policy.

There has been some interest in using techniques from deep learning in the context of SLAM, or more generally, in the context of navigation [Avraham et al., 2019, Fang et al., 2019, Gupta et al., 2017a, Henriques and Vedaldi, 2018, Khan et al., 2018, Oh et al., 2016, Parisotto and Salakhutdinov, 2018, Wierstra et al., 2010, Zhang et al., 2017]. Deep learning-based methods typically require a large number of trials during training and have been rarely considered outside of simulators. However, the growing number of photo realistic environments [Chang et al., 2017, Xia et al., 2018], efficient simulators [Dosovitskiy et al., 2017, Savva et al., 2019] and dedicated methods to transfer from simulated to real environments [Sadeghi and Levine, 2017, Tobin et al., 2017] have fueled the research in deep learning-based navigation methods.

In a separate line of study, there has been great progress in learning image representations through "self-supervised" approaches [Caron et al., 2018, Doersch et al., 2015, Goyal et al., 2019, Zhang et al., 2016]. In these works, using prior knowledge about the basic regularities of images, researchers find pretext tasks that, when solved, give good feature representations for other tasks of interest. While self-supervised learning is interesting for understanding learning methods abstractly, it also promises to be important in applications, as it is often the case that a pretext task is easier to come by and more general than strong supervision.

In this chapter, we introduce an entirely unsupervised method for learning to navigate through simulators like Habitat [Savva et al., 2019] in photorealistic environments and large-scale three-dimensional point clouds such as the Gibson dataset [Xia et al., 2018]. In particular, we assume that the agent only has access to image observations and that the target location is also given as an image. The method is composed of three stages. First, the agent learns a visual representation that can distinguish between nearby and far-away pairs of points in a similar way

to [Savinov et al., 2018b]. The fundamental prior knowledge we use is that in most situations, an agent's representation of the world should not change very fast as it moves; but on the other hand, for most pairs of far-away points, the representations should be different. Next, the agent learns to explore, adding states to a memory buffer when their feature representations are dissimilar to any in the buffer. Finally, the agent trains itself to complete navigation tasks, using the buffer to shape the reward for the navigation policy. An important component of our model is that the agent uses a Scene Memory Buffer for both its policy and reward. In particular, the agent takes actions via a Transformer [Vaswani et al., 2017] applied to the memory buffer. Because our approach can be applied in situations where the practitioner has no control over the environment - and in particular, with no ability to give supervision or move to arbitrary positions in the environment - the method is general. We show that despite this generality, its final navigation policy outperforms other approaches.

Our contributions, in this chapter, are the following:

- We propose a novel three-stage algorithm for learning to navigate using only RGB vision without any external supervision or reward in photorealistic environments that simulates actual houses.
- We introduce several improvements to the exploration policy [Savinov et al., 2018b] such as conditioning on past memory and using discrete rewards.
- We evaluate our model and show that it outperforms all baselines on scenes from the Gibson dataset.

4.2 Related Work

Iteratively building a map of an environment to perform localization or navigation tasks has been extensively studied in robotics in the context of SLAM [Thrun, 2002]. Standard SLAM is composed of multiple hand-crafted modules to fit with the physical constraints of a robot [Mur-Artal et al., 2015].

Recently, several works have replaced components of SLAM with neural networks; for example, Chaplot et al. [2018] replace the localization module. Gupta et al. [2017a] propose a model composed of two successive modules, a mapper to build a latent world map, and a planner, that takes actions based on this map. The mapper does not have dedicated external rewards but the planner performs tasks associated with external rewards and back-propagates the resulting gradients to the mapper. This map has been further extended with image features [Gupta et al., 2017b] or with a dynamic structure [Avraham et al., 2019, Henriques and Vedaldi, 2018]. Other works replace SLAM entirely by deep models with no planning but explicit map-like or SLAM inspired memory structures [Khan et al., 2018, Oh et al., 2016, Parisotto and Salakhutdinov, 2018, Zhang et al., 2017]. Closer to our work, Kumar et al. [2018] use

human-made trajectories stored as sequences of feature representations of views, and Fang et al. [2019] show the potential of the Transformer layer [Vaswani et al., 2017] as a scene memory for navigating realistic environments. As opposed to these works, our model is trained with intrinsic reward only.

Alternatively, several work train deep models to solve a navigation task without explicit world representations. Mirowski et al. [2017] learn a navigation policy with a recurrent network in synthetic mazes, and later, in real-world data from Google Maps [Mirowski et al., 2018]. Similar to our work, they use a surrogate loss on loop closure to help the training of the model, but they use sparse external reward to guide its training. Similarly, Zhu et al. [2017] show the benefit of deep models on a localization task framed as finding an observation taken from the goal. Later, Yang et al. [2018] extend this to navigation to an object described only by its name.

Many works train agents to explore the world with an intrinsic reward [Chentanez et al., 2005, Pathak et al., 2017, Schmidhuber, 1991]. For example, the curiosity-driven reward of Pathak et al. [2017] encourages agents to move to states that are hard to predict. Of particular interest, Chen et al. [2018] propose a coverage reward that encourages the agent to explore every part of its latent map. This reward is quite general and benefits both exploration and navigation, but it does not directly optimize for navigation like ours.

Finally, our approach is most related to a recent line of research that uses multiple stages of learning to build a set or graph of scene observations [Eysenbach et al., 2019, Savinov et al., 2018a,b, Zhang et al., 2018]. Savinov et al. [2018a] internalize a landmark memory obtained from human trajectories. They store representations of the locations visited in the trajectories and build a navigation graph based on their similarities. Our work follows their self-supervised training of a reachability network R to distinguish between nearby observations, but we extend the self-supervision to both exploration and navigation. Savinov et al. [2018b] also use a curiosity-driven intrinsic reward based on a memory buffer. Our exploration phase follows an intrinsic reward inspired by their work, but we also use the memory buffer in our Transformer-based policy. Finally, Eysenbach et al. [2019] propose a method to learn an agent to explore and navigate an environment with intrinsic rewards. Their training follows the same sequence of steps as ours, with the exception that they clean the graph by testing existing edges and adding new ones and then learning to navigate on the graph, and not the environment. Instead, our agent trains itself to navigate the environment directly by shaping dense rewards from the memory buffer. It means that our agent can potentially learn more efficient navigation strategies not constrained to paths on the memory-graph.

4.3 Image-Goal Navigation without External Rewards

In this chapter, we simulate a realistic setting where an agent must learn to navigate in a 3D environment. We formulate this problem using the following assumptions:

- *No extrinsic reward.* We do not have control over the environment and thus cannot add extrinsic reward to guide the training of the agent.
- *No human guidance.* The environment is new and has never been explored. We do not have access to human trajectories or other forms of external information.
- *3d scan environments.* We focus on photo-realistic environments such as the ones in the Habitat platform.

We are interested in the capability of the agent to explore and navigate an environment and we report the following metrics to measure its success:

- *Coverage.* We measure the coverage of an environment by an agent by discretizing the map into C cells of the same size and counting the ratio p_t of visited cells c_t by the agent after t steps.
- *Image driven navigation.* We measure the capability of an agent to navigate the environment to an image target. That is: we give the agent an image observation from the location and we measure the number of steps it takes to reach the destination so that the agent's observation matches the image target, starting from the entry point of the map.

Finally, as a secondary goal, we are also interested in the robustness of an agent to limited sensor data. To that end, we focus on RGB inputs in this chapter. We do not use depth, gps coordinate or relative position as inputs.

4.4 A three-stage approach to Unsupervised Image-Goal Navigation

In this section, we describe our algorithm and its three stage training: first the agent learns a visual representation of the environment from random trajectories, then it learns to explore the environment to build a latent map, and finally it trains itself to navigate using the map. Each step has a module trained without external supervision.

4.4.1 Stage 1: Visual representation of the environment

As the agent moves around the environment, it receives data from its visual sensor, which in this work produces RGB images. From this first-person input, the agent builds a representation of its current location that should encode information to distinguish the current location from other locations, as well as give an idea of

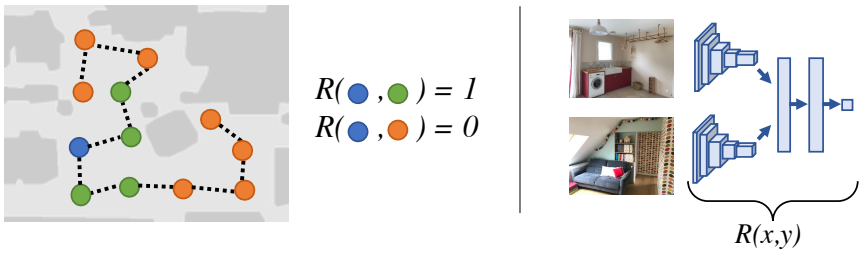


FIGURE 4.2: **Reachability network [Savinov et al., 2018b].** Given a set of observations made by an agent with a random walk policy **(left)**, we train the (local) reachability network R to distinguish between observations that are temporally near or distant. For a given observation (marked in blue), the nearest observations are in green and the distant ones in red. The reachability network **(right)** is a siamese network composed of a convolutional network followed by a fully-connected network.

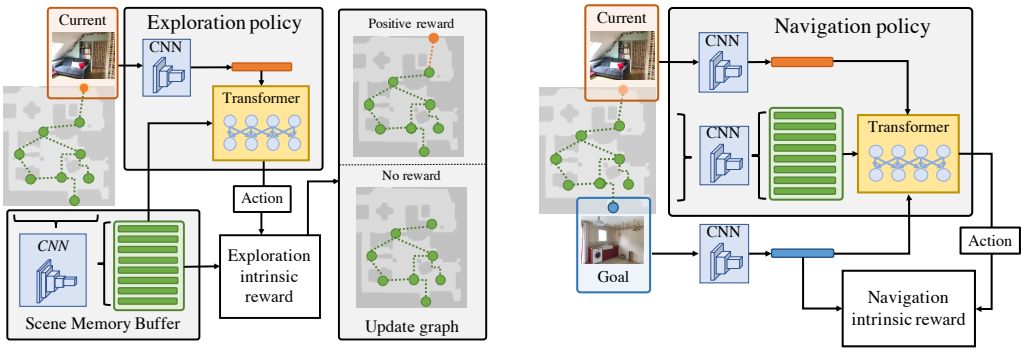


FIGURE 4.3: **Exploration and navigation stages.** The agent first learns to explore **(left)** the environment using a memory of visited regions for its policy and intrinsic reward. Next, the agent learns to navigate **(right)** using memory to set image oriented goals to itself and learn to navigate towards them.

the distance between locations. This is achieved by encouraging nearby locations to have similar representations while pushing distant locations to have different representations. However, in the absence of information about the agent position or a map, we do not have an explicit notion of distance between locations.

As in the previous chapter, we use the reachability network [Savinov et al., 2018b] to learn a visual representation of the environment. This stage is summarized in Fig. 4.2.

4.4.2 Stage 2: Learning to Explore

Once the agent can differentiate images of nearby locations from distant locations, it can explore and map the environment. In this section, we describe how to train our exploration module with a curiosity-driven intrinsic reward, which is our second training stage.

Exploration module. The agent explores the environment to dynamically build an internal map. At each step, this map and the current observation are used to plan an action that moves the agent toward unexplored regions. We model the internal

map as a memory that contains important past observations, and the agent takes actions by applying a Transformer on this memory buffer. This stage is shown in Fig 4.3 (left).

Episodic Memory. In a similar fashion than in the previous chapter, the agent has an Episodic Memory that stores some of its previous observations. At each time step t, the memory $\mathbf{M}_{t-1}$ stores an unstructured set of visual features. Storing every observation is not efficient and we follow the mechanism of Savinov *et al*. [Savinov et al., 2018b], explained in the previous chapter, to select which observation to store in the memory. The idea is to add only observations that are distant (with respect to the Reachability Network) from the current memory vectors. The memory is reset at the end of each episode.

Transformer on the memory. Again, as in the previous chapter, the navigation policy exploits the information contained in the memory through a transformer [Vaswani et al., 2017]. The details for this module are given in Section 3.4.2.

Intrinsic exploration reward. Intrinsic curiosity rewards the agent for exploring parts of the environment that looks unfamiliar to the agent. This reward is based on the agent's intrinsic representation of the environment. In our case, this representation is the memory and a positive reward is given if the current observation has been added to the memory, i.e.,

$$r_{\text{curiosity}}(t) = \alpha \mathbb{1}_{\{s(\mathbf{x}_t, \mathbf{M}_{t-1}) < \tau\}}. \tag{4.1}$$

This reward is a discrete version of the episodic curiosity [Savinov et al., 2018b]. Discretizing the reward removes the trivial solutions noticed in [Savinov et al., 2018b] where the agent stops in a location that gives a reward that is greater than any reachable location.

4.4.3 Stage 3: Learning to Navigate

In this section, we describe the third stage of our algorithm: the training of our navigation module. Every episode will start with an exploration phase where the exploration module builds an internal map of the environment. This is followed by a navigation phase that trains the navigation module to reach a goal sampled from the map. The internal map is also used for generating the intrinsic navigation reward. The trained navigation module does not need to follow the visited locations on the map — these are only used during training to shape the reward. In particular, the navigation policy can be more efficient than policies that plan over visited locations on the map at test time. This stage of the training is depicted in Fig 4.3 (right).

Building an internal map. In the exploration phase of an episode, an internal map of the environment is built by the exploration policy that is already trained in the previous stage. The exploration policy runs for T steps and fills the memory with visual representations of J_T locations. While the memory alone is sufficient for training the navigation module with sparse rewards, we also record the connectivity of those J_T locations to be leveraged in the dense-reward version of the training.

The path followed by the agent connects different vectors in the memory. We use this path to form a graph G_t on top the memory $\mathbf{M}_t$. More precisely, we keep track of the closest element $\mathbf{p}_t$ of the current observation $\mathbf{x}_t$ after updating the memory. Note that this means that $\mathbf{p}_t$ is equal to $\mathbf{x}_t$ if it is added to the memory. If $\mathbf{p}_t$ is different from $\mathbf{p}_{t-1}$, we add an edge $e = (\mathbf{p}_{t-1}, \mathbf{p}_t)$ to G_t. This results in a directed graph representing paths between the vectors of the memory.

Navigation module. The navigation module takes as an input the current observation $\mathbf{x}_t$ as well as a target observation $\mathbf{x}^*$. The module transforms these observations into features with a CNN, and concatenates the resulting features. We then apply a Transformer layer on top of this vector and the memory, resulting in a feature $\mathbf{h}_t$. We compute the feature $\mathbf{h}_t$ as follows:

$$\mathbf{c}_t = [\texttt{CNN}(\mathbf{x}_t), \texttt{CNN}(\mathbf{x}^*)], \tag{4.2}$$

$$\mathbf{e}_t = \texttt{LN}(\texttt{Att}(\mathbf{c}_t, \mathbf{M}_{t-1}) + \mathbf{c}_t), \tag{4.3}$$

$$\mathbf{h}_t = \texttt{LN}(\texttt{MLP}(\mathbf{e}_t) + \mathbf{e}_t). \tag{4.4}$$

Similar to the exploration module, the policy and value function are linear functions of a feature $\mathbf{h}_t$. Note that set of parameters for the attention modules for the exploration and navigation modules are different, but not the CNNs.

Memory based navigation reward. After the exploration phase of an episode, the navigation phase starts by setting a randomly selected element $\mathbf{m}_j$ of the memory as a goal to navigate towards. A positive intrinsic reward is given if the agent considers that it has reached the target location based on its reachability network, i.e.,

$$r_{\text{sparse navigation}}(t) = \beta \mathbb{1}_{R(\mathbf{x}_t, \mathbf{x}^*) > \tau}. \tag{4.5}$$

This is an intrinsic reward built solely on the capability of the agent to perceive if it has reached the goal sampled from its memory. However this reward is sparse and we propose to densify the reward by further exploiting the memory.

Dense intrinsic navigation reward. We leverage the graph G_t to form dense navigation reward by computing a graph based approximation of the distance to the goal. More precisely, at each time step t, we compute the shortest path between $\mathbf{p}_t$ and $\mathbf{x}^*$ in G_t and denote by l_t its length. We thus add a dense reward based on this

TABLE 4.1: Navigation performance as measured by the SPL metric for our method (Ours) and selected baselines on all considered environments.

	Adrian	Albert.	Arkan.	Ballou	Capist.	Goffs	Mosq.	Sanc.	Mean
Random	13.5	19.3	16.4	10.5	26.0	9.3	10.6	12.6	14.8
SPTM	25.5	23.5	20.2	9.7	38.6	9.3	16.9	10.1	19.2
Supervised	27.5	30.5	21.9	11.1	45.8	14.8	13.0	17.4	22.8
Ours-sparse	27.8	39.9	30.6	17.0	60.9	15.1	16.3	**32.9**	30.1
Ours-dense	**35.6**	**45.2**	**32.3**	**27.8**	**65.9**	**16.8**	**18.8**	24.5	**33.4**

distance as:

$$r_{\text{dense navigation}}(t) = \max\left(0, \min_{t'<t} l_{t'} - l_t\right). \tag{4.6}$$

Note that, since we update the graph G_t as we navigate the environment, this reward may change over time for a same target $\mathbf{x}^*$ and memory vector $\mathbf{p}_t$. Note that this bonus reward only absolute progress towards the goal and the total reward accumulated over an episode is equal to the length of the shortest path as estimated at the beginning of the episode. Overall, we use both the dense and sparse reward during the navigation phase.

4.5 Experimental Evaluation

We evaluate both the exploration and the navigation modules. We start by describing the data we use and providing technical details of our experimental setup.

4.5.1 The Gibson Dataset

For a realistic setup we perform all of our experiments on scenes taken from the Gibson dataset [Xia et al., 2018]. We run the simulations for these experiments inside the Habitat-sim framework [Savva et al., 2019]. We have selected a subset of eight scenes from the Gibson dataset, based on the quality of the 3d mesh, surfaces, and the number of floors, following the study presented in [Savva et al., 2019]. The scenes are fairly complex as they have 16 rooms on average spanning multiple floors. The action set contains three actions: moving forward by one meter, and turning right or left by 45 degrees. We only keep the RGB data, discarding the depth channel, and use images of size 160×120 pixels.

In this work, we make the assumption that the agent is always spawned in the same location of a scene. To achieve this, for each scene we manually select a starting position corresponding to the entrance door in the house.

4.5.2 Implementation Details

Visual Representation Learning. We implement the network R as a siamese network with resnet18 as the function g, and use a comparison function f composed of

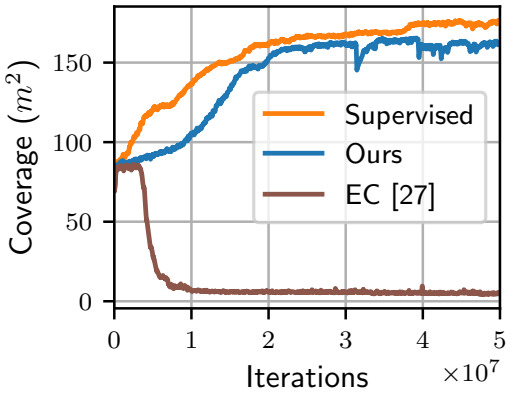


FIGURE 4.4: Performance of exploration policies as measured by the coverage metric in squared meters. We compare the performance of our model to a baseline (EC) and a supervised topline.

two hidden layers of dimension 512. For each scene, we sample 20 random trajectories of 20k steps. From each trajectory we extract $40k$ pairs, yielding a dataset of 800k image pairs. The maximal action distance for a positive pair is set to five steps. We train this network using SGD for 20 epochs with a batch size of 128, a learning rate of 0.1, a momentum of 0.9, a weight decay of 10^{-7} and no dropout. We do not share parameters between scenes.

