Unsupervised Pretraining of State Representations in a Rewardless Environment

Astrid Merckling

Unsupervised Pretraining of State Representations in a Rewardless Environment

by Astrid Merckling

advised by Pr. Stéphane Doncieux,
Dr. Nicolas Perrin-Gilbert and
Dr. Alexandre Coninx

Institute for Intelligent Systems and Robotics (ISIR)
CNRS, Sorbonne University, Paris
École Doctorale Informatique, Télécommunications et Électronique de Paris (EDITE)

PhD thesis in Computer Science and Robotics
Presented and publicly defended on September 22, 2021

In front of a jury composed of:

David Filliat	Professor	U2IS, ENSTA Paris	(reviewer)
Alain Dutech	Research Fellow	LORIA, INRIA	(reviewer)
Catherine Achard	Professor	Sorbonne Université	(examiner)
Alban Laflaquière	Senior Researcher	SoftBank Robotics Europe	(examiner)
Stéphane Doncieux	Professor	ISIR, Sorbonne Université	(advisor)
Nicolas Perrin-Gilbert	CNRS Researcher	ISIR, Sorbonne Université	(co-advisor)
Alexandre Coninx	Associate Professor	ISIR, Sorbonne Université	(co-advisor)

ABSTRACT

This thesis builds on the deep learning techniques that have recently achieved multiple successes for problems as diverse as the type of data used (images, text, DNA, etc.) in large-scale settings (i.e. with large amount of high-dimensional data) thanks to the computational improvements. Reinforcement learning (RL) has leveraged these successes to continuous control tasks in environments that are only partially known, i.e. without knowledge of the transition model, and recently without knowledge of the underlying compact state space that Markov decision processes (MDPs) usually provide. There are many improvements in this last class of methods called *end-to-end deep RL*, but they still remain computationally and memory intensive.

In order to facilitate the applicability of RL algorithms to these large-scale problems, this thesis builds on the burgeoning field of *state representation learning* (SRL). From a global perspective, it takes advantage of the popularity for deep unsupervised pretraining of input representations that offer improvements over input embeddings learned from scratch. Its solution exploits the accumulated experience of agents in the manner of transfer learning, i.e. where automatic prediction for new tasks is performed on examples from a different training distribution.

In this research, we have proposed two new SRL algorithms that address the general question of how to learn state embeddings that improve the performance of RL algorithms (i.e. a better convergence of the optimal policy in terms of computational cost, sampling efficiency and final performance) in large-scale settings and without access to the reward that is generally available. Each of these algorithms respectively answers the question (i) How to represent states so that an agent can perform tasks by imitation learning? (ii) How to represent states so that an agent can predict its near future (the next state and the next observation) by exploring the estimated uncertain unknown transitions?

Finally, this work is a way to improve all types of inputs for RL algorithms (not just image observations) through state representations which verify good properties for the efficiency of bootstrapping and other automatic decision making mechanisms also common to supervised learning.

Keywords

State Representation Learning, Pretraining, Exploration, Unsupervised Learning, Deep Reinforcement Learning

ACKNOWLEDGEMENTS

To my family, my supporters, my friends,

CONTENTS

ABSTRACT iv

1 INTRODUCTION 1
1.1 State Representation Learning . . . 3
1.2 A Brief History of State Estimation . . . 5
1.3 Thesis Outline . . . 6
1.3.1 Summary . . . 7

2 DEEP REINFORCEMENT LEARNING BACKGROUND 8
2.1 Introduction . . . 8
2.1.1 The Curse of Dimensionality . . . 8
2.2 Deep Learning . . . 9
2.2.1 Deep Learning Training Problems . . . 9
2.2.2 Deep Learning Approximators . . . 15
2.3 Reinforcement Learning . . . 19
2.3.1 Reinforcement Learning Formalism . . . 19
2.3.2 Temporal-Difference Learning . . . 25
2.3.3 Policy Gradient . . . 28

3 STATE REPRESENTATIONS FOR REINFORCEMENT LEARNING 32
3.1 Introduction . . . 32
3.1.1 Dimensionality Reduction . . . 33
3.2 Scaling Reinforcement Learning Towards Robotics . . . 36
3.2.1 Pros and Cons of End-to-End Deep Reinforcement Learning 37
3.2.2 Potential Solutions . . . 41
3.3 SRL Formulations . . . 45
3.3.1 Solution Criteria . . . 46
3.3.2 Learning Heuristics . . . 49
3.3.3 Exploration Strategies . . . 52
3.4 Conclusion . . . 54

4 STATE REPRESENTATION LEARNING FROM DEMONSTRATION 55
4.1 Introduction . . . 56
4.2 Related Work . . . 58
4.3 State Representation Learning from Demonstration . . . 58
4.3.1 Demonstrations . . . 58
4.3.2 Imitation Learning from Demonstrations . . . 59
4.4 Goal Reaching . . . 60
4.4.1 Experimental Setup . . . 61

4.4.2 Results and Discussion 63
4.5 Ballistic Projectile Tracking 65
4.5.1 Experimental Setup 66
4.5.2 Results and Discussion 71
4.6 Conclusion 73

5 EXPLORATORY STATE REPRESENTATION LEARNING 75
5.1 Introduction 76
5.2 Related Work 77
5.3 Proposed Method: XSRL 78
5.3.1 State Transition Estimator 78
5.3.2 Discovery in the Face of Uncertainty 80
5.3.3 Optimization Process 84
5.4 Experimental Setup 88
5.4.1 Baselines 89
5.4.2 Environment Details 91
5.4.3 Implementation Details 93
5.5 Experimental Results 97
5.5.1 Evaluations of XSRL Representations and Exploration . . . 97
5.5.2 XSRL Representations Transfer 102
5.6 Discussion 104
5.7 Conclusion 106

6 CONCLUSION 107
6.1 Conclusion 107
6.1.1 General Contributions 107
6.2 Discussion 109
6.2.1 Generalization and State Representation Properties 110

BIBLIOGRAPHY 115

CONTENTS

1

INTRODUCTION

Everything is number.

Pythagoras (570 BC – 495 BC)

The last decade has seen immense progress in machine learning to solve automatic decision making tasks. Indeed, with their data-driven learning principle, learning systems have taken advantage of huge datasets, computational hardware/software improvements, to extract information from data and convert it into numerical knowledge to perform automatic predictions. There are three ways to optimize these learning algorithms, but they can help each other [Jordan and Mitchell, 2015]. However, the difference between them is not as clear as it first appears [Jordan and Rumelhart, 1992].

(1) Supervised learning is similar to function approximation problems, where the training examples are input-output pairs ($\mathbf{x}$, $\mathbf{y}$). Supervised learning systems aim to learn a mapping between $\mathbf{x}$ and $\mathbf{y}$, with the objective of minimizing the expected prediction errors. $\mathbf{x}$ can be of different nature, ranging from low-dimensional vectors to multidimensional inputs such as images, DNA sequences, text, etc. [LeCun et al., 2015]. In particular, it is deep learning that has enabled advances in so many diverse applications. Deep learning consists of multi-layer networks that learn representations to extract features of increasing level of abstraction as the layer approaches the output. Usually, machine learning practitioners used to perform a feature extraction phase before the automatic decision making task [Schmidhuber, 2015]. Today, it is more popular to let deep learning systems automatically learn their representations and make predictions. Learning systems trained without pretraining are said to be end-to-end trainable. In particular, they have led to major changes in the way data is analyzed in computer vision [Krizhevsky et al., 2012].

(2) Reinforcement learning (RL) is akin to problems nested in the fields of control theory and machine learning, which typically involve sequential decision making. It is typically formalized as the optimal control of incompletely known Markov decision processes (MDPs), where the training examples are obtained sequentially through agent-environment interactions to cope with the unknown transition model of MDPs [Sutton and Barto, 2018]. These training examples are transitions which consist of an action performed by an agent and the resulting new state and reward. A state determines what an agent knows about himself (proprioception) and his environment (perception), and a reward is an evaluative feedback provided by the environment. RL systems aim to learn a control strategy

called optimal policy, which is trained to choose actions for any given state, with the objective of maximizing the expected cumulative reward over time. They do this by giving each action a correct credit for its contribution to the cumulative rewards. Thus, unlike supervised systems, they automatically annotate the training examples with intermediate feedbacks provided by the environment.

Traditionally, RL systems rely on specific low-dimensional states with Markovian transitions to ensure full observability [Sutton and Barto, 2018]. In order to make robotics increasingly autonomous, learning systems need to exploit physical information from the real world using various sensors such as cameras. However, such states may be difficult to obtain from raw sensory data, especially without the assumption of the a priori knowledge of the ground truth state, and that complex transitions with distractors are most often present in the environment. It is in this more realistic setting corresponding to one of the most popular scaling, in which we place ourselves in this thesis [Li, 2018, François-Lavet et al., 2018]: simulation-based control tasks with continuous action spaces of up to ten dimensions, and continuous observation spaces formed by camera images. As with supervised learning, end-to-end deep learning has led to great progress in adapting RL systems to such settings in order to represent the necessary state properties in the intermediate layers [Barth-Maron et al., 2018, Kostrikov et al., 2020]. These systems are known as end-to-end deep RL (DRL) and have had great impact in early applications with image observations and discrete actions [Mnih et al., 2015].

(3) Unsupervised learning tackles the problems of learning input embeddings by leveraging unlabeled data. Specifically, unlike the previous two machine learning paradigms, where systems learn representations for a specific automatic decision task, in this paradigm systems learn representations without any supervision or reward [Ghahramani, 2003]. These systems generally make assumptions about the structural properties of the data to better retrieve higher-level features. For example, the assumption that the data lies in a manifold with small intrinsic dimensionality is used by various dimension reduction methods such as PCA (principal component analysis), manifold learning, or autoencoders that make different assumptions about the underlying manifold (e.g. that it is a linear subspace or a smooth nonlinear manifold) [Ghahramani, 2003].

From a general viewpoint, these three paradigms allow to train learning systems as: (1) to reproduce known knowledge (labeled training examples), (2) to discover control strategies (optimal policies) with knowledge available only through interaction (MDPs), (3) to discover previously unknown knowledge from the data. While (1) and (2) are concerned with learning representations for specific tasks of automatic prediction, (3) is concerned with learning useful representations of the input which can be used advantageously by new learning systems. In particular, this thesis will study approaches for pretraining state estimators to generate inputs for RL algorithms. The idea of embedding is to extract the a priori abstract and non-numerical concepts from data into numer-

ical and actionable representations to provide better inputs for new learning systems. To do this, embedding learning immerses data points in a numerical space in order to retrieve them in a machine-readable language. This is what mathematics for computer science allows from the start: privileging numbers as inputs and outputs of algorithms, which motivates our study. Our intuition is that high-dimensional observations of autonomous agents are not an issue *per se* because their intrinsic degrees of freedom matter much more. Deep unsupervised learning would help learn mappings that attempt to "iron" complex manifolds of relevant information into low-dimensional embeddings [Ghahramani, 2003, Van Der Maaten et al., 2009].

This thesis is part of the work on unsupervised input representation pretraining, which can be exploited in transfer learning by new learning systems trained on examples from another data distribution [Bengio, 2012]. These transfer learning methods can leverage huge unlabeled raw datasets to efficiently learn low-dimensional input embeddings, provided that the approximator class matches the complexity of the problem and sufficient computational resources are available during pretraining [Bengio et al., 2007]. In the literature, works to better structure the inputs of learning systems by pretraining input embeddings have been done for several reasons: simplicity, robustness, to overcome the curse of dimensionality, to improve performance with fewer training examples [Hadsell et al., 2006, Collobert et al., 2011, Le, 2013, Jonschkowski and Brock, 2015, Anselmi et al., 2014].

For example, in natural language processing and robotics, the trend is to represent more and more abstract concepts by numbers. On the one hand, in language processing, the scientific explosion of word embedding pretraining approaches popularized by word2vec [Mikolov et al., 2013] and all its variants like GloVe ([Pennington et al., 2014]), BERT ([Devlin et al., 2018]), RoBERTa ([Liu et al., 2019b]), FlauBert ([Le et al., 2019]), camemBERT ([Martin et al., 2019]), have disrupted the previous word embeddings learned from scratch. Indeed, these pretrained word embedding vectors offer better representations of words, sentences, paragraphs, and documents, bringing tremendous progress in all kinds of automatic prediction tasks such as translation, speech recognition, sentiment analysis, document summarization, question answering, language generation, etc. [Young et al., 2018].

On the other hand, in the same years, robotics has operated the same kind of upheaval, where huge datasets consisting of the interaction experience between agents and the environment are exploited to pretrain state embeddings. This is the birth of a whole field of research corresponding to unsupervised pretraining of state estimators popularized by Jonschkowski and Brock [2013] as state representation learning (SRL) and all its variants like [Dwibedi et al., 2018, Ha and Schmidhuber, 2018, de Bruin et al., 2018, Yarats et al., 2019, Sax et al., 2019], that have disrupted previous input embeddings learned from scratch with end-to-end DRL algorithms on various real and simulated robotic tasks.

1.1 STATE REPRESENTATION LEARNING

This thesis will mainly consider SRL for state estimation in the rewardless robotic context. The solution typically takes the form of a state-update function that leverages the accumulated experience of multiple agents to estimate states whose transitions are Markovian, and from which as much unnecessary information is removed as possible. This is intended to make the states simpler and lower dimensional than the observations, and more generally than the history of an agent's trajectory.

Transferring such state representations discovered from task-agnostic experience has many merits. First, it would allow robotics to break free from the limitation of solving tasks known in advance, i.e. to be able to autonomously handle unknown tasks. Second, it is an effective way to move beyond data-hungry versions of end-to-end DRL algorithms [Glasmachers, 2017]. Third, it would effectively train systems to cope with the intrinsic challenges of RL, such as credit assignment due to long-horizon tasks, instability due to stochasticity, the exploration/exploitation tradeoff, complex and unknown environment transitions, and the curse of dimensionality [Henderson et al., 2018, Li, 2018]. Finally, it would be an attempt to recover the theoretically well-studied framework of RL defined with states represented in a very relevant way [Sutton and Barto, 2018].

One of the main challenges in SRL, which arises in many unsupervised learning systems, often more precisely referred to as self-supervised learning systems [Liu et al., 2020], is the choice of a training objective. There are different ways to approach the issue of providing additional information about the SRL solution structure rather than simply using the supervision of a compression signal as is the case with the autoencoder approach [Bourlard and Kamp, 1988]. This can be addressed under the notion of multiple agents that leverage knowledge gained from previous tasks, as it is the case with SRLfD (State Representation Learning from Demonstration), one of the algorithms proposed in this thesis that we present in Chapter 4. Or under the notion of predictability of the next observations by learning a state transition estimator, as it is the case with XSRL (eXploratory State Representation Learning), the other algorithm proposed in this thesis that we present in Chapter 5.

Another main challenge, specific to the SRL framework, is the choice of a task-agnostic exploration strategy (i.e. without extrinsic reward) for autonomous agents to discover their environment [Sutton and Barto, 2018, Lesort et al., 2018]. The agents' experience consists of a collection of sequential agent-environment interactions without extrinsic reward and with a non-stationary state distribution, as the state estimator is trained continuously. While this exploration can be done manually via demonstrations of various transitions, it is complicated to do so in an unsupervised learning context, i.e. without the a priori knowledge of the task. This thesis considers new ways to automate it. Such strategies are intensively studied in the RL context to solve the exploration/exploitation tradeoff for learn-

ing an optimal policy with good generalization performance. In contrast, they have not yet been well studied in the SRL context to learn a state representation generalizable to unknown tasks. In other words, a good exploration strategy in this context should take advantage of the training examples automatically generated by agents and seek for new observations to maximize the generalization likelihood of the learned representation. We propose two approaches, one that generalizes to various tasks: SRLfD, and one that generalizes to complex transitions and from which the model can improve: XSRL.

1.2 A BRIEF HISTORY OF STATE ESTIMATION

Understanding the current state of controlled systems is part of the requirement to learn tasks, such as those of RL. In a context where the state is replaced by a sensory measurement, RL can often be made more efficient by solving first a state estimation problem [Tesauro, 1992, Bertsekas and Tsitsiklis, 1996]. One of the first approaches was proposed by Kalman [1960b] that assumes knowledge of the transition model which must be linear with respect to the state. However, it has been adapted to the nonlinear context with the extended Kalman filter (EKF) [Ljung, 1979]. A more recent type of technique dedicated to navigation tasks is SLAM (Simultaneous Localization and Mapping [Bailey and Durrant-Whyte, 2006a,b]) which provides representations that contain the location and orientation of an agent.

These traditional state estimation techniques have significantly increased the autonomy and intelligence of robots. However, in their usual formulations, they assume the a priori knowledge of the transition model, and the ground truth state in order to define an observation model. Moreover, their state representations focus on self-localization and ego-motion estimation, which may be insufficient for complex tasks that require understanding more abstract concepts of the environment. Moreover, these traditional methods require experts to tune their parameters, which is difficult due to the "curse of manual tuning" [Cadena et al., 2016]. As a collateral effect, they tend to be non-generalizable across tasks, as cross-validation is not an option in an unsupervised learning context.

Feature engineering methods have relaxed the assumption of the a priori knowledge of the transition model and the ground truth state by being solely task specific. Among the most popular techniques are SIFT [Lowe, 1999] and SURF [Bay et al., 2006]. However, it remains challenging to devise ad hoc methods that extract higher-level and more abstract features, even when the task is known. Moreover, in a context of ever-increasing automation, it is necessary to remove the assumption of the a priori knowledge of the task and the environment. This is where SRL comes in, automating the feature engineering process by leveraging experiences discovered by agents to pretrain state estimators [Lesort et al., 2018]. Such pretrained state embeddings can represent numerically in vectors all the concepts of the input, from the least to the most abstract. Thanks to the data-

driven principle of machine learning, the designer remains outside the learning loop, and thus the required expert knowledge only concerns the hyperparameters of the SRL algorithms.

This thesis seeks to extend the capabilities of SRL to help scale DRL algorithms to continuous control tasks with high-dimensional sensory observations. Although end-to-end DRL has recently performed well on such tasks, it relies on many computational resources [Barth-Maron et al., 2018]. This makes it impractical to apply DRL to such tasks with computational restrictions, as in real-world robotics. SRL allows to improve the performance of DRL by providing it with better inputs than the input embeddings learned from scratch with end-to-end strategies. Specifically, this thesis addresses the problem of performing state estimation in the manner of deep unsupervised pretraining of state representations without reward. These representations must verify certain properties to allow for the correct application of bootstrapping and other decision making mechanisms common to supervised learning, such as being low-dimensional and guaranteeing the local consistency and topology (or connectivity) of the environment [Penedones et al., 2018, Morik et al., 2019], which we will seek to achieve through the models pretrained with the two SRL algorithms proposed in this thesis.

1.3 THESIS OUTLINE

In Chapter 2, we provide the background necessary to understand the deep unsupervised pretraining of state embeddings. This takes place in the context of large-scale function approximation problems involved in popular applications of DRL. Once we have a clear understanding of the curse of dimensionality (Section 2.1) (which is ubiquitous in control systems with high-dimensional continuous inputs), we explain the main components of a machine learning algorithm, especially when deep learning techniques are used (Section 2.2). Finally, we present RL and its extension to deep learning approximators, i.e. DRL algorithms, with special attention to the one we use throughout our experiments – SAC (Soft Actor-Critic [Haarnoja et al., 2018b]) – (Section 2.3).

In Chapter 3, we review the main approaches to state representation for the popular large-scale control tasks studied in this thesis with RL that corresponds to robotics in simulation with image observations and continuous actions. We start by recalling works in representation learning with dimensionality reduction to try to overcome the curse of dimensionality in such problems (Section 3.1). We then discuss how end-to-end DRL scales up, but still with limitations (Section 3.2). We then present an effective scaling alternative that is the unsupervised state estimator pretraining known as – state representation learning (SRL). To do so, we lay the foundations of its framework through three main elements: solution criteria (Section 3.3.1), learning heuristics (Section 3.3.2), and exploration

strategies (Section 3.3.3). In the next two chapters, we present two SRL methods that propose a novelty in each of these elements.

In Chapter 4, we introduce the first approach called SRLfD (State Representation Learning from Demonstration). The learned state must here verify the criterion that the agents' experience is represented in a way which abstracts the concepts common to multiple tasks by guaranteeing the Markovianity of state transitions. To this end, SRLfD follows the imitation learning objective of oracle policies on various tasks as a learning heuristic. In this way, the exploration strategy is realized by oracle policies whose diversity guarantees the one of the experience. We evaluate the performance of our pretrained SRLfD models on two control tasks: goal reaching and ballistic projectile tracking. We show comparisons with classical Kalman filtering, popular end-to-end RL, and other representation strategies. Some of the work in this chapter has been presented previously in [Merckling et al., 2020], but it also contains new work in Section 4.5.

In Chapter 5, we introduce the second approach called XSRL (eXploratory State Representation Learning). The learned state must here verify the criterion that the agents' experience is represented in a way which abstracts the concepts to the next observation prediction task by guaranteeing the Markovianity of state transitions. To this end, XSRL follows the next observation prediction objective as a learning heuristic. To ensure an effective exploration, the first training procedure is coupled with a second procedure that trains discovery policies to maximize intrinsic rewards formed by (i) prediction errors of an inverse model trained concurrently in parallel, and (ii) k-step learning progress bonuses of the state representation model. In this way, the trained policies tend to generate transitions that are diverse in controllability and from which the state estimator can be trained efficiently. We first measure the exploration performance of XSRL and compare it to random exploration and entropy maximization strategy. We then evaluate the performance of our pretrained XSRL models on new RL applications which consist of three control tasks: maze navigation, pendulum swing up, and sprint locomotion. We show comparisons with the state-of-the-art RAE (Regularized Autoencoder [Ghosh et al., 2019]) approach, and other representation strategies.

1.3.1 *Summary*

The main chapters of this thesis can be summarized as follows.

- In Chapter 2, the elements to understand DRL and more globally the deep learning techniques used in our research are presented.
- In Chapter 3, the embedding learning methods for DRL are reviewed by developing those in the SRL domain through three major elements: solution criteria, learning heuristics and exploration strategies.

- In Chapter 4, we present a new SRL algorithm (SRLfD) for pretraining state representations from imitation learning of multiple oracle policies sufficiently diverse to provide effective exploration.
- In Chapter 5, we present another new SRL algorithm (XSRL) for pretraining state representations from learning by next observation prediction of the most diverse and learnable unknown transitions.
- In Chapter 6, we summarize the contributions and results of this thesis and the questions they led us to ask.

2

DEEP REINFORCEMENT LEARNING BACKGROUND

> The tool which serves as intermediary between theory and practice, between thought and observation, is mathematics; it is mathematics which builds the linking bridges and gives the ever more reliable forms.
>
> David Hilbert, 1930 *in* The Mind of the Mathematician (2007)

2.1 INTRODUCTION

During the last twenty years, the availability of large and high-dimensional data has dramatically increased. This has raised new questions for data representation in different mathematical domains, added to those already posed in the field of traditional statistical data analysis. Donoho et al. [2000] characterized these new large-scale data analysis problems with *the curse of dimensionality* and *the blessings of dimensionality*. On the one hand, the curse of dimensionality is a central issue, which largely challenges traditional mathematical theories. On the other hand, the blessings of dimensionality are beneficial, they could explain in part the surprising performance of deep learning techniques.

2.1.1 *The Curse of Dimensionality*

Bellman [1961] first coined *the curse of dimensionality* in optimal control problems (or sequential decision problems) to describe the intractability of optimizations involving exhaustive enumerations of high-dimensional spaces. For pedagogical reasons, Bellman explains this problem with this simple illustrative example: a $1/10$ spacing grid on a D-dimensional unit cube has $N = 10^D$ points, which grows exponentially with D. Specifically, this spacing grid may correspond to a D-dimensional discretized state space. Thus, covering all state instances and computing their evaluation simultaneously becomes intractable with the increase of the state dimension, whether it is discrete or continuous. Bellman [1961] first proposed to overcome this problem by using new exhaustive search strategies, in particular with his own method: dynamic programming.

The curse of dimensionality appears in many areas of computer science, in particular Donoho et al. [2000] identified: optimization, function approximation and numerical integration. The main formulation of the curse of dimensionality by Donoho et al. [2000] is as follows:

CURSE OF DIMENSIONALITY If we must approximatively optimize a function of D variables, assumed to be Lipschitz, then to have a tolerance error $0 < \epsilon < 1$ on the approximation objective, we need order $N_\epsilon = (1/\epsilon)^D$ evaluations on the discretized space.

In comparison to the previous illustration, the tolerance error ϵ refers to the spacing grid on the D-dimensional unit cube, while N_ϵ refers to the number of points we need to enumerate all the configurations of this discretized space. For example, to naively approximate a function with a tolerance error $\epsilon \approx 10^{-2}$, in low-dimensionality regimes $D = 1$,2 or 3, the number of required samples becomes unreasonably large with $N_\epsilon = 10^2$, 10^4 or 10^6. Therefore, in settings with high-dimensional data, the number of training examples will never be large enough to naively approximate a function with a neural network.

2.2 DEEP LEARNING

In what follows, we explain the main components used by deep learning techniques. We first describe the optimization problem encountered by their training procedure and the two main tools it uses: stochastic gradient descent and the backpropagation algorithm. Next, we present the most basic deep learning approximators – *feedforward neural networks* – (abbreviated by FNNs), which have given a very popular variant – *convolutional neural networks* – (abbreviated by CNNs). Although this explanation of deep learning techniques is not exhaustive, it may be sufficient to understand the pleasant end-to-end learning perspectives they offer for scaling reinforcement learning algorithms towards robotics (see Section 3.2).

2.2.1 *Deep Learning Training Problems*

Deep learning training problems correspond to optimization problems that are solved by fitting a neural network parameters $\boldsymbol{\theta}$ (a.k.a. weights) to minimize an objective function. An objective function is defined with a loss function ℓ, which in the case of regression can for example be the mean square error (MSE), i.e. based on the L_2 norm. It measures the difference between the output of the target function f and that of the neural network $f_{\boldsymbol{\theta}}$ (a.k.a. prediction), as follows:

$$\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 \tag{1}$$

where $(\mathbf{x}, \mathbf{y})$ corresponds to the input-output pair obtained with evaluation of the unknown target function ($f(\mathbf{x}) = \mathbf{y}$). In practice, the objective function

corresponds to the *empirical risk*, which is computed on a finite training dataset as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \ell\big(f_{\boldsymbol{\theta}}(\mathbf{x}_n), \mathbf{y}_n\big) \tag{2}$$

where $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{1 \leq n \leq N}$ is a training dataset of cardinality N formed from different input-output pairs. Due to the empirical nature of this objective function, training a neural network is similar to an approximate optimization [Bottou and Bousquet, 2008].

Theoretically, an objective function corresponds to the *expected risk* on an unlimited training dataset according to a data distribution $P_{\mathbf{X}}\colon \mathcal{X} \to [0,1]$ (where $\mathcal{X} \subset \mathbb{R}^D$ is the data space) from which input-output pairs $(\mathbf{x}, f(\mathbf{x}))$ are drawn, as follows:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \int_{\mathbf{x}\in\mathcal{X}} P_{\mathbf{X}}(\mathbf{x})\ell\big(f_{\boldsymbol{\theta}}(\mathbf{x}), f(\mathbf{x})\big)d\mathbf{x} \tag{3}$$

However, the knowledge of $P_{\mathbf{X}}$ and the computation of the full integral is impossible in practice. This is why minimizing Eq. 3 is impractical in most problems.

Following Bottou and Bousquet [2008], we assume that the empirical risk $\mathcal{L}(\boldsymbol{\theta})$ and the expected risk $\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$ have a global minimum at $\boldsymbol{\theta}_*$ and $\boldsymbol{\theta}_{\mathcal{D}}$ respectively. Moreover, Bottou and Bousquet [2008] let $\hat{\boldsymbol{\theta}}$ be the practical solution of the empirical risk given by an optimization algorithm. The quality of this solution depends on the time budget $T_{\max}$ needed to obtain a tolerance error $\varepsilon_{T_{\max}}$ such that:

$$\mathbb{E}\left[\mathcal{L}(\hat{\boldsymbol{\theta}}) - \mathcal{L}(\boldsymbol{\theta}_*)\right] < \varepsilon_{T_{\max}} \tag{4}$$

Ideally, the solution $\hat{\boldsymbol{\theta}}$ should yield a good generalization performance. This is expressed by a low error rate of the objective function on the expectation of test datasets (i.e. datasets not used in the optimization process). This is the main goal of any statistical learning method: to approximate a target function using a training dataset to ensure good generalization performance on new samples drawn from the same data distribution. For the sake of clarity, we represent this generalization performance with the expectation $\mathbb{E}\big[\mathcal{L}(\hat{\boldsymbol{\theta}})\big]$.

According to Bottou and Bousquet [2008], through the three different solutions to an optimization problem, the generalization error obtained with a practical solution can be decomposed as follows:

$$\mathbb{E}\left[\mathcal{L}(\hat{\boldsymbol{\theta}})\right] = \underbrace{\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{\mathcal{D}})\right]}_{\mathcal{E}_{\text{app}}} + \underbrace{\mathbb{E}\big[\mathcal{L}(\boldsymbol{\theta}_*) - \mathcal{L}(\boldsymbol{\theta}_{\mathcal{D}})\big]}_{\mathcal{E}_{\text{est}}} + \underbrace{\mathbb{E}\big[\mathcal{L}(\hat{\boldsymbol{\theta}}) - \mathcal{L}(\boldsymbol{\theta}_*)\big]}_{\mathcal{E}_{\text{opt}}} \tag{5}$$

These three error terms allow to show different trade-offs between the choice of the neural network architecture, the number of training examples N, and the desired error rate of the optimization algorithm $\varepsilon_{T_{\max}}$ [Bottou and Bousquet, 2008].

$\mathcal{E}_{\text{app}}$ is the *approximation error* that measures how well functions of the approximator class can approximate the target function f. Since in practice f is most often outside this class, the more complex the approximator class, the smaller the term is according to the theoretical universal approximation results [Cybenko, 1989, Hornik et al., 1990, Pinkus, 1999].

$\mathcal{E}_{\text{est}}$ is the *estimation error* that measures the consequence of minimizing the empirical risk instead of the expected risk. It reflects the generalization performance influenced by the function class and the training dataset of cardinality N. Since N is limited in practice, the dataset cannot cover the entire actual data distribution. Furthermore, the number of samples required for a reasonable function approximation increases with the class complexity of the approximator. Thus, for these two reasons, the larger N and the simpler the approximator class, the smaller the term.

$\mathcal{E}_{\text{opt}}$ is the *optimization error* that measures the consequence of a time-limited optimization solution. It reflects the generalization performance influenced by N, and the properties of the optimization algorithm (convergence speed and cost per iteration). Because the optimization algorithm is time-limited in practice, the more sample efficient and the lower the cost per iteration, the smaller the term.

The two terms $\mathcal{E}_{\text{app}}$ and $\mathcal{E}_{\text{est}}$ constitute the *approximation-estimation tradeoff* (a.k.a. *bias-variance tradeoff*) where high bias is similar to high approximation error known as underfitting, and high variance is similar to high estimation error known as overfitting. In our large-scale setting (i.e. high data dimensionality D and high cardinality N) the generalization performance depends mainly on the optimization algorithm properties and the time budget T_{max}. Bottou and Bousquet [2008], Bottou et al. [2018], Hardt et al. [2016] theoretically analyze the generalization performances of several optimization algorithms, however they do not consider the regime of high-dimensional data. Their results prove that stochastic gradient descent optimization algorithms compare favorably with more sophisticated methods such as Newton's algorithm. This is due to the fact that the former has a cost per iteration independent of N, which is not the case for the latter. Moreover, the computational system, which corresponds to the computational hardware and software, determines the speed of the calculations. Thus, the more sophisticated the computational system is, the faster the iterations are executed.

Wang et al. [2020] propose a detailed survey on the different approaches to scaling machine learning algorithms by the three possible ways mentioned above. Indeed, the various works generally act on these three aspects to scale-up the machine learning algorithms: (i) add hypotheses about the function class to which the approximator belongs, (ii) improve the properties of the optimization algorithm (i.e. the cost per iteration and the convergence time), (iii) improve the resources deployed for numerical computations (in particular with the automatic differentiation algorithms and the parallelization of calculations). All these

improvements have allowed the development of deep learning techniques in the large-scale setting.

The first aspect – assumptions underlying the function class – makes it possible to avoid the exponential increase in the number of parameters needed to approximate high-dimensional target functions. In particular, deep neural networks are able to reach approximation error rates with their number of parameters that depends polynomially on the data dimensionality, whereas for their shallow counterparts it depends exponentially (curse of dimensionality in the parameter space of the approximator) [Cichocki et al., 2017]. This has the benefit of also reducing the estimation error as overfitting is less of a problem with fewer parameters in the approximator (i.e. with a simpler approximator). In the rest of this section, we will present the two components that improve deep learning systems on the two other aspects.

2.2.1.1 *Stochastic Gradient Descent*

The second aspect – properties of optimization algorithms – makes it possible to avoid the exponential increase in the number of calculations needed to approximate high-dimensional target functions. In particular, the stochastic gradient descent (an instance of stochastic optimization algorithms) can find solutions with low estimation error rates without exploring the whole parameter space of a neural network [Zhang et al., 2017a, Nguyen and Hein, 2018], thus reducing the training time needed for convergence.