Exploration and Navigation. For our CNN module, we use a network with 3 convolutional layers with kernels of size $[9, 7, 5]$, strides of size $[5, 4, 3]$ and number of channels $[32, 64, 128]$. For the attention on the memory, we use an attention with two heads, a hidden dimension of 64 and a feedforward network with a hidden dimension of 128. We train the policy using PPO, where each batch consists of 16 full episodes, each with 1000 steps. We run 4 PPO epochs, with γ set to 0.99, an entropy coefficient set to 0.01 and clipping of 0.1. We optimize the parameters using RMSprop with a learning rate of 10^{-4}, a weight decay of 10^{-7}, and parameters α and ϵ set to 0.98 and 10^{-5} respectively. For this model we do use dropout with $p = 0.1$, and a learning rate warm up phase of 300 steps. As with the R network, we do not share parameters between scenes.

4.5.3 Main Results

The main experiment in our evaluation checks how well our agent navigates to new test goals. After training itself to navigate to elements of the memory, the agent can be given a new goal feature as a target. In this experiment, we sampled 100 random locations from each scene, and saved the corresponding RGB observation and location. For each scene, and each target location, we first run 1000 steps of exploration to fill the memory and launch the navigation episode. The total navigation episode lasts for 1000 steps, and as soon as a goal is reached, a new goal is sampled.

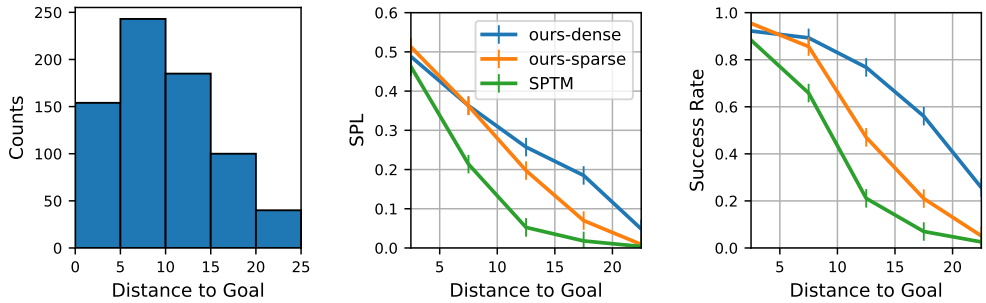


FIGURE 4.5: Navigation performance is broken down by physical shortest distance from start to goal location. **Left**: Histogram of distances from start to goal in our evaluation dataset. We see that most goals are located between 5m and 10m from the entrance. **Center**: Breakdown of the SPL metrics by distance. **Right**: Breakdown of the flat success rate by distance. We clearly see the advantage of using the dense rewards for learning to navigate to far-away locations.

Evaluation Metrics. We evaluate success by computing the number of targets that the agent reached successfully out of 100. The target is considered reached if the agent navigates to a distance of at most one meter from the target location. The first metric that we compute is the *success rate*, which simply corresponds to the fraction of goals that were reached within the allocated 1000 steps. Since this measure does not account for the length of the path taken, the second metric we report is the SPL metric [Anderson et al., 2018]. Let us assume that we have access to the length of the shortest path from the starting location to the goal i, computed by the simulator, that we denote l_i. If we write s_i the indicator of success as defined above, and d_i the metric length of the trajectory obtained with our algorithm, the SPL is defined as follows:

$$SPL = \frac{1}{N} \sum_{i=1}^{N} s_i \frac{l_i}{\max(l_i, d_i)}. \tag{4.7}$$

Baselines. In order to evaluate the quality of our navigation algorithm, we compare our model to three baselines: SPTM, Supervised and Random. We describe these baselines in more detail here.

First, we compare our algorithm to the Semi-Parametric Topological Memory [Savinov et al., 2018a]. In order to adapt SPTM to the environments used in our experiments, we train the action and edge prediction networks on them. For each scene, we train the networks for 300 epochs of 1000 batches each, with a batch size of 64. Samples in the batches are obtained from random trajectories that are sampled online. This number of training iterations amounts to approximately 90M steps in each environment - which is comparable to the number of steps used to train our method (exploration and navigation). Since SPTM requires an expert human-provided exploration trajectory, we use random exploration in place.

Second, we also compare against a feedforward policy trained with supervised rewards (*Supervised*). This policy is trained using RL, assuming that at each step the

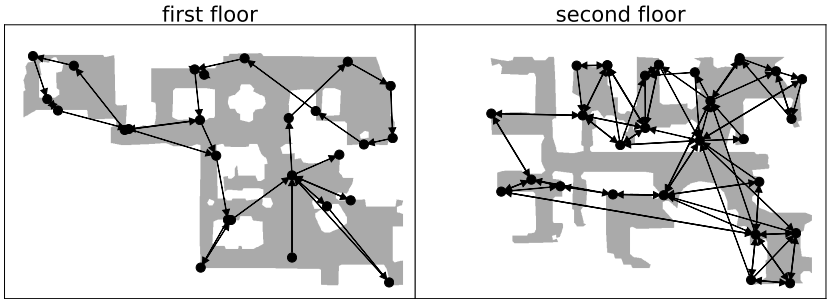


FIGURE 4.6: Visualization of the graph build during exploration in the Ballou environment.

distance $d(t)$ from the agent to the goal is known: $d(t) = \|p_{\text{agent}}(t) - p_{\text{goal}}(t)\|_2$. In that setup, the agent receives a reward of 10 if $d(t) < 1$ - which is equivalent to the success criterion defined above. Please note that this feedforward policy is trained on the same set of 100 goals that are used during evaluation. For reference, we provide the performance of random navigation.

Results. We run the evaluation for the baselines and our method with sparse or dense rewards and report the results for each scene in Table 4.1. There are a couple observations that we can make about this experiment. First of all, we see that our method outperforms all the baselines by a large margin on all of the scenes. Surprisingly, it even works better than the supervised agent which utilizes the location information - data to which our method has no access. However, this can be explained by our architectural choice of conditioning the navigation module on the memory of previously visited states. As opposed to that, the *Supervised* baseline is only a feedforward network, and has no representation of the past observations.

Second, the SPTM baseline performs poorly compared to our method with only little improvement over the randomly acting agent. This can be explained by the fact that SPTM only has access to a random exploration trajectory, therefore limiting the set of goals that it can ever reach. Moreover, SPTM restricts the navigation to its exploration graph, severely limiting the possible routes to the goal. In comparison, our method encourages the agent to reach the goal as fast as possible taking any possible route. Our dense reward does use the graph, but only as a guide that can be completely ignored if more optimal solutions exist.

Finally, we see that the dense reward generated using the graph G_t as described in Sec. 4.4.3 allows to train a better navigation policy, outperforming the sparse reward on most of the scenes. Indeed, for our agent, this dense reward corresponds to a discrete distance over the graph which leads to the goal if minimized. This effect is clearer when we measure the performance for different goal distances as shown in Fig. 4.5. The gap between the dense and sparse reward widens for far-away goals. This is likely because the graph G_t provides intermediate goals, helping a lot when the goal cannot be reached easily.

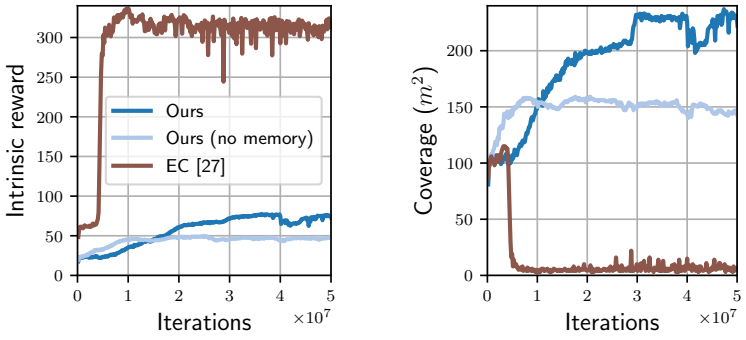


FIGURE 4.7: Ablation study of our exploration policy. We report the performance for our model, the EC baseline, as well as a variant of our model with no attention mechanism on the memory. **Left**: Evolution of the intrinsic bonus reward as a function of iterations. **Right**: Evolution of the coverage metric.

4.5.4 Analysis of Exploration

As mentioned in Sec. 4.4.3, the coverage obtained during the exploration stage is critical for the final navigation task. In this section, we want to evaluate the quality of this exploration stage alone.

Evaluation Metrics. The goal of the exploration stage is to train an agent to explore and map an environment without any form of supervision. For this experiment, we follow previous work and evaluate the quality of the exploration using a coverage metric. In order to define this metric, we discretize the environment using a grid, with cells of size 1×1 meter. At the end of the episode, we report the number of cells that were visited by the agent. Since the environments we consider can have multiple floors, we infer the floors in the environment. We do so by sampling random locations and keeping most frequent heights that are at least .5 meters away by doing non-maximal suppression. We then keep one coverage grid per floor.

Baselines. First, we compare our exploration module to Episodic Curiosity *(EC)* [Savinov et al., 2018b]. In that baseline - unlike our method - the policy has no dependency on the past. Another difference between our method and the *EC* baseline, is the nature of the intrinsic rewards. While the original bonus proposed in [Savinov et al., 2018b] was continuous, we propose to use a discretized version instead. Note that we cannot compare to EC on the navigation task in Sec. 4.5.3 because it does not provide means of navigation without supervision.

Second, we include a *Supervised* policy trained using the "oracle" reward, being the measure that we use for evaluation. In this case, we densely reward the agent every time a new cell is visited. Apart from using a different source of reward, all parameters for this model are taken the same as for our model.

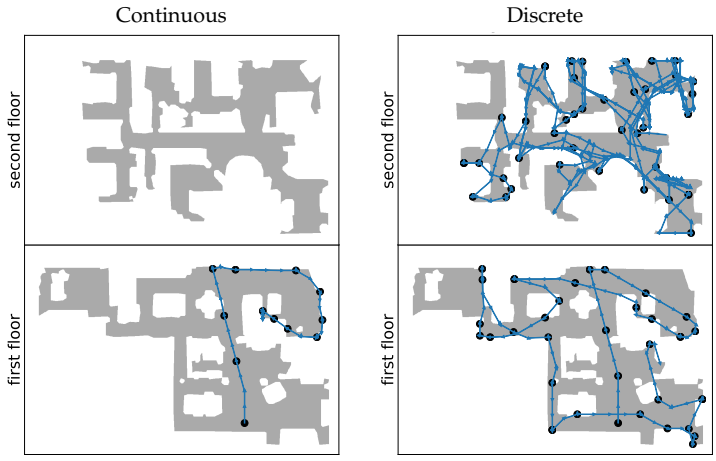


FIGURE 4.8: Visualization of the trajectories obtained with a policy trained with continuous and discrete bonus rewards.

Results. The performance evaluation of our method and the baselines during training is shown in Fig. 4.4. The coverage metric is averaged over the eight scenes. Our method performs comparable to the supervised agent, which can be considered as the upper bound as it directly optimizes the coverage metric. In Fig. 4.6, we show an example of exploration behavior learnt by our agent. The nodes of this graph are states added to the memory by the agent, and they are connected following the rule described in Sec. 4.4.3. We see that the agent has explored most of the house successfully and made connections consistent with its topology, which will assist the training of the navigation module. Surprisingly, we observed that the agent trained using vanilla EC does not learn a good exploration policy. We investigate the reason for this in the following experiment.

Ablation of the exploration model. In Savinov et al. [2018b], the authors propose a continuous curiosity reward: $r(t) = \alpha.(\beta - s(\mathbf{x}_t, \mathbf{M}_{t-1}))$. In this ablation study, we want to exhibit the importance of our improvements over [Savinov et al., 2018b], namely using a discrete bonus and an attention mechanism over the memory. To this end, we show the evolution of the intrinsic reward as well as of the coverage metric for three models on the Ballou environment. We compare our full model to the vanilla EC and an exploration policy such as ours but with no memory in Fig. 4.7.

We observe that using the continuous reward makes the agent find trivial maxima by exploiting the reward design. In that case the total episode reward converges to a value just below $N \times \tau$, where N is the number of steps - see Fig. 4.7 (Left). Despite the fact that the agent trains properly, and optimizes the reward, it does not work well when measured by the metric we care about, the coverage metric, as shown in Fig. 4.7 (Right). We provide a qualitative representation of the phenomenon, by visualizing the agent's path, as well as the spatial location of elements in the memory. We show these visualizations for both continuous and discrete rewards in Fig. 4.8. We see that the agent trained with discrete rewards manages to explore the scene

properly. However, when trained with continuous intrinsic rewards, the agent gets stuck in a specific subpart of the environment where it receives a continuous reward just below the threshold τ.

4.6 Conclusion

We have shown how to train an agent to perform goal-directed navigation in photorealistic environments without using any extrinsic rewards. Our agent trains in a purely self-supervised manner, only using RGB image observations. The model is composed of three interconnected components: one which learns visual representations, a second which explores the environment, and a third that teaches itself to navigate. We have shown that our self-supervised navigation models manage to navigate to novel and unseen test goals.

Chapter 5

Discovering and Reaching Goals Autonomously

In the previous chapter, we presented a method for learning to navigate in photo-realistic environments without position information and external rewards. It is also crucial to consider how the proposed method would scale to different type of tasks, and investigate the design choices that would harm the performance of the model on specific tasks. A probable cause of failure relies on the fact that the representation learning phase is performed only once, on trajectories collected by a random policy, at the first stage of the method. For this representation stage to be successful, the state space needs to be easily covered by random trajectories. If this is a reasonable assumption for classic navigation tasks, where the initial position of the agent is randomized, it is not necessarily the case for more complex environments. In continuous control tasks, such as controlling a robot arm to stack blocks, the state space cannot be covered uniformly with random trajectories, since stacking a block on top of another is a very rare event when performing random actions.

In this chapter, we investigate the more general problem of learning state-reaching policies that can reach any state in the environment, in the absence of external rewards. Building up on the ideas proposed in the previous chapters, we train a Reachability Network to predict the similarity between states, in a self-supervised fashion, on trajectories collected by a random policy. We use it to build a memory over the state space, and to shape a dense intrinsic reward signal for the navigation policy. Contrary to the method from chapter 4, the Reachability Network and the memory are updated along with the navigation policy training, and are thus improved continuously as the agent explores the environment. We apply our method to continuous control navigation and robotic manipulation tasks, where exploration is challenging.

5.1 Introduction

Standard applications of Reinforcement Learning (RL) methods rely on optimizing an objective based on a specific hand-crafted reward function. However, designing these functions for every specific behavior that we want the agent to learn is extremely time-consuming, and sometimes not even feasible. Moreover, optimizing a single reward function restricts the agent to learning a specific behavior [Zhu et al., 2017, Yarats et al., 2022], while it would be more powerful to learn agents that are able to execute diverse skills [Stulp and Sigaud, 2013]. Research efforts have thus been focusing on designing *unsupervised* agents, that are able to learn without external rewards at all. These agents are usually developed in a two-stage protocol: in the first stage, the agent can interact with the environment and acquire experience without external reward, and in the second stage, it is evaluated on human-designed tasks, with or without adaptation.

The resulting agents are most often trained to explore the environment, and then fine-tuned with few interactions to perform specific tasks. This adaptation stage still requires a hand-crafted reward function, and the fine-tuned agent is still able to perform only one particular task. Recently, some works made the link between unsupervised RL and the *goal-conditioned* RL paradigm: in the first stage, the agent would sample goals from its past experience and learn by attempting to reach them, and in the second stage, it would be given user-specified goals to be evaluated on. This allows for a training protocol where the agent does not need any adaptation to the evaluation task, as the goals are specified in the same way at train and test time, and can execute several behaviors, simply by giving it diverse goals. However, this paradigm induces several challenges.

First, the agent must have a set of intrinsic goals on which to train. These goals must be diverse enough, so that the agent learns several behaviors, but not too hard, to make learning feasible. Previous works rely on past states and sample goals from the replay buffer [Nair et al., 2018, Pong et al., 2020], use generative models to create intrinsic goals [Warde-Farley et al., 2018], or exploit an explorer policy to generate novel goals [Mendonca et al., 2021]. Second, in the absence of supervision, the agent must have an intrinsic reward function shaped for goal-reaching. Even though this function can be easily hand-crafted in some simple cases like mazes, it can be infeasible in complex real-world environments. For instance, with high-dimensional inputs like images or complex control tasks such as moving humanoids, assessing whether a goal has been achieved can be infeasible.

In a recent line of study, some methods [Hartikainen et al., 2019, Venkattaramanujam et al., 2019] proposed to tackle these two challenges by learning a distance network in the state space. This network can be directly used to shape the intrinsic reward function towards reaching a goal, and can also be used for sampling training goals in a clever way. These approaches offer a simple and interpretable way of tackling

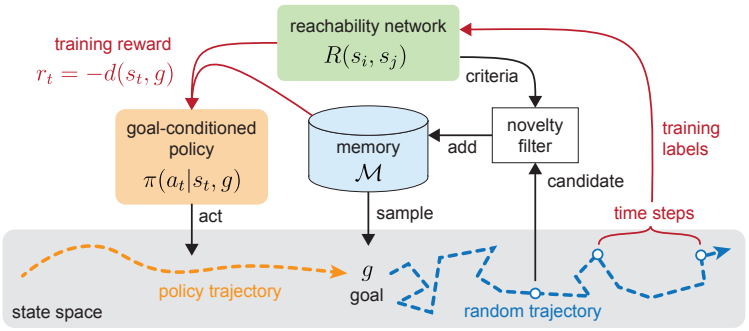


FIGURE 5.1: **Overview of our method.** The agent performs two consecutive stages. In the first stage (orange arrow), it samples a goal g from the memory $\mathcal{M}$ and executes a trajectory with policy π towards a goal with reward $r_t = -d(s_t, g)$. In the second stage (blue arrow), the agent performs random steps in order to discover novel goals to be added to the memory $\mathcal{M}$, and to generate training data for the Reachability Network R.

the unsupervised goal-conditioned problem, as the quality of the learned policy depends mostly on the quality of the distance function.

A possible flaw of these methods is that learning of the distance function and the goal-conditioned policy are intrinsically tied. Indeed, the distance is learned on samples generated by the policy, and the policy is trained using distance-based rewards. In contrast to these works, we propose to learn the distance function independently from the goal-conditioned policy, by learning from randomly generated trajectories in the environment. This self-supervised process, inspired by Savinov et al. [2018b], results in more stability and interpretability when training the distance function jointly with the policy.

5.2 Related Work

In its original formulation, goal-conditioned reinforcement learning was tackled by several methods [Kaelbling, 1993, Schaul et al., 2015, Andrychowicz et al., 2017, Nasiriany et al., 2019]. The policy learning process is supervised in these works: the set of evaluation goals is available at train time as well as a shaped reward function that guides the agent to the goal.

Several works propose solutions for generating goals automatically when training goal-conditioned policies [Ecoffet et al., 2019, Pitis et al., 2020], including self-play [Sukhbaatar et al., 2018b,a, OpenAI et al., 2021], where an agent learns to reach goals with an adversarial objective and a second agent that proposes goals of increasing difficulty. In the same spirit, Campero et al. [2020] presents a student-teacher policy trained in a single module with an adversarial learning loss. These methods assume access to a hand-crafted goal achievement function and therefore require supervision.

In a recent line of research, some works [Nair et al., 2018, Pong et al., 2020, Warde-Farley et al., 2018, Mendonca et al., 2021, Gehring et al., 2021] focused on learning

goal-conditioned policies in an unsupervised fashion. In these works, the objective is to train general agents that can reach any goal state in the environment without any supervision (reward, goal-reaching function) at train time. There are two challenges that these methods have to overcome: first, how to generate goals at train time, and second how to assess whether a goal was reached or not.

To tackle these challenges, Warde-Farley et al. [2018] propose to learn a goal achievement reward function jointly with the goal-conditioned policy with a mutual information objective. Nair et al. [2018] train a variational auto-encoder and generate goals in its latent space while using the Euclidean distance in this space to compute a dense reward function. Pong et al. [2020] improves this approach by "skewing" the set of goals to encourage exploration. More recently, Mendonca et al. [2021] presented a model-based method, composed of an explorer, which proposes novel goals in the latent space and an achiever that learns to reach these goals.

Closer to our work, Venkattaramanujam et al. [2019] and Hartikainen et al. [2019] learn a distance function in the state space jointly with the goal-conditioned policy. This distance function is used to compute rewards for the policy, but also to sample goals in clever ways: by encouraging far-away goals [Hartikainen et al., 2019] or by sampling goals of intermediate difficulty [Venkattaramanujam et al., 2019]. Our work extends this line of research with a new way of learning the distance function: independently from the policy by training it on trajectories sampled by a random policy at every stage of the policy learning process.

5.3 Problem Formulation

In the classical RL setup, the agent observes a state $s_t \in \mathcal{S}$ at time t, selects action a_t according to a policy $\pi(a_t|s_t)$ and receives a reward $r_t \in \mathbb{R}$. The agent learns by maximizing the cumulative return $\sum_{t=1}^{T} r_t$ where T is the episode length. This classical RL setup relies on the design of a hand-crafted reward function to solve a specific task.

In this work, however, we are interested in learning agents that have diverse behaviors, and are not limited to mastering a single task. Given a set of target states $G_{eval} \subseteq \mathcal{S}$, the agent is evaluated on its ability to achieve these goals. We assume G_{eval} is not known during training, so our objective is to learn a goal-conditioned policy $\pi(a_t|s_t, g)$ capable of reaching any goal state $g \in \mathcal{S}$.

5.4 Method

We propose a novel method to tackle the problem of unsupervised training of a goal-conditioned agent. Our method comprises three components as shown in Figure 5.1. The first component is a reachability network (RNet) that learns to predict the similarity between any given two states. We train this RNet using random walk

Algorithm 1 Training of our method

Initialize: random buffer $\mathcal{B} \leftarrow \emptyset$, goal memory $\mathcal{M} \leftarrow \emptyset$
Warm-up: collect random trajectories and store them in $\mathcal{B}$
for each training stage **do**
  Train the reachability network R on trajectories from $\mathcal{B}$
  Update the goal memory $\mathcal{M}$ with states from $\mathcal{B}$ using R as a criteria.
  for each episode **do**
    Sample a goal g from the memory $\mathcal{M}$
    Run the policy $\pi(a_t|s_t, g)$ for T steps and train it with rewards $r_t = -d(s_t, g)$
    Take N random steps and add that trajectory to $\mathcal{B}$
  end for
end for

trajectories generated by a random policy. The second component is a goal memory $\mathcal{M}$ that stores previously seen states that are diverse. To ensure this diversity, we employ the RNet as a criterion to avoid adding similar states to $\mathcal{M}$. The last component is a goal-conditioned policy $\pi(a_t|s_t, g)$ that is trained to reach goals g sampled from $\mathcal{M}$. The policy is trained with rewards $r_t = -d(s_t, g)$ using a distance metric. Since we do not have access to a hand-crafted distance metric, we propose two ways to approximate it using the RNet and $\mathcal{M}$.

The important aspect of our method is continued exploration of an environment that allows the agent to discover new areas. This exploration is accompanied by gradual progression from easy goals to increasingly difficult goals, which acts as curriculum for better training of the policy. This is achieved by first starting an episode with the policy π acting towards a goal sampled $g \in \mathcal{M}$ as shown by the orange trajectory in Figure 5.1. Once the policy trajectory ends, we start a random walk (blue trajectory), which is likely to cover areas that are beyond the reach of the current policy. As both RNet and the goal memory $\mathcal{M}$ are updated by those random trajectories, the agent will be discovering new goals that are sufficiently different from existing ones. This, in turn, will train the policy to reach increasingly further away goals. Since this discovery of increasingly harder goals need to continue throughout the training, all three components are continuously updated as shown in Figure 1. Let us now describe each component in more detail and how they interact with each other.

5.4.1 Reachability Network

As in previous chapters, we approximate the distance between states in the environment by learning a Reachability Network [Savinov et al., 2018b]. During training, we collect random trajectories $(s_1^a, ..., s_N^a)$ in *random buffer* $\mathcal{B}$ where a is a trajectory index. We define a reachability label y_{ij}^{ab} for each pair of observations (s_i^a, s_j^b) that

depends on their distance in the sequence and if a, b are the same trajectory. More precisely, we recall that,

$$y_{ij}^{ab} = \begin{cases} 1 & \text{if } a = b \text{ and } |i-j| \leq \tau_{\text{reach}}, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } 1 \leq i, j \leq T \tag{5.1}$$

where the *reachability threshold* τ_{reach} is a hyperparameter. We train a siamese neural network R, the RNet, to predict the reachability label y_{ij}^{ab} from a pair of observations (s_i^a, s_j^b) in $\mathcal{B}$. The RNet consists of an embedding network g, and a fully-connected network f to compare the embeddings, *i.e.*,

$$R(s_i^a, s_j^b) = \sigma\left[f(g(s_i^a), g(s_j^b))\right], \tag{5.2}$$

where σ is a sigmoid function. A higher R value will indicate that two states easily reachable with random walk, so can be considered close in the environment. The training of the RNet is self-supervised, as the supervised labels needed to train the network are automatically generated.

5.4.2 Goal Memory

Since the set of evaluation goals G_{eval} is not known at train time, the agent needs to come up with goals on which to train on for itself. One possible solution would be to sample goals from states that the agent has previously seen (*i.e.*, from the replay buffer). The main downfall of this solution is that there is no incentive in discovering novel states, and the agent would potentially learn to reach only a small proportion of the state space. Instead, we propose to incrementally build a set of intrinsic goals, or *goal memory* $\mathcal{M}$ on which the agent trains on. We use the random trajectories $\mathcal{B}$ to build $\mathcal{M}$ rather than the policy trajectories because the random walks are more likely to contain novel states.

Memory filtering: The states in $\mathcal{B}$ have to go through a filter: a state is added to the memory $\mathcal{M}$ only if it is distant enough from all other goals in $\mathcal{M}$. More precisely, a state $s \in \mathcal{B}$ is added to $\mathcal{M}$ if and only if $\forall m \in \mathcal{M}, R(s, m) < \tau_{\text{memory}}$, where the *memory threshold* τ_{memory} is a hyperparameter. As presented in previous chapters, this filtering avoids redundancy by preventing similar states to be added to the memory. It also has a balancing effect because it limits the number of goals that can be added from a certain area even if it is visited by the agent many times. This is especially important if episodes always start from the same initial state s_0 and most samples in $\mathcal{B}$ are concentrated around s_0.

Weighted goal sampling: Optionally, the same balancing effect as the filtering can be achieved by giving appropriate sampling weights to the goals. Let us consider a state s_i in the memory $\mathcal{M}$ and its *reachable area* defined as $A_i = \{s_j | R(s_i, s_j) > 0\}$.

The filtering ensures that there is only one state in $\mathcal{M}$ from A_i, so the probability of sampling a goal from A_i is the same as any other reachable area. While this keeps the memory more balanced and evenly distributed, it also limits the number of goals we train on, and can make the policy overfit to these goals. So instead, we can use a weighting scheme that will allow k states from A_i to be added to $\mathcal{M}$. In turn, goals from A_i are sampled with a probability proportional to $1/k$ so each reachable area is still equally represented in the sampled goals.

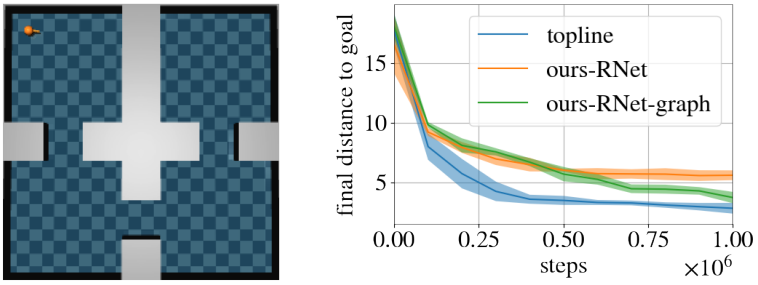
5.4.3 Distance function for policy training

When an appropriate distance function d in the state space is available, designing a reward function to learn a goal-conditioned policy towards a goal g is straightforward, by setting $r_t = -d(s_t, g)$. However, in realistic environments with complex high-dimensional inputs like images, designing a distance between states is generally not feasible. We therefore assume that we do not have access to any kind of distance in the state space *a priori*. Instead, we propose two different ways to construct a distance function.

RNet distance: The RNet predicts the similarity between s_i and s_j so we can directly use it as a distance metric. However, the RNet has a sigmoid function σ for binary classification. We remove it, and define a distance metric as $d(s_i, s_j) = -f(e(s_i), e(s_j))$.

Graph distance: Since the RNet is trained to determine whether states are close to each other or not, there is no guarantee that the aforementioned RNet distance will have good global properties. Instead of using directly the RNet, we construct an unweighted graph on the memory, whose nodes are the memory states, and edges are between nodes that have high reachability score. More precisely, the memory graph contains an edge between states s_i and s_j if and only if $R(s_i, s_j) > \tau_{\text{graph}}$, where τ_{graph}, the *graph threshold*, is a hyperparameter.

Using this memory graph, we can easily derive a distance function d between any pair of states in $\mathcal{M}$ by computing the length of the shortest path in this graph, providing that the graph is connected. Moreover, we can extend this distance to all states in the state space $\mathcal{S}$ by computing, for a state $s \in \mathcal{S}$, its closest node n_s in the memory *w.r.t* the RNet *i.e.* $n_s = \underset{m \in \mathcal{M}}{\text{argmax}} \quad R(s, m)$. The distance d between two states in the state space becomes the length of the shortest path between their respective closest nodes in the graph. This process allows for propagating the good local properties of the RNet in order to get a well-shaped distance function for faraway states.



(A) Maze environment (B) Evaluation performance

FIGURE 5.2: Performance on the Maze environment.

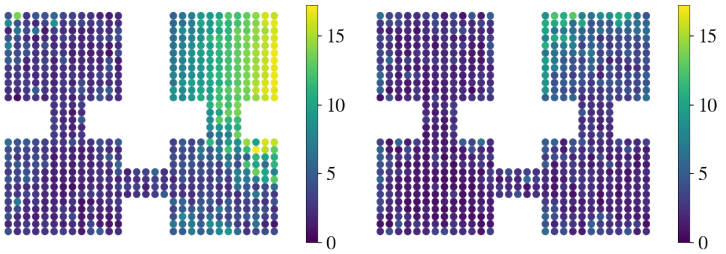
5.5 Experiments

We test our method on two continuous control tasks: maze navigation and robotic arm manipulation. For the policy training, we use Soft actor-critic [Haarnoja et al., 2018]. The policy training steps are limited to 1M steps. For each model, we report mean and standard deviation over 5 random seeds.

5.5.1 Maze environment

We first evaluate our model on a simple maze environment [Kanagawa, 2021] with four rooms, as shown in Figure 5.2a. The point agent always starts at the same position, in the top left corner, and can move in the maze by performing actions in a continuous space. Here, the observations are state-based: they contain the agent's position, direction and velocity. We generate an evaluation set of 500 goals sampled at random in the environment, and we assess the performance of the agent by measuring the distance between its final position and the goal position.

We compare the following three models: **(i) topline** the fully-supervised topline: the agent samples goal uniformly at random in the evaluation set at train time and uses the Euclidean distance in the maze for dense reward computation, **(ii) ours-RNet** our method with rewards computed directly from the RNet distance, and **(iii) ours-RNet-graph** our method with rewards computed as the shortest path in the memory graph.

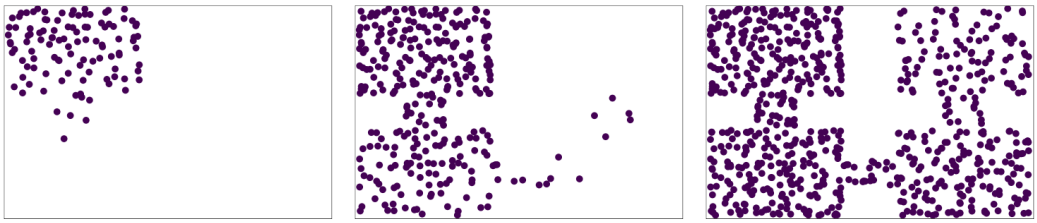


(A) final distance to goal - **ours-RNet** (B) final distance to goal - **ours-RNet-graph**

FIGURE 5.3: Heatmaps of final distance to goal for (A) **ours-RNet** and (B) **ours-RNet-graph**

Evaluating the performance of the model We first show the evaluation performance of the 3 models during training in Figure 5.2b. We see that the unsupervised model **ours-RNet** performs significantly worse than the supervised one (**topline**), but that the gap is largely reduced by using graph-based rewards (**ours-RNet-graph**).

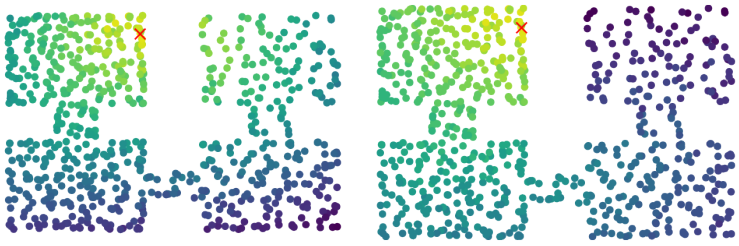
We then perform a qualitative evaluation of the final distance to the goal for the two different rewards in Figure 5.3a and 5.3b. It shows that the policy trained with graph rewards is able to reach almost all goals in the environment, including the ones that are in the fourth room, while the model with the RNet rewards is not able to achieve them.



(A) step = 0 (B) step = 200k (C) step = 1M

FIGURE 5.4: Memory of the **ours-RNet-graph** model during training.

Analysis of the Reachability Network In order to understand how our unsupervised agent learns to discover and achieve novel goals, we visualize the memory $\mathcal{M}$ at several steps of training in Figure 5.4. We see that throughout training, the memory gets filled with vectors that are further and further away from the initial state, and that after 1M steps, it is well distributed in the state space.

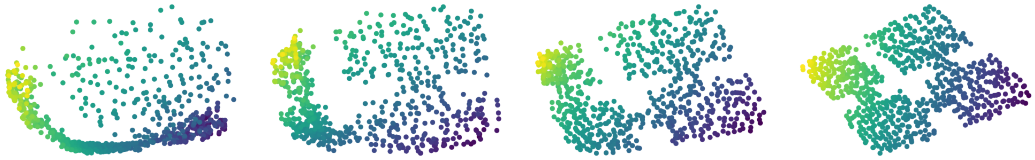


(A) RNet distance (B) Graph distance

FIGURE 5.5: Visualization of rewards computed with the RNet (a) and graph (b) distances.

We then compare the shape of the reward computed from the RNet reward function, and using the graph memory in Figure 5.5. We see that the graph-based reward is necessary in order to have a reward function that takes into account the true dynamics of the environment. Indeed, the shape of the reward based on the RNet distance will make the agent bump into the wall to go from room 1 to room 4, while the graph-based reward has a smooth shape that follows the room order.

Finally, we visualize the features learned by the RNet embedding for the **ours-RNet-graph** model in Figure 5.6. We generate a set of points in the environment, and



(A) step = 0 (B) step = 200k (C) step = 500k (D) step = 1M

FIGURE 5.6: Visualization of the RNet embedding $g(s)$ for the **ours-RNet-graph** model. We compute the 2D-PCA for a set of points sampled in the environment at several steps of training.

compute their features with the embedding part of the RNet. We then reduce the dimension of the embeddings by computing the 2D-PCA and visualize the resulting projection. We see that throughout training, the RNet learns to disentangle observations in consecutive rooms. After 1M steps, the embeddings describe well the entire state space.

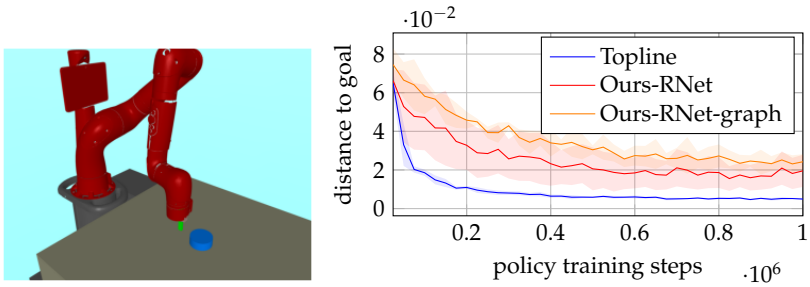


FIGURE 5.7: **Left:** Pusher environment. The robot arm has to move the puck to a specified location. **Right:** Learning curve on the Pusher-Vec environment showing the distance to the goal.

5.5.2 Pusher Task

Next, we apply our method to a realistic robotic environment from Nair et al. [2018]. In particular, we use the *Pusher* task shown in Figure 5.7 (left) where a robot arm (red) needs to push a puck (blue) to a specified location on a table. The performance of this task is measured by the final Euclidean distance $d^*(s_t, g)$ between the puck and its target location. See Nair et al. [2018] for more details about the environment.

We experiment with two versions of this environment: *Pusher-RGB* where observations are RGB images, and *Pusher-Vec* where observations are a vector containing the hand and puck locations. Even with the vector observations, this task is challenging because our method is not given any information particular to this task such as the importance of moving the puck, or the distance between the puck and its target location. Instead, our method has to learn all this solely from interacting with the environment without any external reward.

We start with *Pusher-Vec* and compare our model against a supervised *topline* that is rewarded by the oracle distance $r_t = -d^*(s_t, g)$. As shown in Figure 5.7 (right), the topline learns quickly to move the puck closer to the target location. Our method

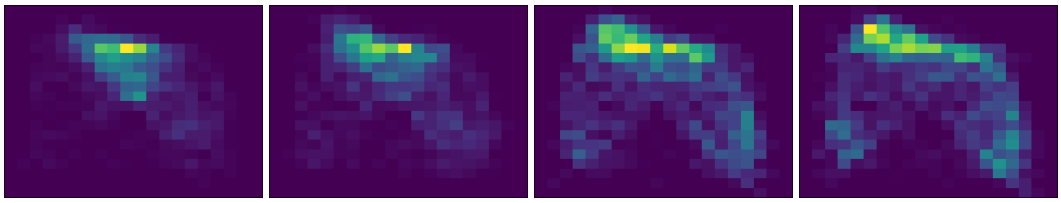
also learns to move the puck to the target location, with the RNet reward working slightly better than the graph reward. For all remaining experiments, we use the RNet reward.