Historically, Robbins and Monro [1951] proposed the stochastic optimization, which allows learning function approximators on large discrete training datasets, by mixing statistics and optimization ideas. Robbins-Monro's theorem shows that the objective function no longer needs to be evaluated directly but only with an unbiased estimate of its gradient with respect to the approximator parameters. It can be seen as an incremental gradient descent algorithm. This eliminates the cardinality dependency that is problematic in large cardinality regimes, as only randomly picked mini-batches of data points are needed instead of the whole dataset in each optimization iteration. One of the first successes brought by this stochastic optimization to train neural networks was obtained with ADALINE (Adaptive Linear Neuron [Widrow and Hoff, 1960]) to solve least-squares problems. Years later, Lenet-5 introduced by LeCun et al. [1998] extended this stochastic optimization to a convolutional neural network to solve document recognition problems.

Nowadays, the stochastic optimization line of work still benefits from many improvements thanks to the considerable interest of the machine learning community. Several scientific avenues have been taken, including optimization algorithms that are faster than the stochastic gradient descent method, such as the momentum method [Polyak, 1964, Sutskever et al., 2013], or an extended version *Adam* [Kingma and Ba, 2014] which adapts the first momentum.

In a nutshell, the Robbins-Monro's theorem allows to manipulate an objective function corresponding to the empirical risk ($\mathcal{L}(\boldsymbol{\theta})$ was previously defined as a large sum over an entire training dataset) without having to compute its values and gradients with respect to the function approximator parameters. Indeed, thanks to Robbins-Monro's theorem, we can simply update the neural network parameters $\boldsymbol{\theta}$ with an unbiased estimate of the true objective function's gradient (denoted $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$) which is typically computed on a randomly sampled mini-batch of data points. The iterative process of parameter adjustment is governed by the following stochastic update rule:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \boldsymbol{\Delta}_k \quad , \quad \boldsymbol{\Delta}_k = \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}} \ell\big(f_{\boldsymbol{\theta}_k}(\mathbf{x}_{i_b}), y_{i_b}\big) \tag{6}$$

where k is an iteration index, $i_b \sim \mathcal{U}_{\mathbb{N}}(1, N)$ is a uniform index parsing the N-cardinality training dataset in small subsamples of size B called mini-batches (which are typically of size 32, 64, 128, 256 or 512, depending on storage capacity and to each application we address). Let $\boldsymbol{\theta}_0$ be a set of neural network parameters at the initial iteration, η_k a learning rate and finally $\boldsymbol{\Delta}_k$ an unbiased estimate of the objective function's gradient. Literally, $\boldsymbol{\Delta}_k$ is an incremental progress, where the learning rate controls how much parameters are changed by it. By the law of large numbers, the stochastic gradient estimation should be close to the real objective function's gradient, i.e. $\boldsymbol{\Delta}_k \approx \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_k)$, however with some random fluctuations [Bottou and Bousquet, 2008].

Starting from $\boldsymbol{\theta}_0$, the k-th iteration of the optimization algorithm uses the forward-propagation and backpropagation (soon detailed in Section 2.2.1.2) to compute $\boldsymbol{\Delta}_k$ and update $\boldsymbol{\theta}_k$. This update (Eq. 6) moves the parameters in the negative direction of $\boldsymbol{\Delta}_k$, therefore moving closer to a global/local minima. A pass through the whole training dataset is called an *epoch*. After some epochs (which is specific to each application we address), the neural network outputs (a.k.a. predictions) must be close to the target function outputs, i.e. $\hat{y} \simeq y$. This stochastic optimization algorithm is guaranteed to converge in supervised learning problems given that Robbins-Monro's conditions on the learning rate are met, i.e. the learning rate is small $0 \leq \eta < 1$, but does not decrease too quickly:

$$\sum_{i}^{\infty} \eta_k = \infty \quad , \quad \sum_{i}^{\infty} (\eta_k)^2 < \infty \tag{7}$$

and with the assumption that the training examples are drawn in an i.i.d. manner[1].

Since the stochastic gradient descent has a low computational cost per iteration, it can scale-up to high cardinality regimes. Specifically, stochastic gradient descent uses a small batch of data points (i.e. a mini-batch) at each update and is therefore more efficient in terms of memory and computational cost. This is

1 (i.i.d. = independently and identically distributed)

in contrast to the vanilla gradient descent algorithm, which passes through the entire training dataset at each update, and therefore has a computational cost per iteration proportional to N.

Deep neural networks are characterized by nonlinearities that generally produce a non-convex objective function [Bengio et al., 2007]. Thus, the minimization of this objective function is far from being a convex optimization problem, which makes the theoretical analysis of its convergence properties more complex. In particular, finding the global optimal solution is impossible under these conditions. In other words, even if the Robbins-Monro conditions are verified, the convergence of the stochastic gradient descent may not be guaranteed. Indeed, the various theoretical results of the neural network approximation only guarantee the existence of a global minimum with respect to the approximation error (i.e. $\mathcal{E}_{\text{app}}$ in Eq. 5) [Cybenko, 1989, Hornik et al., 1990, Pinkus, 1999]. Specifically, a neural network with many parameters is more likely to suffer from the overfitting problem, which can result in a practical solution belonging to one of many poor local minima, i.e. with high estimation and optimization errors ($\mathcal{E}_{\text{est}}$ and $\mathcal{E}_{\text{opt}}$ in Eq. 5) [Erhan et al., 2010, Li et al., 2018].

2.2.1.2 *Backpropagation Algorithm*

The third aspect – computational scalability – makes it possible to avoid the exponential increase in the calculation time needed to approximate high-dimensional target functions. In particular, the backpropagation algorithm (an instance of automatic differentiation algorithms dedicated to neural networks) allows to efficiently compute the derivatives of an objective function with respect to the network parameters [Rumelhart et al., 1986]. This allows to reduce the training time needed for convergence.

As it is not straightforward to compute derivatives of objective functions with respect to the parameters of huge parametric approximators, many studies have been conducted to devise automatic differentiation algorithms in machine learning [Baydin et al., 2017]. The backpropagation is one of the best known automatic differentiation algorithms today which allows to efficiently reduce the derivative calculation cost. It was developed by Rumelhart et al. [1986] for feedforward neural networks (FNNs) and extended to convolutional neural networks (CNNs) by LeCun et al. [1989a]. Different libraries have efficiently implemented the backpropagation algorithm by automatically parallelizing its computations. Among the most popular ones that we used in this thesis are: *PyTorch* [Paszke et al., 2017] and *Tensorflow* [Abadi et al., 2016]. These libraries combined with the increase in computational hardware performance (especially with the increase in the number of CPUs and the availability of GPUs in the case of CNNs) have largely contributed to the scaling of deep learning models.

The backpropagation process is preceded by a forward-propagation phase which computes the neural network outputs. Then the backpropagation algorithm computes with the *chain rule* recursively from the last to the first layer, the

partial derivatives of the objective function with respect to the neural network parameters. In other words, the backpropagation algorithm is a reverse mode automatic differentiation which propagates the excess error by using the chain rule iteratively. Specifically, it first computes the partial derivatives of the objective function with respect to the last layer output:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \hat{\mathbf{y}}} = \frac{\partial}{\partial \hat{\mathbf{y}}} \ell(\hat{\mathbf{y}}, \mathbf{y}) \tag{8}$$

Thanks to the computation of the previous derivatives, it can then update the parameters of the last layer's weight matrix $\mathbf{W}^{(L)}$:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}^{(L)}} \tag{9}$$

Then, the backpropagation iteratively computes for each layer two sets of derivatives. The first set corresponds to the partial derivatives of $\mathcal{L}(\boldsymbol{\theta})$ with respect to the input of the l-th hidden layer $\mathbf{x}^{(l)}$. The chain rule computes them with the already computed derivatives of the $l+1$-th layer, which can be defined recursively by letting $\mathbf{x}^{(0)} = \mathbf{x}$ as follows:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{x}^{(l)}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{x}^{(l+1)}} \frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}} \tag{10}$$

The second set corresponds to the partial derivatives of $\mathcal{L}(\boldsymbol{\theta})$ with respect to the l-th layer parameters $\mathbf{W}^{(l)}$. The chain rule computes them with the first set of derivatives already computed, as follows:

$$\forall i \in [\![1, d^{(l-1)}]\!] \, , \forall j \in [\![1, d^{(l)}]\!] \qquad \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{W}^{(l)}_{ij}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{x}^{(l)}_{j}} \frac{\partial \mathbf{x}^{(l)}_{j}}{\partial \mathbf{W}^{(l)}_{ij}} \tag{11}$$

In other words, the backpropagation uses the first set $\dfrac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{x}^{(l)}}$ to propagate the derivatives of the objective function in the inverse direction from the last layer to the first layer. It can thus compute the derivatives of the objective function with respect to the parameters of each layer with the second set $\dfrac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{W}^{(l)}}$. Once the backpropagation algorithm has computed an estimate of the objective function's gradient, all that remains is to let a stochastic optimization algorithm use these partial derivatives in its update rule (Eq. 6) to minimize the objective function, as explained earlier.

2.2.2 *Deep Learning Approximators*

We now present two basic neural networks: feedforward neural networks (FNNs) and convolutional neural networks (CNNs). They are presented in detail because they are the main parametric approximators used in this thesis.

2.2.2.1 *Feedfoward Neural Networks*

Feedforward neural networks (FNNs) have undergone many extensions since McCulloch and Pitts [1943] introduced them. From a high level viewpoint, FNNs use compositions of simple nonlinear functions called layers to model an unknown target function f as follows:

$$\begin{aligned} y &= f_{\boldsymbol{\theta}}(\mathbf{x}) \\ y &= f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(\mathbf{x}) \end{aligned} \tag{12}$$

where $f^{(l)}$ is the l-th fully-connected layer, $\circ$ is the composition operator, L is the number of layers (a.k.a. the depth of the network), of which the first $L-1$ are characterized as *hidden layers*[2]. A fully connected layer performs a matrix multiplication between an input vector and a weight matrix. If it is a hidden layer, it is followed by an element-wise monotonic increasing nonlinear function independent of $\boldsymbol{\theta}$, which is an activation $\sigma\colon \mathbb{R} \to \mathbb{R}$. A popular choice for the activation is the ReLU (Rectified Linear Unit) [Nair and Hinton, 2010] defined as:

$$\sigma\colon \mathbf{x} \mapsto \max(\mathbf{x}, 0) \tag{13}$$

Thus, each hidden layer performs a simple affine transform followed by a nonlinear transform.

The output of a l-th hidden layer $\mathbf{x}^{(l)}$ (a.k.a. features or neurons) can be defined recursively by letting $\mathbf{x}^{(0)} = \mathbf{x}$, where we deliberately omit the bias terms of the affine transforms for clarity reasons:

$$\mathbf{x}^{(l)} = \sigma\left(\mathbf{W}^{(l)^\tau}\mathbf{x}^{(l-1)}\right) \quad , \quad \forall l \in [\![1, L]\!] \tag{14}$$

where vectors are denoted by lower case bold letters and are assumed to be column vectors. A superscript τ denotes the transpose of a matrix or vector, so that $\mathbf{x}^\tau$ is a row vector. $\mathbf{W}^{(l)}$ is the weight matrix associated to the l-th layer, the set of neural network parameters contains all of them $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}\}_{l \in [\![1,L]\!]}$. Literally, in each l-th hidden layer, the input vector $\mathbf{x}^{(l-1)}$ undergoes a matrix multiplication with the weight matrix $\mathbf{W}^{(l)}$ and an addition with the bias terms to define new origins that we ignore here, and then passes in the activation σ. Mathematically, the weight matrix multiplication performed by the l-th hidden layer is defined as:

$$\mathbf{x}^{(l)} = \left\{ \sigma\left(\sum_{i=1}^{d^{(l-1)}} x_i^{(l-1)} W_{ik}^{(l)} \right) \right\}_{k \in [\![1, d^{(l)}]\!]} \tag{15}$$

where $d^{(l)}$ and $d^{(l-1)}$ are the dimensions of $\mathbf{x}^{(l)}$ and $\mathbf{x}^{(l-1)}$ respectively. For appropriate matrix multiplications, every weight matrix $\mathbf{W}^{(l)}$ must have the same

2 A layer is considered as hidden if it is not connected to the network output.

number of rows as the input vector dimension $d^{(l-1)}$, and the same number of columns as the output vector dimension $d^{(l)}$. This entails that the weight matrix depends on the number of neurons from the input and output of its layer: $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$.

The approximation of a predictor (in supervised learning problems) is of the regression type. In this context, the output of the last hidden layer $\mathbf{x}^{(L-1)}$ passes through a last layer which is an affine transform without activation:

$$\hat{\mathbf{y}} = {\mathbf{W}^{(L)}}^{\tau} \mathbf{x}^{(L-1)} \tag{16}$$

where $\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$ is the last layer's output, which is always a real number if the target function is a value function ($y \in \mathbb{R}$) in the reinforcement learning context, and may be any d-dimensional vector otherwise ($\mathbf{y} \in \mathbb{R}^d$).

2.2.2.2 *Convolutional Neural Networks*

Recent successes in machine learning are largely triggered by the emergence of convolutional neural networks (CNNs). CNNs were introduced by LeCun et al. [1989b] as a special case of FNNs better adapted to tensor data, which continue to benefit from numerous extensions since then. They allow to better bound the approximator complexity with respect to the generalization performance with structured inputs such as images [Poggio et al., 2017].

From a high level viewpoint, CNNs like FNNs use compositions of simple nonlinear layers to approximate a target function f. A hidden convolutional layer denoted $f^{(l)}$ performs a convolution between filters and features of the previous layer, followed by an activation. Thus, it looks like a hidden fully connected layer, with the difference that it performs a special case of affine transform. Indeed, the convolution can be considered as an instance of matrix multiplication which requires vectorizing the input and reducing the filter to a second order tensor (i.e. a matrix), making it a circulant matrix (with many null parameters), as explained by Cichocki et al. [2017].

The output of a l-th hidden convolutional layer is a three-dimensional tensor $\mathbf{x}^{(l)} \in \mathbb{R}^{d_1^{(l)} \times d_2^{(l)} \times d_3^{(l)}}$ whose first two dimensions $(d_1^{(l)}, d_2^{(l)})$ correspond respectively to the spatial height and width of the neurons (a.k.a. feature maps), while the third dimension $d_3^{(l)}$ corresponds to the number of channels. It can be defined recursively by letting $\mathbf{x}^{(0)} = \mathbf{x}$, where we again volontarily omit the bias terms of the affine transforms for clarity reasons:

$$\mathbf{x}_k^{(l)} = \left\{ \sigma \left(\sum_{i=1}^{d_3^{(l-1)}} \mathbf{x}_i^{(l-1)} * \mathbf{W}_{i,k}^{(l)} \right) \right\}_{k \in [\![1, d_3^{(l)}]\!]} \tag{17}$$

The four-dimensional filters $\mathbf{W}^{(l)} \in \mathbb{R}^{w \times w \times d_3^{(l-1)} \times d_3^{(l)}}$ associated to the l-th layer are shared across all spatial coordinates (u, v) by the convolution operator "$*$"

(this is known as the weight-sharing property and results from the convolution invariance to translation). The convolution operator has the advantage that the spatial size w of its filters is independent of the neurons spatial dimensions, which allows w to be small, typically 3 or 5 (but may vary). Indeed, the convolution "$*$" between filters $\mathbf{W}$ and feature maps $\mathbf{x}$ is defined for any spatial coordinate (u, v) as follows:

$$\mathbf{W} * \mathbf{x}(u, v) = \sum_{\substack{1 \leq du \leq w \\ 1 \leq dv \leq w}} \mathbf{x}(s \times u - du, s \times v - dv)\mathbf{W}(du, dv) \tag{18}$$

where $s \in \mathbb{Z}_*^+$ is a stride to perform spatial subsampling. Any spatial coordinate (u, v) in the output results from the convolution between the filter $\mathbf{W}$ and the patch (a.k.a. receptive field) of the input defined as $\{\mathbf{x}(u, v)\}_{1 \leq du, dv \leq w} \in \mathbb{R}^{w \times w}$. This allows to associate a strong numerical value when the patch is positively correlated with the filter. Literally in Eq. 17, the k-th feature map of the l-th convolutional layer results from the the superposition of $d_3^{(l-1)}$ convolutions between the filters $\mathbf{W}_{:,k}^{(l)}$ and the input features maps $\mathbf{x}^{(l-1)}$.

CNNs benefit from four main characteristics: *compositionality*, *locality*, *weight-sharing* and *subsampling*. Compositionality is achieved with the stack of $L - 1$ hidden layers each composed of a linear (parametrized) and nonlinear (non-parametrized) transforms. The subsequent neural network uses the composition of a series of $L - 1$ nonlinear functions, known as a L-depth network.

Locality refers to the fact that a neuron in one layer is connected to only a small number of neurons in the previous layer. On the contrary, FNNs do not have this property as a neuron in one layer is connected to all neurons of the previous layer (hence their name *fully-connected layer*). Eq. 18 clearly shows that CNNs are like superpositions and compositions of a finite number of nonlinear *local-variable functions* [Poggio et al., 2017].

Weight-sharing refers to the fact that the convolution is invariant to the translation operator $\mathcal{T}$ defined as:

$$\forall \mathbf{z} = (z_1, z_2)\ , \quad \mathcal{T}_{\mathbf{z}}\mathbf{x} = \mathbf{x}(u - z_1, v - z_2) \tag{19}$$

Mathematically this implies that for every convolution of a filter $\mathbf{W}$ and an input $\mathbf{x}$ we have:

$$\mathbf{W} * \left(\mathcal{T}_{\mathbf{z}}\mathbf{x}\right) = \mathcal{T}_{\mathbf{z}}\left(\mathbf{W} * \mathbf{x}\right) \tag{20}$$

Specifically, any spatial coordinate (u, v) in the k-th channel of the output feature maps shares the same filters $\mathbf{W}_{:,k}^{(l)}$, hence the weight-sharing property. From a general viewpoint, filters $\mathbf{W}_{:,k}^{(l)}$ extract a type of feature independently of the spatial coordinates.

Subsampling is achieved by reducing spatial patches to a single real value. There are non-parametrized approaches that consist in taking the maximum

or the average of the patches, and on the other side parametrized approaches that are performed by convolutional layers with a stride greater than one. Non-parametrized subsampling operators were used (a.k.a. *pooling layers*) in the first CNN architectures proposed by LeCun et al. [1998] through LeNet-5, and extended by Ranzato et al. [2007] through LeNet-6. Spatial subsampling allows to reduce spatial redundancies and to aggregate the information from previous feature maps (i.e. the responses of activated neurons). It is based on the assumption that a small neighborhood around a spatial coordinate (u, v) in a feature map is likely to contain the same information with possible distractors.

These subsampling operations tend to create a hierarchy of different abstraction levels among the neurons of the subsequent layers [Cohen et al., 2016]. Specifically, through the use of subsampling strategies, CNNs make an additional hierarchical assumption about the composition of local-variable functions. Therefore, CNN approximators belong to a class of functions that have superpositions and compositions of a finite number of nonlinear *hierarchical local-variable functions*. Poggio et al. [2017] equivalently formulated them as *hierarchically local compositional functions*, which tend to be better in terms of generalization performance than *local-variable functions*. Indeed, a line of work in the analysis of the theoretical properties of function approximators tend to show that compositional functions consisting of hierarchically organized local-variable functions are one of the best regularity hypothesis to circumvent the curse of dimensionality in large-scale function approximation problems [Cohen et al., 2016, Poggio et al., 2017].

Locality and subsampling help reduce the network's sensitivity to spatial coordinates and resolutions of the sensory input. As a result, CNNs tend to become translation invariance at the network level. These characteristics are motivated by the fact that in natural images, the semantic meaning of patterns often does not depend on their locations.

In the case of regression tasks (as also in the case of classification tasks), the output of a last hidden convolutional layer denoted $\mathbf{x}^{(L-p)}$ usually goes through a series of p fully-connected layers to predict a vector output. To do this, the input feature maps $\mathbf{x}^{(L-p)}$ are vectorized before passing through the first matrix multiplication $\mathbf{x}^{(L-p+1)} = \sigma\big({\mathbf{W}^{(l)}}^{\intercal}\mathbf{x}^{(L-p)}\big)$. As usual, the last layer of this FNN is an affine transform without activation, and $\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$ is the d-dimensional network output.

2.3 REINFORCEMENT LEARNING

Dynamic programming introduced by Bellman [1961] is a first attempt to mitigate the curse of dimensionality in optimal control problems (see Section 2.1.1). A second major attempt is made by the field of reinforcement learning (RL) to scale to a setting where knowledge of the transition model is incomplete [Bertsekas, 2019]. The RL domain has expanded considerably since its foundations were

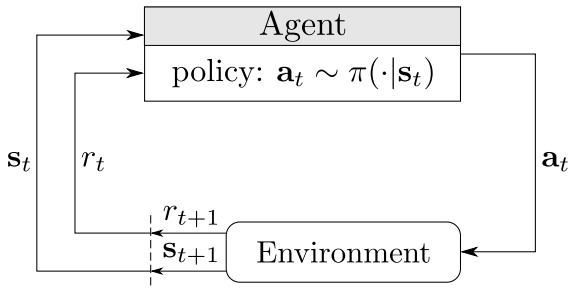


Figure 1. The agent-environment interaction loop for a RL algorithm, following Sutton and Barto [2018]. At time step t, an agent receives a state $\mathbf{s}_t$ and a reward r_t from the environment based on which the algorithm can update its policy. Once an agent performs an action $\mathbf{a}_t$ sampled from the policy, the environment provides him with the next state $\mathbf{s}_{t+1}$ and the next reward r_{t+1}.

comprehensively presented by Sutton and Barto [2018]. RL algorithms use a combination of Bellman equations, which were originally used by dynamic programming methods, and temporal-difference learning optimization. The latter was originally developed for machine learning problems [Sutton, 1988].

2.3.1 *Reinforcement Learning Formalism*

The general optimal control problem addressed by RL has only incomplete knowledge of the environment, which may also be stochastic. More precisely, the transition probabilities are unknown. Thus, RL algorithms generate training examples with agent-environment interaction loops as shown in Fig. 1. The goal of RL is to maximize the expected future cumulative discounted rewards (i.e. the expected return G_t), that an agent receives by interacting with the environment. To do this, a RL algorithm learns an optimal policy (denoted π_*) which predicts actions that maximize the return.

2.3.1.1 *Markov Decision Processes*

The RL optimization process can be formally defined with Markov decision processes (MDPs). An MDP is succinctly denoted as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, whose five elements are defined in the following.

The state and action spaces We consider only continuous states and actions in this thesis, this is why we speak only of spaces and not of sets as would be the case for discrete spaces. $\mathcal{S} \subset \mathbb{R}^{\mathcal{S}_d}$ corresponds to the state space and $\mathcal{A} \subset \mathbb{R}^{\mathcal{A}_d}$ corresponds to the action space. A state $\mathbf{s}_t \in \mathcal{S}$ gathers sufficient information from the environment for an optimal policy to take a best action $\mathbf{a}_t \in \mathcal{A}$. It includes proprioceptive information of the agent, such as the position of its actuators and their velocities. In classical optimal control problems, the state is available, whereas in the problems we study, it is not directly available. The environment provides instead of states, observations ($\mathbf{o}_t \in \mathcal{O} \subset \mathbb{R}^{\mathcal{O}_d}$ where $\mathcal{O}$ is

the observation space) which are most often visual sensory measurements. These are high-dimensional observations (i.e. typically $\mathcal{S}_d << \mathcal{O}_d$) which generally suffer from partial observability of the environment. This corresponds to a state representation learning formalism which will be described thoroughly in Chapter 3.

The starting state distribution In our experiments with a RL algorithm, we use as a starting state distribution *random starts*. This corresponds to an agent being randomly reseted each time an episode is completed. This is one of the conditions for many convergence proofs of RL algorithms [Watkins and Dayan, 1992, Sutton and Barto, 2018]. We will use in Chapter 5 a different starting state distribution with state representation learning experiments[3].

The transition probability P is the transition probability function that maps a pair of state-action at the current time step t to a state-reward distribution at the next step $t+1$. Following Sutton and Barto [2018] it is defined as:

$$P(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) \triangleq \mathbb{P}[\mathbf{s}_{t+1} = \mathbf{s}', r_{t+1} = r \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}] \tag{21}$$

This equation is stationary, which means that the transition probability does not depend on the time step t. $P(\mathbf{s}', r|\mathbf{s}, \mathbf{a})$ is the probability for the agent in state $\mathbf{s}$ to move to the next state $\mathbf{s}'$ after taking action $\mathbf{a}$ by obtaining reward r. Following Sutton and Barto [2018], the state-transition function can be defined as a function of $P(\mathbf{s}', r|\mathbf{s}, \mathbf{a})$:

$$P(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \triangleq \mathbb{P}[\mathbf{s}_{t+1} = \mathbf{s}' \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}] = \sum_{r \in \mathcal{R}} P(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) \tag{22}$$

The reward function R predicts the next reward associated to an action and a state:

$$R(\mathbf{s}, \mathbf{a}) \triangleq \mathbb{E}\left[r_{t+1} \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right] = \sum_{r \in \mathcal{R}} r \sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}', r \big| \mathbf{s}, \mathbf{a}) \tag{23}$$

The reward is obtained indirectly from the environment (i.e. through agent-environment interaction loops) as a privileged information because in most robotic problems we do not have it. This the case for state representation learning algorithms in a rewardless environment studied in this thesis.

The Markovian property In an MDP the transition probabilities have the Markovian property. This is why the transition probability function depends only on current state $\mathbf{s}_t$ and action $\mathbf{a}_t$. Mathematically, this means that the transition probabilities must verify the following relation:

$$P\left(\mathbf{s}_{t+1}, r_{t+1} \big| \mathbf{s}_t, \mathbf{a}_t\right) = P\left(\mathbf{s}_{t+1}, r_{t+1} \big| \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \ldots, \mathbf{s}_0, \mathbf{a}_0\right) \tag{24}$$

3 In the case of purely state representation learning experiments with our XSRL method in Chapter 5, state estimators are learned from a task-agnostic exploration. In this context of pure exploration, we use a constant state as the starting state distribution.

To verify this property in a partially environment, a state must contain information about current and past agent-environment interactions that can potentially influence the future decision process [Sutton and Barto, 2018]. In particular, the Markovian property applies only to the transition probability function and does not apply directly to the state, otherwise the state could not memorize useful experience to restore full observability of the environment.

Episodic MDPs In this thesis we only consider episodic MDPs, also known as finite-horizon MDPs which involve a finite sequence of agent-environment interaction loops. Each interaction is a tuple of the form $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_{t+1})$. An agent is reseted after a given number of time steps called horizon T. A full interaction sequence constitutes a trajectory (a.k.a. episode or rollout) defined as:

$$\mathcal{T} = \mathbf{s}_0, \mathbf{a}_0, r_1, \mathbf{s}_1, \ldots, \mathbf{a}_{T-1}, r_T, \mathbf{s}_T$$

The return The return is the future cumulative discounted rewards. In episodic MDPs, the return corresponds to the sum of rewards obtained so far, discounted by how far in the future they are obtained:

$$G_t \triangleq \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{25}$$

where $\gamma \in [0, 1]$ is the discount factor. Literally the discount factor defines how much preference is given to recent rewards over older ones. As we know, the RL goal is to learn an optimal policy that maximizes the expected return, which can be defined as:

$$J(\pi) \triangleq \mathbb{E}\left[G_t \middle| \pi\right] = \sum_{\mathbf{s}\in\mathcal{S}} \mu_\pi(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) R(\mathbf{s}, \mathbf{a}) \tag{26}$$

where $\mathbb{E}\left[\cdot|\pi\right]$ is the expectation of a random variable conditionned by the policy π which the agent follows, and μ_π is the *stationary distribution* under π (which means that it does not depend on the time step t). Mathematically the optimal policy is defined as follows:

$$\pi_* \triangleq \arg\max_\pi J(\pi) \tag{27}$$

Incomplete knowledge of the MDP Most of the problems addressed by RL algorithms have incomplete knowledge of the MDP, typically the transition probability function is unknown. Being able to deal with optimal control problems with different degrees of knowledge about the environment is one of the main advantages RL has over its dynamic programming counterpart. From a general viewpoint, these degrees of knowledge decompose as follows:

KNOWN TRANSITION MODEL Planning, dynamic programming, and RL (in particular model-based) may be applicable. Dynamic programming methods compute optimal policies by planning through a transition model with Bellman equations.

UNKNOWN TRANSITION MODEL Planning with a learned model and RL may be applicable. According to Moerland et al. [2020], model-based RL is a combination of planning and learning. It uses a model of transition probabilities which can be known or learned, combined with a learned value function and/or policy. On the other hand, planning with a learned model can also be performed as a different approach to model-based RL, as it only learns a model of the transition probabilities and performs planning with it. On another level, model-free RL methods learn a value function and/or a policy solely from training examples obtained from the agent-environment interaction loops, without learning a transition model.

2.3.1.2 *Key Reinforcement Learning Components*

Temporal-difference (TD) learning is the main strategy on which RL relies that allows it to adapt when the transition model is unknown. TD learning is based on computing a value function and/or a policy with samples of past interactions iteratively. As we have seen previously, RL algorithms are distinguished by whether or not they use a model. Thus, RL algorithms fall into two main categories: those that are model-based and those that are model-free. Another main level on which RL algorithms differ is whether or not the policy can be trained with samples obtained from other policies. If it is the case, it is an off-policy RL method. Otherwise, it is an on-policy method.

Fig. 1 shows an agent-environment interaction loop from which a RL algorithm learns an optimal policy to maximize the expected return over trajectories. To interact with the environment, an agent executes an action $\mathbf{a}_t$ sampled from the current policy, then the environment returns a state $\mathbf{s}_{t+1}$ and a reward r_{t+1} determined by the unknown transition probability function $P(\mathbf{s}', r|\mathbf{s}, \mathbf{a})$.

The policy and (action-)value functions The two main components of TD learning are the policy and the value function. In this thesis, we only consider stochastic policies defined as a mapping between the current state and a conditional probability on the actions:

$$\pi(\mathbf{a}|\mathbf{s}) \triangleq \mathbb{P}[\mathbf{a}_t = \mathbf{a}|\mathbf{s}_t = \mathbf{s}] \tag{28}$$

In order to estimate the expected return, RL algorithms learn value functions. The value function estimates the expected return of an agent starting in state $\mathbf{s}$ and then following π [Sutton and Barto, 2018]:

$$V_\pi(\mathbf{s}) \triangleq \mathbb{E}\big[G_t\big|\mathbf{s}_t = \mathbf{s}, \pi\big] \tag{29}$$

The value function predicts how good it is for an agent to be in a state and then following the current policy. In a similar way, the action-value function estimates the expected return of an agent starting in state $\mathbf{s}$ and taking an action $\mathbf{a}$ and then following the current policy:

$$Q_\pi(\mathbf{s}, \mathbf{a}) \triangleq \mathbb{E}\big[G_t\big|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}, \pi\big] \tag{30}$$

These two functions are used to compute the optimal policy. Indeed, the optimal value functions must produce the maximum return as follows:

$$\begin{aligned} V_*(\mathbf{s}) &\triangleq \max_\pi V_\pi(\mathbf{s}) \\ Q_*(\mathbf{s},\mathbf{a}) &\triangleq \max_\pi Q_\pi(\mathbf{s},\mathbf{a}) \end{aligned} \tag{31}$$

Bellman equations The return (Eq. 25) can be expressed recursively with the equation $G_t = r_{t+1} + \gamma\, G_{t+1}$, which is essential for the formulation of Bellman equations as recursive formulations of the value functions. Bellman equations were designed in the dynamic programming literature by Bellman et al. [1957], and then popularized in the RL literature by Sutton and Barto [2018]. The Bellman equation for the value function is:

$$\begin{aligned} V_\pi(\mathbf{s}) &= \mathbb{E}\big[G_t \big| \mathbf{s}_t = \mathbf{s}, \pi\big] \\ &= \mathbb{E}\big[R(\mathbf{s}_t,\mathbf{a}_t) + \gamma G_{t+1} \big| \mathbf{s}_t = \mathbf{s}, \pi\big] \\ &= \sum_{\mathbf{a}\in\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a}) \left[R(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}\left[G_{t+1}|\mathbf{s}_{t+1} = \mathbf{s}', \pi\right]\right] \\ &= \sum_{\mathbf{a}\in\mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a}) \big[R(\mathbf{s},\mathbf{a}) + \gamma V_\pi(\mathbf{s}')\big] \\ &= \mathbb{E}\big[R(\mathbf{s}_t,\mathbf{a}_t) + \gamma V_\pi(\mathbf{s}_{t+1}) \big| \mathbf{s}_t = \mathbf{s}, \pi\big] \end{aligned} \tag{32}$$

Recursivity is observed here because the value function of a state is expressed with respect to the value function on the next state. Similarly the Bellman equation for the action-value function is:

$$\begin{aligned} Q_\pi(\mathbf{s},\mathbf{a}) &= \mathbb{E}\big[G_t \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}, \pi\big] \\ &= \mathbb{E}\big[R(\mathbf{s}_t,\mathbf{a}_t) + \gamma G_{t+1} \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}, \pi\big] \\ &= \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a}) \left[R(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}\left[G_{t+1}|\mathbf{s}_{t+1} = \mathbf{s}', \pi\right]\right] \\ &= \mathbb{E}\big[R(\mathbf{s}_t,\mathbf{a}_t) + \gamma V_\pi(\mathbf{s}_{t+1}) \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}, \pi\big] \end{aligned} \tag{33}$$

Eq. 32 is verified only with the true value function V_π, and for Eq. 33 with the true action-value function Q_π.

Bellman optimality equations Optimal value functions must satisfy the Bellman optimality equation which is for the value function [Sutton and Barto, 2018]:

$$\begin{aligned} V_*(\mathbf{s}) &= \max_{\mathbf{a}} Q_*(\mathbf{s},\mathbf{a}) \\ &= \max_{\mathbf{a}} \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a}) \big[R(\mathbf{s},\mathbf{a}) + \gamma V_*(\mathbf{s}')\big] \\ &= \max_{\mathbf{a}} \mathbb{E}\big[R(\mathbf{s}_t,\mathbf{a}_t) + \gamma V_*(\mathbf{s}_{t+1}) \big| \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\big] \end{aligned} \tag{34}$$

and similarly for the action-value function:

$$\begin{aligned} Q_*(\mathbf{s},\mathbf{a}) &= \sum_{\mathbf{s}'\in\mathcal{S}} P(\mathbf{s}'|\mathbf{s},\mathbf{a})\big[R(\mathbf{s},\mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_*(\mathbf{s}',\mathbf{a}')\big] \\ &= \mathbb{E}\big[R(\mathbf{s}_t,\mathbf{a}_t) + \gamma \max_{\mathbf{a}_{t+1}\in\mathcal{A}} Q_*(\mathbf{s}_{t+1},\mathbf{a}_{t+1})\big|\mathbf{s}_t=\mathbf{s},\mathbf{a}_t=\mathbf{a}\big] \end{aligned} \tag{35}$$

These equations allow dynamic programming methods to directly find an optimal policy, which is in itself a planning problem. However, model-free RL algorithms do not use a transition model and therefore estimate the expectation over the next states thanks to samples of agent-environment interaction loops. Hence the approximation of the true value function V_π, or of the true action-value function Q_π.

2.3.2 *Temporal-Difference Learning*

Temporal-difference (TD) learning introduced by Sutton [1988] scales very well in optimal control problems with an incomplete knowledge of the MDP, typically with an unknown transition model ($P(\mathbf{s}',r|\mathbf{s},\mathbf{a})$). It consists in estimating the expectations over the next states in Bellman equations thanks to interaction samples. Specifically, it learns the value function from its own value function estimates denoted at the k-th iteration $\widehat{V}_k$, hence the bootstrapping. The update rule consists therefore in bringing the current value function closer to the *TD target* of the Bellman equation, which corresponds to $r_{t+1} + \gamma\widehat{V}_k(\mathbf{s}_{t+1})$. Thus the update rule of the value function is as follows:

$$\widehat{V}_{k+1}(\mathbf{s}_t) \leftarrow \widehat{V}_k(\mathbf{s}_t) - \eta_k\left(\widehat{V}_k(\mathbf{s}_t) - \big(r_{t+1} + \gamma\widehat{V}_k(\mathbf{s}_{t+1})\big)\right) \tag{36}$$

similarly the update rule of the action-value function is as follows:

$$\widehat{Q}_{k+1}(\mathbf{s}_t,\mathbf{a}_t) \leftarrow \widehat{Q}_k(\mathbf{s}_t,\mathbf{a}_t) - \eta_k\left(\widehat{Q}_k(\mathbf{s}_t,\mathbf{a}_t) - \big(r_{t+1} + \gamma\widehat{Q}_k(\mathbf{s}_{t+1},\mathbf{a}_{t+1})\big)\right) \tag{37}$$

2.3.2.1 *Q-Learning: Off-policy TD control*

The Q-learning algorithm is a special case of off-policy and model-free TD learning method with tabular representations (a.k.a. look-up table representations), i.e. with discrete and finite states and actions [Watkins, 1989, Watkins and Dayan, 1992]. With tabular state and action spaces, the action-value function estimation $\widehat{Q}$ of Eq. 37 must solve the Bellman equation of the following form:

$$Q(\mathbf{s}_t,\mathbf{a}_t) = r_{t+1} + \gamma Q\big(\mathbf{s}_{t+1},\mathbf{a}_{t+1}\big) \tag{38}$$

In order to estimate the optimal action-value function Q_*, $\widehat{Q}$ must be optimized to verify the following Bellman optimality equation:

$$Q_*(\mathbf{s}_t,\mathbf{a}_t) = r_{t+1} + \gamma \max_{\mathbf{a}_{t+1}\in\mathcal{A}} Q_*\big(\mathbf{s}_{t+1},\mathbf{a}_{t+1}\big) \tag{39}$$

In simple terms, to find the best action, the Q-learning algorithm must maximize the expected return over the next actions $\mathbf{a}_{t+1}$, which is guaranteed by the optimal action-value function Q_*. The update rule of Q-learning is then as follows:

$$\widehat{Q}_{k+1}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow \widehat{Q}_k(\mathbf{s}_t, \mathbf{a}_t) - \eta_k \left(\widehat{Q}_k(\mathbf{s}_t, \mathbf{a}_t) - \left(r_{t+1} + \gamma \max_{\mathbf{a}_{t+1} \in \mathcal{A}} \widehat{Q}_k(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right) \right) \tag{40}$$

where $r_{t+1} + \gamma \max_{\mathbf{a}_{t+1} \in \mathcal{A}} \widehat{Q}_k(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$ is the TD target. Watkins and Dayan [1992] theoretically guarantee that Q-learning converges to the optimal action-value function under mild conditions on the state and action spaces. In addition, all actions and states must be infinitely sampled and the same Robbins-Monro's condition (see Eq. 7) on the learning rate must be satisfied. Once the system has estimated the optimal value function, it can use it to approximate the corresponding optimal policy which takes actions that maximize Q_* as follows:

$$\pi_*(\mathbf{s}_t) = \arg\max_{\mathbf{a}_t \in \mathcal{A}} Q_*(\mathbf{s}_t, \mathbf{a}_t) \tag{41}$$

2.3.2.2 *Deep Q-Network (DQN)*

We know from Section 2.1.1 that when the dimensionality of the state and/or action spaces increase, the number of computations and the memory increase exponentially: it is the curse of dimensionality. In particular, the table of an action-value function cannot store all the state-action pairs. When states and actions are too large to be represented in lookup tables, or when they are continuous, it is necessary to be able to generalize across these large spaces. This is why RL combines with function approximation, as well as the dynamic programming that gave the field of approximate dynamic programming (ADP) its name.

The advantage of these approaches lies in the fact that the parameter set (denoted $\boldsymbol{\theta}$) of an approximator is much smaller than the number of possible state-action pairs. When function approximation is used, the value function and/or the policy are approximated as parametric mappings. There are a wide variety of approximators such as polynomials, wavelets, discretization-interpolation approaches and neural networks [Sutton and Barto, 2018]. RL methods have gone through a few iterations before they could scale to control tasks with continuous state spaces of only a dozen dimensions. Tesauro [1992] proposed for the first time to combine deep neural networks with TD learning [Sutton, 1988] using the stochastic gradient descent algorithm [Robbins and Monro, 1951]. Since stochastic gradient descent has a low computational cost per iteration, it allows to scale-up RL algorithms to high cardinality regimes. Later, Riedmiller [2005] applied this technique to Q-learning, followed by [Mnih et al., 2015] who used CNNs to scale it to image observations, known as DQN (Deep Q-Network).

RL algorithms that combine function approximation with bootstrapping and off-policy generally suffer from instability and divergence convergence issues. This problem is well known as the *deadly triad* [Sutton and Barto, 2018]. Of these three, the function approximation is the most essential. Bootstrapping is useful to increase computational and sample efficiencies. Indeed, Monte Carlo methods that avoid bootstrapping require a lot of memory to store the complete episodes before performing any iteration. According to Sutton and Barto [2018], a key element to enable good bootstrapping is a good state representation. Indeed, states must verify certain properties for TD learning to work on incomplete episodes [Penedones et al., 2018]. Hence, the success of RL algorithms is related to the quality of the state space.

Off-policy learning is a way to draw training examples from a much broader distribution of the transition model underlying the environment rather than being limited to the current policy. Specifically, instead of following a unique distribution whose samples are necessarily correlated during the policy training process, off-policy learning draws samples from a multitude of distributions.

In addition to the deadly triad, another problem concerns the failure of two of the Robbins-Monro conditions for the stochastic gradient descent algorithm to converge. The first condition that true target function predictions are available is broken. This is why TD learning methods use their own target function predictions (known as the TD targets) based on estimates of their current value function, i.e. they bootstrap. The second condition that the training examples are drawn in an i.i.d. manner[4] is broken. On the one hand, since the samples are drawn successively in a same episode, the independent sampling property is not verified. On the other hand, since the samples depend on the exploration policy which changes during the learning process, the identical sampling property is violated.

Mnih et al. [2015] tend to reduce bootstrapping and non-i.i.d. sampling with two tricks. The first trick consists in bringing the training distribution closer to the i.i.d. hypothesis thanks to the experience replay buffer technique introduced by [Lin, 1992]. Indeed, a replay buffer stores the previous episodes in order to randomly draw training examples. The second trick is to stabilize the target function predictions by delaying updates of the *target value function* used in the TD targets. To do this, the parameters of the target value function are obtained as an exponentially moving average of the current value function parameters.

The main elements of DQN as well as other RL algorithms with neural network approximators are as follows. As in the tabular case, the target function prediction is formed by the TD target which corresponds to the right-hand side of the Bellman optimality equation (Eq. 39), defined as follows:

$$\mathbf{y}_t = r_{t+1} + \gamma \max_{\mathbf{a}_{t+1} \in \mathcal{A}} Q_{\boldsymbol{\theta}^-}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \tag{42}$$

4 (i.i.d. = independently and identically distributed)

where $\boldsymbol{\theta}^{-}$ is the parameter set of the target value function. In order to optimize the parameterized action-value function $Q_{\boldsymbol{\theta}_k}$ to approximate the optimal action-value function, the DQN algorithm minimizes the following *TD error*:

$$\ell\big(Q_{\boldsymbol{\theta}_k}(\mathbf{s}_t, \mathbf{a}_t), y_t\big) = \big(Q_{\boldsymbol{\theta}_k}(\mathbf{s}_t, \mathbf{a}_t) - y_t\big)^2 \tag{43}$$

Here the update rule is transformed into a stochastic gradient descent update rule to minimize the TD error:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \boldsymbol{\Delta}_k \quad , \quad \boldsymbol{\Delta}_k = \frac{1}{B} \sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}} \ell\big(Q_{\boldsymbol{\theta}_k}(\mathbf{s}_{i_b}, \mathbf{a}_{i_b}), y_{i_b}\big) \tag{44}$$

where k is an iteration index, $i_b \sim \mathcal{U}_{\mathbb{N}}(1, N)$ is a uniform index parsing the N-cardinality training dataset[5] in small mini-batches of size B.

2.3.3 *Policy Gradient*

Pure TD learning algorithms follow the strategy to learn an (action-)value function to compute an optimal policy. Policy gradient algorithms instead directly learn a parametrized optimal policy denoted $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$. We know that the goal of RL algorithms is to find an optimal policy which maximizes the expected return (Eq. 26). In this context, the expected return corresponds to the objective function which depends on the policy parameters $\boldsymbol{\theta}$ and is defined as:

$$J(\boldsymbol{\theta}) \triangleq \mathbb{E}\left[G_t \middle| \pi_{\boldsymbol{\theta}}\right] = \sum_{\mathbf{s} \in \mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a} \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) R(\mathbf{s}, \mathbf{a}) \tag{45}$$

where $\mu_{\boldsymbol{\theta}}$ is the *stationary distribution* under $\pi_{\boldsymbol{\theta}}$ (which means that it does not depend on the time step t). It can be estimated with the value function as:

$$J(\boldsymbol{\theta}) \approx \sum_{\mathbf{s} \in \mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) V_{\boldsymbol{\theta}}(\mathbf{s}) \tag{46}$$

and similarly with the action-value function as:

$$J(\boldsymbol{\theta}) \approx \sum_{\mathbf{s} \in \mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a} \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) Q_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a}) \tag{47}$$

where $V_{\boldsymbol{\theta}}$ and $Q_{\boldsymbol{\theta}}$ are respectively the value function and action-value function under $\pi_{\boldsymbol{\theta}}$. Policy gradient methods approximate an optimal policy directly by gradient ascent on $J(\boldsymbol{\theta})$. According to Sutton and Barto [2018], policy gradient methods are supposed to better circumvent the curse of dimensionality because they provide a better convergence stability. Indeed the probability of choosing an action with the policy changes smoothly, whereas the action-value function may change smoothly but cause irregular changes in the action probabilities. So policy methods have stronger convergence guarantees than action-value methods.

5 The training dataset is embodied by a replay buffer.

2.3.3.1 *Policy Gradient Theorem*

In order to compute the objective function gradient $\nabla_\theta J(\boldsymbol{\theta})$ it is necessary to disambiguate its dependence on the action selection probabilities determined by $\pi_{\boldsymbol{\theta}}$, and on the stationary state distribution under the exploration policy $\mu_{\boldsymbol{\theta}}(\mathbf{s})$ (depending on $\pi_{\boldsymbol{\theta}}$). However, since the stationary state distribution is a function of the transition model which is unknown, a policy gradient method must ignore it in order to compute $\nabla_\theta J(\boldsymbol{\theta})$. The policy gradient theorem allows us to express an estimate of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ without the dependence on the stationary state distribution as follows [Sutton and Barto, 2018]:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \left(\sum_{\mathbf{s}\in\mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{a\in\mathcal{A}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) \right) \\ &\propto \sum_{\mathbf{s}\in\mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})\end{aligned} \tag{48}$$

where the approximation made is proportional to the true $\nabla_\theta J(\boldsymbol{\theta})$ and provided that $\mu_{\boldsymbol{\theta}}$ is the on-policy distribution under $\pi_{\boldsymbol{\theta}}$.

2.3.3.2 *Soft Actor-Critic (SAC)*

SAC (Soft Actor-Critic [Haarnoja et al., 2018b]) with automatic temperature tuning is the algorithm we use in our experiments to evaluate the performance from different types of state representation. It is according to Haarnoja et al. [2018b] well suited for continuous optimal control problems, achieving state-of-the-art performance and even better stability. This explains why we have chosen this method rather than another one, namely DDPG [Lillicrap et al., 2015]. It belongs to the off-policy actor-critic methods with maximum entropy, whose most recent origin is soft Q-learning [Haarnoja et al., 2017]. An actor-critic method is a policy gradient method which learns in addition to the policy an action-value function.

SAC introduced by Haarnoja et al. [2018a] consists in adding an estimated entropy of the policy conditioned on the state in the objective function. Moreover, the coefficient that regulates this term called the *temperature*, is automatically adjusted in the SAC extension proposed by Haarnoja et al. [2018b]. This eliminates the need to tune the temperature during training (as the policy improves) and over different environments. This new objective function is intended to encourage exploration, as it should make the policy as random as possible while successfully completing the task. According to Haarnoja et al. [2018b], this should improve convergence stability, hence their better results compared to other state-of-the-art methods such as DDPG [Lillicrap et al., 2015].

The goal of SAC is to maximize the following objective function:

$$J(\boldsymbol{\theta}) \triangleq \sum_{\mathbf{s}\in\mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \left[R(\mathbf{s},\mathbf{a}) + \alpha \mathcal{H}\left(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})\right) \right] \tag{49}$$

where α is the temperature, and $\mathcal{H}$ is the entropy measure which is defined as:

$$\mathcal{H}\big(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})\big) = -\sum_{\mathbf{s}\in\mathcal{S}} \mu_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \log\big(\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})\big) \tag{50}$$

Haarnoja et al. [2018a] propose to restrict the policy to a Gaussian distribution. To do this, a network predicts a mean vector μ_π and the diagonal covariance elements of a covariance matrix Σ_π allowing to parameterize the policy as:

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \triangleq \mathcal{N}\left(\mu_\pi(\mathbf{s}_t), \Sigma_\pi(\mathbf{s}_t)\right) \tag{51}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\Sigma\}$. Haarnoja et al. [2018a] use the reparametrization trick [Kingma and Welling, 2014] to sample actions from the policy (i.e. $\mathbf{a}_t^\pi \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$) in order to keep all its parameters differentiable as follows:

$$\mathbf{a}_t^\pi \triangleq \mu_\pi(\mathbf{s}_t) + \boldsymbol{\epsilon}_t \times \Sigma_\pi(\mathbf{s}_t) \quad , \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}\left(\mathbf{0}_{\mathcal{A}_d}, \mathbf{I}_{\mathcal{A}_d}\right) \tag{52}$$

In order to maximize $J(\boldsymbol{\theta})$, Haarnoja et al. [2018a] propose to minimize the Kullback-Leibler divergence between the policy and the exponential *soft action-value function* (ignoring the normalization term which is independent of the policy parameters) expressed as:

$$\begin{aligned} J_\pi(\boldsymbol{\theta}) &\triangleq \mathrm{D}_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t) \middle\| \exp\left(\tfrac{1}{\alpha} Q_{\boldsymbol{\omega}}(\mathbf{s}_t, \cdot)\right)\right) \\ &\propto \mathbb{E}_{\substack{\mathbf{s}_t\sim\mu_{\boldsymbol{\theta}} \\ \mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}}} \left[\log\left(\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\exp\left(\frac{1}{\alpha} Q_{\boldsymbol{\omega}}(\mathbf{s}_t, \mathbf{a}_t)\right)}\right)\right] \\ &\propto \mathbb{E}_{\substack{\mathbf{s}_t\sim\mu_{\boldsymbol{\theta}} \\ \mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}}} \left[\alpha \log\left(\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\right) - Q_{\boldsymbol{\omega}}(\mathbf{s}_t, \mathbf{a}_t)\right] \end{aligned} \tag{53}$$

The estimated gradient of this objective function $\nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta})$ and the update rule for the stochastic gradient descent are then defined as follows:

$$\begin{aligned} &\text{estimated gradient: } (\boldsymbol{\Delta}_\pi)_k = \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}}\left[\alpha \log\big(\pi_{\boldsymbol{\theta}}(\mathbf{a}_{i_b}|\mathbf{s}_{i_b})\big) - Q_{\boldsymbol{\omega}}(\mathbf{s}_{i_b}, \mathbf{a}_{i_b})\right] \\ &\text{update rule: } \boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k (\boldsymbol{\Delta}_\pi)_k \end{aligned} \tag{54}$$

where k is an iteration index, $i_b \sim \mathcal{U}_{\mathbb{N}}(1, N)$ is a uniform index parsing the N-cardinality training dataset in small mini-batches of size B. SAC such as DQN, builds its training dataset with a replay buffer.

Then Haarnoja et al. [2018a] learn a *soft action-value function* $Q_{\boldsymbol{\omega}}$ parameterized by $\boldsymbol{\omega}$ to verify the following *soft Bellman optimality equation*:

$$Q_*(\mathbf{s}_t, \mathbf{a}_t) \triangleq r_{t+1} + \gamma \mathbb{E}_{\substack{\mathbf{s}_{t+1}\sim\mu_{\boldsymbol{\theta}} \\ \mathbf{a}_{t+1}\sim\pi_{\boldsymbol{\theta}}}} \left[Q_*(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log\big(\pi_{\boldsymbol{\theta}}(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})\big)\right] \tag{55}$$

$Q_{\boldsymbol{\omega}}$ is trained to minimize the following TD error:

$$J_Q(\boldsymbol{\omega}) \triangleq \mathbb{E}_{\substack{\mathbf{s}_t\sim\mu_{\boldsymbol{\theta}} \\ \mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}}} \left[Q_{\boldsymbol{\omega}}(\mathbf{s}_t, \mathbf{a}_t) - Q_*(\mathbf{s}_t, \mathbf{a}_t)\right]^2 \tag{56}$$

The estimated gradient of this objective function $\nabla_{\boldsymbol{\omega}} J_Q(\boldsymbol{\omega})$ and the update rule for the stochastic gradient descent are defined as follows:

$$\begin{aligned} &\text{estimated gradient: } (\boldsymbol{\Delta}_Q)_k = \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{\omega}} \big(Q_{\boldsymbol{\omega}}(\mathbf{s}_{i_b}, \mathbf{a}_{i_b}) - \mathbf{y}_{i_b}\big)^2 \\ &\text{update rule: } \boldsymbol{\omega}_{k+1} \leftarrow \boldsymbol{\omega}_k - \eta_k (\boldsymbol{\Delta}_Q)_k \end{aligned} \tag{57}$$

where $\mathbf{y}_{i_b} = r_{i_b+1} + \gamma \left[Q_{\boldsymbol{\omega}^-}(\mathbf{s}_{i_b+1}, \mathbf{a}_{i_b+1}) - \alpha \log\big(\pi_{\boldsymbol{\theta}}(\mathbf{a}_{i_b+1}|\mathbf{s}_{i_b+1})\big)\right]$ and $\boldsymbol{\omega}^-$ is the parameter set of the target action-value function which is obtained as an exponentially moving average of the soft action-value function parameters (in order to mitigate bootstrapping), as with DQN [Mnih et al., 2015].

To automatically tune the temperature, Haarnoja et al. [2018b] propose to approximate the policy with a constraint optimization. It is formed by a minimum policy entropy threshold denoted $\bar{\mathcal{H}}$ such that $\mathcal{H}\big(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s})\big) \geq \bar{\mathcal{H}}$. They use the Lagrangian to transpose this constrained optimization into the previously presented policy and soft actor-critic optimizations, with an additional optimization process for the dual variable (i.e. the temperature). This third optimization must minimize the following objective function:

$$J_\alpha(\alpha_t) \triangleq \mathbb{E}_{\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}} \left[-\alpha_t \log\big(\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\big) - \alpha_t \bar{\mathcal{H}}\right] \tag{58}$$

The estimated gradient of this objective function $\nabla_\alpha J_\alpha(\alpha_t)$ and the update rule for the stochastic gradient descent are defined as follows:

$$\begin{aligned} &\text{estimated gradient: } (\boldsymbol{\Delta}_\alpha)_k = \alpha_k \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}} \big[-\log\big(\pi_{\boldsymbol{\theta}}(\mathbf{a}_{i_b}|\mathbf{s}_{i_b})\big) - \bar{\mathcal{H}}\big] \\ &\text{update rule: } \boldsymbol{\alpha}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k (\boldsymbol{\Delta}_\alpha)_k \end{aligned} \tag{59}$$

3

STATE REPRESENTATIONS FOR REINFORCEMENT LEARNING

> Well-connected representations let you turn ideas around in your mind, to envision things from many perspectives until you find one that works for you. And that's what we mean by thinking!
>
> Marvin Minsky *in* Will Robots Inherit the Earth?, 1994

3.1 INTRODUCTION

The challenge of representing high-dimensional data occupies a key place in machine learning. Finding representations boils down to finding patterns in these data. It is an essential prerequisite for any of the machine learning systems we envision. On the one hand, proper representations do not require much further processing and simple decision mechanisms are sufficient to achieve good results. On the other hand, poor representations require additional task-specific knowledge to achieve good results.

Knowing how to represent data is an essential skill that helps solving many mathematical and engineering problems. Traditionally, researchers seperate feature extraction and automatic decision making tasks into two distinct problems [Schmidhuber, 2015]. In particular, in reinforcement learning (RL) when only a sensory measurement is available instead of a state, it is recommended to first solve a state estimation problem which could correspond to a feature extraction problem. A proper feature extractor involves discovering patterns and relationships within data in order to communicate them as meaningful representations to help future learning systems. Indeed, systems must be able to interpret and analyze data in order to make useful predictions. Different representations from the same data can convey different information. Thus, a conceptor has to devise an appropriate representation for each automatic decision making task under consideration. In general, this means reducing the data dimensionality by focusing on useful information and neglecting unnecessary information (such as distractors) with respect to the task at hand, and of course reducing redundancy.

The current century is undoubtedly one in which data plays a key role in mathematical discoveries [Donoho et al., 2000, Bottou, 2015]. This is due to the easier collection of data in large quantities, which gives rise to the large-scale context. The major change due to the large-scale setting, is however not based on the increase of data cardinality N, but on the increase of data dimensionality D. In traditional data analysis settings, the input variables are carefully designed by human experts, resulting in reasonable D values. Instead, in the large-scale settings observations come from various sensors (e.g. visual sensors, motor sensors, contact sensors, radar sensors), and thus are composed of many more variables, hence the increase of data dimensionality. In addition, these variables are usually corrupted by noise from various error sources. This is why in large-scale settings it is much more complex to circumvent the *curse of dimensionality* which leads to unbounded computational and storage complexity (see Section 2.1.1). The general strategy is to apply dimensionality reduction techniques which involve transforming observations into smaller representations to reduce redundancy and noise in sensory measurements.

Moving from traditional statistical methods to sophisticated learning methods, the machine learning literature tends to show that deep neural networks are very efficient for dimensionality reduction [Bengio et al., 2007, Ur Rehman et al., 2016]. These techniques are either applied to reduce observation dimension immediately or are customized to solve specific problems. The latter approach allows a context-aware analysis of data stream in the learning pipelines. In the particular case of RL systems, this amounts to extract features from the agent's observation in order to approximate a value function and/or a policy. Specifically, it must extract features based on the transition probabilities of the environment through agent-environment interaction loops.

3.1.1 *Dimensionality Reduction*

Representation learning is a long-standing problem in machine learning which is one of the main goal of unsupervised learning [Bengio et al., 2013a, LeCun et al., 2015]. Dimensionality reduction is the process of mapping a high-dimensional data space to a new low-dimensional space acting as an embedding. This space can also be called depending on the context, *code space*, *feature space*, *bottleneck*, or *state space*. Many RL approaches use data compression-decompression techniques to perform dimensionality reduction [Schmidhuber, 2015]. Indeed, this unsupervised learning approach extracts features from data with greater relevance and autonomy than feature engineering approaches that require task-specific knowledge [Kober et al., 2013]. These dimensionality reduction techniques may be integrated into a RL system to improve its computational speed and performance [Böhmer et al., 2015, Schmidhuber, 2015].

A fundamental hypothesis of machine learning is that data distributions live in a lower d-dimensional manifold than the much higher D-dimensional data

space (i.e. $d << D$) [Cayton, 2005, Bishop, 2006, Bengio et al., 2007, Murphy, 2012]. A classical pedagogical example consists in the i.i.d.[1] uniform sampling of each pixel of an image: it inevitably produces meaningless images. In fact, even complex data such as images lie on a manifold of lower intrinsic dimensionality than their dimensionality (which is the number of pixels multiplied by the number of channels). All dimensionality reduction methods are based on this low-dimensional manifold hypothesis for data.

One of the first methods proposed was PCA (Principal Component Analysis [Pearson, 1901]). This method finds a linear projection of the data space to a lower-dimensional space corresponding to the data compression. This projection is such that it minimizes the quadratic reconstruction error between the decompressed code and the original data. Moreover, this projection transforms correlated observations into linearly uncorrelated representations. Indeed, the first dimension of the code space, called the first principal component, captures the maximum variance of the data. This is followed by the other principal components, each capturing the maximum variance uncorrelated with all the previous components. This technique was successfully used as a state embedding strategy for RL systems. For example Curran et al. [2016] has used PCA to reduce a visual data space from many demonstrations to extract a low-dimensional state space.

However, feature extraction techniques based on data covariance (such as PCA or factor analysis) do not necessarily produce useful features on a dataset obtained from transitions of a control system, as they may lose their physical meaning [Rao, 1999]. Indeed, the most important features in this case are clearly those that correlate with the changes between transitions, which generally do not coincide with the directions of greatest variation in the dataset. In particular, PCA and factor analysis are more adapted to non-time series data.

The hypothesis that data points lie on manifolds of low intrinsic dimension embedded in the high-dimensional data space is at the basis of the manifold learning approaches that have attracted the attention of many machine learning researchers [Narayanan and Mitter, 2010, Bengio et al., 2013b, Chui and Mhaskar, 2018]. The machine learning literature tends to prove that it is generally impossible to find a linear mapping between high-dimensional data and manifolds [Bengio et al., 2007, 2013b]. This is why researchers propose nonlinear methods to retrieve subtle information from the data stream to find a low-dimensional representation of the data.

The PCA traditional method was then extended in two nonlinear versions: (i) the kernel PCA [Schölkopf et al., 1998] which models a projection with kernel evaluations in a Reproducing Kernel Hilbert Space (RKHS), (ii) the autoencoder [Bourlard and Kamp, 1988, Kramer, 1991, Vincent et al., 2010] which models the projection by two neural networks for compression and decompression. An encoder is the part of the autoencoder which compresses the data, and the decoder is the part that decompresses it so as to minimize the reconstruction

1 i.i.d. = independently and identically distributed

objective. We believe the principle on which deep learning systems are able to solve this problem of reorganizing manifold information is based on *the blessings of dimensionality* [Donoho et al., 2000] and the *blessing of compositionnality* [Poggio and Liao, 2018].

Autoencoders have proven to be one of the most efficient techniques for compressing data [Vincent et al., 2010]. We recall that data compression is a special case of dimensionality reduction where the original data can be reconstructed from the code space without too many information loss. In the RL context, autoencoders can be used as a state estimator in a rewardless environment. For example, Lange and Riedmiller [2010] first applied this approach to NFQ (neural fitted Q-learning algorithm [Riedmiller, 2005]). [Lange and Riedmiller, 2010] is one of the first approaches to solve a deep RL problem from visual observations with a state representation learning strategy. This illustrates the difference in performance gain that is achieved by reducing dimensionality and better structuring information from data in a pretraining phase to later solve RL tasks, whereas learning in an end-to-end fashion is much more computationally inefficient. Indeed, applying directly deep RL algorithms to visual data streams is only possible since the breakthrough of convolutional neural networks [Mnih et al., 2013, 2015].

GANs (Generative Adversarial Networks [Goodfellow et al., 2014]) consist of a minimax game between a generator neural network on the one hand, and a discriminator (a.k.a. critic) neural network on the other hand, in order to generate data (from noise variables following some prior distribution) that the discriminator must not distinguish from the real data. This extended in multiple variants like AAE (Adversarial Autoencoders [Makhzani et al., 2015]), InfoGAN (Information maximizing Generative Adversarial Networks [Chen et al., 2016]), ALI (Adversarially Learned Inference [Dumoulin et al., 2017]), and BiGAN (Bidirectional Generative Adversarial Network [Donahue et al., 2017]). These four methods transform an autoencoder into a generative model, which means that they also learn an encoder and a decoder that map data to code space and vice versa. This compression approach may help reduce data dimensionality in the RL context. For example Shelhamer et al. [2017] use BiGAN to learn state estimators.

As confirmed by previous works, such dimensionality reduction methods to build a state estimator may help RL systems [Lange and Riedmiller, 2010, Lange et al., 2012]. However, dimensionality reduction performed with observation reconstruction, may not match the space where the actual RL is performed [Böhmer et al., 2015, Shelhamer et al., 2017]. Indeed, the training process does not take into account the physical world information (specifically the transition probabilities), useful for an agent's control. For example, these methods remove redundant information, but can also lose nonredundant information from environment transitions as empirically confirmed by many authors [Watter et al., 2015, Böhmer et al., 2015, Lesort et al., 2018, de Bruin et al., 2018]. Specifically, the information lost during data compression could have been useful to approximate target functions such as (action-)value functions and/or a policy. As a consequence,

these classical dimensionality reduction methods tend to perform poorly with RL systems in practice.

Another dimensionality reduction approach is SFA (Slow Feature Analysis [Wiskott and Sejnowski, 2002]), which is as the compression-decompression approach an unsupervised learning technique. In addition to the low-intrinsic dimensionality hypothesis of a manifold, it makes the slowness hypothesis. Specifically it decomposes data with respect to different time scales. In other words, it amounts to making temporally close data samples similar in the learned state space. This approach is efficient to learn state representations from high-dimensional sensory data for RL, as proved by these extensions of the original SFA technique [Wiskott and Sejnowski, 2002]: IncSFA (Incremental SFA Kompella et al. [2011a]), H-IncSFA (Hierarchical IncSFA Legenstein et al. [2010]), and AutoIncSFA by (Autoencoder IncSFA Kompella et al. [2011b]). AutoIncSFA proposes a combination of the best of the autoencoder and IncSFA by training an IncSFA on top of the compressed data to extract abstract *spatio-temporal features* for state estimation.

3.2 SCALING REINFORCEMENT LEARNING TOWARDS ROBOTICS

Our goal is to show that end-to-end deep RL (DRL) has limitations for large-scale tasks studied in this thesis, i.e. with a continuous action space and a high-dimensional observation space[2]. Although the assertion that this end-to-end strategy is unsuitable remains unproven, in this thesis we study an alternative to solve such control tasks – *state representation learning* (SRL). SRL is a way to simplify end-to-end DRL optimization problems, by first solving the problem of learning state representations in a rewardless environment, which we will explain in Section 3.3.