Our method does not know that moving the puck is the goal. Instead, it tries to match the current state to the target state according to its own distance metric. In fact, we can see the effect of this if we look at the hand distance. The target observation contains a hand location, but it has no effect on the evaluation metric. While the topline has no incentive to match the hand location, our method also learned to match the hand position (hand distance 0.08 vs 0.02), which can be a useful skill in general.

Method	Goal distance ($\cdot 10^{-2}$)
Skew-fit	4.9
LEXA	2.3
Topline	1.34 ± 1.09
Our method	4.11 ± 0.53

TABLE 5.1: The performance on Pusher-RGB

We compare our method against two existing baselines: Skew-Fit [Pong et al., 2020] and LEXA [Mendonca et al., 2021] on the Pusher-RGB task in Table 5.1. We evaluated our model on the same set of evaluation goals than these methods, and reported the performance available in their paper. We see that our method performs better than the existing model-free method Skew-Fit, but is still far behind the model-based model LEXA. One possible explanation for this bad performance is the quality of the RNet on RGB, which could be greatly improved to match the performance of the model on Pusher-Vec.



(A) step=160k (B) step=331k (C) step=673k (D) step=995k

FIGURE 5.8: Distribution of the puck locations in random buffer $\mathcal{B}$ at different stages of training.

Importance of keep updating the reachability network One important aspect of our method is that the reachability network is kept updated throughout the training. Let us demonstrate why that is important. Figure 5.8 shows the puck location distribution within the collected random trajectories at different stages of training.

Early in the training, the goals in the memory are concentrated around the initial state s_0, so a policy trajectory does not go far from the initial state. Since a random trajectory starts from the last state of a policy trajectory, it is also biased towards the

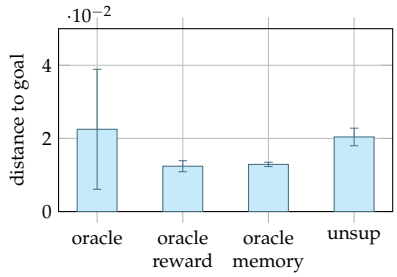
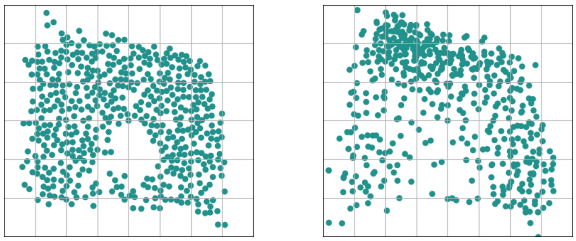


FIGURE 5.9: Ablation on Pusher-Vec where the reachability network is replaced by the oracle puck distance. We compare the performance of the unsupervised model against the three oracle variations.

initial state, which can be seen in Figure 5.8a. Note that a similar thing will happen if random trajectories are started from s_0 instead.

However, as training progress, more states are added to the memory that are farther away from the initial state s_0. So a random trajectory is more likely to start from a state that is less biased towards s_0 and will be more evenly distributed. This, in turn, allows the reachability network to be trained on more evenly distributed samples, and even farther away states to be added to the memory. We can see this trend towards more uniformity in random trajectories shown in Figure 5.8.



(A) Oracle distance (B) Reachability network

FIGURE 5.10: Puck positions of the states stored in the memory for the oracle model (A) and the unsupervised one (B).

Effectiveness of the Reachability Network. The RNet $R(s_i, s_j)$ is an integral part of our method and it has to function well for our method to succeed. Here, we test the quality of the RNet with an ablation study. We replaced the RNet in our method with the oracle distance metric $d^*(s_i, s_j)$. Since our method rely on $R(s_i, s_j)$ in two places, when giving a reward to the policy and as a criterion for adding to the memory, we ended up with three oracle variations:

- **oracle-reward:** the oracle distance is used as a reward $r_t = -d^*(s_t, g)$ for the policy.
- **oracle-memory:** the oracle distance is used as a criterion for adding a state to the memory. A state s is added when $d^*(s, m) > 0.0075$ for $\forall m \in \mathcal{M}$.
- **oracle:** the oracle distance is used for both, replacing the RNet completely.

The results are shown in Figure 5.9 where we do not see much difference between our method and its oracle variations in terms of performance. This indicates that the RNet is as effective as the oracle distance metric.

However, the RNet predictions differ from the oracle distance, which can be seen from the states stored in the memory. Figure 5.10 shows the puck positions of the states that are stored in the memory. The oracle distance is based on the exact puck position, so the memory looks more uniform and evenly spaced. This is not true when the RNet is used, which takes into account the hand position as well. It also depends on the agent's experience, so the memory is more biased towards puck locations that are relatively easy to achieve.

Memory Filter	Threshold τ_{memory}	Goal weighting	Goal distance ($\cdot 10^{-2}$)
✓	0.5	×	2.05 $\pm$ 0.21
✓	0.73	×	2.47 $\pm$ 0.84
✓	0.95	×	5.75 $\pm$ 2.27
×	-	×	6.04 $\pm$ 2.32
✓	0.95	✓	1.77 $\pm$ 0.21

TABLE 5.2: Effect of the memory filtering and goal weighting on the Pusher-Vec performance.

The Effect of The Memory Filtering. As discussed in Sec. 5.4.2, one goal of filtering states added to the memory is to make the memory more evenly distributed over explored areas. To measure the effect of the filtering, we trained several variations of our method. First, we made the filter loose by increasing its threshold τ_{memory} from its default value of 0.5, to allow more states to be added to the memory. This clearly had a negative impact on the performance as shown in the top 3 rows of Table 5.2. Removing the filter altogether, as shown in the 4th row, leads to the worst performance. Next, we use the weighting scheme from Sec. 5.4.2 that take on the role of the filtering and keep the goals well balanced. When it is applied to a loose filtering threshold of 0.95, it improved the performance and actually gave the best performance we obtained on Pusher-Vec. The reason why the weighting worked better than the filtering alone is probably because it had more goals that made over-fitting less likely.

5.6 Conclusion

In this chapter, we proposed a novel method for training a goal-conditioned agent without any external supervision. The method utilizes random walk to learn the similarity between states, which then is used to build a goal memory that is diverse and well-balanced. On the maze task, we showed that our method can discover increasingly difficult goals. In this task the graph distance built on the memory worked better than the similarity metric, mainly because the global structure of the

rooms is more important than the local dynamics. The Pusher task showed that our method can learn to manipulate an object without any external supervision.

Chapter 6

Learning State-Reaching Policies Offline

In online reinforcement learning, the agent interacts with the environment in order to collect experience to improve its policy. However, in some cases, this live interaction can be impractical, expensive, or even unsafe. In robotics for example, performing actions on a physical robot can be very slow, and applying online RL algorithms, that require millions of environment interactions, for such tasks is not scalable. Moreover, it can be unsafe to train a policy from scratch on such robots, as it can cause damage to the robot or its environment. The same argument applies to other types of tasks, like autonomous driving, where deploying a learning agent in the real world must be done with extreme caution. Recently, a line of study has emerged that uses pre-collected datasets of trajectories, and trains policies offline, *i.e.*, without any additional interaction in the environment [Yarats et al., 2022, Lambert et al., 2022]. This setup is particularly useful in robotics, where data collection is extremely time-consuming: disentangling data collection and policy learning in this context allows for faster policy iteration. Leveraging such datasets to learn multi-task policies has been done from the state-reaching perspective [Chebotar et al., 2021, Yang et al., 2022, Li et al., 2022a], by training agents to reach any state in the dataset. However, such methods either rely heavily on human supervision [Yang et al., 2022, Li et al., 2022a], or are subject to the pitfall of learning from sparse rewards [Chebotar et al., 2021].

In this chapter, we introduce a self-supervised learning stage in order to shape a dense reward function for training a goal-conditioned policy offline. This learning stage aims to learn the structure and dynamics of the environment without human intervention or prior knowledge. We evaluate our method on complex continuous control tasks, and compare it to previous state-of-the-art offline approaches [Chebotar et al., 2021, Andrychowicz et al., 2017]. We show that, contrary to prior work that uses datasets collected with a policy trained with supervised rewards [Chebotar et al., 2021], our method allows for learning goal-conditioned policies even from datasets of poor quality, *e.g.*, containing trajectories sampled with a random policy.

Our method is thus the first to learn goal-conditioned policies from offline datasets without any supervision, as it does not require any hand-crafted reward function at any stage: data collection, policy training and evaluation.

6.1 Introduction

A recent line of study has emerged that uses pre-collected datasets of trajectories and trains policies offline (*i.e.*, without additional interactions with the environment) [Yarats et al., 2022, Lambert et al., 2022]. More precisely, given a dataset of reward-free trajectories and a reward function designed to solve a specific task, the agent learns offline by relabeling the transitions in the dataset with the reward function. This setting is particularly relevant in robotics, where data collection is extremely time-consuming: disentangling data collection and policy learning in this context allows for faster policy iteration. However, it would require designing one specific reward function and learning one policy for each individual task.

An important question to scale offline robot learning is therefore to find ways of learning multi-task policies from already collected datasets. Recent works [Chebotar et al., 2021, Yang et al., 2022, Li et al., 2022a], have targeted this problem from a goal-conditioned perspective: given a dataset of previously collected trajectories, the objective is to learn a goal-oriented agent that can reach any state in the dataset. The advantages of this formulation are two-fold: first, it makes it easy to interpret skills, and second it does not require any adaptation at test time. Making this framework unsupervised requires to break free from hand-crafted rewards, as proposed by Chebotar et al. [2021], where they learn goal-conditioned policies offline through hindsight relabeling [Andrychowicz et al., 2017]. However, their approach is subject to the pitfall of learning from sparse rewards, and can be inefficient in long-horizon tasks.

In this work, we present a self-supervised reward shaping method that enables building an offline dataset with dense rewards. To this end, we develop a self-supervised learning phase that aims at learning the structure and dynamics of the environment before training the policy. During this phase, we: (i) train a reachability network [Savinov et al., 2018b] to estimate the local distance in the state space $\mathcal{S}$, then (ii) extract a set of representative states that covers $\mathcal{S}$, and finally (iii) build a graph on this set to approximate the global distance in $\mathcal{S}$. When training the goal-conditioned policy, we use the graph in two ways: to compute rewards through shortest path distance, and to create transitions of intermediate difficulty on the path to the goal.

We evaluate our method on complex continuous control tasks, and compare it to previous state-of-the-art offline [Chebotar et al., 2021, Andrychowicz et al., 2017] approaches. We show that our graph-based reward method learns good goal-conditioned policies by leveraging transitions from a dataset of past experience with

neither any additional interactions with the environment nor manually-designed rewards. Moreover, we show that, contrary to prior work that uses datasets collected with a policy trained with supervised rewards [Chebotar et al., 2021], our method allows for learning goal-conditioned policies even from datasets of poor quality, *e.g.*, containing trajectories sampled with a random policy. Our work is thus the first to learn goal-conditioned policies from offline datasets without any supervision, as it does not require any hand-crafted reward function at any stage: data collection, policy training and evaluation.

6.2 Related Work

Goal-conditioned RL. In its original formulation, goal-conditioned reinforcement learning was tackled by several methods [Kaelbling, 1993, Schaul et al., 2015, Andrychowicz et al., 2017, Nasiriany et al., 2019]. The policy learning process is supervised in these works: the set of evaluation goals is available at train time as well as a reward function that guides the agent to the goal. Several works propose solutions for generating goals automatically when training goal-conditioned policies, including self-play [Sukhbaatar et al., 2018b,a, OpenAI et al., 2021], and adversarial student-teacher policies [Campero et al., 2020]. A recent line of research [Warde-Farley et al., 2018, Nair et al., 2018, Ecoffet et al., 2019, Pong et al., 2020, Venkattaramanujam et al., 2019, Hartikainen et al., 2019, Mendonca et al., 2021, Mezghani et al., 2022a] focuses on learning goal-conditioned policies in an unsupervised fashion. The objective is to train general agents that can reach any goal state in the environment without any supervision (reward, goal-reaching function) at train time. In particular, Mendonca et al. [2021] trains a model-based agent that learns to discover novel goals with an explorer model, and reach them with an achiever policy via imagined rollouts.

Offline RL. The data collection technique is an important aspect when studying the training of policies from pre-collected datasets. In this context, the first works assumed access to policies trained with task-specific rewards [Fu et al., 2020, Gulcehre et al., 2020]. More recently, methods proposed to leverage unsupervised exploration to collect datasets for offline RL [Yarats et al., 2022, Lambert et al., 2022]. In particular, Yarats et al. [2022] creates a dataset of pre-collected trajectories, ExoRL, on the DeepMind control suite [Tassa et al., 2018] generated without any hand-crafted rewards. Similar to URLB [Laskin et al., 2021], ExoRL benchmarks a number of exploration algorithms [Pathak et al., 2017, Eysenbach et al., 2018, Pathak et al., 2019, Yarats et al., 2021], and evaluates the performance of a policy trained on the corresponding offline datasets relabeled with task-specific rewards.

Multi-task Offline RL. Recent works proposed to learn multiple tasks from pre-collected datasets, starting with methods [Endrawis et al., 2021] that generate goals

to improve the offline data collection process in a self-supervised way. This connection has also been studied in the supervised setting [Yang et al., 2022, Ma et al., 2022] and when learning hierarchical policies [Li et al., 2022a]. In a setting closely related to our work, Actionable Models [Chebotar et al., 2021] considers the problem of learning goal-conditioned policies from offline datasets without interacting with the environment, and with no task-specific rewards. They employ goal-conditioned Q-learning with hindsight relabeling [Andrychowicz et al., 2017]. As opposed to their work that relies on learning from sparse rewards, we propose to leverage a self-supervised training stage to densely shape rewards.

6.3 Preliminaries

Let $\mathcal{E} = (\mathcal{S}, \mathcal{A}, P, p_0, \gamma, T)$ define a reward-free Markov decision process (MDP), where $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a state-transition probability function, $p_0 : \mathcal{S} \to \mathbb{R}_+$ is an initial state distribution, γ is the discount factor, and T is the task horizon. In the goal-conditioned setting, the objective is to learn a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ that maximizes the expectation of the cumulative return over the goal distribution, where $\mathcal{G}$ denotes the goal space. Here, we make the common assumption that states and goals are defined in the same form, *i.e.*, $\mathcal{G} \subset \mathcal{S}$.

We assume that we have access to a dataset $\mathcal{D}$ of pre-collected episodes generated by using any data collection algorithm in $\mathcal{E}$. Each episode is stored in $\mathcal{D}$ as a series of (s, a, s') tuples, where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$. In the general offline formulation introduced by Yarats et al. [2022], the dataset $\mathcal{D}$ can be relabeled by evaluating any reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ at each tuple in $\mathcal{D}$, and adding the resulting tuple $(s, a, r(s, a), s')$ in the relabeled dataset $\mathcal{D}_r$. We can extend this protocol to the goal-oriented setting by considering a goal distribution $p_\mathcal{G}$ in the goal space, and any goal-conditioned reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. Given a tuple (s, a, s') in $\mathcal{D}$, we relabel it by sampling a goal $g \sim p_\mathcal{G}$, computing $r(s, a, g)$ and adding the resulting tuple $(s, a, g, r(s, a, g), s')$ in the relabeled dataset $\mathcal{D}_{r,p_\mathcal{G}}$.

Once the relabeled dataset $\mathcal{D}_{r,p_\mathcal{G}}$ is generated, we can learn a goal-conditioned policy by executing any offline RL algorithm. The algorithm runs completely offline, by sampling tuples from $\mathcal{D}_{r,p_\mathcal{G}}$ and without any interaction with the environment. The goal-conditioned policy is then evaluated online in $\mathcal{E}$ on a set of fixed evaluation goals that is not known during training.

6.4 Self-Supervised Reward Shaping

We now describe our self-supervised reward shaping method. It comprises three stages that we will detail below. In the first stage, we train a Reachability Network (RNet) [Savinov et al., 2018b] on the trajectories in $\mathcal{D}$ to predict whether two states

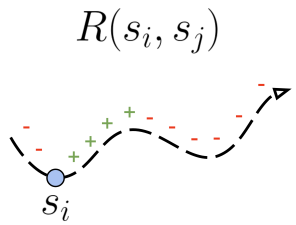


FIGURE 6.1: Training labels generation for RNet: given a state s_i, positive pairs are sampled in the same trajectory within a threshold τ_{reach}, and the rest of the trajectory forms negative pairs.

are reachable from one another. The second stage consists in building a directed graph $\mathcal{M}$ whose nodes are a subset of states in $\mathcal{D}$, and edges connect reachable states. We employ the RNet as a criterion to avoid adding similar states to $\mathcal{M}$ so that its nodes cover the states in $\mathcal{D}$ uniformly. The final stage consists in training the goal-conditioned policy on transitions and goals sampled from $\mathcal{D}$. It is trained with dense rewards computed as the sum of a global (based on the graph distance in $\mathcal{M}$) and local (based on the RNet) distance terms. The important aspect of our method is that the whole training only uses trajectories from the pre-collected dataset $\mathcal{D}$ without running a single action in the environment. We now describe each component in more detail.

6.4.1 Reachability network

In order to learn a good local distance between states in $\mathcal{D}$, we adopt an asymmetric version of the Reachability Network (RNet) [Savinov et al., 2018b]. The general idea of RNet is to approximate the distance between states in the environment by the average number of steps it takes for a random policy to go from one state to another. We adapted the original formulation with two modifications: first, we use exploration trajectories from $\mathcal{D}$ instead of random trajectories and second, we leverage the temporal direction because a state can be reachable from another without the converse being true. Let $(s_1^a, ..., s_T^a)$ denote a trajectory in $\mathcal{D}$, where a is a trajectory index. We define a *reachability label* y_{ij}^{ab} for each pair of observations (s_i^a, s_j^b) by

$$y_{ij}^{ab} = \begin{cases} 1 & \text{if } a = b \text{ and } 0 \leq j - i \leq \tau_{\text{reach}}, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } 1 \leq i, j \leq T, \tag{6.1}$$

where the *reachability threshold* τ_{reach} is a hyperparameter. The reachability label is equal to 1 *iff* the states are in the same trajectory and the number of steps from s_i^a to s_j^b is below τ_{reach}, as shown in Figure 6.1. Note that $y_{ij}^{ab} \neq y_{ji}^{ab}$. We train a siamese neural network R, the RNet, to predict the reachability label y_{ij}^{ab} from a pair of observations (s_i^a, s_j^b) in $\mathcal{D}$. The RNet consists of an embedding network g, and a

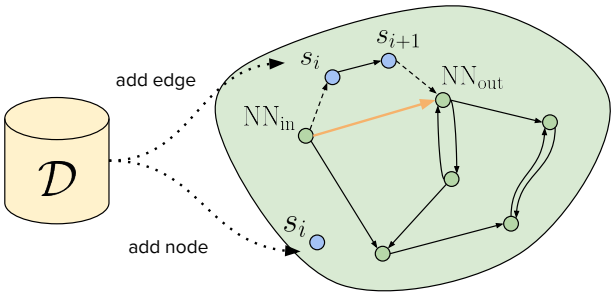


FIGURE 6.2: Overview of the graph building algorithm. Given a transition $(s_i, s_{i+1}) \in \mathcal{D}$, we add s_i as node if it is distant enough from existing nodes in the graph. Moreover, we add an edge in the graph between the incoming nearest neighbor of s_i and the outgoing nearest neighbor of s_{i+1}.

fully-connected network f to compare the embeddings, *i.e.*,

$$R(s_i^a, s_j^b) = \sigma\left[f(g(s_i^a), g(s_j^b))\right], \tag{6.2}$$

where σ is a sigmoid function. A higher R value indicates two states reachable easily with random walk, so they can be considered close in the environment. More precisely, R takes values in $(0, 1)$ and s' is reachable from s if $R(s, s') \geq 0.5$. RNet is learned in a self-supervised fashion, as the ground-truth labels needed to train the network are generated automatically.