The unsupervised representation learning techniques presented in Section 3.1.1, based on dimensionality reduction, correspond to the first SRL methods. They were essential for the early applications of RL in robotics [Lange and Riedmiller, 2010, Legenstein et al., 2010] by bypassing the curse of dimensionality. However, these applications used toy control tasks, for example a pixelized grid-world (with 30×30 images) [Lange and Riedmiller, 2010], a visual slot car racer task (with 52×80 images) [Lange et al., 2012], and a "Morris water maze task" (with 155×155 images) [Legenstein et al., 2010]. In general, it is not the image input dimensions which make the task difficult (e.g. the images have 155×155 dimensions for the task studied by Legenstein et al. [2010]), but the intrinsic dimensions of the manifolds underlying the data space, and also the dimension of the continuous action space. Although these previous methods of dimensionality reduction have allowed to gain in autonomy with respect to traditional feature engineering

2 In this thesis we focus on control tasks with image observations which are predominant in the literature, however other sensory observations may be used.

techniques, they are still limited in terms of generalization capacity [Jaderberg et al., 2017, Shelhamer et al., 2017].

It is therefore necessary to further improve the performance of SRL methods in order to apply them to large-scale control tasks[3]. The ubiquitous strategy for this is to take advantage of recent advances in deep learning to approximate high-dimensional target functions. Thanks to the data-driven principle on which the Machine learning is based, deep learning techniques can take full advantage of many computational improvements to remove the human expert from the loop of the problem solving process. Indeed, the human work required to implement algorithms with deep learning is typically reduced to the design of different modules in a neural network model, as well as their architectures and other training hyperparameters (see Section 2.2).

The ability to perform both representation learning while solving an automatic decision making task belongs to the popular *end-to-end training* trend. It has been popularized by Krizhevsky et al. [2012] for image classification, and has subsequently been widely studied across all the machine learning domains: from image detection [Szegedy et al., 2015], image representation [Zhang et al., 2017b], image generation [Goodfellow et al., 2014], image deblurring [Eboli et al., 2020], etc. [Jordan and Mitchell, 2015]. On the control theory side, Mnih et al. [2013] popularized the combination of DRL techniques with end-to-end training, giving rise to many new DRL algorithms until today [Kostrikov et al., 2020].

3.2.1 *Pros and Cons of End-to-End Deep Reinforcement Learning*

In what follows, we outline the pros and cons of end-to-end DRL. This list is certainly not exhaustive, and is intended to provide an overview of how we perceive this approach to solve the large-scale tasks studied in this thesis. However, it is not our intention to judge it on the basis of arguments that weigh more on one side than the other. The objective is to explain the reason that pushes us with large-scale tasks, to use an approach opposite to that of end-to-end training. This approach, chosen as the object of study for this thesis, is the one proposed by state representation learning (SRL), which generally uses deep learning techniques in order to automate the feature engineering process just like the end-to-end approach. In contrast, SRL differs by the fact that it uses an independent training from the control task and without reward.

3.2.1.1 *Pros of End-to-End Deep Reinforcement Learning*

The first merit is based on the capacity of deep learning systems to approximate high-dimensional functions [Glasmachers, 2017]. This is mainly possible thanks

3 In this thesis we study the most common scaling of RL algorithms. It concerns control tasks in simulation with continuous action spaces with no more than ten dimensions, and continuous observation spaces formed by visual sensory measurements of thousands of dimensions.

to their capacity to compactly retrieve the hidden variables underlying the observations as explained by Bengio et al. [2007]. End-to-end DRL uses this capacity to simultaneously represent data and approximate a value function and/or a policy. In particular, their models are composed of different modules, with a main one serving as the state representation module, and one or two others to approximate a value function and/or a policy. Traditionally, an input processing module was designed using state estimation techniques such as the Kalman filter [Kalman, 1960b] or SLAM techniques [Bailey and Durrant-Whyte, 2006a,b] which require expert-based knowledge, or using feature engineering methods such as SIFT [Lowe, 1999] or SURF [Bay et al., 2006] which require task-specific knowledge. This module was then used to train the rest of a RL model to solve an automatic decision making task. Thus, in end-to-end DRL systems, deep learning eliminates the tedious phase of the task-specific feature engineering process, in a manner suitable to the RL context [Kober et al., 2013, Arulkumaran et al., 2017].

The second merit stems from the previous one, which is the fact that end-to-end training of a whole model tends towards a sole goal [Silver et al., 2021]. In other words, the formulation of this problem fits harmoniously with the data-driven learning principle on which the Machine learning is based [Glasmachers, 2017]. Specifically, the optimization problem is only based on the maximization of rewards. According to the hypothesis of Silver et al. [2021], this is enough for agents to achieve knowledge, learning, perception, etc. However, this remains a controversial point as it has not yet been possible to realize such applications.

Due to the merits listed first by Glasmachers [2017], the end-to-end training approach has gained popularity in the RL community to solve large-scale control tasks (especially image-based). These merits were first reflected with discrete-action tasks on the ALE benchmark (Arcade Learning Environment [Bellemare et al., 2013]) with the DQN (Deep Q-Network [Mnih et al., 2013]) algorithm. These end-to-end approaches have particularly benefited from stochastic optimization techniques specific to supervised learning, as explained in Section 2.3.2.2. DQN is a pillar in the RL field, as it is the first DRL algorithm to be able to approximate a value function directly from the visual sensory flow of an agent.

However, the end-to-end approach took several years before it could be applied on continuous control tasks from visual inputs. These tasks correspond to those studied in this thesis. Moreover, while the first methods of end-to-end DRL in continuous robotics often apply to manipulation tasks [Levine et al., 2016, 2018], we focus on different tasks of varying difficulty in terms of partial observability and controllability. It is only a few years later that Tassa et al. [2018] propose the first benchmark of such control tasks, called DeepMind Control Suite (DMControl), on the MuJoCo physic simulator [Todorov et al., 2012]. The observation space in each environment can correspond to the output of a camera tracking the robot; or the traditional low-dimensional (dozens of dimensions) ground truth state composed of the angular positions and velocities of the robot's articulations (or joints) and some of his cartesian coordinates.

Prior to the DMControl benchmark, Brockman et al. [2016] proposed the OpenAI Gym benchmark including most of the DMControl tasks (also on the MuJoCo simulator) and other tasks. However, it only provides by default compact vector observations. This implies that the majority of works evaluated on this benchmark do not fit into the large-scale setting studied in this thesis [Duan et al., 2016, Schulman et al., 2017]. On the other side, despite their low-dimensional observation spaces, these MuJoCo benchmark tasks are particularly complex to solve since the dimensions of the action spaces and the degrees of freedom of robots are relatively high. This explains, why it is so difficult for deep learning systems to approximate a value function and/or a policy, even with compact vector observations. In order to solve these complex tasks, the DRL community has proposed many new algorithms [Lillicrap et al., 2015, Barth-Maron et al., 2018, Haarnoja et al., 2018b]. Some of these algorithmic enhancements for RL have led to promising performance when the camera is used in place of the ground truth state on DMControl benchmark tasks [Tassa et al., 2018], especially with D4PG (Distributed Distributional Deep Deterministic Policy Gradient [Barth-Maron et al., 2018]).

3.2.1.2 *Cons of End-to-End Deep Reinforcement Learning*

In what follows, we present the limitations of end-to-end DRL, which have been pointed out in particular by Glasmachers [2017], Stoica et al. [2017], Sünderhauf et al. [2018].

The first limitation concerns the learning signal used to jointly train modules with different roles in a DRL model. According to Glasmachers [2017], training a module with an inappropriate learning signal may hinder the overall model training. This is particularly the case when a module dealing with visual inputs (called state representation module) is trained with very different modules responsible for approximating a value function and/or a policy, based on rewards. Indeed, rewards may be sparse, contain little information, and be delayed (the longer the horizon of the task, the more it is delayed).

The second limitation is that end-to-end training of a DRL model cannot explicitly take advantage of the role the conceptor has assigned to each of its modules [Glasmachers, 2017]. Indeed, the conceptor of a RL model, generally chooses one module to process the visual inputs (often a CNN), and one or two modules to approximate a value function and/or a policy (often a MLP). According to Glasmachers [2017], this can affect the success of the optimization problem, as the modules may interact with each other in a noxious way. This is particularly the case when the state embedding module has overfitted on the first training observations. Indeed, deep neural networks are particularly prone to be attracted to poor local optima due to their non-convexity [Erhan et al., 2010]. Therefore, the more modules there are, the more likely it is that a module will get stuck in one of them and thus hinder the training of the rest of the model.

The third limitation is the instability and slow convergence of their optimization problems. Instability is mainly caused by non-convex objective functions due to the nature of deep neural networks [Bengio et al., 2007]. As a consequence, the landscape of the objective function is full of many local optima, most of which have poor generalization performance. Thus, in spite of the efficiency of the stochastic gradient descent to find a solution that can generalize to a large dataset, it may quickly get stuck in one of these poor local minima. In other words, the neural network risks overfitting on the first training examples [Erhan et al., 2010]. Slow convergence is mainly caused by ill-conditioning due to rank-deficient Jacobians[4] of deep neural networks [Saarinen et al., 1993]. This makes them data intensive, i.e. they have a low sample efficiency.

The fourth limitation, is the lack of interpretability of the modular functioning of their models on control tasks, even if the overall functioning remains mysterious because of black box neural networks [Busoniu et al., 2018, Li, 2018]. Furthermore, according to Busoniu et al. [2018], when learning end-to-end a value function, it encodes the whole transition model underlying the environment without discovering its parameters. The lack of interpretability to the point that even the role of each module is imprecise, makes end-to-end DRL models difficult to apply in the real world [Stoica et al., 2017] where systems must comply with very strict safety regulations if there is a risk to humans. However, this is impossible in the current state of our means to interpret DRL systems, as well as to explain their decisions (which is well explained by Stoica et al. [2017], who propose a view of systems challenges for artificial intelligence).

The first three limitations of end-to-end DRL algorithms listed above, tend to make the convergence of an optimization algorithm slow and unstable. Even in spite of efforts to circumvent the curse of dimensionality by reducing the dimension through the state embedding (for example, Tassa et al. [2018] use an output dimension of 50 for all their DMControl benchmark tasks), training simultaneously multiple modules may fail. Indeed, since these optimization problems are subject to overfitting and slow convergence, it seems more challenging to circumvent the curse of dimensionality in this context. Moreover, they are all the more complex the higher the dimension of the actions, and the longer the horizon of the task.

To address these problems, works in end-to-end DRL have recently taken advantage of data augmentation techniques commonly used in computer vision tasks, which act as regularizers [Laskin et al., 2020, Kostrikov et al., 2020]. Laskin et al. [2020] use a simple data augmentation technique which is random image translations (and random amplitude scaling), without modifying the DRL algorithms (which are SAC [Haarnoja et al., 2018b] and PPO [Schulman et al., 2017]). In contrast, Kostrikov et al. [2020] use random cropping (among others) and regularized Q-functions with consequent modifications to the SAC algorithm. These two end-to-end DRL algorithms are evaluated on the DMControl benchmark

4 A Jacobian is rank-deficient as soon as it has two almost linearly dependent columns.

tasks [Tassa et al., 2018] from images, and achieve the same performances as when ground truth states are used.

However, these successful performances remain inefficient in terms of sample efficiency as pointed out in particular by Srinivas et al. [2020]. Therefore, we need to develop other alternatives to end-to-end DRL approaches if we are to extend RL algorithms to real-world control tasks, such as robotics. Indeed, the conditions in robotics are different because of the limited availability of data. It is therefore necessary to find solutions to the convergence issues (i.e. unstable and slow) encountered by RL algorithms.

The limitations of end-to-end DRL are still little studied in the literature. Among these rare studies, that of Glasmachers [2017] has made it possible to show, through a detailed empirical analysis, the limitations listed above. In order to highlight only the complexity of end-to-end training and not the complexity of the task, they used toy tasks in their experiments, in the contexts of supervised learning and RL. Their studies focus in particular on the phenomena of dependencies between modules (like representation learning and memory formation) of a deep learning system during their joint training. Their results tend to show that these phenomena are one of the main causes of the poor performance of these approaches. Indeed, the joint end-to-end training of several modules can often collapse. In addition, they achieve better performance with modules that are properly pretrained individually and then frozen to train the rest of models. Although these results are by no means conclusive, they do suggest that training models consisting of several modules (as with the end-to-end DRL strategy) may benefit from pretraining one or more of their modules.

3.2.2 *Potential Solutions*

We have previously reviewed four main limitations of end-to-end DRL. The main objective of this section, is to introduce solutions to this end-to-end approach, in order to scale RL algorithms to continuous control tasks with high-dimensional sensory inputs. Although there are many domains that propose solutions to this approach, we focus on those that deal with improving the state representation process. In this category there are two mains strategies. The first is known as RL with auxiliary tasks which are specific to train state embeddings. However, it still trains the model in an end-to-end manner, including the module(s) of the value function and/or the policy, unlike the second approach. The second constitutes our field of study, – *state representation learning* (SRL) –, which solves the state representation problem independently of the RL problem, i.e. in the manner of an unsupervised state embedding pretraining without reward for RL.

3.2.2.1 *Reinforcement Learning with Auxiliary Tasks*

RL with auxiliary tasks, which aim at improving the state representation process, can be divided into two main categories. The first is placed under the perspective of learning the dynamics in a compact latent space, with in particular model-based RL algorithms. Here, the auxiliary tasks train a transition model simultaneously with a state embedding. The second is placed under the perspective of model-free RL. Here, the auxiliary tasks focus on the state representation training from which the value function and/or the policy of the model-free RL algorithm are learned.

We now review some of the works from the perspective of model-free RL. Their auxiliary tasks are often borrowed from the self-supervised techniques used in unsupervised learning, without necessarily being related to the control task. This is particularly the case for auxiliary tasks based on constrative learning [Dwibedi et al., 2018, Srinivas et al., 2020]. Other auxiliary tasks are more related to the control. This is particularly the case for those based on dynamic constraints, for example by minimizing the errors of an inverse model in the state space [Shelhamer et al., 2017, Hansen et al., 2020]. Similarly, Munk et al. [2016] do so with a forward model, in which the state and reward must be predicted with respect to the previous action and observation. Finally, Jaderberg et al. [2017] introduce as an auxiliary task, the maximization of a pseudo-reward function. Generally in these works, there are various auxiliary objective functions to be minimized within a single training, in addition to the one related to the RL optimization. Moreover, the observation reconstruction auxiliary tasks were reported as less efficient than the other auxiliary self-supervised tasks in the following works [Jaderberg et al., 2017, Shelhamer et al., 2017, de Bruin et al., 2018].

Now we review some of the most promising works from the perspective of model-based RL. PlaNet [Hafner et al., 2018] is one of the leading approaches that has scaled-up model-based RL on the DMControl benchmark from images [Tassa et al., 2018]. They use a stochastic (or deterministic) transition model that uses a learned compact latent space. This compact latent space is learned in the manner of an autoencoder with dimensionality reduction. Thus, the transition model can predict many future sequences in parallel thanks to its memory efficiency. Numerous works have been inspired by it until today to obtain better results on DMControl tasks from images [Hafner et al., 2019, Lee et al., 2019, Sekar et al., 2020]. The advantage of these methods with two of the previous strategy [Srinivas et al., 2020, Hansen et al., 2020], is that they work even on partially observable tasks, as is the case with those of the DMControl benchmark.

Since [Hafner et al., 2018, 2019, Lee et al., 2019, Srinivas et al., 2020] evaluate their methods on the same DMControl benchmark tasks and use the same model-free RL algorithm with continuous actions – SAC (Soft Actor-Critic [Haarnoja et al., 2018b]) – we try to see how each ranks. First, we observe from the results obtained in these four works that the performance obtained with SAC using

their methods is better than that obtained with the classical end-to-end approach without auxiliary tasks. In addition, they are all superior to those obtained with autoencoder reconstruction techniques (i.e. AE and variants such as VAE [Kingma and Welling, 2014]). However, in the results of SLAC [Lee et al., 2019] and CURL [Srinivas et al., 2020] the performance of PlaNet [Hafner et al., 2018] is inferior to the autoencoder compression technique. In addition, the performance of SLAC slightly outperforms that of DrQ [Kostrikov et al., 2020][5]. Finally, CURL has better performances than SLAC and Dreamer [Hafner et al., 2019]. At the end of this analysis of the results obtained in the literature, it would seem that, approaches with auxiliary tasks performed according to contrastive learning methods are promising in order to optimize state representation models for RL algorithms.

The above empirical results show that, some of the limitations of the end-to-end approach can be overcome by carefully selected auxiliary tasks. Specifically, they are intended to replace the poor learning signal of RL based on rewards, to better train state embeddings. Indeed, these approaches combine various auxiliary objectives, which only concern the state representation module, typically in order to avoid slow convergence and especially to bypass poor local optima [Jaderberg et al., 2017, Hafner et al., 2018].

Furthermore, this improvement on the optimization of the state representation learning, combined to the dimensionality reduction, helps the optimization of RL to bypass the curse of dimensionality. For example, Srinivas et al. [2020] use a state representation dimension of 50 on all DMControl benchmark tasks [Tassa et al., 2018]. This is why these approaches tend to improve the optimization of end-to-end DRL algorithms, and thus to obtain better performance on control tasks. However, these methods are too computationally intensive to be easily applicable (especially in our hardware setting). In addition, they can suffer from sample inefficiency as shown in the results of Shelhamer et al. [2017], where optimization iterations of the order 10^7 are required for a policy to converge. This is mainly due to the non-stationary distribution that the state representations follow. Indeed, during the state embedding training, the RL inputs evolve, thus breaking the identically distributed Robbins-Monro condition for the stochastic gradient descent to converge (see Section 2.2.1.1).

3.2.2.2 *State Representation Learning (SRL)*

SRL is a domain of machine learning, recently considered as such by Jonschkowski and Brock [2013], for the first time reviewed by Böhmer et al. [2015], and popularized in contexts different from that of state estimator pretraining by Lesort et al. [2018]. In this thesis, we restrict the study of SRL to the context of state estimator pretraining in an environment without reward for future unknown

5 SLAC [Kostrikov et al., 2020], previously presented with the classical end-to-end approaches in Section 3.2.1.2, does not use an auxiliary task strategy to help the state representation learning process, unlike the methods proposed in this section.

RL applications. In other words, to a context that explicitly decomposes the two sub-problems that end-to-end DRL algorithms are confronted with: state representation learning and learning a control task. On the contrary, the strategy with auxiliary tasks solves these two different optimization problems jointly [Shelhamer et al., 2017]. Thus, SRL removes the challenging twofold training procedure of a state representation module with the rest of the model, i.e. the value function and/or the policy module(s) [Glasmachers, 2017].

The first limitation which is overcome is that the state representation module is trained with an appropriate learning signal instead of the extrinsic reward signal returned by the environment. Indeed, the latter tends towards the unique goal of solving a control task. However, although a reward signal is most often sparse and delayed, it can sometimes help eliminate unnecessary information, as is the case with the previously discussed strategy of using auxiliary tasks or with the SRL strategy in a context with rewards [Jonschkowski and Brock, 2015, Lesort et al., 2017].

The second limitation of not exploiting the compositionality principle is totally bypassed thanks to the SRL pretraining strategy. Indeed, as the state representation module respects its assigned role, the rest of the modules related to RL can also respect their roles, which is to approximate the value function and/or the policy. The effectiveness of this compositional learning paradigm is widely recognized for scaling machine learning algorithms to increasingly complex tasks, especially in RL [Bottou and Bousquet, 2018, Lake and Baroni, 2018, Glasmachers, 2017, Stoica et al., 2017, Sünderhauf et al., 2018]. In particular, they support the idea that it is more efficient to focus an optimization on the approximation of the value function and/or the policy than on a more complex problem involving representation learning. This idea is based according to Glasmachers [2017] on the principle of "divide and conquer", and according to Sünderhauf et al. [2018] on the principle of "complex problems should be solved by decomposition and re-composition".

The third limitation SRL tends to overcome is the convergence issues due to non-convex and ill-conditioned optimization process. Although the SRL does not solve them completely, it tends to improve their convergence thanks to a simplification of the patterns present in the state representations. According to Glasmachers [2017], non-convexity causes stochastic gradient descent to most often converge to poor local optima; ill-conditioning causes stochastic gradient descent to converge slowly. Independently and properly pretrained modules (which should be the case with SRL solutions), will generally improve the training of the rest of a RL model on a new control task [Erhan et al., 2010, Glasmachers, 2017]. Numerous works have tried to understand the reasons for the efficiency of pretraining on deep learning models [Erhan et al., 2010, Yosinski et al., 2014, Glasmachers, 2017, Liu et al., 2019a]. Liu et al. [2019a] propose a theoretical and empirical analysis to clarify how stochastic gradient descent can escape poor local optima. These improvements would be due, according to them, mainly to

the fact that partial derivatives are removed in the pretrained parameters, which stabilizes the magnitude of the estimated gradient of the objective function, and thus improves the objective function landscape. However, it is not yet clear, how effective pretraining is for DRL models, but promising results with other pretraining than SRL have already been achieved. In particular, as mentioned earlier in this section, Glasmachers [2017] have empirically analyzed that pretraining one of the modules of a deep learning model (in supervised and RL contexts) makes the stochastic gradient descent converge faster to better local optima.

Finally, the fourth merit concerns more the deployability of DRL algorithms, in particular their interpretability to some extent (indeed, the features of learned representations are not necessarily interpretable, although more and more tools exist to try to make sense of extracted features). Thanks to the compositionality principle guaranteed by SRL, it is possible to interpret the role of each neural network module. Furthermore, according to Busoniu et al. [2018], a better understanding of the representation learning process can help interpret a learned value function and/or policy. However, due to the nonlinear nature of deep neural networks, we still cannot fully understand their workings. The deep learning community has tried to alleviate this problem over the last decade with deep neural network interpretability algorithms [Zeiler and Fergus, 2014, Yosinski et al., 2015, Karpathy et al., 2015, Li et al., 2015, Olah et al., 2017, Nguyen et al., 2017, Samek et al., 2017]. In simple terms, these consist of identifying the input properties of the network that are responsible for a particular output. For example, Karpathy et al. [2015], Zeiler and Fergus [2014] use statistical analysis of unit activations, and ablation studies where specific units are disconnected or deactivated.

Thus, SRL methods can overcome three additional limitations of end-to-end DRL compared to approaches with auxiliary tasks. However, we do not pretend to know the relationship of these improvement factors to the problem of DRL optimization. We only know that its success seems to be related to the input dimensions, the training signal quality of each module, the stability at the modular interaction level, the stability and convergence speed of the optimization algorithm, and for security reasons to the interpretability at the modular functioning of its model (while the interpretability at the global functioning still remains impractical).

We now present our general SRL framework, to build on it in the remainder of this chapter and manuscript. This framework is restricted to unsupervised pretraining of state representations without reward, to later solve an unknown control task with a RL algorithm. As a result, many other domains beyond this framework will not be studied in this thesis. Below, we describe the three mains elements of this framework, which are common to many works in the SRL literature:

SOLUTION CRITERIA The solution criteria concern the a priori knowledge about the solution, which is necessary by the fact that SRL problems are

ill-posed[6] because they can have multiple solutions. They are based on the physical world properties the SRL solution should verify, and are surveyed in Section 3.3.1.

LEARNING HEURISTICS Learning heuristics translate the criteria on the SRL solution into mathematical language (more specifically the optimization language). They take the form of objective functions that provide learning signals and must be designed appropriately for each method. As this is a SRL framework without extrinsic reward[7], these heuristics must be reward agnostic. They will be reviewed through the different main SRL formulations from the literature in Section 3.3.2.

EXPLORATION STRATEGIES Exploration strategies train discovery policies which must be task-agnostic, and are typically optimized based on self-calculated rewards called intrinsic rewards [Burda et al., 2018]. Without proper exploration, learned representations would only have a very limited range of validity and usefulness. The few approaches currently used will be discussed in Section 3.3.3.

3.3 SRL FORMULATIONS

In this thesis, we study state representation learning (SRL) as a deep unsupervised pretraining of state representations without reward. As explained previously, this is what makes it possible to train RL models by overcoming the four limitations of the end-to-end approach listed in Section 3.2.1.2: (i) the poor learning signal of temporal-difference optimization (consequently the non-optimal dimensionality reduction), (ii) the lack of guarantee that the roles assigned to the modules are respected, (iii) the instabilities and slow convergence of their optimization problems, (iv) the lack of interpretability on the modular functioning of their models.

The general problem of SRL is to retrieve from sensory observations[8] compact representations about the agent's proprioceptive information which generally also require knowledge of the environment's properties (related to perception). In our setting, an agent receives an observation denoted $\mathbf{o}_t \in \mathcal{O} \subset \mathbb{R}^{\mathcal{O}_d}$, where generally $\mathbf{o}_t$ will be an image with $\mathcal{O}_d$ the number of pixels. Unlike the RL context, here the extrinsic reward signal $r_t \in \mathbb{R}$ is inaccessible. A SRL method learns a state estimator similar to a mapping φ which outputs state vectors belonging to a state embedding denoted $\mathcal{S} \subset \mathbb{R}^{\mathcal{S}_d}$. Such a mapping typically updates the state

6 A problem is ill-posed if it does not satisfy one of the three Hadamard criteria [Hadamard, 1902]: (i) a solution exists, (ii) the solution is unique, (iii) the solution depends continuously on the initial conditions (especially the data).

7 Extrinsic rewards are obtained indirectly from the environment (by an agent-environment interaction loop) as privileged information, because in real robotic settings we do not have it.

8 This thesis focuses on visual sensory observations, given its predominance in the literature. However, SRL problems can be applied to other sensory observation types.

estimation $\mathbf{s}_t \in \mathcal{S}$ as new information is received, from the current observation $\mathbf{o}_t$ and/or the past observation $\mathbf{o}_{t-1}$, possibly from the past action $\mathbf{a}_{t-1} \in \mathcal{A} \subset \mathbb{R}^{\mathcal{A}_d}$ and the previous state $\mathbf{s}_{t-1}$ as follows:

$$\mathbf{s}_t \triangleq \varphi\left(\mathbf{o}_t, \mathbf{o}_{t-1}, \mathbf{a}_{t-1}, \mathbf{s}_{t-1}\right) \tag{60}$$

End-to-end DRL algorithms build an input embedding from scratch which is a kind of φ, at the same time that they learn an automatic decision making task. In contrast, SRL problems are ill-posed (as explained in Section 3.2.2.2) and estimate φ based on learning heuristics that restrict the class of admissible solutions, to mitigate the ill-posedness. The main learning heuristics used in the SRL literature will be examined in Section 3.3.2. In general, they correspond to unsupervised/self-supervised tasks, which allow training φ in a similar way to supervised learning. Therefore, they have the advantage of fitting into the classical framework of approximation theory (we have detailed this framework in the context of deep learning for supervised problems in Section 2.2).

3.3.1 *Solution Criteria*

In what follows, we review recently proposed criteria for devising learning heuristics to deal with the ill-posedness of SRL problems, i.e. by formulating the SRL optimization problem. In other words, here we explain what the SRL criteria represent in order to understand what the corresponding SRL solutions mean and how they are modeled from data. These criteria have in common that they translate a priori knowledge of the physical world in order to guarantee the physical plausibility of the SRL solution. Indeed, purely mathematical criteria cannot be used to mitigate the ill-posedness of the SRL problem. More precisely, they rely on a physical analysis of a robot interacting with the physical world.

The first criterion on the solution to the SRL problem is that it must retrieve, in a compact representation, the agent's degrees of freedom in space and time [Jonschkowski and Brock, 2013, Böhmer et al., 2015, Lesort et al., 2018]. We draw a parallel between this criterion and the hypothesis of the broader problems of representation learning. It is the hypothesis that there are multiple manifolds underlying the observation space, which justifies dimensionality reduction. This capacity to retrieve the manifold information in a compact way, is the most fundamental, as it facilitates RL algorithms to overcome the curse of dimensionality.

The second criterion is that the SRL solution must model the local consistency and topology (or connectivity) of the environment [Jonschkowski and Brock, 2015, Lesort et al., 2017, Morik et al., 2019]. While the former concerns the transition model (i.e. how an agent transitions from one state to another), the latter concerns agent-independent information. Both must be retrieved by the state embedding in order to provide complete observability of the environment. In particular, the knowledge of topology is necessary to determine whether a point is more or less accessible depending on where the agent is located. This

criterion is one of the most difficult to satisfy because of its abstract and subjective nature, which can therefore only be obtained qualitatively (e.g. topologically). With this knowledge of local consistency and topology, the temporal-difference optimization on which RL algorithms are based could effectively project into future time steps (by bootstrapping).

As a pedagogical example, consider a navigation task to be learned with a RL algorithm, in an environment with walls, and where the observation space is a first-person perspective camera. In this case, a state estimator obtained with a SRL formulation should retrieve the spatial proximity of the different camera poses and taking into account the environment's topology. In other words, the camera poses on either side of a wall must be separated in the learned state embedding, although they are spatially close. If the state estimator does respect this criterion, the RL policy could learn to go around the wall so that an agent reaches a goal behind the wall. The reason for the effectiveness of such a state embedding according to Böhmer et al. [2013] would come from the fact that unlike the Euclidean metric defined with the observation space of first-person perspective images, the metric defined with the state embedding would have distances consistent with the amount of time steps away from them.

The third criterion is that the SRL solution must ignore the distractor source of information, and to retrieve the two information sources relevant to the automatic decision making process[9] (i.e. policy) [Jonschkowski and Brock, 2015, Jonschkowski et al., 2017]. The distractor source of information generally corresponds to elements that are beyond the agent's control and that do not affect him. The first source of relevant information is the controllable one, which corresponds to elements that can be controlled by an agent. The second one is the non-controllable source of information, it corresponds to the elements that an agent cannot control but that can affect him.

The fourth criterion that the SRL solution must verify is that the state transition (defined by the right term in Eq. 61) is Markovian [Sutton and Barto, 2018]. Indeed, RL algorithms are generally defined in the formalism of Markov decision processes (MDPs). An MDP is composed of a reward signal relative to the control task (which is ignored) in addition to a state space $\mathcal{S}$ (to which the φ outputs belong), an action space $\mathcal{A}$, and a Markovian state transition $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ (which is unknown in the regular RL context, for more details see Section 2.3.1.1). Formulated mathematically, for the state transition to be Markovian, it must verify the following equation:

$$P\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right) = P\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \ldots, \mathbf{s}_0, \mathbf{a}_0\right) \tag{61}$$

for all $\mathbf{s}_{t+1}, \mathbf{s}_t \in \mathcal{S}$ and $\mathbf{a}_t \in \mathcal{A}$. It differs from the Markovianity equation of the classical MDP formalism (see Eq. 24) since the reward is not available. For this

9 In a control task it may be necessary to detect elements inside the distractor source of information. In this case, the corresponding information can be concatenated later to the learned state representation.

reason, the SRL solution (i.e. φ) must guarantee the Markovianity of the state transition, so that it can be used as a state estimator in a RL model. In simple terms, this means that the current state must retrieve the necessary information about the past agent-environment interactions, so that the policy has no ambiguity on the actions to predict. Therefore, in a general partial observability context, state representations must be non-Markovian in order to be able to retrieve useful past information for a policy. However, the Markovian property is regularly used on the learned state representations as a shortcut [Böhmer et al., 2015, Jonschkowski and Brock, 2015]. Indeed, the SRL works in the literature often assume that the environments are totally observable [Lesort et al., 2018]. Therefore, to guarantee the Markovianity of the state transition in this context, it is sufficient to assume state representations are Markovian.

The fifth and last criterion is proposed as a potential avenue for the design of future SRL algorithms. This concerns the nature of the SRL solution itself, which we limit here for the sake of clarity, to a mapping of the observation space to a state space. In particular, this criterion could concern a well-chosen regularity imposed on this mapping. We suppose that this can favor the existence and the uniqueness of the solution. In other words, it can help mitigate the ill-posedness of the SRL problem. For example, we could choose to impose the L-Lipschitz continuity property on the mapping, which should therefore verify the following equation, for any consecutive observations $\mathbf{o}_{t+1}, \mathbf{o}_t \in \mathcal{O}$:

$$\|\varphi(\mathbf{o}_{t+1}) - \varphi(\mathbf{o}_t)\| \leq L\|\mathbf{o}_{t+1} - \mathbf{o}_t\| \tag{62}$$

Literally, this means that state variations must be bounded by observation variations, which is a stronger regularity than continuity.

This fifth criterion corresponds to regularization properties on the SRL solutions which may help to filter out distractors. Indeed, when a continuity constraint such as Lipschitz is used on φ, it allows to control output (state) variations with respect to input (observation) variations. This is why a distractor which disturbs observation variations will necessarily disturb less or not at all state variations. Looking in this direction is in our opinion promising, and we could benefit from techniques already developed in deep learning to implement the Lipshitz property on neural networks [Mescheder et al., 2018, Miyato et al., 2018].

As mentioned earlier, learning heuristics correspond to unsupervised/self-supervised tasks that formulate the SRL problem into a better-posed mapping approximation problem. Before to review them in Section 3.3.2, we roughly explain how to choose an approximator class which is the most compatible with this mapping. Indeed, it would be inefficient to learn a mapping if the chosen approximator could only give a poor state estimation. Moreover, as explained with the first criterion, the SRL solution must retrieve information about the agent's degrees of freedom in space and time in a compact way. This seems to be linked to the hypothesis that observations are located on a manifold of

intrinsic dimensionality lower than that of the observation space, due to strong correlations between the observation dimensions [Cayton, 2005, Bishop, 2006]. As the solution of such a mapping is generally nonlinear, the approximators must be nonlinear, hence the use of deep learning techniques [Bengio et al., 2007, 2013b]. Indeed, they are very effective in the context of learning compact representations of manifolds underlying the observation space, as proved by the literature reviewed in Section 3.1.1. For these reasons, deep neural networks are generally used to model the SRL solution via learning heuristics.

3.3.2 *Learning Heuristics*

We have previously listed some criteria that SRL solutions should meet. They are usually used to carefully devise learning heuristics. These heuristics mitigate the ill-posedness of the SRL problem by formulating it through self-supervised tasks. There are four main families of learning heuristics: (i) observation reconstruction, (ii) near-future prediction, (iii) state-based constraints, (iv) and constrastive prediction. The first family has already been reviewed in Section 3.1.1, and belongs to the more general domain of representation learning with dimensionality reduction. This section therefore reviews the other three families known in the SRL literature.

3.3.2.1 *SRL with Transition Estimation*

Most of the SRL solution criteria can be formulated via a near-future prediction objective. Specifically, it consists in a self-supervised task of next state prediction. It is based on the principle that, consecutive state vectors must verify roughly the Markovian transition equation (see Eq. 61). Mathematically, Eq. 61 implies that a SRL solution φ, estimates states so that their transitions are Markovian. Literally, this forces φ to retrieve the information of the physical world necessary to predict the next state from the current action and state. There are similarities between this approach and learning a transition model (i.e. model-based RL), which is why some approaches combine learning a model and a representation [Schrittwieser et al., 2019, Hafner et al., 2018, 2019]. However, in representation learning the prediction objective can be combined with other objectives, which shows that prediction is here not an end but the mean to construct a representation with relevant properties.

In this approach, some works sometimes assume that the transition model in the estimated state space is linear [Watter et al., 2015, van Hoof et al., 2016]. This allows to force the transition model to approximate affine dynamics, which in their view is appropriate for control. However, the majority of these approaches learn complex nonlinear transition models [Assael et al., 2015, Wahlström et al., 2015].

The above works formulate the SRL problem, with two main distinct learning heuristics. The first one consists of an observation compression objective, which allows to train a state estimator in the manner of an autoencoder, where φ consists of an encoder. That is, they use a reconstruction loss between the state and the current observation, which may have the drawbacks of the approaches previously examined in Section 3.1.1 [Jaderberg et al., 2017, Shelhamer et al., 2017]. This learning signal is as follows:

$$\|\mathcal{D}\left(\varphi(\mathbf{o}_t)\right) - \mathbf{o}_t\|_2^2 \tag{63}$$

where φ denotes the encoder which outputs the state $\mathbf{s}_t$, and $\mathcal{D}$ denotes the decoder which reconstructs the observation $\hat{\mathbf{o}}_t$.

The second one consists of the near-future prediction objective (mentioned above), which trains a state transition model from the φ outputs. Thus, φ is constrained to estimate states so that, their transitions are Markovian, i.e. they verify Eq. 61. The corresponding learning signal must roughly follow this loss function:

$$\|\mathcal{F}(\mathbf{s}_t, a_t) - \mathbf{s}_{t+1}\|_2^2 \tag{64}$$

where φ is a state estimator which predicts the "real" state $\mathbf{s}_t$ and next state $\mathbf{s}_{t+1}$, and $\mathcal{F}$ is a forward model which predicts the next state $\hat{\mathbf{s}}_{t+1}$.

In a simple way, these approaches minimize errors from an observation reconstruction objective, and a near-future state prediction objective. These models are typically used to solve long temporal predictions in a compact state space. In contrast to these approaches, Ha and Schmidhuber [2018] distinctly train an autoencoder (in particular a VAE [Kingma and Welling, 2014]) in a first phase, and then a state transition in a second phase. All these methods allow to guarantee the spatial and temporal coherence of the physical world information within the compact outputs of φ. However, their limitation is that they build φ with a reconstruction heuristic on the observation space. This implies that φ outputs keep information from the images that may not be useful and thus impede dimension reduction. The next SRL formulation, takes this weakness into account by relying solely on dynamic criteria of the physical world.

3.3.2.2 *SRL with Dynamic Constraints*

The solution criteria may be formulated with constraints directly on the state space. One popular formulation of this strategy, is the combination of a slowness and diversity constraints on the learned states [Jonschkowski and Brock, 2013, 2015, de Bruin et al., 2018]. The slowness constraint is typically translated into mathematical language with the following loss function:

$$\|\mathbf{s}_{t+1} - \mathbf{s}_t\|_2^2 \tag{65}$$

and for the diversity, with the following loss function:

$$e^{-\|\mathbf{s}_{t'}-\mathbf{s}_t\|_2^2} \tag{66}$$

where $(\mathbf{s}_{t+1}, \mathbf{s}_t)$ are consecutive states while $(\mathbf{s}_{t'}, \mathbf{s}_t)$ are temporally distant states predicted by a state estimator φ. The slowness constraint is based on the assumption that states should not change quickly [Wiskott and Sejnowski, 2002]. However, this formulation on the one hand cannot take into account information about a fast-moving agent, and on the other hand, cannot filter out slow distractors. Furthermore, this constraint has a trivial solution which corresponds to a constant state, and therefore does not allow to retrieve relevant information. This is why another diversity constraint accompanies it, in order to move the states away as a function of the number of time steps, as shown by Eq. 66. However this formulation can cause issues if the agent remains stationary for a while. Below is a strategy to remedy this.

One way to improve the retrieval of the two relevant information sources (see Section 3.3.1), is to use other constraints based on the trainings of forward/inverse models [de Bruin et al., 2018, Raffin et al., 2019]. The loss function to train an inverse model can be expressed mathematically as follows [10]:

$$\|\mathcal{I}(\mathbf{s}_{t+1}, \mathbf{s}_t) - \mathbf{a}_t\|_2^2 \tag{67}$$

and the one to train a forward model is:

$$\|\mathcal{F}(\mathbf{a}_t, \mathbf{s}_t) - \mathbf{s}_{t+1}\|_2^2 \tag{68}$$

where $\mathcal{I}$ is an inverse model which predicts the executed action $\hat{\mathbf{a}}_t$ from the state $\mathbf{s}_t$ and the next state $\mathbf{s}_{t+1}$ predicted by a state estimator φ. It allows to retrieve the controllable information of the physical world by an agent. $\mathcal{F}$ is a forward model which predicts the next state $\hat{\mathbf{s}}_{t+1}$. It allows to retrieve the (non-)controllable information source of an agent (and which can affect him). Moreover, these learning heuristics make it possible to ignore the distractor source of information.

Other constraints have been formulated by Jonschkowski and Brock [2015]. In addition to the constraints of slowness and diversity, they propose two other main constraints. A first one is the proportionality constraint, which is based on the principle that, the information retrieved by φ must change proportionally to the action's magnitude. A second one is the repeatability constraint, which is based on the principle that, the information retrieved by φ must change in the same way when the actions and the state vectors are similar. Jonschkowski et al. [2017] extend these robotic priors in the context of acceleration-dependent control tasks with a speed estimated from state changes. Subsequently, Lesort et al. [2017] study the applicability of these robotic priors to simulators in the physical world. However, they obtain unstable results, usually lacking local consistency

10 For simplicity, we only give examples with the L_2 norm, but other distances may be more relevant in special cases.

with respect to different locations in the environment (see Section 3.3.1). That is, similar trajectories are removed by φ. To counter this problem, they have introduced the reference point prior, which allows different trajectories belonging to a same region to be brought closer together. There seems to be a parallel with clustering, which for example, must bring together pairs of distinct points that belong to the same semantics. A limitation of these methods, is that they assume total observability of the environments. To remedy this, Morik et al. [2019] extend [Lesort et al., 2017] (i.e. with the reference point prior) on partially observable environments. To do so, they adapt robotic priors with LSTMs networks to manage partial observability.

Although these works propose promising heuristics due to their originality and their physical interpretability, they have not yet been validated on DMControl benchmark tasks [Tassa et al., 2018], or with other continuous control tasks with image inputs, studied in particular in this thesis.

3.3.2.3 *SRL with Constrastive Learning*

A last heuristic family to formulate the SRL problem, is to use discriminatory tasks. Discriminatory tasks are originally formulated under the supervised learning paradigm [Le-Khac et al., 2020, Jaiswal et al., 2021]. In contrast, in the SRL context these tasks must be carefully chosen in such a way as to meet the SRL solution criteria. This is the case with the works by Sermanet et al. [2018], Dwibedi et al. [2018], which use time as a supervision signal to learn the structure present in the videos, and build a robust viewpoint-invariant visual mapping. Dwibedi et al. [2018] has extended [Sermanet et al., 2018] to take speed into account. This work builds on the discriminative loss proposed by Sohn [2016]. However, these methods require a sufficiently large number of labeled demonstrations, which is costly in terms of the conceptor's time.

A class of discriminative methods recently proposed, called contrastive learning, is based on a self-supervised task formulation directly on the state space [Le-Khac et al., 2020]. This removes the need to generate labeled demonstrations. One of the most popular, CPC (Contrastive Predictive Coding [Oord et al., 2018]), maximizes mutual information between more or less spatially distant states with InfoNCE (Noise-Contrastive Estimation [Gutmann and Hyvärinen, 2010]). These constraints imposed by contrastive learning, make it possible to force similarity between pairs of similar points within a mini batch, and to force diversity between other points. More specifically, the state space is learned by maximizing mutual information between similar points in a mini-batch, and minimizing mutual information with respect to others.

These techniques have been used massively in computer vision [Chen et al., 2020, He et al., 2020], and have also allowed to formulate SRL problems [Stooke et al., 2020, Zhan et al., 2020]. However, these constraints are based solely on the slowness and spatial proximity principle [Anand et al., 2019]. They may therefore in some cases, not be able to retrieve the controllable but fast information source,

and retrieve on the contrary the slow distractor information source. Just as it is the case with formulations based on the combination of a slowness and diversity constraints presented earlier.

3.3.3 *Exploration Strategies*

This section succinctly examines a currently little-studied issue in the SRL field: exploration strategies. It is almost absent in the field of supervised learning and unsupervised learning which generally is not applied directly to controlled systems. In contrast, SRL must generate its training data itself via agent-environment interactions. In the absence of reward, the agent should seek for a large diversity of observations, as it is the only way for the learned representation to be useful for various tasks. It is therefore useful to devise an exploration strategy, which in the rewardless context may correspond to learning a task-agnostic discovery policy. However, although such a discovery policy is crucial to solve the SRL problem, to our knowledge to date there are no studies on its development, as also noticed by Sutton and Barto [2018], Lesort et al. [2018].

First, let us distinguish what an effective exploration strategy is in the SRL context versus the RL context. In the RL context, it solves the exploration/exploitation tradeoff to find a policy with good generalization performance. In the SRL framework, it solves the underfitting/overfitting tradeoff (presented in Section 2.2.1), also known as the bias/variance tradeoff from the statistical viewpoint [Lawrence et al., 1998]. While the latter tradeoff is intensively studied in the supervised learning context [Caruana, 1995], it is beginning to receive more and more attention in the RL context [Zhang et al., 2018]. However, it has not yet been studied in the SRL context. In our view, the difference is that in this context there may be several objective functions against which this trade-off (i.e. the generalization error of approximation versus estimation) can be estimated.

Hence, unlike typical unsupervised learning methods, SRL can leverage its application to controlled systems to perform exploration that addresses the underfitting/overfitting tradeoff. This is one of the core elements that allow these transfer learning methods to outperform more traditional task-specific approaches. In particular, by solving this tradeoff, exploration can help optimization algorithms circumvent the overfitting danger without resorting to other explicit (e.g. weight decay with ℓ_2 or ℓ_1 norms or a decreasing learning rate) or implicit (dropout or early stopping) regularization techniques. Thus, SRL exploration acts as a regularizer that avoids overfitting even when the pretraining time is small and the networks are overparameterized, whereas most unsupervised learning approaches avoid overfitting by increasing the dataset, or reducing the network complexity, or by using regularization techniques [Bengio, 2012]. Moreover, this is essential to make SRL algorithms sample efficient and thus reduce the pretraining time. In other words, an exploration strategy must allow a SRL algorithm to converge faster and better than if it were a random exploration.

The strategies proposed so far in the literature are based on random policies or demonstrations. The previously cited works that use a random exploration are for example [Watter et al., 2015, van Hoof et al., 2016, Jonschkowski and Brock, 2015, Yarats et al., 2019], and (expert) demonstrations are for example [Sermanet et al., 2018, Dwibedi et al., 2018, Stooke et al., 2020, Zhan et al., 2020]. The latter allows to have a training dataset that is sufficiently representative of the environment. This exploration strategy is used by our SRLfD pretraining approach proposed in Chapter 4. Therefore, it seems necessary to address the following question for future SRL development: How to develop an exploration strategy consistent with the SRL problem?

To develop such an exploration strategy, it should rely on the learning heuristics on which the SRL method is formulated. Furthermore, it should observe transitions from which the SRL model can learn the most, i.e. transitions which are complex with respect to the learning heuristics. For SRL methods based on a near-future prediction objective, an appropriate exploration strategy may be to observe as many diverse transitions as possible with respect to their controllability. Indeed, in this context, to efficiently train a state estimator it is not sufficient for the agent to seek for a large diversity of observations because, for example, first-person perspective images have a metric (distance defined with the L_2 norm) that varies greatly when the orientation changes from one transition to another. Moreover, since the prediction objective adequate to construct representations with relevant properties may not be adequate to promote such efficient exploration, a controllability criterion on environment transitions can therefore be specially constructed to allow the training of discovery policies for the most diverse transitions. XSRL presented in Chapter 5 proposes such an exploration strategy.

We believe that the development of exploration strategies can lead to new SRL algorithms that perform better in terms of generalization, and thus better scaling-up RL algorithms to complex control tasks.

3.4 CONCLUSION

In this section, we have presented the SRL problem studied in this thesis. It can be broken down into three main elements: (i) solution criteria, (ii) learning heuristics, (iii) exploration strategies. We have detailed the specificity of these generally ill-posed problems, and how to solve them with a priori knowledge of the physical world related to the control of an agent. This allowed us to list the specific criteria that a SRL solution (i.e. the mapping denoted φ between the observation space and a state space) should verify. We then reviewed four major families of learning heuristics used in the SRL literature that meet some of the listed criteria. Finally, we discussed the currently used exploration strategies and how new ones could be designed given the learning heuristics.

The next two chapters each propose a novel SRL method. Each one has a learning heuristic belonging to a different class. For SRLfD presented in Chapter 4, it belongs to the dynamic constraints class. It is the loss of multi-task imitation learning. We will see that it allows us to take advantage of knowledge from other tasks. For XSRL presented in Chapter 5, it belongs to the transition estimation class. It is the prediction loss of the next observation. We will see that it takes advantage of the task-agnostic interaction between an agent and its environment. These two methods allow us to model state embeddings verifying our criteria since they reduce the dimension of the observations performing well on new tasks with an RL algorithm.

4

STATE REPRESENTATION LEARNING FROM DEMONSTRATION

> The object of pure physics is the unfolding of the laws of the intelligible world; the object of pure mathematics that of unfolding the laws of human intelligence.
>
> James Joseph Sylvester

ABSTRACT

Robots could learn their own state and world representation from perception and experience without supervision. This desirable goal is the main focus of our field of interest, state representation learning (SRL). Indeed, a compact representation of such a state is beneficial to help robots grasp onto their environment for interacting. The properties of this representation have a strong impact on the adaptive capability of the agent. In this chapter we present an approach based on imitation learning. The idea is to train several policies that share the same representation to reproduce various demonstrations. To do so, we use a multi-head neural network with a shared state representation feeding a task-specific agent. If the imitation tasks are diverse, the trained representation will eventually contain the information necessary for all tasks, while discarding irrelevant information. As such, it will potentially become a compact state representation useful for new tasks. We call this approach SRLfD (State Representation Learning from Demonstration). Our experiments confirm that when a controller takes SRLfD-based representations as input, it can achieve better performance than with other representation strategies and promote more efficient reinforcement learning (RL) than with an end-to-end RL strategy.

KEYWORDS

State Representation Learning, Pretraining, Learning from Demonstration, Unsupervised Learning, Deep Reinforcement Learning

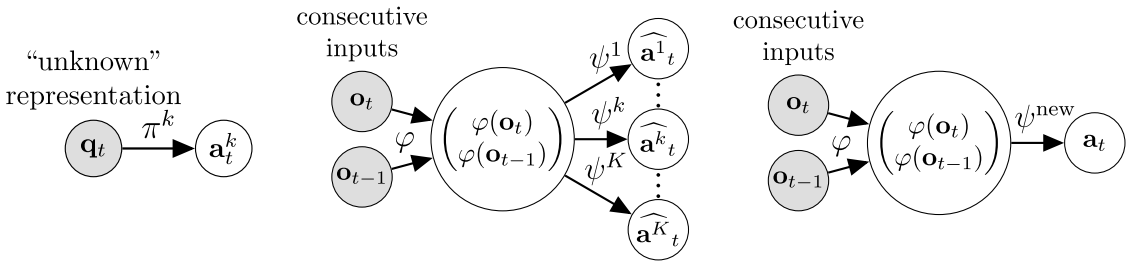
4.1 INTRODUCTION

Recent reinforcement learning (RL) achievements might be attributed to a combination of (i) a dramatic increase of computational power, (ii) the remarkable rise of deep neural networks in many machine learning fields including robotics, which take advantage of the simple idea that training with quantity and diversity helps. The core idea of this work consists of leveraging task-agnostic knowledge learned from several task-specific agents performing various instances of a task.

Learning is supposed to provide animals and robots with the ability to adapt to their environment. RL algorithms define a theoretical framework that is efficient on robots [Kober et al., 2013] and can explain observed animal behaviors [Schultz et al., 1997]. These algorithms build policies that associate an action to a state to maximize a reward. The state determines what an agent knows about itself and its environment. A large state space – raw sensor values, for instance – may contain the relevant information but would require a too large exploration to build an efficient policy. Well-thought feature engineering can often solve this issue and make the difference between the failure or success of a learning process. In their review of representation learning, Bengio et al. [2013a] formulate the hypothesis that the most relevant pieces of information contained in the data can be more or less entangled and hidden in different representations. If a representation is adequate, functions that map inputs to desired outputs are somewhat less complex and thus easier to construct via learning. However, a frequent issue is that these adequate representations may be task-specific and difficult to design, and this is true in particular when the raw data consists of images, i.e. 2D arrays of pixels. One of the objectives of deep learning methods is to automatize feature engineering to make learning algorithms effective even on raw data. By composing multiple nonlinear transformations, the neural networks on which these methods rely are capable of progressively creating more abstract and useful representations of the data in their successive layers.

The intuition behind our work is that many tasks operated in the same environment share some common knowledge about that environment. This is why learning all these tasks with a shared representation at the same time is beneficial. The literature in imitation learning [Pastor et al., 2009, Kober et al., 2012] has shown that demonstrations can be very valuable to learn new policies. To the best of our knowledge, no previous work has focused on constructing reusable state representations from raw inputs solely from demonstrations, therefore, here we investigate the potential of this approach for SRL.

In this chapter, we are interested in solving continuous control tasks via RL or supervised learning, using state estimates as inputs, without having access to any other sensor, which means in particular that the robot configuration, which we will call *ground truth representation*, is unknown. We assume that at all times the consecutive high-dimensional observations $(\mathbf{o}_{t-1}, \mathbf{o}_t)$ contain enough information to know the ground truth state $\mathbf{q}_t$ and that the controller/predictor only needs



(a) Preliminary phase (b) Pretraining phase (c) Transfer learning phase

Figure 2. SRLfD (State Representation Learning from Demonstration) consists of three phases. **(a)** Preliminary phase: for K different tasks, we assume to have access to oracle policies (π^k) that solve each task, and compute their outputs with an "unknown" state representation. **(b)** Pretraining phase: learning of one shared representation function φ with imitation learning of K specific heads ψ^k by observing π^k from high-dimensional observations. Each head ψ^k defines a sub-network that contains the parameters $\boldsymbol{\theta}_\varphi$ of φ and the parameters $\boldsymbol{\theta}_{\psi^k}$ of ψ^k. The set of all network's parameters is $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\varphi, \boldsymbol{\theta}_{\psi^1}, \ldots, \boldsymbol{\theta}_{\psi^K}\}$. **(c)** Transfer learning phase: the pretrained network φ provides representations to learn an unseen decision making task ψ^{new}.

to rely on this representation to choose actions. Intuitively, $\mathbf{q}_t$ could probably be a much better input for a RL algorithm than the raw images, but without prior knowledge, it is not easy to get $\mathbf{q}_t$ from $(\mathbf{o}_{t-1},\mathbf{o}_t)$. In robotics, SRL [Lesort et al., 2018] aims at constructing a mapping from high-dimensional observations to lower-dimensional representations which, similarly to $\mathbf{q}_t$, can be advantageously used instead of $(\mathbf{o}_{t-1},\mathbf{o}_t)$ to form the inputs of a policy.

Our proposed experimental setup consists in three different phases:

1. Preliminary phase (Fig. 2(a)): we have K controllers called oracle policies π^k, each solving a different task. For example, we could define them in laboratory conditions with better sensors (e.g. motion capture), the goal being to reproduce them with a different perception (e.g. images) where in this setting, building a representation extracted from the raw inputs makes sense. For the sake of the experiments, we used almost fake tasks.

2. Pretraining phase (Fig. 2(b)): we derive a state representation that can be relied on to reproduce any of these oracle policies. We do so via imitation learning on a multi-head neural network consisting of a first part that outputs a common state representation $\mathbf{s}_t \triangleq \big[\varphi(\mathbf{o}_{t-1}), \varphi(\mathbf{o}_t)\big]$ used as input to K heads ψ^k trained to predict actions $\mathbf{a}^k_t$ executed by the oracle policies π^k from the previous phase.

3. Transfer learning phase (Fig. 2(c)): we use the previously trained representation $\mathbf{s}_t$ as input to a new learning process ψ^{new} in the same environment.

This method, which we call SRLfD (State Representation Learning from Demonstration), is presented in more detail in Section 4.3, after an overview of the existing related work in Section 4.2. We show that using SRLfD learned representations instead of raw images can significantly accelerate RL (using the popular

SAC algorithm [Haarnoja et al., 2018a]). When the state representation is chosen to be low-dimensional, the speed up brought by our method is greater than the one resulting from state representations obtained with deep autoencoders, or with principal component analysis (PCA).

4.2 RELATED WORK

Learning state representations from demonstrations of multiple policies solving different tasks instances, as we propose, has some similarities with multi-task and transfer learning [Taylor and Stone, 2009]. Multi-task learning aims to learn several similar but distinct tasks simultaneously to accelerate the training or improve the generalization performance of the learned policies, while transfer learning strives to exploit the knowledge of how to solve a given task to then improve the learning of a second task. Not all multi-task and transfer learning works rely on explicitly building a common representation, but some do, either by using a shared representation during multiple task learning [Pinto and Gupta, 2017] or by distilling a generic representation from task-specific features [Rusu et al., 2015]. The common representation can then be used to learn new tasks. However, all these techniques rely on the end-to-end RL approach, which is less sample-efficient than the self-supervised learning approach followed by SRLfD.

In another perspective, the learning from demonstration literature typically focuses on learning from a few examples and generalizing from those demonstrations, for example by learning a parameterized policy using control-theoretic methods [Pastor et al., 2009] or RL-based approaches [Kober et al., 2012]. Although those methods typically assume prior knowledge of a compact representation of the robot and environment, some of them directly learn and generalize from visual input [Finn et al., 2017] and do learn a state representation. However, the goal is not to reuse that representation to learn new skills but to produce end-to-end visuomotor policies generalizing the demonstrated behaviors in a given task space. Several works have also proposed using demonstrations to improve regular deep RL techniques [Večerík et al., 2017, Nair et al., 2018], but the goal is mostly to improve exploration in environments with sparse rewards. Those works do not directly address the problem of state representation learning.

4.3 STATE REPRESENTATION LEARNING FROM DEMONSTRATION

4.3.1 *Demonstrations*

Let us clarify the hierarchy of the objects that we manipulate and introduce our notations. This work focuses on simultaneously learning K different tasks[1]

1 Roughly, different tasks refer to goals of different natures, while different instances of a task refer to a difference of parameters in the task. For example, reaching various locations with a robotic arm is considered as different instances of the same reaching task.

sharing a common state representation function φ and with K task-specific heads for decision $(\psi^1, \psi^2, \ldots, \psi^K)$ (see Fig. 2(b)). For each k-task, the algorithm has seen demonstrations in a form of paths $P_1^k, P_2^k, \ldots, P_P^k$ from an initial random position to the same goal corresponding to the k-task generated by running the oracle policy π^k obtained in the preliminary phase (see Fig. 2(a)). Specifically, during a path P_p^k, an agent is shown a demonstration (or data point) of $(\mathbf{o}_{t-1}^{k,p}, \mathbf{o}_t^{k,p}, \mathbf{a}_t^{k,p})$ from which it can build its own world-specific representation. Here, $\mathbf{o}_{t-1}^{k,p}$ and $\mathbf{o}_t^{k,p}$ are consecutive high-dimensional observations (a.k.a. measurements), and $\mathbf{a}_t^{k,p}$ is a real-valued vector corresponding to the action executed right after the observation $\mathbf{o}_t^{k,p}$ was generated.

4.3.2 *Imitation Learning from Demonstrations*

Following the architecture described in Fig. 2(b), we use a state representation neural network φ that maps high-dimensional observations $\mathbf{o}_t^{k,p}$ to a smaller real-valued vector $\varphi(\mathbf{o}_t^{k,p})$. This network φ is applied to consecutive observations $(\mathbf{o}_{t-1}^{k,p}, \mathbf{o}_t^{k,p})$ to form the state representation $\mathbf{s}_t^{k,p}$, as follows:

$$\mathbf{s}_t^{k,p} = \left[\varphi(\mathbf{o}_{t-1}^{k,p}), \varphi(\mathbf{o}_t^{k,p})\right] \tag{69}$$

This state representation $\mathbf{s}_t^{k,p}$ is sent to the ψ^k network, where ψ^k is one of the K independent heads of our neural network architecture. $\psi^1, \psi^2, \ldots, \psi^K$ are head networks with similar structure but different parameters, each one corresponding to a k-task. Each head has continuous outputs with the same number of dimensions as the action space of the robot. We denote by $\psi^k(\mathbf{s}_t^{k,p})$ the output of the k-th head of the network on the input $(\mathbf{o}_{t-1}^{k,p}, \mathbf{o}_t^{k,p})$. We train the global network to imitate all the oracle policies via supervised learning. Specifically, our goal is to minimize the quantities: $\|\psi^k(\mathbf{s}_t^{k,p}) - \mathbf{a}_t^{k,p}\|_2^2$ that measure how well the oracle policies are imitated. The optimization problem we want to solve is thus the minimization of the following objective function:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{PT} \sum_{p=1}^{P} \sum_{t=1}^{T} \|\psi^k(\mathbf{s}_t^{k,p}) - \mathbf{a}_t^{k,p}\|_2^2 \tag{70}$$

for $k \in [\![1, K]\!]$, and where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\varphi, \boldsymbol{\theta}_{\psi^1}, \ldots, \boldsymbol{\theta}_{\psi^K}\}$ as explained in Fig. 2(b). We give an equal importance to all oracle policies by uniformly sampling $k \in [\![1, K]\!]$, and performing a training step on $\mathcal{L}(\boldsymbol{\theta})$ to adjust $\boldsymbol{\theta}$. Algo. 1 describes this procedure.

The network of SRLfD is trained to reproduce the demonstrations, but without direct access to the ground truth representation of the robot. Each imitation can only be successful if the required information about the robot configuration is extracted by the state representation $\left[\varphi(\mathbf{o}_{t-1}^{k,p}), \varphi(\mathbf{o}_t^{k,p})\right]$. However, a single task may not require the knowledge of the full robot state. Hence, we cannot

Algorithm 1 SRLfD algorithm

1: **Input:** A set of instances of tasks T^k, $k \in [\![1, K]\!]$, and for each of them a set of paths P_p^k, $p \in [\![1, P]\!]$ of maximum length T.
2: **Initialization:** A randomly initialized neural network following the architecture described in Fig. 2(b) with parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\varphi, \boldsymbol{\theta}_{\psi^1}, \dots, \boldsymbol{\theta}_{\psi^K}\}$.
3: **while** $\boldsymbol{\theta}$ has not converged **do**
4: Pick uniformly a k-task
5: Predict current state representations with Eq. 69:

$$\mathbf{s}_t^{k,p} = \left[\varphi(\mathbf{o}_{t-1}^{k,p}), \varphi(\mathbf{o}_t^{k,p})\right]$$

6: Compute $\mathcal{L}(\boldsymbol{\theta})$ with Eq. 70:

$$\mathcal{L} \leftarrow \frac{1}{PT} \sum_{p=1}^{P} \sum_{t=1}^{T} \|\psi^k(\mathbf{s}_t^{k,p}) - \mathbf{a}_t^{k,p}\|_2^2$$

7: Perform a training step on $\mathcal{L}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$
8: **end while**

be sure that reproducing only one instance of a task would yield a good state representation. By learning a common representation for various instances of tasks, we increase the probability that the learned representation is general and complete. It can then be used as a convenient input for new learning tasks, especially for a RL system.

4.4 GOAL REACHING

In this section, we study a transfer learning phase (see Fig. 2(c)) corresponding to a RL optimization problem to solve a torque-controlled reaching task with image observations. This is a challenging problem despite the simplicity of the task. Indeed, when high-dimensional observations are mapped to a lower-dimensional space before feeding a RL system, a lot of information is compacted and valuable information for control may be lost. The purpose is to verify that state representations learned with SRLfD are useful representations for RL algorithms. In this thesis, we only conduct experimental validations, but a fundamental question that we will not answer is at stake: what constitutes a good representation for state-of-the-art deep RL algorithms? Should it be as compact and as disentangled as possible, or on the contrary, can redundancy of information or correlations be useful in the context of deep RL? A definitive answer seems beyond the current mathematical understanding of deep RL.

We consider a simulated 2D robotic arm with 2 torque-controlled joints, as shown in Fig. 3. We use the environment *Reacher* adapted from the OpenAI Gym [Brockman et al., 2016] benchmark to PyBullet [Coumans et al., 2018]. An

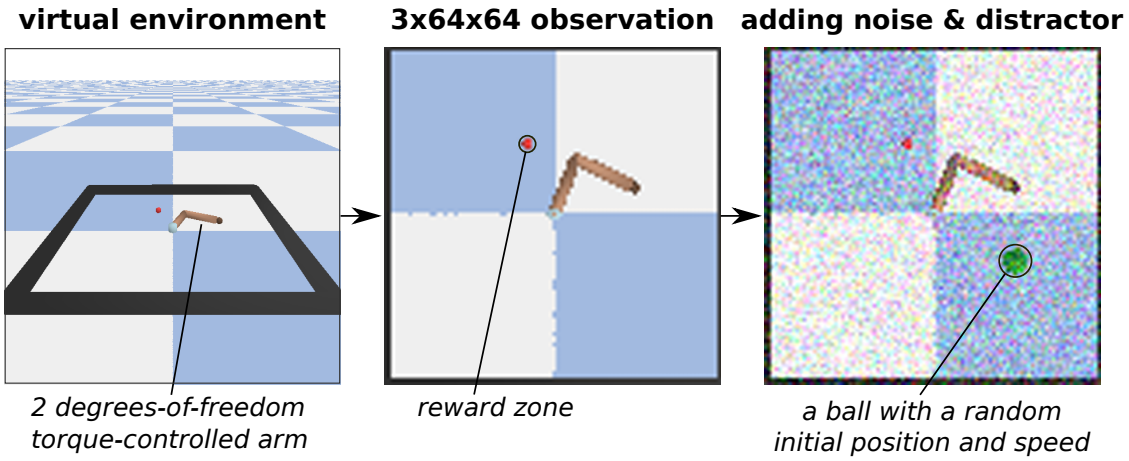


Figure 3. The *Reacher* environment, with a reward of 1 when the end-effector reaches a position close to the goal, and 0 otherwise. For more challenging inputs (on the right), we add Gaussian noise with zero mean and standard deviation 10, and a ball distractor is added in the environment with random initial position and velocity.

instance of this continuous control task is parameterized by the position of a goal that the end-effector of the robot must reach within some margin of error (and in limited time). We use as raw inputs RGB images of 64×64 pixels. As the heart of our work concerns state estimation, we have focused on making perception challenging, by adding in some cases randomly moving distractor and Gaussian noise, as shown in Fig. 3. We believe that the complexity of the control part (i.e. the complexity of the tasks) is less important to validate our method, as it depends more on the performance of the RL algorithm. To solve even just the simple reaching task, the configuration of the robot arm is required and needs to be extracted from images for the RL algorithm to converge. Indeed our results show that this is the case when SRLfD learned representations are used as inputs of SAC [Haarnoja et al., 2018a].

4.4.1 *Experimental Setup*

Baseline Methods We compare state representations obtained with our method (SRLfD) to five other representation strategies:

- Ground truth: as mentioned in Section 4.4.1, what we call *ground truth* representation of the robot configuration is represented as a vector of size four: the two torques angles and velocities.
- Principal Component Analysis (PCA) [Jolliffe, 2011]: we perform PCA on the demonstration data, and the 8 or 24 most significant dimensions are kept, thus reducing observations to a compact vector that accounts for a large part of the input variability.