6.4.2 Directed graph

In the next phase, we use trajectories in $\mathcal{D}$ to build a directed graph $\mathcal{M}$ that captures high-level dynamics of the environment, as illustrated in Figure 6.2. We want the nodes of $\mathcal{M}$ to evenly represent the states in $\mathcal{D}$. This is achieved by filtering the states in $\mathcal{D}$: a state is added to $\mathcal{M}$ only if it is distant enough from all the other nodes in $\mathcal{M}$. More precisely, a state $s \in \mathcal{D}$ is added to $\mathcal{M}$ if and only if

$$R(s, n) < 0.5 \text{ and } R(n, s) < 0.5, \quad \text{for all } n \in \mathcal{M}. \tag{6.3}$$

Note that we require both the directions to be novel. This filtering avoids redundancy by preventing similar states to be added to the memory. It also has a balancing effect because it limits the number of states that can be added from a certain area even if it is visited by the agent many times in $\mathcal{D}$.

Once the nodes are selected, we connect pairs that are reachable from one to another. To this end, we employ trajectories in $\mathcal{D}$ because they contain actual feasible transitions. Given a transition $s_i \to s_j$ in $\mathcal{D}$, we add edge $n_i \to n_j$ if s_i can be reached from node n_i and node n_j can be reached from s_j. This way, we have a chain $n_i \to s_i \to s_j \to n_j$ and can assume n_j is reachable from n_i. Concretely, we select node n_i to be the incoming nearest neighbor ($\mathrm{NN}_{\mathrm{in}}$) to s_i, and n_j to be the outgoing

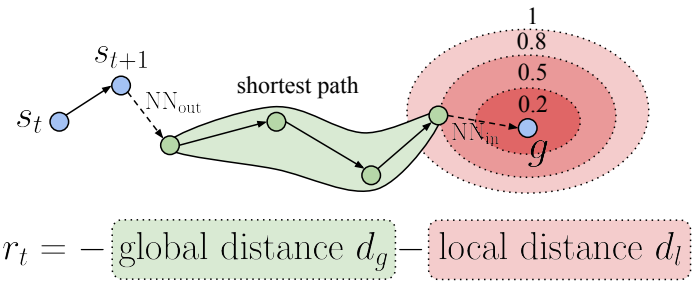


FIGURE 6.3: Visualization of the reward computation process. The total reward is a combination of **(i) a global distance term (green)**, computed with the shortest path in the graph between the outgoing nearest neighbor (NN_{out}) of the state s_{t+1} and the incoming nearest neighbor of the goal (NN_{in}), and **(ii) a local distance term (red)** computed using the RNet value between NN_{in} and g.

Algorithm 2 Building the directed graph $\mathcal{M}$

Input: pre-collected dataset $\mathcal{D}$, Reachability Network R
Initialize: $\mathcal{M} = \{\}$
/ * Build the set of nodes * /
for each state s in $\mathcal{D}$ **do**
  if $R(s, \mathcal{M}) < 0.5$ and $R(\mathcal{M}, s) < 0.5$ **then**
    Update $\mathcal{M} := \mathcal{M} \cup \{s\}$
  end if
end for
/ * Build edges * /
for each transition (s_t, s_{t+1}) in $\mathcal{D}$ **do**
  Let $n_t := \text{NN}_{\text{in}}(s_t) = \text{argmax}_{n \in \mathcal{M}} R(n, s_t)$
  Let $n_{t+1} := \text{NN}_{\text{out}}(s_{t+1}) = \text{argmax}_{n \in \mathcal{M}} R(s_{t+1}, n)$
  Create directed edge from n_t to n_{t+1}
end for

nearest neighbor (NN_{out}) from s_j, *i.e.*,

$$n_i = \text{NN}_{\text{in}}(s_i) = \underset{n \in \mathcal{M}}{\text{argmax}}\, R(n, s_i), \quad n_j = \text{NN}_{\text{out}}(s_j) = \underset{n \in \mathcal{M}}{\text{argmax}}\, R(s_j, n). \tag{6.4}$$

By performing this action over all the transitions in $\mathcal{D}$, we turn $\mathcal{M}$ into a directed graph where edges represent reachability from one node to another. The directed graph building process is summarized in Algorithm 2.

6.4.3 Distance function for policy training

We then use the obtained directed graph to compute a global distance in the state space. Indeed, RNet predicts reachability between s_i and s_j so we can directly use it

Algorithm 3 Building replay buffer $\mathcal{B}$ for offline policy training

Input: pre-collected dataset $\mathcal{D}$, Reachability Network R, directed graph $\mathcal{M}$
Initialize: $\mathcal{B} = \{\}$
while $\mathcal{B}$ is not full **do**
  Sample a transition (s_t, a_t, s_{t+1}) at random in $\mathcal{D}$
  Sample a goal g at random in $\mathcal{D}$
  Compute $d_l(s_{t+1}, g) := 1 - R(s_{t+1}, g)$
  Let $n_{t+1} := \text{NN}_{\text{out}}(s_{t+1}) = \text{argmax}_{n \in \mathcal{M}} R(s_{t+1}, n)$
  Let $n_g := \text{NN}_{\text{in}}(g) = \text{argmax}_{n \in \mathcal{M}} R(n, g)$
  Compute $d_g(s_{t+1}, g) := \text{ShortestPathLength}(n_{t+1}, n_g)$
  Compute reward $r_t := -(d_l(s_{t+1}, g) + d_g(s_{t+1}, g))$
  Relabel transition with goal g and reward r_t, and
  Push $(s_t, a_t, g, r_t, s_{t+1})$ to $\mathcal{B}$
end while

as a distance metric

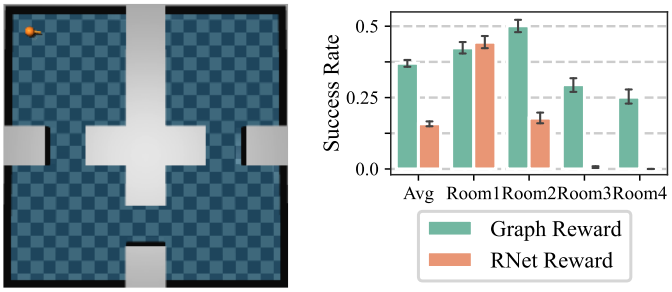
$$d_l(s_i, s_j) = 1 - R(s_i, s_j), \quad \forall s_i, s_j \in \mathcal{S}. \tag{6.5}$$

However, this reachability metric is confined to a certain threshold, so there is no guarantee that the RNet predictions will have good global properties.

In contrast, the directed graph $\mathcal{M}$ captures high-level global dynamics of the environment. We can easily derive a distance function $d_{\mathcal{M}}(n_i, n_j)$ between any pair of nodes in $\mathcal{M}$ by computing the length of the shortest path in this graph, provided the graph is connected. In practice, we can use a trick to connect the graph if necessary, by adding an edge between the pair of nodes from different connected components with the maximum RNet value. Moreover, we can extend this distance $d_{\mathcal{M}}$ to a global distance function d_g in the state space $\mathcal{S}$ by finding, for any pair s_i and s_j in $\mathcal{S}$ their nearest neighbors in the corresponding direction. More precisely,

$$d_g(s_i, s_j) = d_{\mathcal{M}}(\text{NN}_{\text{out}}(s_i), \text{NN}_{\text{in}}(s_j)), \quad \forall s_i, s_j \in \mathcal{S}. \tag{6.6}$$

The distance d_g between two states in the state space becomes the length of the shortest path between their respective closest nodes in the graph. This process, summarized in Figure 6.3, propagates the good local properties of RNet to get a well-shaped distance function for states that are further away. Since d_g captures global distances while d_l captures local fine-grained distance, we use their combination as a final distance function: $\forall s_i, s_j \in \mathcal{S}, \quad d(s_i, s_j) = d_g(s_i, s_j) + d_l(s_i, s_j)$.



(A) UMaze environment (B) RNet vs. Graph Rewards

FIGURE 6.4: (A) Visualization of UMaze environment, and (B) Performance of the goal-conditioned policy trained with RNet and graph-based rewards on UMaze.

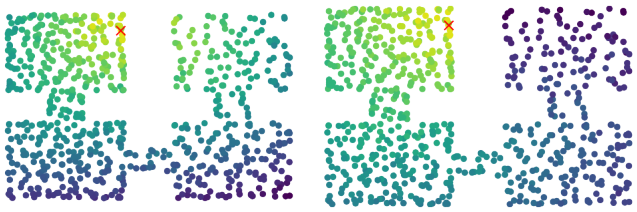
6.4.4 Policy training

The last phase of our method is training the goal-conditioned policy offline. Here, we create an offline replay buffer $\mathcal{B}$ that is filled with relabeled data. We randomly sample a transition (s_t, a_t, s_{t+1}) from $\mathcal{D}$ as well as a goal g and relabel the transition with reward $r_t = -d(s_{t+1}, g)$. We then push the relabeled transition $(s_t, a_t, g, r_t, s_{t+1})$ to $\mathcal{B}$. The pseudo-code of this process is given in Algorithm 3. In order to create a curriculum that artificially guides the agent towards the goal, we experimented with two different transition augmentation techniques:

Sub-goal augmentation. Let $(s_t, a_t, g, r_t, s_{t+1})$ denote a relabeled transition and $(n_0, ..., n_{P-1})$ the shortest path in the graph $\mathcal{M}$ between $n_0 = \text{NN}_{\text{out}}(s_t)$ and $n_{P-1} = \text{NN}_{\text{in}}(g)$. The augmentation technique consists in adding to the replay buffer every transition $(s_t, a_t, n_i, r_t^i, s_{t+1})$ for all $i \in \{0, P-1\}$, where $r_t^i = -d(s_{t+1}, n_i)$. In other words, given a transition (s_t, a_t, s_{t+1}) and a goal g from $\mathcal{D}$, we push to the replay buffer a set of relabeled transitions with all goals on the shortest path from s_t to g (and their corresponding rewards).

Edge augmentation. Similar to the subgoal augmentation technique, we consider a relabeled transition $(s_t, a_t, g, r_t, s_{t+1})$ and the associated shortest path $(n_0, ..., n_{P-1})$. This time, we keep the same goal g for every augmented transition, but for every edge $(n_{i-1}, n_i), i \in \{1, P-1\}$, we add the relabeled transition $(s_t^i, a_t^i, g, r_t^i, s_{t+1}^i)$ to $\mathcal{B}$ where $(s_t^i, a_t^i, s_{t+1}^i) \in \mathcal{D}$, $\text{NN}_{\text{out}}(s_t^i) = n_{i-1}$, $\text{NN}_{\text{in}}(s_{t+1}^i) = n_i$ and $r_t^i = -d(s_t^i, g)$. Note that the existence of such a transition in $\mathcal{D}$ is guaranteed by construction: an edge is added to the graph from one node to another *iff* there exist a transition in $\mathcal{D}$ whose corresponding nearest neighbors are these two nodes (in the same order).

Once the replay buffer $\mathcal{B}$ is filled, the goal-conditioned policy can be trained using any off-policy algorithm. In our implementation, we chose Soft Actor-Critic [Haarnoja et al., 2018], as it is known to require few hyper-parameter tuning, and is widely used in the literature.



(A) RNet distance (B) Graph distance

FIGURE 6.5: Heatmaps of rewards computed with RNet (A) and graph (B) distances. High rewards are shown in yellow, and low rewards in black.

6.5 Experiments

6.5.1 Environments & data collection

We perform experiments on three continuous control tasks with state-based inputs.

UMaze [Kanagawa, 2021]. The first environment, shown in Figure 6.4a, is a two-dimensional U-shaped maze with continuous action space and a fixed initial position. We generate the training data for this environment by deploying a random policy with randomized start position in the maze. We collect 10k trajectories of length 1k. We evaluate the goal-conditioned agent by giving the agent a goal sampled at random in the environment and computing the final Euclidean distance to the goal.

RoboYoga Walker [Mendonca et al., 2021]. Introduced by Mendonca et al. [2021], the challenging RoboYoga benchmark is based on the Walker domain of the DeepMind control suite [Tassa et al., 2018], and consists of 12 goals that correspond to body poses inspired from yoga (*e.g.*, lying down, raising one leg or balancing). We consider the state-based version of the task, and use the task-agnostic dataset from Yarats et al. [2022] generated with an unsupervised exploration policy. It contains 10k trajectories of length 1k obtained by deploying the "proto" [Yarats et al., 2021] algorithm in the Walker domain. The success metric of the evaluation policy is assessed by the pose of the humanoid at the end of the episode.

Pusher [Nair et al., 2018]. We also apply our method on *Pusher*, a realistic robotic environment shown in Figure 6.13 (left), where a robot arm (red) needs to push a puck (blue) to a specified location on a table. To build the offline dataset, we generated 10k random trajectories of length 200. Similar to prior works [Nair et al., 2018, Pong et al., 2020, Mezghani et al., 2022a], we generated 500 goals at random in the state space, and we measured the performance as the final Euclidean distance between the puck and its target location.

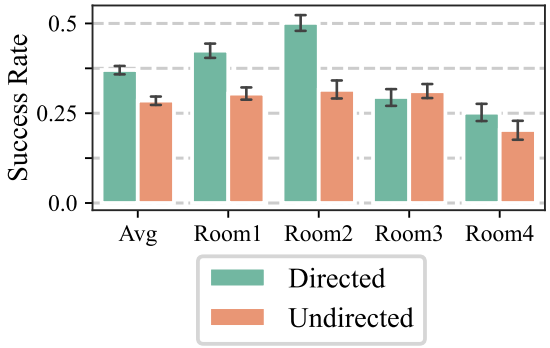


FIGURE 6.6: Importance of graph directness on the UMaze task

6.5.2 Ablation & design choices

We first show that the graph structure is necessary for long-term planning. Then, we explain the importance of the directness of the graph on tasks with asymmetric behaviors. Finally, we show the impact of transition augmentation techniques when labeling data for the goal-conditioned policy.

Necessity of graph-based rewards. An important component of our method is the construction of the graph $\mathcal{M}$ that enables computing a distance with good global properties. To empirically validate this hypothesis, we performed a comparison between the goal-conditioned policy trained with RNet rewards (*i.e.*, by using the distance d_l from equation (5)) and the one trained with both distance terms as reward. We run this experiment on the UMaze environment, and show results in Figure 6.4b. We note that the model trained with graph rewards outperforms the one trained with RNet rewards overall, particularly for distant goals (ie. rooms 3 and 4). We also notice that the model trained with RNet rewards is slightly better for goals that are close to the initial position. This highlights the fact that RNet is good at estimating local distances. The qualitative visualization in Figure 6.5a & 6.5b confirms this observation, as it shows low values between states in the first and fourth rooms.

Importance of graph directness. We then investigate the importance of the asymmetry of the RNet and the directness of the graph. To this end, we implement an undirected version of our method where the RNet is symmetric and the graph is undirected. All other components of our method are unchanged. First, we compare the performance of both variants in the UMaze task in Figure 6.6, and note that asymmetric RNet and directed graph in our approach significantly improve the goal-conditioned policy performance (+11% on success rate), especially on goals close to the initial location, *i.e.*, goals in rooms 1 and 2. We then analyze qualitative visualizations of the shortest path in the undirected and directed graphs in the RoboYoga task, as shown in Figure 6.7. In the undirected case, the humanoid defies the laws of gravity and is encouraged to stand its head by flipping backwards, which might

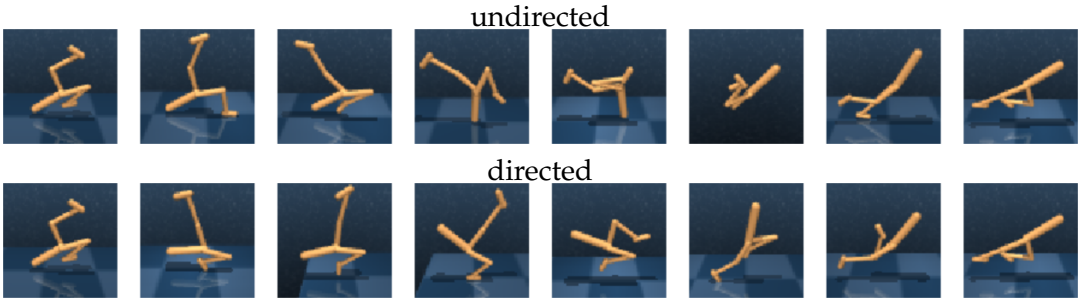


FIGURE 6.7: Shortest Path visualization for undirected (top) and directed (bottom) graphs in the RoboYoga Walker task.

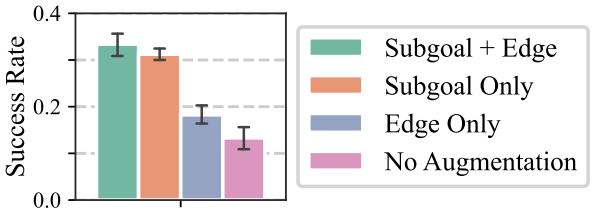


FIGURE 6.8: Impact of Transition Augmentation on the RoboYoga Walker task.

be extremely difficult, or even infeasible. In the directed case, the shortest path fosters the agent to first get back on its legs, and then lean forward. In this example, the gravity makes the dynamics of the environment non-symmetric and non-fully reversible, which justifies the directed formulation described in our method.

Transition sampling strategy. As a final ablation study, we study the utility of the transition augmentation techniques described in section 6.4.4. We evaluate four possible variants of our method: (i) without any augmentation, (ii) with edge augmentation only, (iii) with subgoal augmentation only, and (iv) with both augmentations. We execute this experiment on the RoboYoga task, and show results in Figure 6.8. We observe that both of the augmentation techniques improve the performance of the goal-conditioned agent, with subgoal augmentation showing greater improvement. Moreover, we note that combining both augmentations improves the performance further. For the reminder of the experiments, we use both these augmentation techniques.

6.5.3 Comparison to prior work

Baselines. We compare our method to prior work on unsupervised goal-conditioned policy learning. We perform an apples-to-apples comparison by implementing the baselines using the same learning framework as our method, and changing the reward relabeling process. We compare with the following baselines:

- **Hindsight Experience Replay [HER] [Andrychowicz et al., 2017]** This is a re-implementation of the standard unsupervised RL technique, adapted to the

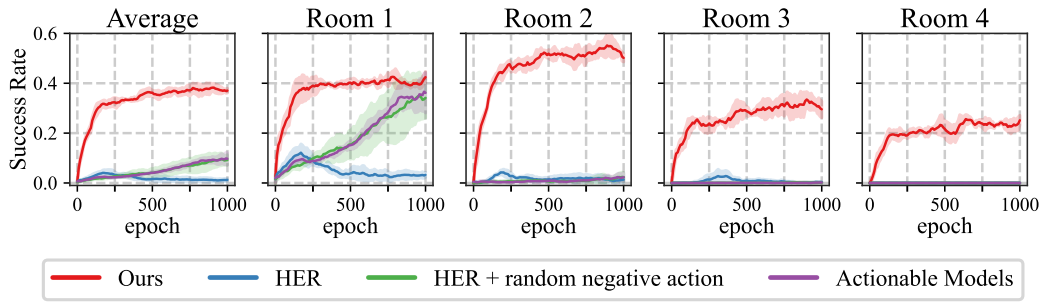


FIGURE 6.9: Performance on the UMaze task. We show the success rate for goals sampled at random in each of the four rooms, as well as the average over all rooms.

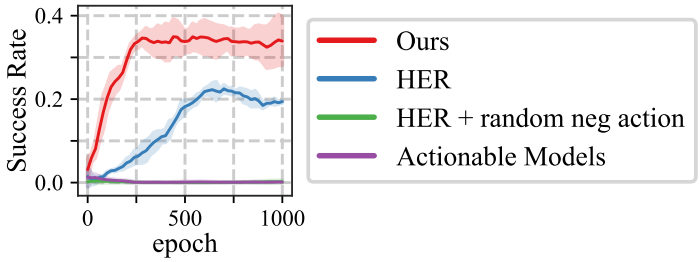


FIGURE 6.10: Comparison to baselines on the RoboYoga Walker task.

offline setting. More precisely, we relabel sub-trajectories from $\mathcal{D}$ with a sparse reward, which is equal to 1 only for the final transition of the sub-trajectory, and 0 everywhere else. Following Chebotar et al. [2021], we also label sub-trajectories with goals sampled at random in $\mathcal{D}$ and zero reward.

- **HER [Andrychowicz et al., 2017] with random negative action** is a variant of HER where, for a transition in $\mathcal{D}$ we sample an action uniformly at random in the action space and label it with zero reward. This helps overcoming the problem of over-estimation of the Q-values for unseen actions mentioned in Chebotar et al. [2021].

- **Actionable Models [Chebotar et al., 2021]** This approach is based on goal-conditioned Q-learning with hindsight relabeling. We re-implemented the goal relabeling procedure that uses the Q-value at the final state of sub-trajectories in $\mathcal{D}$ to enable goal chaining, as well as the negative action sampling trick.

Comparison on UMaze. We compare our method to the baselines on the UMaze task, and show results in Figure 6.9. We observe that our model outperforms all baselines overall, and shows greater improvements on challenging goals that are far from the initial position. Interestingly, we note that Actionable Models reaches goals in the first room only. This confirms the intuition that sparse rewards make it difficult for the policy to learn long-horizon tasks.

Comparison on RoboYoga Walker. In a second experiment, we compare our method to baselines on the RoboYoga task, as shown in Figure 6.10. Here again, our method outperforms prior work, and Actionable Models does not make any significant improvement over HER. The results for each goal are shown in Figure 6.11. These goals are illustrated in Figure 6.12. We see that our method masters most of the goals that do not require balancing (Lie Back & Front, Legs Up, Lunge), and succeeds quite well at more complicated goals like Side Angle, Lean Back and Bridge, but is unable to progress in complex goals like Head Stand or Arabesque. Overall these results suggest that our dense reward shaping method allows for faster and more robust offline goal-conditioned policy training.

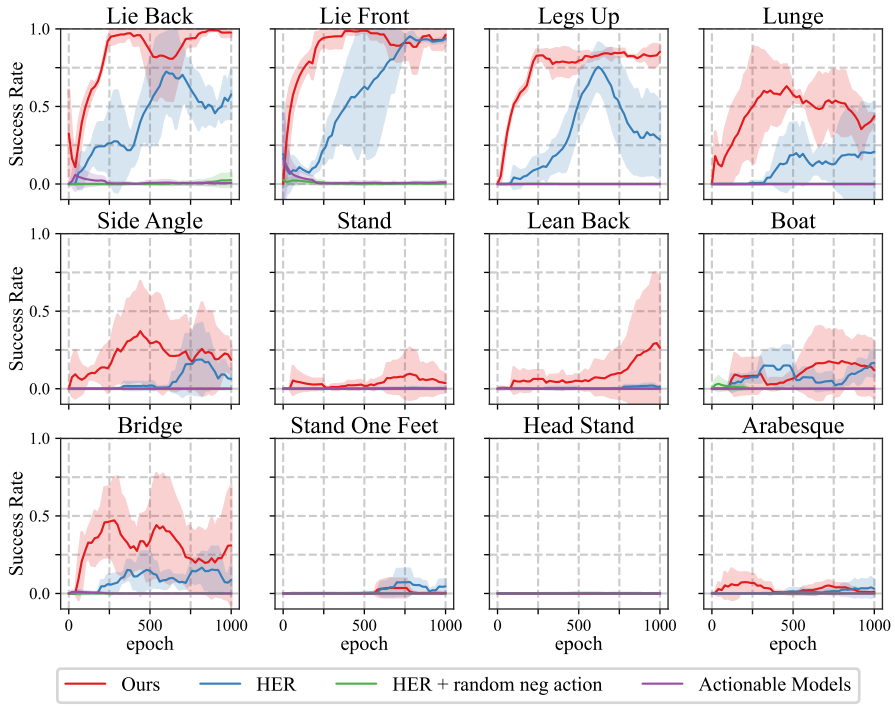


FIGURE 6.11: Performance on the RoboYoga Walker talk for each of the 12 goals.

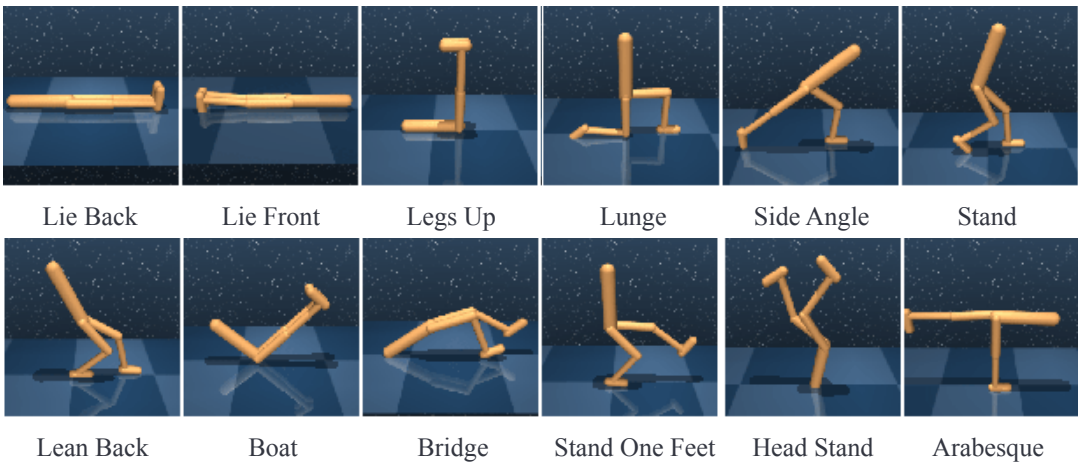


FIGURE 6.12: Visualization of the 12 evaluation goals for the RoboYoga Walker task.

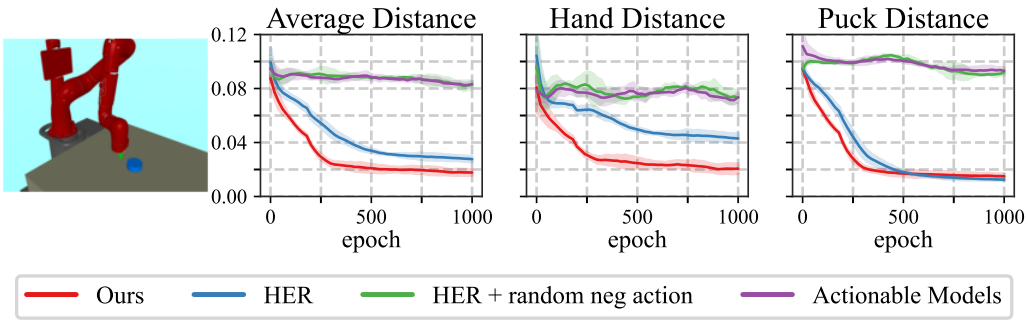


FIGURE 6.13: Performance on the Pusher task (lower is better). We report the final average, hand, and puck distance to the goal for our model and all baselines.

Comparison on Pusher. As a final experiment, we compared our method to prior work on a realistic robotic environment, as shown in Figure 6.13. Our policy trained offline is evaluated by sampling a goal at random in the state space, and measuring three different metrics: (i) the *hand distance*, which corresponds to the final distance between the end of the robot arm and the target, (ii) the *puck distance*, which measures the distance between the final puck location and the target, and (iii) the *average distance*, the average of the first two metrics. Our method outperforms the baselines on this task, and our goal-conditioned agent is able to sequentially place the puck at the goal location, and then place the hand at its target location. On the contrary, **HER [Andrychowicz et al., 2017]** places the puck at the target location with a performance similar to our method, but lacks precision on the hand location.

6.6 Conclusion

In this chapter, we proposed a method for learning multi-task policies from pre-generated datasets in an offline and unsupervised fashion, *i.e.*, , without requiring any additional interaction with the environment, nor manually designed rewards. Our method leverages a self-supervised stage that aims at learning the dynamics of the environment from the offline dataset, and that allows for shaping a dense reward function. It shows significant improvement over prior works based on hindsight relabeling, especially on long-horizon tasks, where dense rewards are crucial for learning a good policy.

The main limitation of our method is that it relies on the availability of a pre-collected dataset of trajectories, with a sufficiently large coverage of the state space for proper policy learning. Although such data can be already available, as for the RoboYoga Walker task, or that offline dataset collection could be done with random policies, as we did on the UMaze and Pusher tasks, this step can be challenging for other environments.

Chapter 7

Using Text as Supervision for Language-Conditioned Agents

When developing goal-oriented agents, goal specification is an important design choice. In Chapters 3 to 6, we studied the state-reaching case, where the goal is specified as a state (or an observation) in the environment. There are, in fact, some limitations to such a formulation. It might be difficult to access some states of the environment that we want to use as goals, since it requires to actually put the agent in the goal position and collect the observation from there. It is the case, for example, for the RoboYoga task presented in the previous chapter, where defining the "head stand" state needs manual engineering. Another important limitation is that all desirable goals might not be defined as a state. For example, if the desired goal is to perform a back-flip in the RoboYoga case, the final state represents the humanoid standing up, and will therefore not capture the correct behavior.

Recent works [Fontoura et al., 2014, Küttler et al., 2020] suggest that a more natural and general way of specifying goals is to use language instructions. Text is indeed a flexible and interpretable way to express a desired behavior, and can be used to either describe a specific goal state, or to specify the way in which the goal should be reached. However, using text in RL requires access to environments with a rich language component, and to have models that understand the alignment between the behavior of the agent and natural language. In this chapter, we propose to leverage subgoal descriptions in the BabyAI environment [Chevalier-Boisvert et al., 2019] to train a language-conditioned agent. We collect a dataset fo expert trajectories on several levels of the environment, as well as text aligned with the intermediate subgoals reached by the expert. We train a transformer-based agent on these expert trajectories, by predicting both actions and language-based subgoals. We evaluate the performance of our model zero-shot, on language-specified instructions, and show that it outperforms the text-free baseline.

7.1 Introduction

Transformer-based [Vaswani et al., 2017] large language models (LLMs) trained on a massive amount of text data have shown impressive results in various language understanding tasks [Brown et al., 2020]. Their application also goes beyond language domains, ranging from instruction following [Hill et al., 2020] to vision-language navigation [Majumdar et al., 2020]. This progress suggests that LLMs are powerful not only for purely linguistic modeling, but they can also be used in setups that require planning and sequential reasoning. Interestingly, Li et al. [2022b] show that this even holds for tasks that do not involve actual language at all, but only sequential string tokens, which highlights the compositional power of these models.

Along with the rise of LLMs, the field of reinforcement learning (RL) has also been impacted by the advances in sequential decision-making. Indeed, the development of offline RL [Levine et al., 2020], that is, exploiting pre-collected datasets to learn policies without interacting with the environment, has enabled a new way of learning controllable agents—from the sequence modeling perspective [Janner et al., 2021]. In this light, Decision Transformer [Chen et al., 2021] shows how to learn a policy using a causal transformer [Vaswani et al., 2017], and follow-up works [Putterman et al., 2021, Correia and Alexandre, 2022] propose ways of training them without rewards, in a goal-conditioned formulation.

Replicating the success of LLMs in RL also requires large-scale training data, such as Minedojo [Fan et al., 2022] that gathers, among other things, a collection of YouTube video tutorials on MineCraft. These videos contain captions that often explain what the human is doing while playing, creating an exciting opportunity to leverage language-based reasoning with decision making in RL.

In this chapter, we propose a method for training an agent that is capable of interlacing language reasoning with performing actions in an environment. This allows us to fully leverage captions in offline training data, in addition to actions and observations. Our method relies on a unified policy that is capable of choosing between thinking verbally and acting. We achieve this by equipping the policy with additional word outputs so it can decide when to act and when to perform language reasoning. To this end, we train an auto-regressive transformer, conditioned on the instruction and observations, to predict both actions and text captions in a unified way. At test time, given a language instruction, our model predicts actions towards the goal, as well as text tokens to reason, as shown in Fig. 7.1, where the captions define which subgoals it has to reach to solve the task.

In order to study this problem in a controlled setting, we experiment on BabyAI [Chevalier-Boisvert et al., 2019], a grid world platform to study language-grounded tasks. Specifically, we focus on language reasoning that lays out the next subgoal.

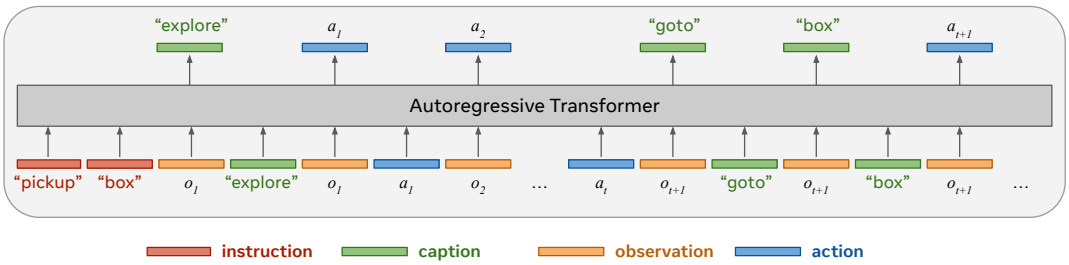


FIGURE 7.1: Given an instruction, our transformer policy can generate language based reasoning tokens interleaved with sequence of actions in the environment.

We generated a dataset of trajectories with the expert bot provided by Chevalier-Boisvert et al. [2019], and leveraged the underlying subgoals used by the bot to create text descriptions aligned with the trajectories, *i.e.*, captions. We evaluate the performance of our model zero-shot, on language-specified instructions, and study its generalization to unseen maze configurations on several setups, including the BossLevel, the most difficult task of BabyAI. We show that leveraging text captions greatly improves the performance compared to the caption-free baseline, particularly for harder tasks that require planning and long-term reasoning.

In summary, we make the following contributions: **(i)** we propose an algorithm for generating text-augmented expert trajectories on BabyAI [Chevalier-Boisvert et al., 2019] that constitutes a toy environment for emulating tutorial videos, **(ii)** we present an auto-regressive transformer architecture to learn to predict both actions and captions in a unified way, and **(iii)** we show that it outperforms the caption-free baseline, particularly on tasks that involve long-term sequential planning.

7.2 Related Work

Offline datasets in RL. The data collection technique is an important aspect when studying the training of policies from pre-collected datasets. In this context, prior works assumed access to policies trained by task-specific rewards [Fu et al., 2020, Gulcehre et al., 2020] or proposed to leverage unsupervised exploration policies [Eysenbach et al., 2018, Yarats et al., 2021] to collect datasets for offline RL [Yarats et al., 2022, Lambert et al., 2022]. These works are often limited to learning a restricted set of tasks, specified with hand-crafted reward functions. More recently, the accessibility of large-scale datasets [Baker et al., 2022, Fan et al., 2022] from the internet enabled new possibilities for learning offline policies with language-based task specification. Indeed, these works propose to exploit a dataset of Youtube videos, that are augmented with text sources at several stages, including captions throughout the video. However, this data source has one crucial drawback, as the actions that the player is executing (*i.e.*, the keyboard buttons or the mouse movements) are missing, and consequently, the videos cannot be considered as *expert trajectories*. To overcome

this issue, Baker et al. [2022] propose to collect a small dataset of trajectories annotated by human experts. Using this data, the authors propose to train an Inverse Dynamics Model [Nguyen-Tuong et al., 2008] to predict the actions given past and future frames, and use it to annotate the large dataset of YouTube videos with corresponding actions.

Transformers in RL. Using Transformers in sequential decision-making [Li et al., 2023] was first proposed by Janner et al. [2021], and Chen et al. [2021]. The latter present Decision Transformer, a causal transformer model with masked attention layer used to train policies on offline datasets. One particularity of this model is that it uses returns-to-go to derive optimal behavior from the training data. By leveraging these cumulative rewards, the model learns the interesting transitions, even when the offline dataset is not collected with an expert policy. The drawback of this method, however, is that the initial return-to-go value must be arbitrarily chosen at test time. To avoid relying on these, Correia and Alexandre [2022] propose a hierarchical formulation of the decision transformer, where a high-level policy outputs subgoals for the low-level policy. Alternatively, Putterman et al. [2021] propose a goal-conditioned version of the Decision Transformer, where demonstrations are preceded with a language instruction describing the goal of the trajectory. In this setup, trajectories must be collected by expert policies since the transformer only imitates the demonstrations present in the dataset. Related to this line of works, our model leverages not only language instructions, but also text description along the trajectory, for faster learning and better generalization.

Language-conditioned Offline RL. Advances in both offline RL and transformer models for sequential decision-making led to an interesting research direction, that aims at taking advantage of text data for offline RL. Several works showed promising results on datasets as large as MineDojo [Fan et al., 2022] by designing specific reward functions on top of the offline data, either manually for a specific task like crafting diamonds [Baker et al., 2022], or by learning the alignment between text and images in the video [Fan et al., 2022], in a CLIP Radford et al. [2021] fashion. A very recent line of study propose to use large language models on offline datasets for decision-making tasks: Zhang et al. [2022] show that task instructions can be effectively used to pre-train offline policies and Carta et al. [2023] train an adaptive LLM-based policy in language-grounded tasks. Closely related to our work, Li et al. [2022b] show the effectiveness of LLMs on decision-making tasks, specifically on the BabyAI [Chevalier-Boisvert et al., 2019] environment. Their idea is to convert actions and observations to text input, and fine-tune a pre-trained GPT-2 Radford et al. [2019] language model on a set of expert trajectories. Apart from the observation conversion part, our model is trained on the same type of data than this work, and our goal is to show the importance of using subgoal descriptions for the stability and performance of training.