- Autoencoder-based representation [Hinton and Salakhutdinov, 2006]: φ is replaced by an encoder learned with an autoencoder. The latent space representation of the autoencoder (of size 8 or 24) is trained with the same demonstrations (but ignoring the actions) as in the SRLfD training.
- Random network representation [Gaier and Ha, 2019]: we use the same neural network structure for φ as with SRLfD, but instead of training its parameters, they are simply fixed to random values sampled from a Gaussian distribution of zero mean and standard deviation 0.02.
- Raw pixels: the policy network is modified to receive directly $(\mathbf{o}_{t-1}, \mathbf{o}_t)$ in input, with the same dimensionality reduction after φ as other methods, but all of its parameters are trained simultaneously in the manner of end-to-end RL.

The representations obtained with these methods use the same demonstration data as SRLfD method, and share the same neural network structure for φ or replace it (with ground truth and PCA) in the architecture of Fig. 2(c) whose output size is 8 or 24[2].

Generating Demonstrations For simplicity, the preliminary phase of training K oracle policies π^k (see Fig. 2(a)) is done by running the SAC [Haarnoja et al., 2018a] RL algorithm. Here, the "unknown" representations used as inputs are the ground truth representations. SAC also exploits the cartesian cordinates of the goal position[3]. It returns a parameterized policy capable of producing reaching trajectories to any goal position.

For the pretraining phase of SRLfD (see Fig. 2(b)), the previously learned parameterized policy generates $K = 16$ oracle policies π^k (which represent different instances of the reaching task), with each of them 238 paths for training and 60 paths for validation of maximal length $T = 50$, computed from various initial positions. We then simultaneously train all the heads for computational efficiency. Specifically, for each optimization iteration, we uniformly sample for every head ψ^k a mini-batch of 64 demonstrations from the P paths corresponding to the k-task.

Implementation Details For SRLfD network architecture (adapted from the one used in [Mnih et al., 2013]), φ (see Fig. 2) sends its $3 \times 64 \times 64$ input to a succession of three convolutional layers. The first one convolves 32 8×8 filters with stride four. The second layer convolves 64 4×4 filters with stride two. The third hidden layer convolves 32 3×3 filters with stride one. It ends with a fully

2 The number of 24 dimensions has been selected empirically (not very large but leading to good RL results).

3 The purpose of our method is to generate state representations from (possibly noisy) inputs that are hard to exploit (such as raw images), so only the preliminary phase has access to the "unknown" representation.

connected layer with half as many output units as the chosen state representation dimension (because state representations have the form $\left[\varphi(\mathbf{o}_{t-1}), \varphi(\mathbf{o}_t)\right]$). The heads ψ^k take as input the state representation and are composed of three fully connected layers, the two first ones of size 256 and the last one of size two, which corresponds to the size of the action vectors (one torque per joint).

For SAC network architecture we choose a policy network that has the same structure as the heads ψ^k used for imitation learning, also identical to the original SAC implementation [Haarnoja et al., 2018a], and use the other default hyperparameters.

The Rectified linear units (ReLU) is used for the activation functions between hidden layers. We use ADAM [Kingma and Ba, 2014] with a learning rate of 10^{-4} to train the neural network φ, and 10^{-3} to train all the heads ψ^k and the policy network.

4.4.2 *Results and Discussion*

In this section, we report our results with the quantitative evaluation of SRLfD when a goal reaching task is used in the transfer learning phase (see Fig. 2(c)). Specifically, we evaluate the transferability of SRLfD learned state representations used as inputs for a RL algorithm (SAC [Haarnoja et al., 2018a]) to solve a new instance of the reaching task chosen randomly, and compare the success rates to the ones obtained with state representations originating from other methods. The performance of a policy is measured as the probability to reach the goal from a random initial configuration in 50 time steps or less. We expect that better representations yield faster learning progress and better convergence (on average).

Table 1 displays the success rates corresponding to the end of the learning curves of Fig. 4 obtained with SAC and different state representations in four different contexts: with a representation of either 16 or 48 dimensions (except for the ground truth representation, of size 4), and on "clean" i.e. raw observations (in the middle of Fig. 3) or observations with noise and a randomly moving ball distractor (on the right of Fig. 3). As expected, the best results are obtained with the ground truth representation, but we see that out of the five other state representations, only SRLfD, PCA, and VAE representations can be successfully used by SAC to solve reaching tasks when noise and a distractor are added to the inputs. SAC fails to train efficiently (in an end-to-end manner) the large neural network that takes raw pixels in input, whether its representation is of size 16 or 48. Using fixed random parameters for the first part of its network (random network representation) is not a viable option either.

The results show that with fewer dimensions our method (SRLfD) leads to better RL performances than with observation compression methods (PCA and VAE). We assume that the information from the robotic arm can be filtered through the small size of the bottleneck due to the observation reconstruction objective

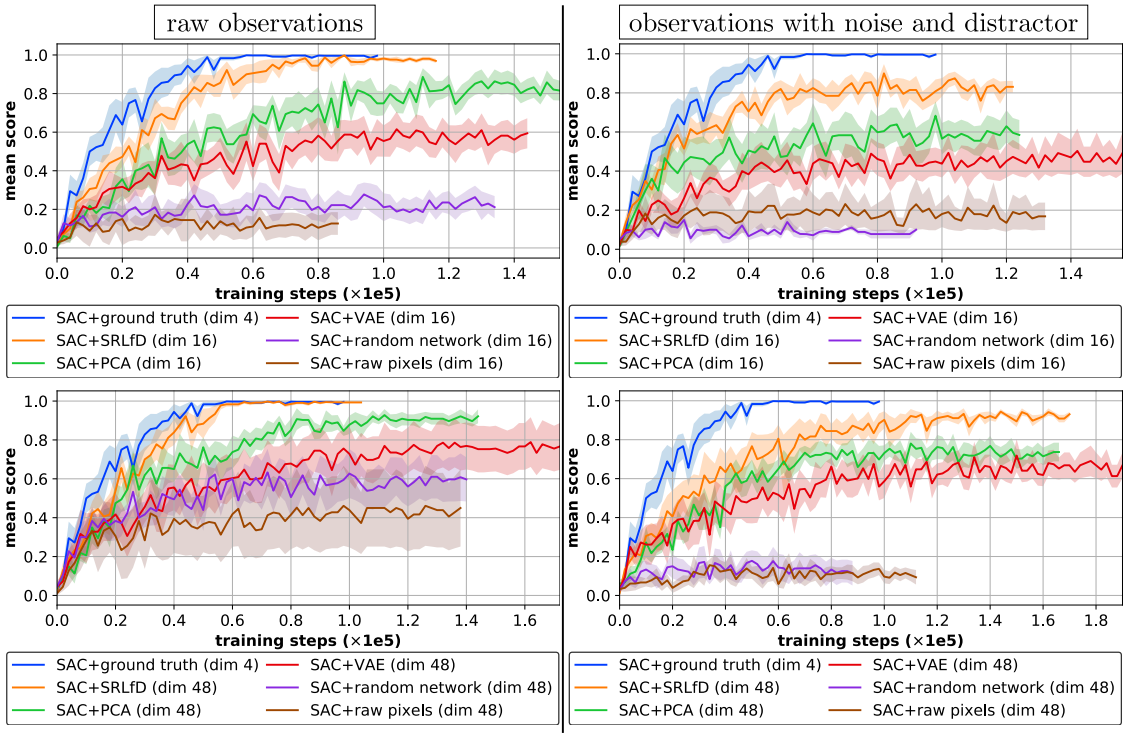


Figure 4. Learning curves of the episode returns averaged over 50 episodes (mean in lines and half standard deviation in shaded areas over 8 runs; the higher the better) with SAC algorithm with a random position of the goal for each run, based on various state representations. The indicated dimensions for SRLfD, PCA, VAE, random network and raw pixels correspond to the size of the state representation $\left[\varphi(\mathbf{o}_{t-1}), \varphi(\mathbf{o}_t)\right]$. The use of our SRLfD state representation (red) in SAC outperforms all the other baselines, except the case in which SAC is given a direct access to the ground truth representation.

(and dramatically more on the challenging observations). This explains why PCA and VAE tend to require additional dimensions than the minimal number of dimensions of our robotic task (four dimensions: the two torques angles and velocities). This clearly shows that with a carefully chosen unsupervised learning objective, such as the one used for SRLfD, it is possible to compact into a minimal number of dimensions only the information necessary for robotic control.

Another surprising observation is that PCA outperforms VAE in our results. By design, VAE is trained to encode and decode with as few errors as possible, and it can generally do this better than PCA by exploiting the nonlinearities of neural networks. Moreover, as first explained by Bourlard and Kamp [1988], Kramer [1991], the autoencoder is an extension of PCA that transforms correlated observations into nonlinearly uncorrelated representations. However, it is not clear that such uncorrelated input variables lead to better RL performances. This is because when data are obtained with transitions from a control system, the most important variables are those correlated with changes between transitions, which generally do not coincide with the directions of greatest variation in the data.

Table 1. Mean episode returns (mean ± standard deviation over 8 runs; the higher the better) corresponding to the end of the curves in Fig. 4.

Method	Mean score	
	Raw observations	With noise and distractor
SAC+SRLfD (dim 48)	**0.992 ± 0.0080**	**0.928 ± 0.029**
SAC+SRLfD (dim 16)	**0.980 ± 0.022**	**0.833 ± 0.082**
SAC+PCA (dim 48)	0.908 ± 0.072	0.725 ± 0.10
SAC+PCA (dim 16)	0.832 ± 0.13	0.591 ± 0.18
SAC+VAE (dim 48)	0.749 ± 0.27	0.650 ± 0.13
SAC+VAE (dim 16)	0.574 ± 0.16	0.448 ± 0.18
SAC+raw pixels (dim 48)	0.365 ± 0.39	0.106 ± 0.063
SAC+raw pixels (dim 16)	0.118 ± 0.13	0.149 ± 0.14
SAC+random network (dim 48)	0.552 ± 0.21	0.143 ± 0.14
SAC+random network (dim 16)	0.239 ± 0.13	0.0863 ± 0.037
SAC+ground truth (dim 4)	0.995 ± 0.0054	0.995 ± 0.0054

4.5 BALLISTIC PROJECTILE TRACKING

In this section, we study a transfer learning phase (see Fig. 2(c)) corresponding to a simple supervised learning system to solve a ballistic projectile tracking task. Specifically, it consists in training a tracker from learned representations to predict the next projectile position. This task has the advantage of not needing K oracle policies π^k in a preliminary phase (Fig. 2(a)). Instead, we derive π^k directly from the ballistic trajectory equations (Eq. 82). This enables us to easily perform the experimental study of the main hyperparameters of our SRLfD method: the state dimension $\mathcal{S}_d$ and the number of oracle policies K. This also allows us to conduct a comparative quantitative evaluation against other representation strategies. Furthermore, we study the possibility of using a recursive loop for the state update instead of the state concatenation. Thus, SRLfD can handle partial observability by aggregating information that may not be estimable from a single observation. In particular, to solve a simple projectile tracking task, projectile velocity information is required and must be extracted from past measurements for the supervised learning algorithm to converge. Mathematically, the observation (i.e. measurement) $\mathbf{o}_t$ is concatenated to the previous state estimate $\mathbf{s}_{t-1}$ to form the input of SRLfD model φ which estimates the current state as follows:

$$\mathbf{s}_t \triangleq \varphi\left(\left[\mathbf{o}_t, \mathbf{s}_{t-1}\right]\right) \tag{71}$$

where $\mathbf{s}_0 \sim \mathcal{N}\left(\mathbf{0}_{\mathcal{S}_d}, 0.02 \times \mathbf{I}_{\mathcal{S}_d}\right)$. Literally, this recursive loop conditions the current state estimate on all previous states.

An instance of this projectile tracking task is parameterized by the initial velocity and angle of the projectile. Specifically, a tracker receives as input the

state estimated by φ, and must predict the projectile's next position $\hat{\mathbf{o}}_{t+1}$ as follows:

$$\psi^{\text{new}}\left(\varphi\left(\left[\mathbf{o}_t, \mathbf{s}_{t-1}\right]\right)\right) = \hat{\mathbf{o}}_{t+1} \tag{72}$$

A tracker is then trained by supervised learning to minimize this objective function:

$$\mathcal{L} = \frac{1}{PT}\sum_{p=1}^{P}\sum_{t=1}^{T} \|\psi^{\text{new}}\left(\varphi\left(\left[\mathbf{o}_t^p, \mathbf{s}_{t-1}^p\right]\right)\right) - \mathbf{o}_{t+1}^p\|_2^2 \tag{73}$$

where the notations correspond to those defined in Section 4.3.1.

4.5.1 *Experimental Setup*

Baseline Methods We compare the state representations learned with SRLfD to five other representation strategies:

- the ground truth is a vector of size 4 formed from the 2D cartesian positions and velocities of the projectile: $(x_t, y_t, v_{x,t}, v_{y,t})$;
- the position corresponds to the 2D cartesian coordinates of the projectile: (x_t, y_t);
- a random network representation with the same φ architecture and state recursive loop which is not trained[4];
- an end-to-end representation learning strategy, i.e. it builds its state estimate with the same φ architecture and state recursive loop, while solving the tracking task;
- a Kalman filter estimated from positions with unknown initial cartesian velocities of the projectile.

The Kalman filter designed by Kalman [1960b] is a classical method for state estimation in state-linear control problems, where the ground truth state is not directly observable, but sensor measurements are observed instead. Kalman et al. [1960] create a mathematical framework for the control theory of the LQR (Linear-Quadratic-Regulator) problem and create for this purpose the notions of *controllability* and *observability* refined later in [Kalman, 1960a]. Witsenhausen [1971] conducted one of the first attempts to survey the literature on the separation of state estimation and control. In particular, this led to the two-step procedure, composed of the resolution of Kalman filtering and then of LQR,

4 As in the previous reaching task, for random network representation the parameters of φ are simply fixed to random values sampled from a Gaussian distribution of zero mean and standard deviation 0.02.

known as LQG (Linear-Quadratic-Gaussian). However, the Kalman filter is also commonly used in recent RL applications [Ng et al., 2003, 2006, Abbeel et al., 2007, Abbeel, 2008].

The Kalman filter has then undergone many extensions including the popular extended Kalman filter (EKF) which can handle nonlinear transition models [Ljung, 1979]. However, a major drawback of these classical state estimation methods is that they require knowledge of the transition model. This constraint has been relaxed by feature engineering (like SIFT [Lowe, 1999] and SURF [Bay et al., 2006]) techniques which require knowledge of the subsequent task [Kober et al., 2013]. Indeed, good hand-crafted features are task-specific and therefore costly in human expertise. These drawbacks were then overcome by SRL methods popularized by Jonschkowski and Brock [2013], which benefit from the autonomy of machine learning and the generalization power of deep learning techniques.

Kalman filter We briefly describe below the operations of the Kalman filter, the reader willing a complete presentation can refer to [Bertsekas, 2005]. The equations for updating the positions (x_t, y_t) and velocities $(v_{x,t}, v_{y,t})$ of the projectile are related to their previous values, to the acceleration due to the gravitational force (g), and to the time elapsed between each update (Δt), as follows:

$$\begin{aligned} x_t &= x_{t-1} + v_{x,t-1}\Delta t \\ y_t &= y_{t-1} + v_{y,t-1}\Delta t - \frac{1}{2}g(\Delta t)^2 \\ v_{x,t} &= v_{x,t-1} \\ v_{y,t} &= v_{y,t-1} - g\Delta t \end{aligned} \tag{74}$$

The ground truth state is defined as $\mathbf{s}_t = \left[x_t,\, y_t,\, v_{x,t},\, v_{y,t}\right] \in \mathbb{R}^4$. Thus, the update procedure of the transition model is described with a single linear equation as follows:

$$\begin{aligned} &\mathbf{s}_t = \mathbf{A}\mathbf{s}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{w}_{t-1} \\ &\mathbf{A} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad , \quad \mathbf{B} = \begin{pmatrix} 0 \\ \frac{(\Delta t)^2}{2} \\ 0 \\ \Delta t \end{pmatrix} \quad , \quad \mathbf{w}_{t-1} \sim \mathcal{N}\left(0, \mathbf{Q}\right) \end{aligned} \tag{75}$$

where $\mathbf{Q}$ is the *process noise covariance matrix* and $\mathbf{w}_t$ is the *process noise* assumed to be white (i.e. normally, independently and identically distributed at each time step).

The observation of this transition model is the projectile position defined as $\mathbf{o}_t = [x_t,\, y_t] \in \mathbb{R}^2$ which is obtained from the ground truth state in the following way:

$$\mathbf{o}_t = \mathbf{H}\mathbf{s}_t + \boldsymbol{\nu}_t \quad , \quad \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad , \quad \boldsymbol{\nu}_t \sim \mathcal{N}\left(0, \mathbf{R}\right) \tag{76}$$

where $\mathbf{R}$ is the *measurement noise covariance matrix* (a.k.a. *sensor noise covariance matrix*) and $\boldsymbol{\nu}_t$ is the *measurement noise* also assumed to be white. $\mathbf{R}$ and $\mathbf{Q}$ are ignored in our experiments for simplicity.

The aim of the Kalman filter is to solve the problem of estimating the ground truth state $\mathbf{s} \in \mathbb{R}^4$. To do this, the Kalman filter assumes to know $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{H}$. In this experiment, we assume there is no noise in the sensory inputs (i.e. $\mathbf{R}$ has only zero values), and no uncertainty in the transition model (i.e. $\mathbf{Q}$ has only zero values). Moreover, in this projectile tracking task, there is no control vector ($a_t \triangleq 0$), because as the force exerted by gravitation is constant, it can be incorporated in the matrix $\mathbf{A}$. The state of the Kalman filter is initialized with the initial measurement (i.e. the initial 2D cartesian position of the projectile) and zeros instead of the true initial velocities of the projectile. In order to let the Kalman filter fix the initial velocities of the projectile and to ensure its convergence, we initialize the diagonal coordinates corresponding to the velocities of the *state covariance matrix* (denoted $\mathbf{P}$) to 100.

The Kalman filter follows a twofold procedure: (i) a prediction step which uses knowledge of the transition model, (ii) an update step which combines the model and the measurement knowing that both may be imperfect. The equations of the prediction step consists in the prediction of the state estimate Eq. 77, and the prediction of the state covariance estimate Eq. 78.

$$\hat{\mathbf{s}}_t^- = \mathbf{A}\hat{\mathbf{s}}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} \tag{77}$$

$$\mathbf{P}_t^- = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^\tau + \mathbf{Q} \tag{78}$$

This corresponds to a priori estimates.

The equations in the update step first update the Kalman gain matrix Eq. 79, then update the state estimate by incorporating the measurement and the a priori state estimate Eq. 80, and similarly update the state covariance matrix Eq. 81.

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^\tau\left[\mathbf{H}\mathbf{P}_t^-\mathbf{H}^\tau + \mathbf{R}\right]^{-1} \tag{79}$$

$$\hat{\mathbf{s}}_t = \hat{\mathbf{s}}_t^- + \mathbf{K}_t\left(\mathbf{o}_t - \mathbf{H}\hat{\mathbf{s}}_t^-\right) \tag{80}$$

$$\mathbf{P}_t = \left(\mathbf{Id} - \mathbf{K}_t\mathbf{H}\right)\mathbf{P}_t^- \tag{81}$$

Therefore, at each iteration, the Kalman filter predicts the next a priori state estimate which is then used to update the next state estimate which corresponds to an a posteriori estimate. This implies that the Kalman filter is a recursive linear state estimation, whereas SRLfD is a recursive nonlinear state estimation (thanks to the state recursive loop). In particular, while SRLfD may use a linear network φ for simple tasks, it may still take advantage of nonlinear approximations such as multilayer perceptrons defining its ψ^k heads, such as in this ballistic projectile tracking task.

Generating Demonstrations The projectile tracking task does not require in its preliminary phase K oracle policies π^k. It is the ballistic trajectory equations

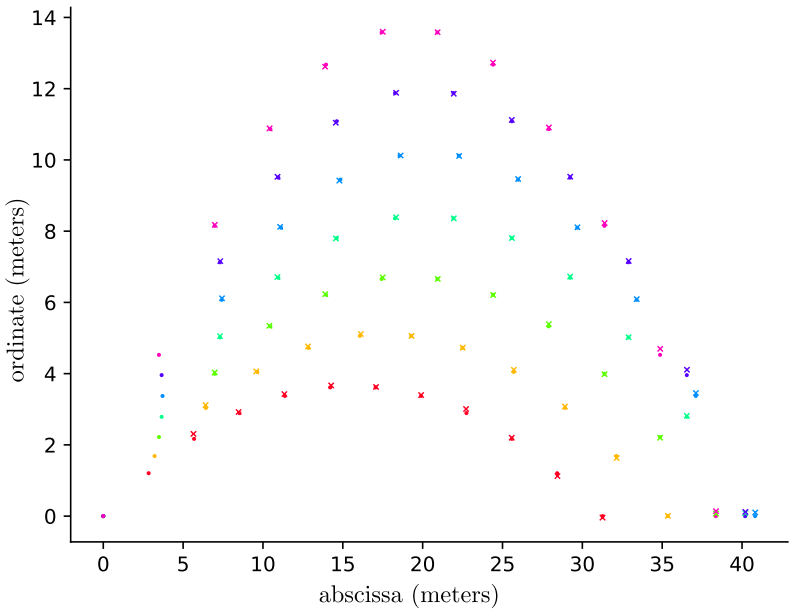


Figure 5. Predictions (crosses) with a tracker trained from SRLfD representations, over 7 trajectories with zero initial coordinate (i.e. $x_0 = 0$ and $y_0 = 0$), initial angles $\alpha_0 \in \{25, 30, 35, 40, 45, 50, 55\}$ (in degrees), and initial velocity 20 (in meter per second), composed of 12 projectile position measurements (points).

(Eq. 82) that allow us to define these oracle policies π^k. As we ignore all forces except the gravitational one, the trajectory of the projectile corresponds to a ballistic trajectory. The temporal equations of the ballistic trajectory are defined with the force of gravity $g = 9.81$ m/s^2, the initial angle of launch of the projectile α_0, and its initial velocity v_0 as well as its initial y-coordinate y_0, as follows:

$$\begin{aligned} x(t) &= v_0 \cos(\alpha_0)\, t \\ y(t) &= -\frac{1}{2} g t^2 + v_0 \sin(\alpha_0)\, t + y_0 \\ v_x(t) &= v_0 \cos(\alpha_0) \\ v_y(t) &= -gt + v_0 \sin(\alpha_0) \end{aligned} \tag{82}$$

The total horizontal distance $x_{\max}$ covered until the projectile falls back to the ground is given by:

$$x_{\max} = \frac{v_0}{g} \cos\alpha_0 \left(v_0 \sin\alpha_0 + \sqrt{(v_0 \sin\alpha_0)^2 + 2 g y_0} \right) \tag{83}$$

This allows us to calculate the corresponding time of flight $t_{\max}$:

$$t_{\max} = \frac{x_{\max}}{v_0 \cos\alpha_0} \tag{84}$$

These equations provide an oracle policy parameterized by v_0 and α_0, which can generate ballistic trajectories at any initial ordinate y_0. Each generated trajectory is of fixed length $T = 12$ which corresponds to 10 demonstration samples (as the

first two are used during initialization) of the form $(\mathbf{o}_t^{k,p}, \mathbf{a}_t^{k,p})$ where the actions $\mathbf{a}_t^{k,p}$ correspond to the next positions of the projectile, i.e. $\mathbf{a}_t^{k,p} = \mathbf{o}_{t+1}^{k,p}$. To do this, we define for each trajectory the time between each update Δt, as follows:

$$\Delta t = \frac{t_{\max}}{T-1} \tag{85}$$

where $t_{\max}$ is defined in Eq. 84.

For the pretraining phase of SRLfD (see Fig. 2(b)), each oracle policy π^k has a fixed initial velocity and angle (v_0^k, α_0^k), and generate P ballistic trajectories with different random initial ordinates such that $y_0 \in [0, 30]$. We uniformly pick K' tasks (such that $K' \leq K$) to simultaneously train the corresponding heads ψ^k for computational efficiency. Each of them is trained on demonstrations sequentially generated from $P = \frac{B}{K'}$ different paths where B is a desired batch size for training φ. This way, every optimization iteration to train φ is performed on a fixed number B of demonstrations, independently of the total number of tasks K. We use $B = 256$ and $K' = \min(K, 6)$ in our experiments. For the tracker training, the batch size is also 256, which implies $P = 256$.

For the SRLfD training validation, we measure the average of tracking prediction errors over all oracle policies on initial ordinates defined as $y_0 \in \{0, 10, 20, 30\}$ (in meters). For the tracker training validation, we measure the average of tracking prediction errors on fixed trajectories with the same initial velocity $v_0 = 20$ and the initial angles defined as $\alpha_0 \in \{25, 30, 35, 40, 45, 50, 55\}$ (in degrees) and on initial ordinates defined as $y_0 \in \{0, 10, 20, 30\}$ (in meters). Fig. 5 shows some qualitative results of this validation with a tracker trained from 4-dimensional SRLfD representations with 6 oracle policies.

Implementation Details φ is a linear neural network of input dimension $(2 + \mathcal{S}_d)$ and output dimension $\mathcal{S}_d$. When not specified, the default number of oracle policies K is 6. We used a state recursive loop to remove the state concatenation. For the random network and the end-to-end baselines, the networks have the same structure as for φ, where in the former the parameters are kept fixed, while in the latter φ is trained jointly with the tracker ψ^{new}. The heads ψ^k and the tracker ψ^{new} are one-hidden neural networks, with the hidden one of size 32 and the last one of size two, which corresponds to the size of the action vectors (i.e. the next projectile positions). The previous nonlinear networks are necessary because Δt changes on all ballistic trajectories, so unlike the Kalman filter which knows this value, for the tracker and SRLfD heads they must relate the input to the output nonlinearly.

We use ADAM [Kingma and Ba, 2014] with a learning rate of 10^{-4} to train φ, the heads ψ^k, and the tracker ψ^{new}. For the SRLfD and tracker trainings, we use early stopping of patience 40 epochs. In one epoch 10 000 iterations are performed, φ (during SRLfD training) or ψ^{new} (during tracker training) see exactly 1 000$\times$256 different trajectories composed of 10 demonstrations (i.e. data

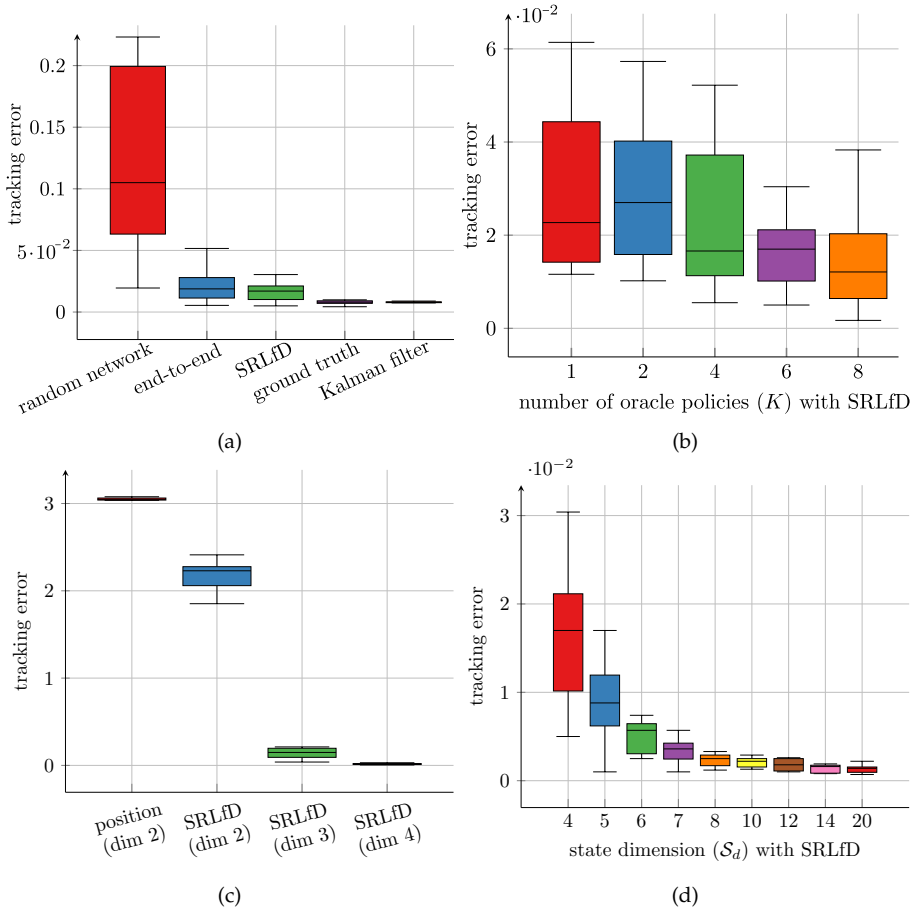


Figure 6. Mean tracking errors over last 5 epochs (average on 10 runs; the lower the better) obtained from five representation strategies with different hyperparameters for SRLfD. **(a)** Five representation strategies of size 4. **(b)** SRLfD representations of size 4 with different number of oracle policies K. **(c)** Representation strategies of size 4 and lower. **(d)** SRLfD representations with 6 oracle policies of varying sizes $\mathcal{S}_d$.

points or samples). The Leaky Rectified Linear Unit (Leaky ReLU) is used for the activation function [Xu et al., 2015].

4.5.2 *Results and Discussion*

We learned the projectile tracking task from six representation strategies. Fig. 6(a) shows the boxplot of the average tracking errors obtained with all different strategies of the same size 4. Our SRLfD outperforms the end-to-end representation, confirming the empowerment provided by "divide and conquer" techniques [Dasgupta et al., 2008]. Ground truth representations outperform our method because they know the complete projectile configuration. With a random network, the supervised learning system fails to track the projectile, implying that state representation learning is required in the context of recursive state estimation.

Regarding the Kalman filter, it uses the knowledge of the real transition model in order to provide a complete recursive state estimation. It is therefore not

surprising that it achieves the same performance as the ground truth baseline. Unlike this classical method, SRLfD does not use the a priori knowledge of the tracking task but only that available in the oracle policies. The performance obtained with the SRLfD representations in this comparative evaluation shows that they extract the position and velocity of the projectile. In other words, there is enough diversity in the oracle policies to be imitated by SRLfD network heads, so that their joint state space is close to the real state that makes the system fully observable. The comparative quantitative evaluation presented by Fig. 6(b) confirms this hypothesis since the performance obtained with the size 4 SRLfD representations increases with the number of oracle policies used during the pretraining phase of SRLfD.

Table 2 displays the average tracking errors displayed in Fig. 6, with the mean and standard deviation for better insight. Fig. 6(c) shows that as the state dimension decreases, the information lost by SRLfD significantly degrades the performance of the trained trackers, while for a size of 2, it is still better than the position baseline. On the other hand, Fig. 6(d) shows that as the state dimension increases, the performance of the trained trackers improves until it even outperforms ground truth starting at 6 dimensions (see Table 2). Although these results may be surprising, one can assume that adding redundancy to the representations makes them easier to build (since the dimension of the ground truth vector is 4). Indeed, larger state embeddings could be more regular and thus be build with simpler neural networks which are less subject to the overfitting problem. However, the question of what is an ideal representation for deep learning algorithms is far from being answered. Recently works have started to investigate this question [Ota et al., 2020, 2021], but the search for a definitive answer leads far beyond the scope of this thesis.

Table 2. Tracking errors corresponding to Fig. 6 (mean ± standard deviation on 10 runs; the lower the better): **(a)** obtained with the five baselines, **(b)** obtained with SRLfD representations of size 4 with different number of oracle policies K, **(c)** obtained with SRLfD representations with 6 oracle policies of varying sizes $\mathcal{S}_d$.

(a)

Method	**Tracking error** ($\times 10^{-3}$)
Ground truth (dim 4)	7.97 ± 1.64
Kalman filter (dim 4)	8.14 ± 0.6
Position (dim 2)	$3\,000 \pm 26.9$
End-to-end (dim 4)	22.4 ± 13.7
Random network (dim 4)	130 ± 74.5

(b)

K	**Tracking error** ($\times 10^{-3}$)
1	31.5 ± 19.7
2	31.4 ± 16.1
4	26.3 ± 16.4
6	$\mathbf{17.1} \pm 7.46$
8	18.4 ± 16.8

(c)

$\mathcal{S}_d$	**Tracking error** ($\times 10^{-3}$)
2	$2\,200 \pm 175$
3	147 ± 60.4
4	17.1 ± 7.46
5	9.37 ± 4.56
6	5.21 ± 1.78
7	3.61 ± 1.37
8	2.41 ± 0.733
10	2.16 ± 0.574
12	1.83 ± 0.686
14	1.44 ± 0.44
20	$\mathbf{1.38} \pm 0.432$

4.6 CONCLUSION

We presented a method (SRLfD) for learning state representations from demonstrations, more specifically from runs of oracle policies on different instances of a task. Our results indicate that the learned state representations can advantageously replace raw sensory inputs to learn policies on new task instances via regular RL. By simultaneously learning an end-to-end technique for several tasks sharing common useful knowledge, SRLfD forces the state representation to be general, provided that the tasks are diverse. Moreover, since the representation is trained together with heads that imitate the oracle policies, we believe that it is more appropriate for control than other types of representations (for instance ones that primarily aim at enabling a good reconstruction of the raw inputs). Our experimental results tend to confirm this belief, as SRLfD state representa-

tions were exploited more effectively by the SAC RL algorithm and a supervised learning system, than several other types of state representations.