Algorithm 4 Sequence from trajectory

Input: trajectory $\mathcal{T} = (o_t, a_t, c_t)_{0 \leq t < T}$
$s_0, x_0 := o_0, c_0$
$s_1, x_1 := o_0, a_0$
$i := 1$
for $t = 1$ **to** $T - 1$ **do**
  if $c_t \neq c_{t-1}$ **then**
    $i := i + 1$
    $s_i, x_i := o_t, c_t$
  end if
  $i := i + 1$
  $s_i, x_i := o_t, a_t$
end for

7.3 Learning Language-Conditioned Agent Offline

We consider a goal-conditioned partially observable Markov decision process (POMDP), defined by a set of states, actions, observations, goals and a transition model, that maps the current state and action to the next state. In a POMDP, the observation o_t at timestep t only captures a portion of the underlying state, and an optimal policy must take as input not only the last observation o_t, but also the history of previous observations and actions [Parr and Russell, 1995] $h_t = (o_0, a_0, ..., o_{t-1}, a_{t-1})$. Our goal is to learn a goal-conditioned policy $\pi(a_t|m, h_t, o_t)$ which, given an instruction (or mission) m, history of previous observations and actions h_t, and last observation o_t, outputs a probability over the set of actions.

In this work, we assume that we do not have access to the POMDP at train time, but instead, that the policy must be trained offline, on a dataset of pre-collected expert trajectories $\mathcal{D}$. $\mathcal{D}$ contains trajectories $(m, (o_t, a_t)_{0 \leq t < T})$ where m is the goal of the trajectory specified with natural language (and can be thought of as an instruction), such that the sequence $(o_t, a_t)_{0 \leq t < T}$ satisfies goal m. We also assume that the train trajectories are augmented with natural language captions c_t at every timestep, that is, trajectories are of the form: $(m, (o_t, a_t, c_t)_{0 \leq t < T})$. However, these captions are not available during the evaluation stage. The goal is therefore to learn a policy, which at test time, is given a instruction, and must perform the actions that satisfies the goal described by the instruction.

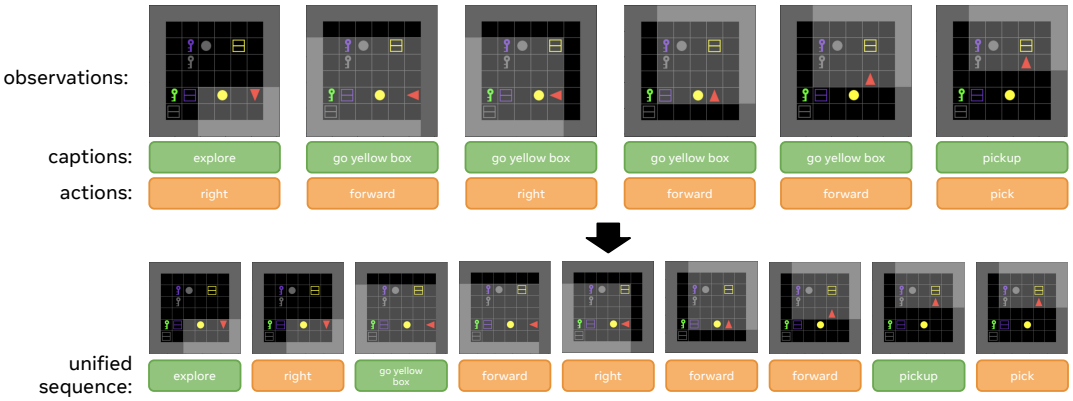


FIGURE 7.2: Visualization of the trajectory to sequence process. The top row shows the original trajectory, with observations, actions and captions, and the bottom row shows how observations are duplicated, and how action and captions organized in the unified sequence.

7.4 Transformer-based model for unifying actions and language reasoning

We will now present our approach, which relies on the idea that language annotations, e.g., captions from YouTube gaming tutorials, provide useful information about which subgoals the player is solving while playing. In MineCraft for instance, in a tutorial on how to craft diamonds, the video will explain the different steps that should be executed to achieve the goal, including crafting a table and a pickaxe. Since captions are not available at test time, *i.e.* when the agent is deployed in the environment, we propose to train an auto-regressive transformer on the offline dataset that, given the instruction and sequence of observations, will not only learn to predict actions, but also captions.

7.4.1 Unifying actions and language reasoning

In the formalism described in Sec. 7.3, we assume that captions and actions have the same temporal granularity, *i.e.* that a new caption appears exactly when a new action is executed. In practice however, in tutorial videos, the granularity of action and caption sequences might be different, as same captions can last for several steps, or conversely, a long caption can occur while no new action appeared.

In our approach, we propose to unify actions and language reasoning in a joint sequence, and train a model that can predict both modalities. We therefore create a new sequence $(s_i, x_i)_{0 \leq i < N}$ from the $\mathcal{T} = (o_t, a_t, c_t)_{0 \leq t < T}$ trajectory. The newly created sequence includes a list of tokens $(x_i)_{0 \leq i < N}$ that can be either caption tokens or actions, as well as a list $(s_i)_{0 \leq i < N}$ of observations, obtained by duplicating elements in $(o_t)_{0 \leq t < T}$. The general idea is to go through the trajectory $\mathcal{T}$, and whenever a new caption is encountered, insert its tokens in the sequence by duplicating the corresponding observation. In practice, we convert actions to natural language, as done in prior work [Li et al., 2022b, Carta et al., 2023], so that actions and captions

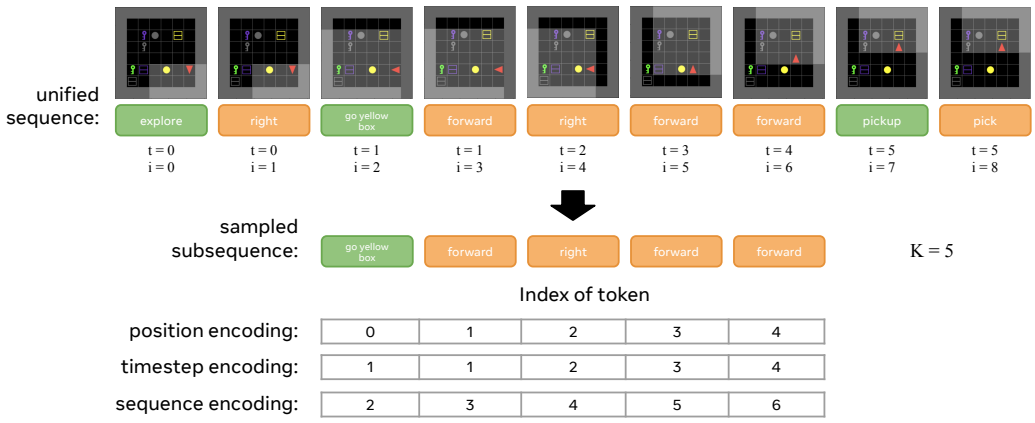


FIGURE 7.3: Example of the different encoding types. We first sample a sub-sequence of length K (here, 5) from the unified sequence, and then show the index of token fed to the transformer for the three different types of encodings: position, timestep and sequence.

stand in the same vocabulary. The process is described in detail in Alg. Figure 4, and an illustrated example is shown in Fig. 7.2. The dataset of trajectories $\mathcal{D}$ is therefore transformed into a dataset $\mathcal{B}$ of unified action-caption sequences of the form $(m, (s_i, x_i)_{0 \leq i < N})$.

7.4.2 Auto-regressive transformer for generating both language and actions

We train an auto-regressive transformer on the set of sequences $\mathcal{B}$, as shown in Fig. 7.1. The model first encodes the instruction sentence m by tokenizing it into a sequence of length $m = (m_0, ..., m_{n-1})$, and embedding every token. For computational efficiency, we sample sub-sequences $(s_i, x_i)_{t \leq i < t+K}$ of length K from the full sequences in $\mathcal{B}$. We pass the observations $(s_i)_{t \leq i < t+K}$ through a convolutional network, and embed the action or caption tokens $(x_i)_{t \leq i < t+K}$ before passing them to the transformer model. The model predicts the action or caption token at every step in the trajectory in an auto-regressive manner, and is trained to minimize the token prediction error across the trajectory using the cross-entropy loss.

Similarly to Vaswani et al. [2017], each element in the sequence is summed with a positional encoding, to make sure that the model takes advantage of the order of the sequence. While instruction tokens m_j are summed with encodings of j indicating their absolute positions, s_i and x_i tokens are summed with the encoding of i informing their position in the sequence. Note that this is different from timestep encodings because i is incremented by both caption tokens and environmental steps. The different types of encodings are illustrated in Fig. 7.3.

At test time when the agent has to act in the environment, we sequentially generate tokens from the transformer model by feeding the current observation. If the generated token belongs to the caption vocabulary, we simply feed it back to the model

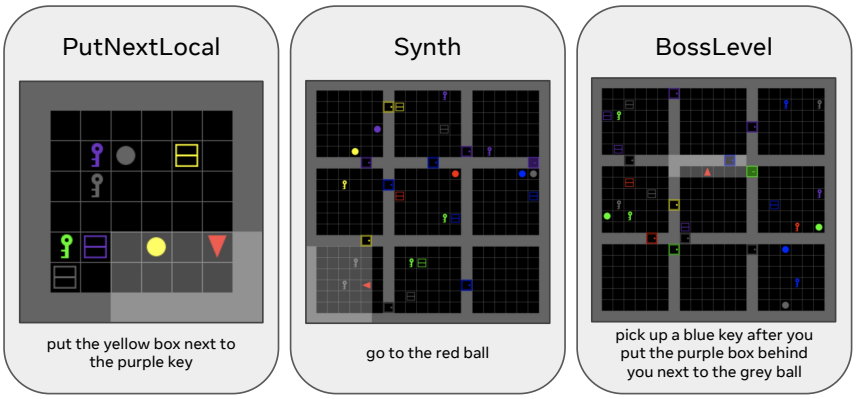


FIGURE 7.4: The three levels from BabyAI that we consider in this work, and an example of observation and instruction for each of these environments.

and continue the generation similar to language models. Only when the model generates an action token, we perform this action in the environment and feed the updated observation to the model. This way the model itself decides when to reason and when to act, interleaving multiple reasoning captions throughout an episode.

7.5 Experiments

7.5.1 Data generation in BabyAI

We generate a dataset of expert trajectories aligned with natural language captions in the BabyAI [Chevalier-Boisvert et al., 2019] environment. BabyAI is a grid world platform developed to train agents grounded with natural language instructions. The platform contains several levels that vary in difficulty, with different types of instructions and distractors. In this chapter, we consider the following three levels, of increasing difficulty:

- **PutNextLocal**: the agent must put an object next to another object, with distractors.
- **Synth**: the agent must perform one of the four possible instruction **(i)** *open* a door specified by its color, **(ii)** *go to* an object specified by its nature and color, **(iii)** *pick up* an object specified by its nature and color, and **(iv)** *put* an object next to another object.
- **BossLevel**: the hardest task of the BabyAI suite. The mission is a combination of tasks from the **Synth** level, where objects can be specified by their type and color, but also by their location on the map (*e.g.* "the door on your left" or "the ball behind you"). Moreover, the task combination might have to be realized in a particular order.

An example of environment and text instruction for each level is shown in Fig. 7.4. In all three environments, the observations cover the 7×7 area visible to the agent,

TABLE 7.1: Success rate (mean and standard error) on the three BabyAI levels with different amounts of training data. We show results for our method, as well as two caption-free baselines, including **BabyAI-Ori-1M**, the imitation learning baseline from the original BabyAI paper.

	PutNextLocal-100k	**Synth-500k**	**BossLevel-1M**
BabyAI-Ori-1M	99.2	97.3	77.0
Baseline	**99.5** ± 0.2	**97.9** ± 0.2	73.6 ± 8.0
Our Method	**99.6** ± 0.1	96.4 ± 0.3	**85.2** ± 0.5

and is given as a tensor of size $7 \times 7 \times 3$, where the last dimension describes the type of object present at the corresponding location, its color and its state (*e.g.* for doors, either *open*, *locked* or *closed*).

Together with the platform, Chevalier-Boisvert et al. [2019] also provide an expert bot, that can solve any language-grounded instruction in the environment. It is implemented by setting subgoals to itself and solving them sequentially. We therefore used this bot to collect expert trajectories, and processed the underlying subgoals used by the bot to generate natural language descriptions of the sub-tasks that the agent is achieving.

Note that for each trajectory, the environment configuration is different, *i.e.* the location of doors, objects, as well as the initial agent location if randomized. Moreover, the seed for generating evaluation episodes is distinct from the train one, which means that environment configurations from the evaluation set are not seen during training. We thus generate datasets of varying size, ranging from 50k to 1M trajectories, and evaluate all models on a fixed set of 512 environment configurations and text instruction pairs.

7.5.2 Training details

To implement our auto-regressive transformer model, we used the GPT-2 Radford et al. [2019] model architecture and tokenizer from the HuggingFace Library Wolf et al. [2020]. In all experiments, we used a reduced architecture with 4 layers, 2 attention heads and a dropout of 0.1, and we tested two hidden size configurations for the transformer: 256 and 512. We train our model on the generated dataset using AdamW optimizer [Loshchilov and Hutter, 2019] with a learning rate of 1e-4.

At every epoch, we perform 5000 model updates, during which we sample 128 sub-trajectories of size K from the dataset. Instructions, actions and captions are tokenized using the pre-trained GPT-2 tokenizer, and embedded with a common embedding layer. Observations are encoded using a 3-layer convolutional network, then flattened with a linear layer. To predict the action or caption token, we use a linear layer on the transformer hidden state followed by softmax.

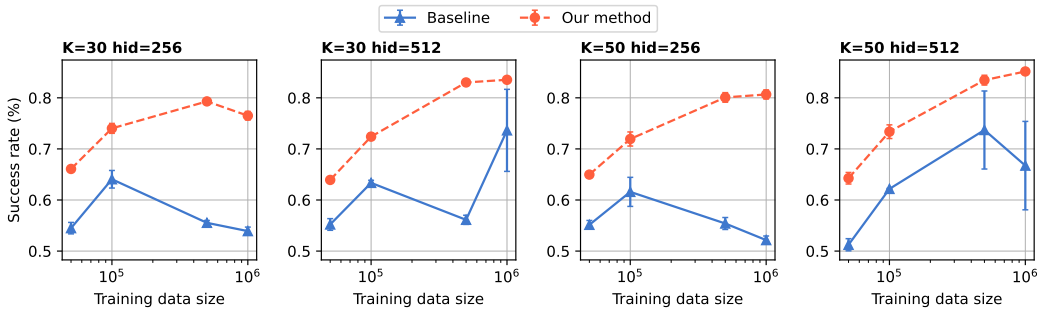


FIGURE 7.5: Success rate on BossLevel for varying amount of training samples in 4 different settings (varying sub-trajectory length K and the model hidden size). Our method consistently outperforms the caption-free baseline. We repeat each experiment 3 times and plot standard error.

7.5.3 Comparison to caption-free Baselines

We first compare our unified policy to the baseline that does not use subgoals. Our baseline is trained using the exact same process as the caption-based policy, except that the captions were removed from the dataset, and only actions remain. This baseline can therefore been seen as reasoning-free model, which at train time, only sees the instruction, observations and the actions. We run all experiment on 3 random seeds on the three babyAI levels, and show results in Table 7.1. We also reported the numbers from the Imitation Learning baseline (**BabyAI-Ori-1M**) of the original BabyAI paper [Chevalier-Boisvert et al., 2019]. This model was trained on expert demonstrations with language-grounded instructions, in the exact same setup than our baseline, with the only difference being the dataset size: all models from the original BabyAI imitation learning baseline were trained on 1M trajectories, vs. twice as less for our baseline on the **Synth** task, and 10 times less on **PutNextLocal**. We see that both baselines perform similarly on all three levels.

Concerning the caption-based model, we observe that it perform similarly to the baselines on **PutNextLocal-100k**, the simplest task, but that the baseline is slightly better on **Synth-500k**. On the harder **BossLevel-1M** task however, the model that leverages captions outperforms baselines, showing on average +10% success rate. These results can be explained by the nature of the tasks. Indeed the BossLevel requires long-term reasoning and sequential planning, while a good exploration behavior and a decent understanding of the instruction is enough for solving most of the tasks from the Synth environment. This result is strengthen by the graphs shown in Fig. 7.5, which compare our method and the baseline for varying training dataset sizes, sub-trajectory length K and hidden size. We see that our method consistently outperforms the caption-free baseline for all three parameters, highlighting the efficiency of leveraging captioned subgoals for challenging sequential tasks.

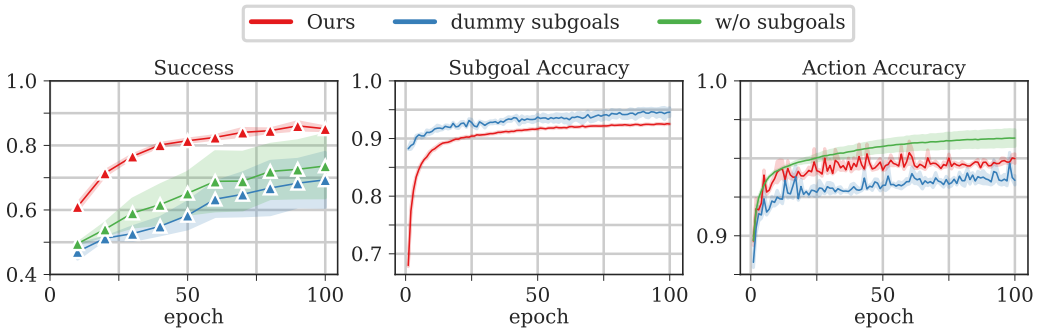


FIGURE 7.6: Success rate, subgoal and action prediction accuracy on BossLevel for our method and two ablated variants: (i) *dummy subgoals*, the method where subgoals are replaced by dummy tokens, and (ii) *w/o subgoals*, the caption-free baseline. We see that the model with dummy captions, despite showing good accuracy at predicting subgoals at train time, performs much worse than the model with captions. This confirms that the model with captions really learns to leverage the information contained in the captions to solve the task.

7.5.4 Highlighting the role of captions

The model with captions therefore outperforms the baseline on the harder task, that require planning and long-term reasoning. However, it is not clear whether the model really learns to understand and exploit the captions, or if it simply uses the caption tokens as an extra compute power to predict the right actions. In order to validate the importance of captions for sequential reasoning, we perform an additional ablation experiment, shown in Fig. 7.6. We compared the performance of our method with captions to a model trained on dummy captions, *i.e.*, captions that are arbitrarily sampled from the vocabulary. More precisely, we replaced the subgoal tokens by a sequence of tokens of the same length as the original caption, but that have no meaning for the task. For this sequence to be still predictable by the model, we generate it always with the tokens in the same order.

Interestingly, we see that the model with dummy captions performs much worse than the model with captions, and its performance is similar to the caption-free baseline. Moreover, we see from the subgoal accuracy plot that the model with dummy captions is better at predicting subgoals than the model with captions. This means that the model knows when subgoals should be predicted. Overall, this experiment confirms the theory that the model with captions really learns to understand and exploit the information contained in the captions.