5

EXPLORATORY STATE REPRESENTATION LEARNING

> To know what you know and what you do not know, that is true knowledge.
>
> Confucius

ABSTRACT

Not having access to compact and meaningful representations is known to significantly increase the complexity of reinforcement learning (RL). For this reason, it can be useful to perform state representation learning (SRL) before tackling RL tasks. However, obtaining a good state representation can only be done if a large diversity of transitions is observed, which can require a difficult exploration, especially if the environment is initially reward-free. To solve the problems of exploration and SRL in parallel, we propose a new approach called XSRL (eXploratory State Representation Learning). On one hand, it jointly learns compact state representations and a state transition estimator which is used to remove unexploitable information from the representations. On the other hand, it continuously trains an inverse model, and adds to the prediction error of this model a k-step learning progress bonus to form the maximization objective of a discovery policy. This results in a policy that seeks complex transitions from which the trained models can effectively learn. Our experimental results show that the approach leads to efficient exploration in challenging environments with image observations, and to state representations that significantly accelerate learning in RL tasks.

KEYWORDS

State Representation Learning, Pretraining, Exploration, Unsupervised Learning, Deep Reinforcement Learning

5.1 INTRODUCTION

Recent improvements in computational power and deep learning techniques have been combined with reinforcement learning (RL) to create deep RL (DRL) algorithms capable of solving complex control tasks with continuous state and action spaces [Li, 2018]. These improvements have popularized end-to-end DRL techniques, which involve letting deep learning systems automatically learn their representations and make predictions simultaneously (i.e. without performing a feature extraction as a preliminary phase). However, despite its simplicity of design, this end-to-end approach has four main limitations discussed in Section 3.2.1. In the context studied during this thesis of continuous control tasks with visual observations, we reviewed methods that have improved on these end-to-end techniques, but they face a significant computational challenge. In Section 3.2.2 we presented two other alternatives to the end-to-end approach that focus on the state representation learning process of which state representation learning (SRL) is one.

We propose here a new SRL method that tends to circumvent the four main limitations of end-to-end DRL, namely (i) it removes the typically long horizon related to the control task to focus only on representation learning with dimensionality reduction; (ii) it guarantees the compositionality principle since the roles assigned to the modules are respected; (iii) it improves sample efficiency and convergence stability; (iv) it improves interpretability at the modular level. As explained in Section 3.3, this is possible by performing unsupervised state embedding pretraining from agents' experiences which contain knowledge about themselves (proprioceptive information) and the environment properties (related to perception) common to different unknown tasks. In other words, this knowledge is task-independent information about the agent-environment, such as the agent's configuration, the transition model of the environment (i.e. local consistency), and its structure (i.e. topology) [Morik et al., 2019] (for more details see Section 3.3.1).

For state representations to be good as inputs to an unseen RL task, a SRL training must observe a large diversity of transitions. Since it is often impossible to randomly explore all environment transitions, we optimize discovery policies. A standard RL policy is trained to associate an action to a state to maximize a reward. In a pure exploration context, a policy optimization can only use intrinsic rewards which estimate a degree of uncertainty about the trained models [Bubeck et al., 2009]. Our exploration strategy is one that explores the most diverse and learnable unknown transitions. However, previous strategies that also focus on such exploration belong to the context of model-based RL with planning [Shyam et al., 2019, Sekar et al., 2020], without aiming at learning an intermediate state. Instead, we propose a novel exploration strategy to learn a state embedding model, called XSRL (eXploratory State Representation Learning), which has the advantage of being computationally lighter.

XSRL consists of a twofold training procedure. In the first training procedure, XSRL learns state representations whose transitions are Markovian while advantageously reducing the image observation dimensions by filtering out unexploitable information with respect to the objective of the next observation prediction. In the second training procedure, XSRL learns discovery policies which draw actions considered as uncertain by an inverse model, and from which the state transition estimator can learn the most. Finally, in order to cope with the two sources of non-stationarity due to evolving state representations and inverse model predictions, we train two discovery policies in parallel and, given their mutual performance, reset one of them after a given number of training steps (as explained in Section 5.3.2.1). We use an online training with a set of agents, each half of whom follows one of the two policies.

Contributions In this chapter, we propose a new SRL algorithm – XSRL (eXploratory State Representation Learning) – whose main contributions can be summarized as follows. First, we introduce a novel SRL architecture which learns a state transition estimator (denoted φ and composed of three different modules: α, β and γ) through the self-supervised next observation prediction objective to provide a recursive state estimation. The recursion allows the state representation to memorize information about past time steps in order to verify Markovian transitions, so as to restore full observability to environments whose real state has been replaced by image observations (Section 5.3.1). Second, XSRL provides a consistent exploration strategy in a rewardless environment, which is novel compared to other pure exploration strategies (Section 5.3.2). Third, we demonstrate the validity of XSRL representations as well as its discovery policies through quantitative and qualitative evaluations on three different environments (Section 5.5.1). Finally, we show improvements over other representation strategies through a comparative quantitative evaluation on unseen control tasks with a popular RL algorithm (SAC [Haarnoja et al., 2018b]) (Section 5.5.2).

5.2 RELATED WORK

Several other SRL algorithms with a near-future prediction objective have been proposed recently [Watter et al., 2015, Assael et al., 2015, Wahlström et al., 2015, van Hoof et al., 2016], and reviewed in Section 3.3.2.1. However, they separately learn state representations with the reconstruction objective on observations, and train a forward model on the learned states. The forward model forces the representations to retrieve information to make their transitions Markovian. The main limitation of these approaches is the inefficiency of the reconstruction objective, which forces the representations to contain unnecessary information from the observations. Because of this limitation, many empirical results in the literature show a poor generalization performance of this representation strategy to RL systems [Böhmer et al., 2015, Jaderberg et al., 2017, Shelhamer

et al., 2017, de Bruin et al., 2018]. Instead, XSRL jointly learns a state transition estimator with a next observation predictor with the next observation prediction objective. On the one hand, this forces the learned state representations to retrieve information and memorize it through the recursive loop in order to restore the observability of the environment (in this work, the partial observability is due to image observations) and to verify the Markovian property. On the other hand, this forces the learned state representations to filter out unnecessary information, in particular information about distractors (i.e. elements which are not controllable or do not affect an agent).

The XSRL exploration strategy is inspired by the line of work that maximizes intrinsic rewards corresponding to prediction errors of a trained forward model, which is a form of dynamics-based curiosity [Hester and Stone, 2012, Pathak et al., 2017, Burda et al., 2018]. These strategies are often used by model-free RL algorithms such as ICM [Pathak et al., 2017], which combine intrinsic rewards with extrinsic rewards to solve the complex exploration/exploitation tradeoff. Instead, XSRL applies to rewardless environments in the SRL context, i.e. it focuses only on the complex non-stationary training of state representation models. In addition, XSRL differs in two other ways: (i) while ICM is applied to discrete actions, XSRL is applied to continuous actions, (ii) while ICM uses prediction errors of a trained forward model, XSRL uses those of an inverse model because they only depend on the controllability properties (see Eq. 5.3.2.2).

While the previous prediction errors of an inverse model give an uncertainty estimation of actions with respect to their controllability, k-step learning progress bonuses of the transition model (φ) give an uncertainty estimation of actions with respect to their learnability. Learning progress estimation was initially proposed in the field of developmental robotics [Oudeyer et al., 2007]. Lopes et al. [2012] initiated the estimation of learning progress bonuses to solve the exploitation/exploration tradeoff in the model-based RL domain with finite MDPs. Achiam and Sastry [2017] have scaled this approach to continuous MDPs, however it remains limited to compact observations of several dozen dimensions. We now scale the work of Achiam and Sastry [2017] to image observations and in the SRL context. Thus, XSRL trains discovery policies to also maximize k-step learning progress bonuses of φ to favor learnable unknown transitions. This way, the XSRL exploration strategy exploits learnable unknown transitions with a high controllability diversity criterion.

5.3 PROPOSED METHOD: XSRL

5.3.1 *State Transition Estimator*

The goal of SRL is to transform high-dimensional observations into machine-readable compact representations which retrieve information about an agent and the environment and disentangle their degrees of freedom [Lesort et al., 2018].

To do this, we make the assumption with XSRL that a good state representation must contain the information needed to predict the next observation from the previous time step.

Our state transition estimator φ consists of two neural network parts (α, β), and a common network head γ. While α is a convolutional neural network (CNN) to process image observations, β is a multilayer perceptron (MLP) to process the concatenated action and state vectors. Finally, the common network head γ is a MLP to process the concatenated output vectors of the two first network parts to estimate next state vectors ($\mathbf{s}_{t+1}$).

According to the graph in Fig. 7(a), from current observation $\mathbf{o}_t$, previous action $\mathbf{a}_t$ and state $\mathbf{s}_t$, information is compactly merged into a next state $\mathbf{s}_{t+1}$ through the intermediate functions (α, β, γ). Because of the recursive loop on the state representation, φ bootstraps from an initial state drawn from a Gaussian distribution of mean zero and standard deviation 0.02. Putting all the functions together, we get the following definition of φ predictions:

$$\begin{aligned} \mathbf{s}_{t+1} &= \gamma\left([\alpha(\mathbf{o}_t),\ \beta\left([\mathbf{s}_t, \mathbf{a}_t]\right)]\right) \\ \mathbf{s}_{t+1} &= \varphi([\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]) \end{aligned} \tag{86}$$

where we abbreviate the state transition estimator network (α, β, γ) by φ and their parameters are concatenated into the following parameter set $\boldsymbol{\theta}_\varphi = \{\boldsymbol{\theta}_\alpha, \boldsymbol{\theta}_\beta, \boldsymbol{\theta}_\gamma\}$. The implementation details of the whole neural network are displayed in Table 6.

φ is trained jointly with a next observation predictor ω thanks to the next observation prediction objective. ω is a CNN with transposed convolution layers[1] to deterministically predict from the outputs of φ (i.e. $\mathbf{s}_{t+1}$) the next observations as follows $\omega(\mathbf{s}_{t+1}) = \hat{\mathbf{o}}_{t+1}$. This yields the following prediction error:

$$\|\hat{\mathbf{o}}_{t+1} - \mathbf{o}_{t+1}\|_2^2 \tag{87}$$

All the parameters of ω are gathered in a single parameter set $\boldsymbol{\theta}_\omega$. The corresponding training process will be described with the complete XSRL training process in Section 5.3.3.

Thanks to this joint training of φ and ω, XSRL builds compact state representations which contain the information needed to predict the next observations, which is deterministic and simple enough to be modeled. In the context where the robot's state space follows Markovian transitions but is unknown and only image observations are available, the environment becomes partially observable, which may be due to perceptual aliasing or dynamic transitions. We therefore force φ to memorize in the state representations (through the recursive loop) the information of past time steps in order to restore the Markovian property of the learned state transitions.

Indeed, to predict the next observation with ω, the next state representation $\mathbf{s}_{t+1} = \varphi([\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t])$ must contain the information of past and current time steps.

1 We used the 2D transposed convolution operator provided by PyTorch.

As this information cannot only be retrieved from $\mathbf{o}_t$ and $\mathbf{a}_t$, some of it must be memorized in $\mathbf{s}_t$ through the state recursive loop. In this way, the state representations learned by XSRL form Markovian transitions that translate mathematically as follows:

$$P\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right) = P\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \ldots, \mathbf{s}_0, \mathbf{a}_0\right) \tag{88}$$

for all states $\mathbf{s}_{t+1}, \mathbf{s}_t \in \mathcal{S} \subset \mathbb{R}^{\mathcal{S}_d}$ and actions $\mathbf{a}_t \in \mathcal{A} \subset \mathbb{R}^{\mathcal{A}_d}$. In particular, this forces state representations to verify the local consistency and often the topology (or connectivity) of the environment, since otherwise they would not be able to restore the observability [Morik et al., 2019].

The local consistency of an environment is related to the transition model, i.e. the way an agent transitions from one state to another, but without the reward information. In order for the state representations to verify this property in environments with acceleration, it is necessary for them to be linked to the environment dynamics. For example, a state representation of a torque-controlled robot requires to verify this property to retrieve his velocities and positions, which are necessary to predict the next observation.

Alternatively, the topology of an environment corresponds to its structure, i.e. properties independent of an agent. For example, a state representation of a navigator robot (like the one in TurtleBot Maze) requires to verify in addition to the local consistency this property to retrieve his orientation and position. Indeed, to localize an agent with perceptual aliasing, the state representation must memorize the environment structure independently of the agent and so invariant with respect to his orientation and position as noticed previously by Böhmer et al. [2013].

Another advantage of XSRL state representations is that they remove unnecessary information for predicting next observations so as to preserve useful information to predict the change produced by an action in the next observation. This has two main merits: (i) XSRL can effectively cope with dimensionality reduction, (ii) aleatoric uncertainty such as random noise or other random distractors will be removed by state representations, which is key to the success of XSRL pure exploration strategy. As shown by Burda et al. [2018], policies learned with a dynamics-based curiosity tend to be attracted by aleatoric uncertainty related to the environment transitions. Our results in Section 5.5.1 will show that XSRL trained discovery policies are not attracted to such transitions in TurtleBot Maze environment where one of the walls has a randomly sampled color at each time step.

5.3.2 *Discovery in the Face of Uncertainty*

5.3.2.1 *Over-Commitment*

A problem that arises in pure exploration with dynamics-based curiosity is the non-stationarity of intrinsic rewards. Specifically, as in other dynamics-based

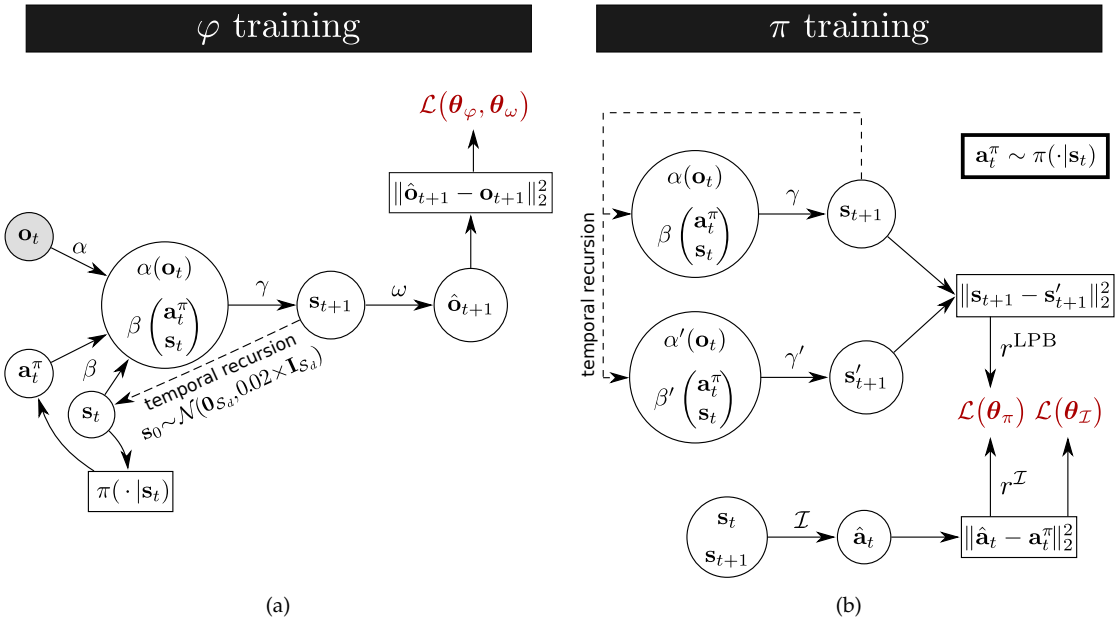


Figure 7. **(a)** XSRL learning process of state representations by jointly training a state transition estimator φ formed by (α, β, γ) and a head ω with the next observation prediction objective, where actions are sampled from one of the two discovery policies π_1 and π_2 (each policy is used on half of the agents in an online manner, as explained in Section 5.3.3). **(b)** XSRL learning process of a discovery policy by minimizing $\mathcal{L}(\boldsymbol{\theta}_\pi)$ which is related to the intrinsic rewards (for clarity reasons, π represents either π_1 or π_2). Intrinsic rewards are formed of two main terms. (i) $r^{\mathcal{I}}$: prediction errors of an inverse model $\mathcal{I}$ (also used in $\mathcal{L}(\boldsymbol{\theta}_\mathcal{I})$); (ii) r^{LPB}: k-step learning progress bonuses of φ where the parameters of φ' formed by (α', β', γ') are delayed by k training steps and kept fixed.

curiosity explorations from image observations, two sources of non-stationarity emerge [Burda et al., 2018]: (i) the model trained concurrently with discovery policies improves during training and its prediction errors minimize on visited transitions, (ii) the state representations evolve during φ training. Such a non-stationary training signal tends to attract policies in poor local optima. Indeed, when transitions maximizing intrinsic rewards are sufficiently visited, they become known and the policy generally cannot escape from this solution stuck in a poor local optima. Shyam et al. [2019] have popularized this problem as "over-commitment". The latter proposed to circumvent it by training from scratch a new policy. This is what we propose to do with XSRL by training two discovery policies in parallel called π_1 and π_2, and every T_{reset} iterations reset the policy with the lowest cumulative intrinsic rewards without the entropy term.

5.3.2.2 *Intrinsic Rewards*

The intrinsic rewards to be maximized by XSRL discovery policies are a combination of the following terms: (i) prediction errors of an inverse model which should be maximized on transitions which are complex with respect to their controllability, (ii) k-step learning progress bonuses of the state transition estimator

(φ) which should be maximized on transitions with high learnability with respect to the next observation prediction, (iii) a policy entropy estimation to ensure convergence stability. Fig. 7(b) shows the graph corresponding to the calculation of the two main terms (i) and (ii).

Inverse model While previous dynamics-based curiosity methods [Pathak et al., 2017, Burda et al., 2018] typically use a forward model to indirectly estimate an action uncertainty, we use an inverse model. Pathak et al. [2017] train an inverse model to learn state representations (without the goal to transfer them to new learning tasks). Prediction errors of a forward model on such representations would depend only on the elements controllable by an agent. Instead, we train a state transition model with a next observation predictor to learn state representations (with the goal to transfer them to new learning tasks). Prediction errors of a forward model on such representations would depend on what affect the camera due to the next observation prediction objective. For example, large variations in pose or illumination would be considered difficult to predict. Thus, maximizing prediction errors of a forward model from XSRL learned representations would favor actions that cause large observation changes which are not useful to discover the environment. Alternatively, prediction errors of an inverse model are only related to the difficulty of the state transitions. In summary, with XSRL learned representations, while prediction errors of a forward model depend on visibility properties, those of an inverse model depend on controllability properties. It is this controllability diversity criterion that motivated us to train an inverse model to indirectly estimate an action uncertainty.

An inverse model, takes as input a pair of consecutive states $(\mathbf{s}_t, \mathbf{s}_{t+1})$ to predict the action $\hat{\mathbf{a}}_t = \mathcal{I}(\mathbf{s}_{t+1}, \mathbf{s}_t)$ executed by an agent to obtain the next state $\mathbf{s}_{t+1}$. The prediction errors to be maximized by the discovery policies and minimized by the inverse model are calculated as follows:

$$r^{\mathcal{I}}\left(\hat{\mathbf{a}}_t, \mathbf{a}_t^{\pi}\right) = \|\hat{\mathbf{a}}_t - \mathbf{a}_t^{\pi}\|_2^2 \tag{89}$$

where the action $\mathbf{a}_t^{\pi}$ is sampled from π_1 and π_2 equally because half of the set of agents is associated with one of them. The training process of an inverse model is detailed later in Section 5.3.3.

Learning progress bonuses To ensure that actions considered uncertain by the inverse model (given the state representations learned by φ) lead to learnable unknown transitions, we use k-step learning progress bonuses of φ. This is a way to approximate the amplitude change in the parameter space produced by k training steps [Achiam and Sastry, 2017]. The larger this measure is on a new transition, the more φ and ω can reduce the prediction error of the corresponding next observation and thus generalize better. Maximizing these bonuses allows, during XSRL training, to progressively increase the complexity of the observed transitions, thus ensuring that some unknown transitions, too complex to be

learned with the current φ solution, are not favored until other easier transitions are observed. In other words, it ensures that the difficulty of transitions with respect to controllability intersects with their learnability by φ. Furthermore, once the inverse model has converged, these bonuses will complete the convergence of φ by increasing the diversity of transitions with respect to the next observation prediction objective. For example, if at new transitions there is deterministically predictable information, these bonuses allow visiting them to further improve the generalization performance of the model φ.

We adapted the k-step learning progress bonus proposed by Achiam and Sastry [2017] into the deterministic setting of our state transition model φ. To do so, we introduce φ' formed by $(\alpha', \beta', \gamma')$ with parameters delayed by k iterations and kept frozen. The k-step learning progress bonuses of φ to be maximized by the two discovery policies result in:

$$r^{\text{LPB}}\left(\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t^{\pi}\right) = \left\| \varphi([\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t^{\pi}]) - \varphi'([\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t^{\pi}]) \right\|_2^2 \tag{90}$$

where the action $\mathbf{a}_t^{\pi}$ is sampled from the discovery policy π_1 or π_2 that an agent follows.

Policy entropy estimation For better convergence stability, discovery policies must maximize in addition to intrinsic rewards their entropy estimation. Following Haarnoja et al. [2018b], we use an automatic weight tuning of this term $w_{\mathcal{H}}$ (a.k.a. temperature). This technique uses an optimization algorithm of the gradient descent type to automatically adjust this hyperparameter with respect to the difference between the estimated entropy and a target value $\bar{\mathcal{H}}$ to match as follows:

$$w_{\mathcal{H}} \left[\mathcal{H}\left(\pi(\cdot|\mathbf{s}_t)\right) - \bar{\mathcal{H}} \right] \tag{91}$$

As in the original implementation of the automatic tuning of the entropy term weight, the target entropy to match $\bar{\mathcal{H}}$ is equal to minus the action dimension $-\mathcal{A}_d$, see [Haarnoja et al., 2018b] for more details.

5.3.2.3 *Discovery Policies*

Now that we have detailed the three terms for computing intrinsic rewards, we explain how we train discovery policies to maximize them. In this work, we study environments with continuous action spaces. The general approach to learn a policy in this case is to model it as a multivariate Gaussian distribution with a diagonal covariance matrix from states to actions [Haarnoja et al., 2018b]. To do this, we use a neural network with a first common part, then one head μ_π with parameters $\boldsymbol{\theta}_\mu$ to predict a mean vector, and a second head Σ_π with parameters $\boldsymbol{\theta}_\Sigma$ to predict the diagonal covariance elements of a covariance matrix. The outputs of these two heads, which have the same dimensions as the action

space, allow us to parameterize a policy, so that it follows a Gaussian distribution defined as:

$$\pi(\cdot|\mathbf{s}_t) \triangleq \mathcal{N}\left(\mu_\pi(\mathbf{s}_t), \Sigma_\pi(\mathbf{s}_t)\right) \tag{92}$$

All parameters of a discovery policy are gathered in a single parameter set $\boldsymbol{\theta}_\pi = \{\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\Sigma\}$. The reparametrization trick [Kingma and Welling, 2014] is used to sample an action from a policy (i.e. $\mathbf{a}_t^\pi \sim \pi(\cdot|\mathbf{s}_t)$) to keep all its parameters differentiable:

$$\mathbf{a}_t^\pi \triangleq \mu_\pi(\mathbf{s}_t) + \boldsymbol{\epsilon}_t \times \Sigma_\pi(\mathbf{s}_t) \quad , \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}\left(\mathbf{0}_{\mathcal{A}_d}, \mathbf{I}_{\mathcal{A}_d}\right) \tag{93}$$

The two discovery policies (π_1 and π_2) can be optimized directly from the intrinsic reward gradients. Indeed, intrinsic rewards are computed with prediction errors of an inverse model, k-step learning progress bonuses of φ, and a policy entropy estimation, all of which use actions sampled from π_1 or π_2. Thus, intrinsic reward gradients can be used to train discovery policies in a supervised learning manner, as was done previously by Pathak et al. [2019]. Thus, our discovery policy training strategy is based on minimizing the following loss function:

$$-\left(w_{\mathcal{I}} r^{\mathcal{I}}\left(\hat{\mathbf{a}}_t, \mathbf{a}_t^\pi\right) + w_{\text{LPB}} r^{\text{LPB}}\left(\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t^\pi\right) + w_{\mathcal{H}} \mathcal{H}\left(\pi(\cdot|\mathbf{s}_t)\right)\right) \tag{94}$$

Minimizing this loss function, which amounts to maximize the intrinsic rewards, allows a gradient descent type of optimization. The corresponding training process is described in the next section.

The fact that we minimize a deterministic loss does not mean that there is no probabilistic interpretation. Indeed as explained previously, sources of prediction errors of an inverse model may be related to action uncertainty dependent on the controllability of learned state transitions; k-step learning progress bonuses of φ may be related to action uncertainty dependent on the learnability of state transitions. Furthermore, the estimated entropy of a policy is related to action uncertainty with respect to learned state representations. Thus, our two discovery policies learn a probability over the action space that tends to sample actions which maximize these uncertainties.

5.3.3 *Optimization Process*

Let us define the notations for the training examples we manipulate in our online training procedure. There is an even number $B \geq 2$ of agents in parallel, where $b \in [\![1, B]\!]$, and each of them is initialized in the same fixed configuration so that an effective exploration is required to visit the most diverse transitions of the environment. At time step t, a training example for (φ, ω) is an element of the form $\left(\mathbf{o}_{t+1}^{(b)}, \mathbf{o}_t^{(b)}, \mathbf{s}_t^{(b)}, \mathbf{a}_t^{\pi(b)}\right)$, composed respectively of the next observation and current observation, a state representation estimated at previous time step (i.e. $t-1$), and an action sampled from one of the two discovery policies as

$\mathbf{a}_t^{\pi(b)} \sim \pi(\cdot|\mathbf{s}_t^{(b)})$ (following the sampling process defined in Eq. 93). Specifically, each half of the set of B agents follows one of the two policies (π_1 and π_2). A state transition estimator φ composed of three modules (α, β, γ) estimates from the triplet input $\left(\mathbf{o}_t^{(b)}, \mathbf{s}_t^{(b)}, \mathbf{a}_t^{\pi(b)}\right)$ the next state $\mathbf{s}_{t+1}^{(b)}$, from which ω predicts the next observation $\hat{\mathbf{o}}_{t+1}^{(b)}$.

The optimization problem to simultaneously train φ and ω, is the minimization of the following objective function (based on the next observation prediction error of Eq. 86):

$$\mathcal{L}(\boldsymbol{\theta}_\varphi, \boldsymbol{\theta}_\omega) = \frac{1}{B}\sum_{b=1}^{B} \|\omega(\mathbf{s}_{t+1}^{(b)}) - \mathbf{o}_{t+1}^{(b)}\|_2^2 \tag{95}$$

After each b-agent has executed the action $\mathbf{a}_t^{\pi(b)}$, the backpropagation computes the partial derivatives of this objective function with respect to the parameter sets $\boldsymbol{\theta}_\varphi$ and $\boldsymbol{\theta}_\omega$ in order to perform a training step.

5.3.3.1 *Different Update Interval*

The inverse model and the two discovery policies are trained in parallel to the above training. Instead of performing a training step after every agent performs an action, it is performed after a chosen update interval (T_π). Since the policy optimization is much more sensible to the i.i.d. hypothesis, we use the largest possible sampling period k for these two types of optimization (k also corresponds to the number of training steps whose the parameters of φ' are delayed). To do this, we specify an update interval $T_\pi \in \mathbb{Z}^+$ which is the number of time steps before a training step is performed on the parameters of the inverse model and the parameters of the two discovery policies. Then, given a chosen batch size $B_\pi \in \mathbb{Z}^+$ and the number B of agents running in parallel, a batch of training examples is formed of $\lfloor \frac{B_\pi}{B} \rfloor$ samplings. Then, to maximize the independence between each of these samplings, we define a sampling period to be $k = \lfloor \frac{T_\pi B}{B_\pi} \rfloor$.

The optimization problem to train the inverse model is the minimization of the following objective function (based on the action prediction error of Eq. 89):

$$\mathcal{L}(\boldsymbol{\theta}_\mathcal{I}) = \frac{1}{B_\pi}\sum_{i=0}^{\lfloor \frac{B_\pi}{B} \rfloor - 1}\sum_{b=1}^{B} \left\| \mathcal{I}(\mathbf{s}_{t+1-ki}^{(b)}, \mathbf{s}_{t-ki}^{(b)}) - \mathbf{a}_{t-ki}^{\pi\,(b)} \right\|_2^2 \tag{96}$$

The backpropagation computes the partial derivatives of this objective function with respect only to the parameter set $\boldsymbol{\theta}_\mathcal{I}$.

The optimization problem to train the two discovery policies is the minimization of the following objective function (based on the loss of Eq. 94):

$$\mathcal{L}(\boldsymbol{\theta}_\pi) = \frac{1}{B_\pi} \sum_{i=0}^{\lfloor \frac{B_\pi}{B} \rfloor - 1} \sum_{b=1}^{B} - \Big[w_{\mathcal{I}} r^{\mathcal{I}}\big(\hat{\mathbf{a}}^{\pi}_{t-ki}{}^{(b)}, \mathbf{a}^{\pi}_{t-ki}{}^{(b)}\big) + w_{\text{LPB}} r^{\text{LPB}}\big(\mathbf{o}^{(b)}_{t-ki}, \mathbf{s}^{(b)}_{t-ki}, \mathbf{a}^{\pi}_{t-ki}{}^{(b)}\big) - w_{\mathcal{H}} \log\big(\mathbf{a}^{\pi}_{t-ki}{}^{(b)}\big) \Big] \tag{97}$$

where the parameter set $\boldsymbol{\theta}_{\mathcal{I}}$ is frozen, and that of φ' is updated every k iterations with that of φ and kept frozen. More specifically, the backpropagation computes the partial derivatives of this objective function with respect to the parameter set of one of the two discovery policies (i.e. $\boldsymbol{\theta}_{\pi_1}$ or $\boldsymbol{\theta}_{\pi_2}$). This objective function is low where the inverse model fails to predict actions, and learning progress bonuses are large.

Finally, to automatically tune the temperature $w_{\mathcal{H}}$, we minimize the following objective function:

$$\mathcal{L}(w_{\mathcal{H}}) = \frac{1}{B_\pi} \sum_{i=0}^{\lfloor \frac{B_\pi}{B} \rfloor - 1} \sum_{b=1}^{B} w_{\mathcal{H}} \Big[-\log\big(\mathbf{a}^{\pi}_{t-ki}{}^{(b)}\big) - \bar{\mathcal{H}} \Big] \tag{98}$$

As explained in Section 5.3.2.1, we choose to simultaneously train two discovery policies to mitigate the "over-commitment" [Shyam et al., 2019]. Specifically, our XSRL algorithm (as displayed in Algo. 2) resets the policy with the lowest intrinsic rewards without the entropy term as follows:

$$w_{\mathcal{I}} r^{\mathcal{I}}\big(\hat{\mathbf{a}}^{\pi}_{t-ki}{}^{(b)}, \mathbf{a}^{\pi}_{t-ki}{}^{(b)}\big) + w_{\text{LPB}} r^{\text{LPB}}\big(\mathbf{o}^{(b)}_{t-ki}, \mathbf{s}^{(b)}_{t-ki}, \mathbf{a}^{\pi}_{t-ki}{}^{(b)}\big)$$

accumulated over T_{reset} time steps (defined in Table 6). It is also a way to re-explore already learned transitions by performing suboptimal actions. Indeed, a policy trained from scratch must go through all types of transitions again. This ensures a better diversity of transitions visited throughout the XSRL training procedure and thus mitigates the overfitting peculiar to deep neural networks.

In summary, our XSRL algorithm described in Algo. 2, performs four types of optimization: (i) of a state transition estimator with Eq. 95, (ii) of an inverse model with Eq. 96, (iii) of two distinct discovery policies with Eq. 97, (iv) of an automatic temperature tuning with Eq. 98. This XSRL training procedure is repeated until the convergence of the parameters updated with the next observation prediction objective (i.e. $\boldsymbol{\theta}_\varphi$ and $\boldsymbol{\theta}_\omega$). See Table 6 for more details on the hyperparameters of our XSRL implementation.

Fig. 8 shows the two phases of XSRL considered in this work. (a): the twofold training procedure that XSRL follows in order to provide good state representations to solve unseen control tasks. (b): shows the deployment of such a solution φ to predict state representations for an unseen RL task. In particular, we no

Algorithm 2 XSRL algorithm

1: **Initialization:** Prepare an even number $B \geq 2$ of agents, reset to the same fixed starting state in an instance of the same environment $\mathbf{env}^{(b)}$, the first $\frac{B}{2}$ agents will be associated with π_1, and the other $\frac{B}{2}$ agents will be associated with π_2. Choose for the inverse model and discovery policies an update interval $T_\pi \in \mathbb{Z}^+$, a batch size $B_\pi \in \mathbb{Z}^+$ and compute the sampling period as $k = \lfloor \frac{T_\pi B}{B_\pi} \rfloor$; choose the reset interval T_{reset}; choose intrinsic reward weight terms: $w_{\mathcal{I}}$, w_{LPB}; choose the entropy target $\bar{\mathcal{H}}$ for the automatic tuning of the temperature (i.e. $w_{\mathcal{H}}$).