7.5.5 Importance of positional encoding

Finally, we investigate the role of the positional encoding in our model and the baseline. The positional encoding embeds the relative position of elements stored in the batch fed as input to the transformer. Its role is of tantamount importance as its allows the model to put attention on the right elements. We compare three different types of encodings: **(i)** the *position* encoding, which is simply the absolute position

TABLE 7.2: Comparison of encoding strategies for the caption-free baseline and our method.

	Encoding	PutNextLocal-100k	BossLevel-1M
Baseline	position	98.9 ± 0.1	57.8 ± 3.9
	timestep/sequence	99.5 ± 0.2	73.6 ± 8.0
Our Method	position	99.0 ± 0.3	78.1 ± 0.6
	timestep	-	79.6 ± 1.1
	sequence	99.6 ± 0.1	85.2 ± 0.5

in the sub-sequence fed to the model, as in the original transformer implementation, **(ii)** the *timestep* encoding, that encodes the actual environment timestep of the corresponding observation and action, and **(iii)** the *sequence* encoding which corresponds to the index in the modified sequence, a default one as described in Section 7.4.1. Note that for the caption-free baseline, **(ii)** and **(iii)** are the same since that, in the absence of captions, the modified sequence is the same as the original trajectory.

We report results in Table 7.2, for **PutNextLocal-100k** and **BossLevel-1M** tasks. We see that , on the simpler task, the switching to position encoding only slightly degrades the performance (< 1%) for the baseline and our method. On BossLevel however, the performance of position encoding is much worse than the sequence one: -16% for the baseline, and -7% for our method. This confirms the intuition that sequence encoding is necessary for tasks that require knowing where the agent is in the episode for predicting the right actions and subgoals. We also notes that timestep encoding, while being slightly better than position encoding, is much worse than sequence ones. A potential reason for that is the fact that actions and captions from the same timesteps have the same encoding, and it might make it hard for the model to disentangle both elements.

7.6 Conclusion

We proposed a unified policy that is capable of switching between reasoning with language and selecting the next action to take. When trained on offline data that contains subgoal descriptions, we saw consistent improvements over the baseline that skips this language reasoning. One possible explanation for this improvement is that language reasoning splits complex tasks into smaller chunks that are easier for the model to learn. This promising result suggests the possibility of improving RL training efficiency with language reasoning.

Chapter 8

Conclusion

In this thesis, we studied the problem of learning goal-conditioned agents under limited supervision by focusing on three main challenges: relying on minimal observation modalities, training RL agents in the absence of external rewards, and leveraging fixed datasets for learning agents. We presented novel approaches for learning goal-oriented agents under various conditions, and evaluated them on a variety of tasks, including continuous control, navigation, and language-conditioned tasks. In this last chapter, we summarize the contributions of the thesis in Section 8.1 and then conclude with perspectives for future research in Section 8.2. Table 8.1 provides a summary – and comparison – of the different results presented in this thesis.

8.1 Summary of contributions

Navigation and planning from pixels. In Chapter 3, we presented the problem of image-goal navigation with generalization to unseen environments, and introduced a novel memory-augmented approach for learning to visually navigate in real-world environments. The proposed method builds on an attention-based end-to-end model that leverages an episodic memory to learn to navigate. Our approach was validated with extensive evaluations, and we showed that our model establishes a new state of the art on unseen environments from the challenging Gibson dataset [Xia et al., 2018].

Learning to discover and reach states without supervision. In Chapter 4 we tackled the problem of learning to visually navigate in photo-realistic environments without any supervision, that is, from RGB input only and without access to neither a reward function, nor a notion of distance in the environment. We presented a three-stage approach for this problem that consists in first, learning a representation of the environment in a self-supervised fashion, then exploring the environment to build a graph over the states and then learning to navigate by leveraging this graph. In Chapter 5, we extended this method to other tasks, such as locomotion and manipulation, and showed that our approach can be used to learn to reach states in a variety of environments without supervision. More precisely, we proposed to jointly

Chapter	External Rewards	RGB Observations	Setup	Goal specification	Environments
Chapter 3	✓	✓	online	obs	Habitat
Chapter 4	-	✓	online	obs	Habitat
Chapter 5	-	-	off-policy	obs	UMaze, Pusher
Chapter 6	-	-	offline	obs	+ RoboYoga
Chapter 7	-	-	offline	text	BabyAI

TABLE 8.1: Summary of the results presented in this thesis. The table shows the different setups used in each chapter, and the environments used for the experiments. It compares the availability of external rewards, the type of observations used, the setup used to train the agent (online, off-policy or offline), the type of goal specification (observations or text), and the environments used for the experiments.

learn the representation of the environment and the state-reaching policy by alternating between goal-oriented trajectories and random exploration. This process allowed for fostering exploration at all stages of the training of the goal-conditioned policy, and further improved the representation of the environment.

Leveraging pre-collected datasets for learning goal-conditioned agents. Finally, in Chapter 6, we showed how pre-collected trajectories can be used to learn policies that can reach any state in the environment, offline and without supervision. To this end, we deployed a self-supervised representation learning stage on the environment, and then used the resulting representations to shape a dense reward signal for the goal-conditioned policy. In a complimentary direction, we proposed in Chapter 7 to provide self-supervision to the agent by leveraging text-aligned video data, *i.e.*, , datasets of trajectories augmented with text captions. We showed that leveraging this type of offline data allows for learning language-conditioned agents, that can, given a natural language instruction, execute a task in the environment.

8.2 Perspectives for future work

This thesis takes place at the starting point of the field of self-supervision for learning goal-oriented agents. We now discuss some of the (subjectively) most exciting directions for future investigation in this field, as well as the practical limitations that remain.

Representation learning in RL. In the absence of supervision, and prior knowledge on the tasks, the agent needs to learn a representation of the environment that is useful for solving the task. Indeed, in goal-oriented RL, the agent needs to understand the goals that it must achieve, and for that, it needs to understand the structure of the environment from its observations and experience. In this thesis, we have shown several approaches for learning such representations, mainly based on the following self-supervised model: the reachability network [Savinov et al.,

2018b]. We have shown that this method has several advantages, including the fact that it can be trained on trajectories generated by a random policy, and that it can generalize well from RGB inputs, even in situations unseen during training. Other works have shown that self-supervised learning techniques could be applied to RL, such as the prototypical representations [Yarats et al., 2021] that draws inspiration from SwAV [Caron et al., 2020] and BYOL [Grill et al., 2020], or CURL [Laskin et al., 2020] that extracts features using contrastive learning [Henaff, 2020, He et al., 2020, Chen et al., 2020]. These results open up an exciting opportunity for future research in applying recent progress in self-supervised learning on images and videos to the field of RL.

RL as a sequential decision-making problem. The impressive progress of large language models (LLM) in the field of natural language processing, has brought a novel view point to RL. Indeed, training an agent to achieve a task with RL amounts to a sequential decision-making problem, where the agent needs to predict the best action at every timestep. The same models and recipes that are applied for LLMs, can therefore be applied to train RL agents, as done in recent work [Chen et al., 2021, Li et al., 2022b, Carta et al., 2023]. This opens up for promising opportunities, at the crossroads between RL and natural language processing, especially for tasks that involve learning agents with text-based modalities, such as WebShop [Fontoura et al., 2014] and NetHack [Küttler et al., 2020].

Datasets for RL and language. A crucial point for research in offline RL is the availability of datasets of pre-collected trajectories. Learning agents offline makes sense only when such datasets are available – the data collection process can otherwise be more complex than the downstream task itself [Endrawis et al., 2021]. These datasets can come from a number of sources, including human demonstrations [Chebotar et al., 2021], and trajectories generated automatically with specific policies [Laskin et al., 2021, Yarats et al., 2022]. In the absence of a reward annotation strategy, the question of how to exploit such data without supervision remains unclear. A promising direction, explored in Chapter 7, is to use offline data for which there exist a text-alignment modality. These can be of the form of human demonstrations with captions, as is the case for tutorial videos on YouTube. In the same way that progress in computer vision and natural language processing happened thanks to the availability of massive datasets [Deng et al., 2009, Chen et al., 2017], a revolution in RL can come from the our ability to harness billions of demonstrations. This perspective is more than ever possible, since the release of large datasets of videos on MineCraft [Fan et al., 2022, Baker et al., 2022].

Exploration vs. exploitation. The emergence of methods for learning agents from offline datasets raises the question of the importance of exploration. Reinforcement learning algorithms are indeed known to suffer from the exploration-exploitation

dilemma [Sutton, 1988, Colas et al., 2018], that is, finding the right balance between exploiting the learned policy by performing the best action, and exploring the environment to discover new states and actions. When learning an agent offline, however, the agent cannot further interact with the environment, and the exploration is therefore limited to state-action transitions available in the training set. As showed by Yarats et al. [2022], the quality of the learned agent is then highly correlated to that of the offline dataset. This, by essence, naturally disentangles the challenge of exploration from the problem of learning from offline datasets, which, as such, constitute distinct research directions.

Bibliography

Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding language to autonomously-acquired skills via goal generation. In *ICLR*, 2021.

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NeurIPS*, 2017.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34:26–38, 2017.

Gil Avraham, Yan Zuo, Thanuja Dharmasiri, and Tom Drummond. Empnet: Neural localisation and mapping using embedded memory points. In *ICCV*, 2019.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. In *NeurIPS*, 2022.

Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*, 2020.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Egomap: Projective mapping and structured egocentric memory for deep RL. In *ECML PKDD*, 2020a.

Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf. Learning to plan with uncertain topological maps. In *ECCV*, 2020b.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *NeurIPS*, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *ICLR*, 2018.

Andres Campero, Roberta Raileanu, Heinrich Kuttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. In *ICLR*, 2020.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. *arXiv preprint arXiv:2302.02662*, 2023.

Marvin Chancán and Michael Milford. MVP: Unified motion and visual self-supervised learning for large-scale robotic navigation. *arXiv preprint arXiv:2003.00667*, 2020.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *3DV*, 2017.

Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. In *ICLR*, 2018.

Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural SLAM. In *ICLR*, 2019.

Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *NeurIPS*, 2020a.

Devendra Singh Chaplot, Helen Jiang, Saurabh Gupta, and Abhinav Gupta. Semantic curiosity for active visual learning. In *ECCV*, 2020b.

Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological SLAM for visual navigation. In *CVPR*, 2020c.

Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, and Sergey Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. In *ICML*, 2021.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017.

Kevin Chen, Juan Pablo De Vicente, Gabriel Sepulveda, Fei Xia, Alvaro Soto, Marynel Vázquez, and Silvio Savarese. A behavioral approach to visual navigation with graph localization networks. *arXiv preprint arXiv:1903.00445*, 2019.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, 2021.

Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *ICLR*, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *NeurIPS*, 2005.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for gymnasium, 2018. URL https://github.com/Farama-Foundation/Minigrid.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *ICLR*, 2019.

Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137, 2001.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In *ICML*, 2018.

Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *ICML*, 2019.

Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Dominey, and Pierre-Yves Oudeyer. Language as a cognitive tool to imagine goals in curiosity driven exploration. In *NeurIPS*, 2020.

Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research*, 74:1159–1199, 2022.

André Correia and Luís A Alexandre. Hierarchical decision transformer. *arXiv preprint arXiv:2209.10447*, 2022.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *CVPR*, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Alessandro Devo, Giacomo Mezzetti, Gabriele Costante, Mario L Fravolini, and Paolo Valigi. Towards generalization in target-driven visual navigation by using deep reinforcement learning. *IEEE Transactions on Robotics*, 36:1546–1561, 2020.

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CoRL*, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Shadi Endrawis, Gal Leibovich, Guy Jacob, Gal Novik, and Aviv Tamar. Efficient self-supervised data collection for offline robot learning. In *ICRA*, 2021.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2018.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *NeurIPS Datasets and Benchmarks Track*, 2022.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *CVPR*, 2019.

Marcus Fontoura, Wolfgang Pree, and Bernhard Rumpe. The webshop e-commerce framework. *arXiv preprint arXiv:1409.6596*, 2014.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Jonas Gehring, Gabriel Synnaeve, Andreas Krause, and Nicolas Usunier. Hierarchical skills for efficient exploration. In *NeurIPS*, 2021.

Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. In *ICCV*, 2019.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent-a new approach to self-supervised learning. In *NeurIPS*, 2020.

Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gomez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matt Hoffman, Ofir Nachum, George Tucker, Nicolas Heess, and Nando de Freitas. RL unplugged: A suite of benchmarks for offline reinforcement learning. In *NeurIPS*, 2020.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017a.

Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Unifying map and landmark based representations for visual navigation. *arXiv preprint arXiv:1712.08125*, 2017b.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. Dynamical distance learning for semi-supervised and unsupervised skill discovery. In *ICLR*, 2019.

Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *ICML*, 2019.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *ICML*, 2020.

Joao F Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *CVPR*, 2018.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*, 2020.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NeurIPS*, 2016.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, 2021.

Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *CVPR*, 2018.

Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 2020.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, 1993.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

Yuji Kanagawa. mujoco-maze, 2021. URL https://github.com/kngwyu/mujoco-maze.

Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. Memory augmented control networks. In *ICLR*, 2018.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23:4909–4926, 2021.

Ashish Kumar, Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path following. In *NeurIPS*, 2018.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In *NeurIPS*, 2020.

Nathan Lambert, Markus Wulfmeier, William Whitney, Arunkumar Byravan, Michael Bloesch, Vibhavari Dasagi, Tim Hertweck, and Martin Riedmiller. The challenges of exploration for offline reinforcement learning. *arXiv preprint arXiv:2201.11861*, 2022.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *ICML*, 2020.

Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. URLB: Unsupervised reinforcement learning benchmark. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2021.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Jinning Li, Chen Tang, Masayoshi Tomizuka, and Wei Zhan. Hierarchical planning through goal-conditioned offline reinforcement learning. *IEEE Robotics and Automation Letters*, 2022a.

Shuang Li, Xavier Puig, Yilun Du, Clinton Wang, Ekin Akyurek, Antonio Torralba, Jacob Andreas, and Igor Mordatch. Pre-trained language models for interactive decision-making. In *NeurIPS*, 2022b.

Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A survey on transformers in reinforcement learning. *arXiv preprint arXiv:2301.03044*, 2023.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Hao Liu and Pieter Abbeel. Aps: Active pretraining with successor features. In *ICML*, 2021.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. In *IJCAI: Survey Track*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *IJCAI*, 2019.

Yecheng Jason Ma, Jason Yan, Dinesh Jayaraman, and Osbert Bastani. How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression. *arXiv preprint arXiv:2206.03023*, 2022.

Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. Improving vision-and-language navigation with image-text pairs from the web. In *ECCV*, 2020.

Oleksandr Maksymets, Vincent Cartillier, Aaron Gokaslan, Erik Wijmans, Wojciech Galuba, Stefan Lee, and Dhruv Batra. THDA: Treasure hunt data augmentation for semantic navigation. In *CVPR*, 2021.

Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. In *NeurIPS*, 2021.

Lina Mezghani, Sainbayar Sukhbaatar, Arthur Szlam, Armand Joulin, and Piotr Bojanowski. Learning to visually navigate in photorealistic environments without any supervision. *arXiv preprint arXiv:2004.04954*, 2020.

Lina Mezghani, Piotr Bojanowski, Karteek Alahari, and Sainbayar Sukhbaatar. Walk the random walk: Learning to discover and reach goals without supervision. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022a.

Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski, Alessandro Lazaric, and Karteek Alahari. Learning goal-conditioned policies offline with self-supervised reward shaping. In *CoRL*, 2022b.

Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari. Memory-augmented reinforcement learning for image-goal navigation. In *IROS*, 2022c.

Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski, and Karteek Alahari. Think before you act: Unified policy for interleaving language reasoning with actions. In *ICLR Workshop on Reincarnating Reinforcement Learning*, 2023.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017.

Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, and Raia Hadsell. Learning to navigate in cities without a map. In *NeurIPS*, 2018.

Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NeurIPS Workshop on Deep Learning*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31: 1147–1163, 2015.

Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. Task-agnostic exploration via policy gradient of a non-parametric state entropy estimate. In *AAAI*, 2021.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *NeurIPS*, 2018.

Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *NeurIPS*, 2019.

Duy Nguyen-Tuong, Jan Peters, Matthias Seeger, and Bernhard Schölkopf. Learning inverse dynamics: a comparison. In *European Symposium on Artificial Neural Networks*, 2008.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *ICML*, 2016.

OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique Ponde de Oliveira Pinto, Alex Paino, Noh Hyeonwoo, Lilian Weng, Qiming Yuan, Casey Chu, and Wojciech Zarembe. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*, 2021.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *ICML*, 2017.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *ICLR*, 2018.

Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *IJCAI*, 1995.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *ICML*, 2019.

Alexandre Péré, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. In *ICLR*, 2018.

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *ICML*, 2020.

Vitchyr Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *ICML*, 2020.

Aaron L Putterman, Kevin Lu, Igor Mordatch, and Pieter Abbeel. Pretraining for language conditioned imitation with transformers. In *NeurIPS Workshop on Offline Reinforcement Learning*, 2021.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., 1995.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. In *RSS*, 2017.

Hongrui Sang, Rong Jiang, Zhipeng Wang, Yanmin Zhou, and Bin He. A novel neural multi-store memory network for autonomous visual navigation in unknown environment. *IEEE Robotics and Automation Letters*, 2022.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018a.

Nikolay Savinov, Anton Raichuk, Damien Vincent, Raphael Marinier, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In *ICLR*, 2018b.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *CVPR*, 2019.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *ICML*, 2015.

Jürgen Schmidhuber. Curious model-building control systems. In *IJCNN*, 1991.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

Freek Stulp and Olivier Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4:49–61, 2013.

Sainbayar Sukhbaatar, Emily Denton, Arthur Szlam, and Rob Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning. *arXiv preprint arXiv:1811.09083*, 2018a.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *ICLR*, 2018b.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 2017.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45:52–57, 2002.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017.

Nicola Tomatis, Illah Nourbakhsh, and Roland Siegwart. Combining topological and metric: A natural integration for simultaneous localization and map building. In *European Workshop on Advanced Mobile Robots*, 2001.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Srinivas Venkattaramanujam, Eric Crawford, Thang Doan, and Doina Precup. Self-supervised learning of distance functions for goal-conditioned reinforcement learning. *arXiv preprint arXiv:1907.02998*, 2019.

David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised control through non-parametric discriminative rewards. In *ICLR*, 2018.

Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18:620–634, 2010.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2019.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *EMNLP: System Demonstrations*, 2020.

Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuandong Tian. Bayesian relational memory for semantic visual navigation. In *ICCV*, 2019.

Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, 2018.

Rui Yang, Yiming Lu, Wenzhe Li, Hao Sun, Meng Fang, Yali Du, Xiu Li, Lei Han, and Chongjie Zhang. Rethinking goal-conditioned supervised learning and its connection to offline rl. In *ICLR*, 2022.

Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *ICML*, 2021.

Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. In *ICLR Workshop on Generalizable Policy Learning in Physical World*, 2022.

Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur Szlam. Composable planning with attributes. In *ICML*, 2018.

Jesse Zhang, Karl Pertsch, Jiahui Zhang, Taewook Nam, Sung Ju Hwang, Xiang Ren, and Joseph J Lim. Sprint: Scalable semantic policy pre-training via language instruction relabeling. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2022.

Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural SLAM: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

Xiaoming Zhao, Harsh Agrawal, Dhruv Batra, and Alexander G Schwing. The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation. In *ICCV*, 2021.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017.