At each reset of a b-th agent: randomly initialize the state $\mathbf{s}_0^{(b)}$ from a Gaussian distribution of mean zero and standard deviation 0.02.

Randomly initialize: a state representation transition network φ formed by (α, β, γ) following the architecture described in Fig. 7**A** with parameters $\boldsymbol{\theta}_\varphi = \{\boldsymbol{\theta}_\alpha, \boldsymbol{\theta}_\beta, \boldsymbol{\theta}_\gamma\}$ and use these parameters to initialize φ' formed by (α', β', γ'); a next observation predictor network ω with parameters $\boldsymbol{\theta}_\omega$; an inverse model $\mathcal{I}$ with parameters $\boldsymbol{\theta}_\mathcal{I}$; two distinct discovery policies π_1 and π_2 with parameters $\boldsymbol{\theta}_{\pi_1}$ and $\boldsymbol{\theta}_{\pi_2}$ respectively.

2: **Output:** A task-independent state transition estimator φ formed of three modules (α, β, γ) to predict compact state representations.

3: **while** ($\boldsymbol{\theta}_\varphi$, $\boldsymbol{\theta}_\omega$) have not converged **do**

4: Sample actions from each of the discovery policies (π_1 and π_2) on half of the B agents as:

$$\mathbf{a}_t^{\pi(b)} = \mu_\pi(\mathbf{s}_t^{(b)}) + \boldsymbol{\epsilon}_t^{(b)} \times \Sigma_\pi(\mathbf{s}_t^{(b)}) \quad , \quad \boldsymbol{\epsilon}_t^{(b)} \sim \mathcal{N}\left(\mathbf{0}_{\mathcal{A}_d}, \mathbf{I}_{\mathcal{A}_d}\right)$$

5: Perform the action with every agent: $\mathbf{o}_{t+1}^{(b)} \leftarrow \mathbf{env}^{(b)}\left(\mathbf{a}_t^{\pi(b)}\right)$

6: Predict next state representations for all B agents:

$$\mathbf{s}_{t+1}^{(b)} = \gamma\left(\left[\alpha(\mathbf{o}_t^{(b)}),\ \beta\left(\left[\mathbf{s}_t^{(b)}, \mathbf{a}_t^{\pi(b)}\right]\right)\right]\right)$$

7: Compute $\mathcal{L}(\boldsymbol{\theta}_\varphi, \boldsymbol{\theta}_\omega)$ from Eq. 95.

8: Perform a training step on $\mathcal{L}(\boldsymbol{\theta}_\varphi, \boldsymbol{\theta}_\omega)$ w.r.t. $\boldsymbol{\theta}_\varphi$ and $\boldsymbol{\theta}_\omega$.

9: **every** T_π **iterations do**
Compute $\mathcal{L}(\boldsymbol{\theta}_\mathcal{I})$ with Eq. 96 and perfom a training step w.r.t. $\boldsymbol{\theta}_\mathcal{I}$.
Compute $\mathcal{L}(\boldsymbol{\theta}_{\pi_1})$ with Eq. 97 and perfom a training step w.r.t. $\boldsymbol{\theta}_{\pi_1}$.
Compute $\mathcal{L}(\boldsymbol{\theta}_{\pi_2})$ with Eq. 97 and perfom a training step w.r.t. $\boldsymbol{\theta}_{\pi_2}$.
Compute $\mathcal{L}(w_\mathcal{H})$ with Eq. 98 and perform a training step w.r.t. $w_\mathcal{H}$.
Update the parameters of φ' with those of φ and keep them frozen.

10: **every** T_{reset} **iterations do**
Reset one of the two discovery policies with the lowest intrinsic reward without the entropy term (i.e. $w_\mathcal{I} r^\mathcal{I}\left(\hat{\mathbf{a}}_{t-ki}^{\pi\ (b)}, \mathbf{a}_{t-ki}^{\pi\ (b)}\right) + w_{\text{LPB}} r^{\text{LPB}}\left(\mathbf{o}_{t-ki}^{(b)}, \mathbf{s}_{t-ki}^{(b)}, \mathbf{a}_{t-ki}^{\pi\ (b)}\right)$) accumulated over T_{reset} time steps.

11: **end while**

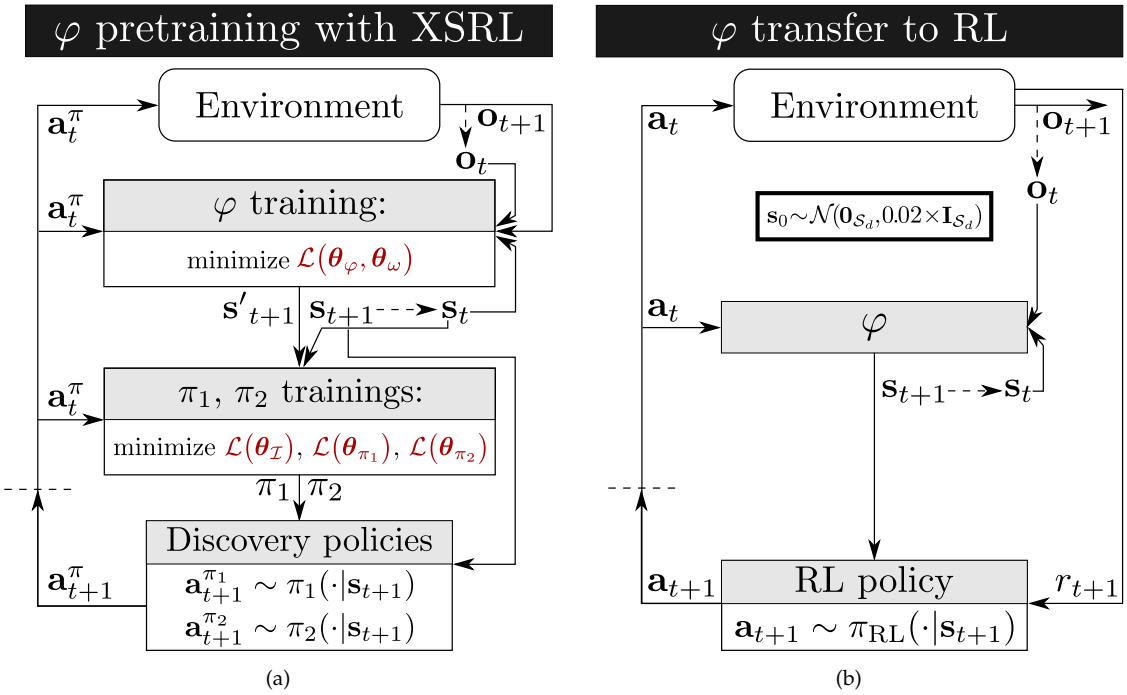


Figure 8. **(a)** Schematic representation of the XSRL twofold training procedure to provide compact state representations by jointly training a state transition estimator φ with a next observation predictor ω, guided by two discovery policies π_1 and π_2 (for clarity reasons, in the exponents of sampled actions π represents either π_1 or π_2) in an online manner with several agents in parallel (the first half of the agents following π_1 and the other π_2). Here $\mathbf{s}_{t+1} = \varphi([\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t])$, and $\mathbf{s}'_{t+1} = \varphi'([\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t])$ where φ' is the same network of φ whose parameters are delayed by k iterations and kept frozen. **(b)** A schematic illustration of the transfer of the pretrained state representation model (φ) to an unknown RL task.

longer use discovery policies, an inverse model, and a k-step delayed model φ', all corresponding to the exploration strategy of XSRL. We also discard the network head ω related to the self-supervised next observation prediction objective.

5.4 EXPERIMENTAL SETUP

This section describes a systematic evaluation of all criteria that our XSRL algorithm must verify. XSRL must learn state representations which (i) retrieve information (possibly by memorizing through past time steps) to garantee that their transitions are Markovian, (ii) filter unnecessary information. Furthermore, XSRL must learn discovery policies which (iii) guide agents quickly through most diverse transitions while avoiding unlearnable ones. Finally, after XSRL pretraining, the state transition estimator φ must (iv) provide advantageous inputs to solve unseen control tasks.

We evaluate the criterion (i) with the average of the next observation prediction errors on a training dataset and a test dataset. While the former is formed from samples generated during the training process, the latter is carefully designed for each environment, as described in Section 5.4.2.6. The lower this error measure is, the more the state transitions verify the Markovian property, because the next observation prediction depends only on the previous time step (as explained in Section 5.3.1). Furthermore, we measure the quantitative performance obtained by RAE (Regularized Autoencoder [Ghosh et al., 2019]) with the average of observation reconstruction errors to give a quantitative comparison with a state-of-the-art algorithm. However, since it is more complicated to predict the next observation than to reconstruct it, it is expected that the latter will perform better.

We evaluate the criterion (ii) of state representations and the criterion (iii) of discovery policies by training XSRL in our TurtleBot Maze environment by injecting aleatoric uncertainty into its transitions. To do this, we randomly sample at every time step a color for the small vertical wall in the lower branch of the maze, just in front of the initial state location of the robot, as shown in the top row of Fig. 11. We obtain a quantitative evaluation of our results with the same error measure as before, while also qualitatively evaluating our results by observing that XSRL does not attempt to predict the random wall color.

In addition to verify the criterion (iii), we perform other exploration evaluations during the state embedding pretraining of XSRL. First qualitatively by visualizing the coverage performance of XSRL discovery policies, then quantitatively by counting the average number of training steps before one of the 32 agents reaches the other end of the maze. Furthermore, since $\|\mathbf{o}_{t+1} - \hat{\mathbf{o}}_{t+1}\|_2^2$ is a useful prediction error measure to quantitatively evaluate the generalization performance of ω which is directly related to the performance of discovery policies, a high error measure will indicate that the XSRL exploration strategy is not efficient enough. To complete the evaluation of the discovery policy criterion, we also compare with two XSRL ablations:

- XSRL-MaxEnt: trains a policy to maximize its entropy estimation;
- XSRL-random: uses a random policy which samples actions uniformly from the action space.

Here, XSRL-random is expected to give minimal performance, while XSRL-MaxEnt should be worse than XSRL, as it only depends on the policy distribution.

We evaluate the criterion (iv) with the transfer of XSRL φ solution to unseen RL tasks. During RL training, the environment provides an agent with extrinsic rewards to train an optimal policy, while φ provides compact observations as shown in Fig. 8(b). To rigorously conduct this evaluation, we use a popular RL algorithm with continuous actions – SAC (Soft Actor-Critic) [Haarnoja et al., 2018b] – on each of the tasks in the three environments shown in Fig. 9. These continuous control tasks (presented in detail in Section 5.4.2), are challenging because of their high-dimensional observation spaces consisting of a camera. In order to obtain a quantitative evaluation of our results, we compare the performance between other representation strategies detailed below.

5.4.1 *Baselines*

We compare the performances of XSRL representations on unseen RL tasks to the following five baselines: ground truth, open-loop, position, RAE, random network.

Of all these baselines, only RAE (Regularized Autoencoder [Ghosh et al., 2019]) is a state-of-the-art SRL method. We train it using the same three rewardless environments with fixed state initializations as for XSRL described in Section 5.4.2.4. However, since it has no associated exploration strategy to generate observations, we use either a random policy as previously done by Yarats et al. [2019], or an effective exploration (indicated by the suffix *-explor*). In TurtleBot Maze, this effective exploration corresponds to a random policy with 50 time steps and random initialization, while in the two torque-controlled environments, it has 0.5 probability to take a random action and otherwise to take an action sampled from an optimal policy pretrained in the RL context (i.e. where extrinsic rewards are available) with SAC from the ground truth representations.

RAE is a deterministic alternative to the variational autoencoder (VAE) [Kingma and Welling, 2014], which preserves the regularizing effect of the latter. To the best of our knowledge, we do not know of any other method than RAE, belonging to the SRL context, that achieves state-of-the-art performance on the torque-controlled tasks of the DeepMind Control Suite (DMControl) benchmark [Tassa et al., 2018] with visual observations considered in this work. Specifically, on the DMControl benchmark, Yarats et al. [2019] obtain results where RAE perfoms as well as PlaNet [Hafner et al., 2018] with the SAC algorithm [Haarnoja et al., 2018b].

We also use a random network representation where, instead of training a network, its parameters are simply fixed to random values sampled from a Gaussian distribution of mean zero and standard deviation 0.02. This strategy without any training was popularized for classification tasks by Jarrett et al. [2009] and then for RL task by Gaier and Ha [2019].

We use only in the InvertedPendulum environment, the position baseline which corresponds to position measurements without velocities. The absence of velocities allows us to show the relevance of such dynamic information to solve the swing up task. Comparison with its performance would show that XSRL extracts this information from the observation information of consecutive time steps (by memorizing through the recursive loop) and thus transmits it to the RL system.

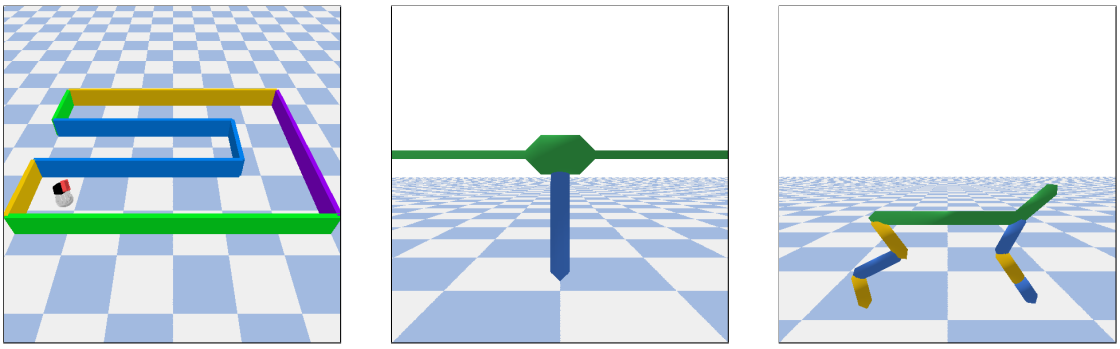
Finally, we use a ground truth baseline, which is a state directly extracted from the environment dynamics (see Section 5.4.2 for details in each environment), and an open-loop baseline, where the state is defined as the time step of an agent. The ground truth baseline is expected to constitute an upper bound on RL performance. The open-loop baseline serves as a sanity check. Indeed, the performance of SAC with this baseline on the three tasks allows us to validate whether these tasks really require closed-loop policy optimization. That is, whether it is necessary to use the agent's perception and proprioceptive information to solve the task, or whether open-loop policy learning strategies may be sufficient. In particular, this gives the minimum performance to beat to show the relevance of different state representation strategies.

We justify the absence of state-of-the-art end-to-end RL baselines such as [Lee et al., 2019, Kostrikov et al., 2020, Laskin et al., 2020, Srinivas et al., 2020], despite their open source implementations, by their too high computational complexity which is impractical in our hardware setting and limited computational time.

5.4.2 *Environment Details*

We perform our experiments on the three environments presented in Fig. 9 which are all partially observable due to image observations. InvertedPendulum and HalfCheetah belong to the MuJoCo torque-controlled benchmark [Todorov et al., 2012], implemented on PyBullet [Coumans and Bai, 2016–2019] (an open source library unlike MuJoCo). However, the MuJoCo benchmark was initially used from compact states directly provided by the simulator [Todorov et al., 2012]. In particular, while InvertedPendulum is easiest to control with a single torque, HalfCheetah, with its six torques, requires a state-of-the-art RL algorithm to solve its locomotion task even with compact states [Lillicrap et al., 2015].

We use the same number of action repetition as most works [Hafner et al., 2018, Yarats et al., 2019] (see Table 4). In many RL applications, the action is repeated several times to reduce the task horizon and make the control dynamics more stable. When the action is repeated, the number of observed time steps is reduced. For example with an action repetition of four, an episode of 1,000 total time steps is reduced to 250 observed time steps.



(a) (b) (c)

Figure 9. High-rendered images of the three continous control environments in PyBullet [Coumans and Bai, 2016–2019]. **(a)** The novel TurtleBot Maze environment proposed in this work, where the observation space corresponds to a first-person perspective camera, used to quantify XSRL exploration performance, and to provide a goal reaching task with navigation. **(b)** The InvertedPendulum environment provides a swing up task. **(c)** The HalfCheetah environment provides a locomotion task. **(b)**, **(c)** are two popular torque-controlled benchmark environments where the observation space corresponds to the camera tracking an agent, as in the DMControl benchmark [Tassa et al., 2018].

5.4.2.1 *TurtleBot Maze*

We have implemented this environment as a U-shaped maze with the TurtleBot robot from PyBullet [Coumans and Bai, 2016–2019], inspired by Ant Maze from OpenAI Gym [Brockman et al., 2016] used by Shyam et al. [2019]. The two-dimensional action applies a velocity to each of the left and right wheels of the robot. The three-dimensional ground truth state is formed by the cartesian coordinates in x and y axis of the robot, and the angle of his orientation. In this environment, the task consists in a goal reaching task with sparse rewards and a long horizon[2]. Thus, it is a challenge for a RL algorithm to address the exploration/exploitation tradeoff. Specifically, this task provides a RL algorithm with a sparse reward of +1 each time the robot reaches the goal, a reward of -1 each time he touches a wall, and 0 otherwise, within a maximum of 100 time steps before the robot and the goal are randomly reinitialized. In addition, this task provides a RL algorithm with the position of the goal, which is concatenated to the state representation. Indeed, since the goal position is task-dependent, it cannot be learned by state representations in a rewardless SRL context.

5.4.2.2 *InvertedPendulum*

The InvertedPendulum is attached to a pivot point on a cart sliding on a ramp. The one-dimensional action applies a force to the cart, which is limited to linear movement on the ramp. The five-dimensional ground truth state is formed by the x-axis position and velocity of the cart, the angular position in Cartesian space (i.e. cosine and sine of the angle) and angular velocity of the pendulum. In this

2 In TurtleBot Maze, an agent must perform 47 actions of maximum amplitude to cross the maze.

environment, the task consists in a swing up task where the pendulum must swing up several times before balancing upward. Specifically, this task provides a RL algorithm with a reward for keeping the pendulum up vertically, within a maximum of 1,000 time steps before the pendulum is reset to a random state.

5.4.2.3 *HalfCheetah*

The HalfCheetah is composed of eight rigid links, the torso and the back, and two legs each composed of three rigid and controllable links. The six-dimensional action applies torques to each of the six joints of the two legs. The 17-dimensional ground truth state is formed by the angular positions and velocities of the six joints, as well as agent cartesian position. In this environment, the task consists in a locomotion task where an agent must run to progress as far as possible. Specifically, this task provides a RL algorithm with a reward for moving the robot as fast as possible, in a maximum of 1,000 time steps and with a constraint that resets it to a random state as soon as it gets too close to the ground (which is not applied during XSRL and RAE trainings).

5.4.2.4 *Rewardless Environments*

We detail some of the differences in the three environments used without reward in the SRL context and the three tasks described above used in the RL context. In the SRL context (i.e. during XSRL and RAE pretraining), an agent is reset after a longer horizon, and is initialized to a fixed state. For the TurtleBot Maze the horizon is 500 time steps, hence the need of an effective exploration to reach the other end of the maze, which is at the opposite of the fixed initial state. For the other two torque-controlled environments (InvertedPendulum and HalfCheetah), the horizon is 2,000 time steps (so 500 after repeating the action four times). The remaining common hyperparameters of the three environments for the SRL and RL contexts are displayed in Table 4.

5.4.2.5 *Image Preprocessing*

The image preprocessing performed in these environments follows basically the same state-of-the-art approaches. Specifically, we divide the camera images by 255 to normalize them to $[0, 1]$. Then we downscale the image size to $3 \times 64 \times 64$ pixels just like [Lillicrap et al., 2015, Sekar et al., 2020]. When the action repeat is one, an observation corresponds to the image $\mathbf{o}_t = \mathbf{I}_t$. When it is four, an observation corresponds to the stack of the three consecutive images $\mathbf{o}_t = [\mathbf{I}_{t'-2}, \mathbf{I}_{t'-1}, \mathbf{I}_{t'}]$ of size $9 \times 64 \times 64$, just like [Lillicrap et al., 2015, Yarats et al., 2019], where t' corresponds to a time scale four times smaller than that of t (i.e. $t' = 4 \times t$). For our XSRL method, this concatenation of images obtained by repeating the last action three times allows not to loose all the information on these time steps. Moreover, the estimation of the state at each time step where the

action is repeated is far too computationally intensive to be feasible. Thus, this concatenation of images solves the trade-off between computational complexity and information loss. In practice for XSRL, φ uses $\mathbf{o}_t$ to predict $\mathbf{s}_{t+1}$, from which ω predicts the next observation $\hat{\mathbf{o}}_{t+1}$. In practice, RAE encodes $\mathbf{o}_t$ into $\mathbf{s}_t$ and decodes it into $\hat{\mathbf{o}}_t$.

5.4.2.6 *Test Datasets*

For quantitative performance evaluation of our XSRL algorithm, we use an error measure of the next observation prediction, and for the state-of-the-art RAE baseline, we use an error measure of the next observation reconstruction. To perform those evaluations, we need an appropriate test dataset for each of the three environments described above. To do this, we carefully collected a wide variety of 400 transitions formed of observation-action pairs into a dataset. We generated them in two different ways. In the case of TurtleBot maze, we hand-designed expert trajectories that follow the U-shape of the maze. In the case of InvertedPendulum and HalfCheetah, we executed a policy learned by SAC from the ground truth baseline.

5.4.3 *Implementation Details*

We now detail the implementation of the training procedures for XSRL and SAC. Our implementation uses the deep learning library PyTorch [Paszke et al., 2017]. The hyperparameter details for XSRL are displayed in Table 6, and for SAC, when different from the original implementation of Haarnoja et al. [2018b] in Table 5. Preliminary experiments showed that the hyperparameters $w_{\mathcal{I}}$ and w_{LPB} (to solve the tradeoff during discovery policy training between maximizing the prediction error of an inverse model and maximizing the k-step learning progress bonus of φ) had little impact on final performance.

For a fair comparison with RAE baseline, the same architecture as α (a convolutional neural network) and ω (a transposed convolutional neural network) is used for the encoder and decoder respectively. Similarly, for the random network baseline, the same architecture as α is used to produce state representations but its randomly initialized parameters remain fixed. We choose as state dimensions for the TurtleBot Maze and InvertedPendulum environments 20 and for HalfCheetah 30 which correspond to heuristically chosen values, i.e. not very large but leading to good RL results.

We use the same architecture for the policy (a.k.a. actor model) and the action-value function (a.k.a. critic model) of the SAC algorithm as for the discovery policies, the inverse model and γ of our XSRL algorithm. As detailed in Table 6, our three-hidden layer architecture is different from the two-hidden layer architecture typically used in these same environments with image observations [Yarats et al., 2019, Hansen et al., 2020]. This deeper network architecture allows

us to reduce the total number of parameters and thus the computational complexity. As Yarats et al. [2019], we use double Q-learning [Van Hasselt et al., 2015] for the critic model.

The Leaky Rectified Linear Unit (Leaky ReLU) is used for the activation functions between hidden layers, which removes the vanishing gradients encountered with the ReLU and improves the convergence speed and stability (which we observed empirically on preliminary experiments); see [Xu et al., 2015] for details.

In our RL experiments, the SAC algorithm is only used to test the generalization of the XSRL state representation to unseen control tasks. This implies that we keep the parameters of φ fixed. Due to memory constraints, for all experiments, we use a reduced buffer capacity unlike work comparable to ours: 100,000 instead of 1,000,000 with [Yarats et al., 2019].

5.4.3.1 *Hardware Details*

All our experiments are performed on three computers, each containing 40 cores and a Titan xp GPU provided by Nvidia.

Table 4. Hyperparameters used in the PyBullet environments [Coumans and Bai, 2016–2019].

Hyperparameter	Value
Image rendering size	$3 \times 96 \times 96$
Image size after downscaling	$3 \times 64 \times 64$
Action repeat	1 TurtleBot Maze
	4 otherwise

Table 5. Hyperparameters used for SAC (Soft Actor-Critic [Haarnoja et al., 2018b]) experiments.

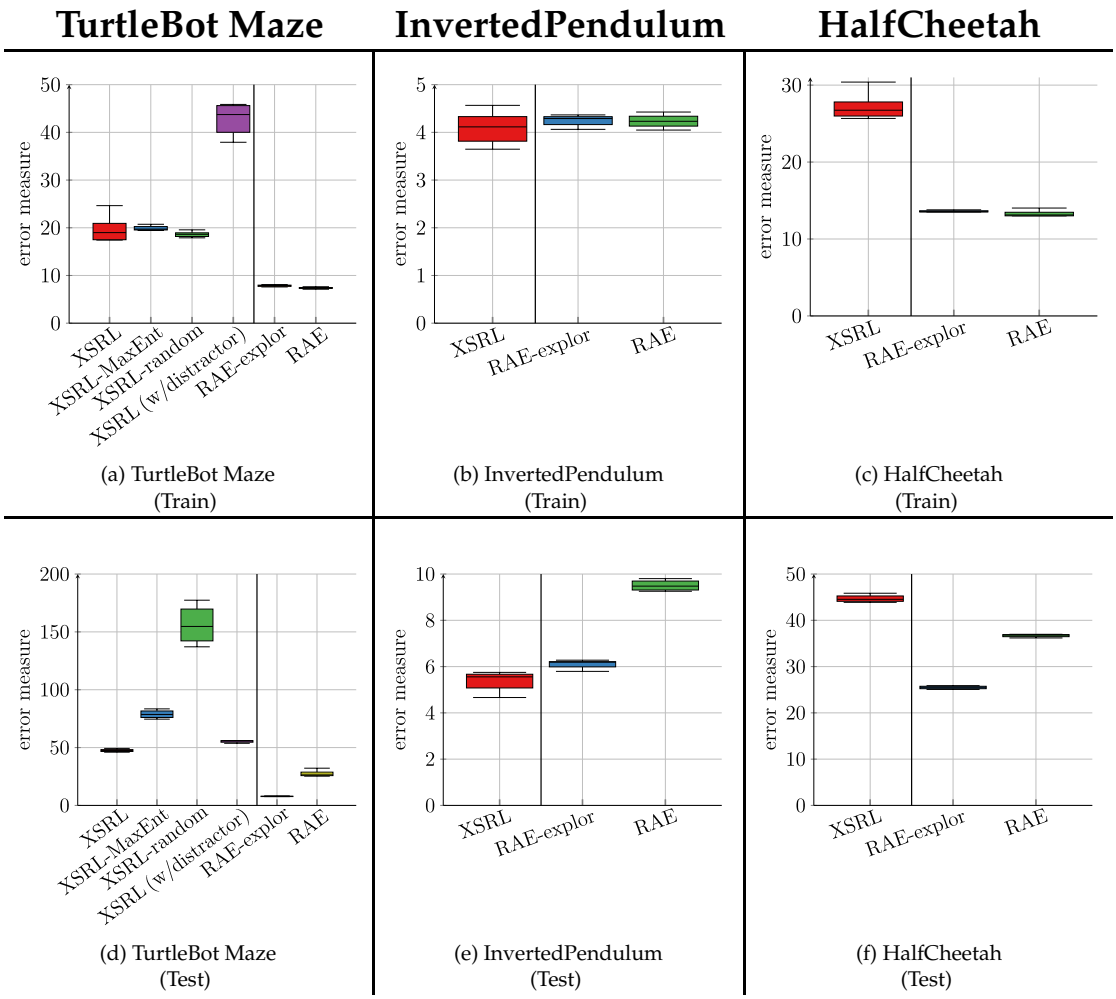
Hyperparameter	Value
Episode length of the environments	100 TurtleBot Maze
	1,000 otherwise
Discount facor γ	0.99
Replay buffer capacity	100,000
Optimizer	Adam [Kingma and Ba, 2014]
Batch size	256
Critic target update frequency	2
Actor update frequency	2
Learning rate for the critic model	5e-4
Learning rate for the actor model	5e-4
Learning rate for the automatic temperature tuning	5e-4
Entropy target $\bar{\mathcal{H}}$	$-\mathcal{A}_d$
Hidden units of critic/actor models	128, 512, 128

Table 6. Hyperparameters used for XSRL experiments.

Hyperparameter	Value
Episode length for all the environments (after action repeat)	500
State dimension $\mathcal{S}_d$	20 TurtleBot Maze; InvertedPendulum 30 HalfCheetah
α output dimension	30
β output dimension	$(\mathcal{S}_d + \mathcal{A}_d)$
Intrinsic reward weight terms	$w_{\mathcal{I}} = 0.5$, $w_{\text{LPB}} = 1$
Optimizer	Adam [Kingma and Ba, 2014]
Batch size B for α, β, γ, ω	32
Batch size B_π for $\mathcal{I}$ and π	128
Update interval T_π for $\mathcal{I}$ and π	512
Reset interval T_{reset}	4,096 for both discovery policies
Learning rate for α, β, γ, ω	1e-4
Learning rate for $\mathcal{I}$ and π	1e-4
Learning rate for $w_{\mathcal{H}}$	1e-4
Entropy target $\bar{\mathcal{H}}$	$-\mathcal{A}_d$
Hidden units of $\mathcal{I}$, π, γ	128, 512, 128
Hidden units of β	128, 512, 32
Hidden units of α, ω:	
α	CNN (strides and filters): (2, 32), (2, 64), (2, 128), (2, 256) MLP hidden units: 1024, 256, 32
ω	MLP hidden units: 32, 256, 1024 transposed CNN (strides and filters): (1, 256), (2, 128), (2, 64), (2, 32)

5.5 EXPERIMENTAL RESULTS

5.5.1 *Evaluations of XSRL Representations and Exploration*



(a) TurtleBot Maze (Train) (b) InvertedPendulum (Train) (c) HalfCheetah (Train)

(d) TurtleBot Maze (Test) (e) InvertedPendulum (Test) (f) HalfCheetah (Test)

Figure 10. Error measure results (the lower the better) obtained on a training dataset (top row) and a test dataset (bottom row) (which is defined for each environment in Section 5.4.2.6), averaged over last 20 epochs across 5 runs (with a random seed). This measure corresponds to the prediction of $\mathbf{o}_{t+1}$ with XSRL, and to the reconstruction of $\mathbf{o}_{t+1}$ with RAE. XSRL (w/ distractor) is performed in TurtleBot Maze with a randomly sampled wall color.

In this section, we show the results of our quantitative and qualitative evaluations to validate whether XSRL verify criteria (i), (ii), and (iii) which we defined in Section 5.4.

Fig. 10 reports the results of the error measure obtained on a training dataset and a test dataset (defined in Section 5.4.2.6) on each of the three environments. This error measure corresponds to the prediction error of the next observation for XSRL and the two ablations (XSRL-random and XSRL-MaxEnt); it corresponds to the reconstruction error of the next observation for RAE [Ghosh et al., 2019]

(following a random exploration) and RAE-explor (following an effective exploration, as explained in Section 5.4.1). We observe on the two environments, TurtleBot Maze and HalfCheetah, that the error measure for XSRL is higher than that for RAE and RAE-explor on both training and test datasets. This does not correspond to a poor exploration performance of XSRL but to the objective function which is more complicated than RAE. Indeed, all information in the next observation that cannot be predicted from the current time step is ignored as it is the case for random distractors or too complex information from the transition model, which tends to increase the prediction error. Furthermore, the qualitative results in Fig. 11 show that XSRL is able to predict very well what is relevant to predict the next observation, but ignores less useful/redundant information. For example, in TurtleBot Maze it predicts the walls very well, but not the exact checkerboard pattern on the floor (see Fig. 11(i)).

These results therefore imply that representations learned by XSRL guarantee Markovian transitions which is criterion (i). In other words, the representations learned by XSRL succeed in retrieving information that allows the prediction of the next observation from the current time step. First, on the InvertedPendulum and HalfCheetah environments, since they are torque-controlled robots, the learned representations must retrieve the angular positions and velocities (which requires that the state representations memorize the information of the previous time step through the recursive loop) of their joints so that ω can predict next observations. Second, on the TurtleBot Maze environment, the observation space is a first-person perspective camera, which constitutes a particularly complex partial observability setting. This complexity is amplified because the walls of the maze, as shown in Fig. 9(a) have only three colors for a total of eight walls. Therefore, when an agent is facing one of them, he cannot know his exact configuration (i.e. his position and orientation) or even which wall it is. This phenomenon is known as perceptual aliasing [Cadena et al., 2016]. Thus, for XSRL representations to allow ω to predict next observations, they must contain the robot's current configuration, which is only possible if they also verify the consistency and topology of the environment. The latter can be possible thanks to the recursive loop on state representations, which allows to memorize the information from the previous time steps (as explained in Section 5.3.1).

In TurtleBot Maze with a distractor represented by a randomly sampled wall color, the gray wall predicted by ω in Fig. 11(j) shows that this random color is ignored by φ. This confirms that XSRL learns state representations which filter out stochastic information and more generally unnecessary information from observations, which is the criterion (ii). This also explains that the higher error measure results for XSRL are only related to unexploitable information with respect to its prediction objective.

We evaluated XSRL discovery policies through a comparative visualization of maze coverage displayed in Fig. 12, accompanied by a quantitative evaluation of maze exploration presented in Fig. 13. We observe in Fig. 12 that on a trajectory

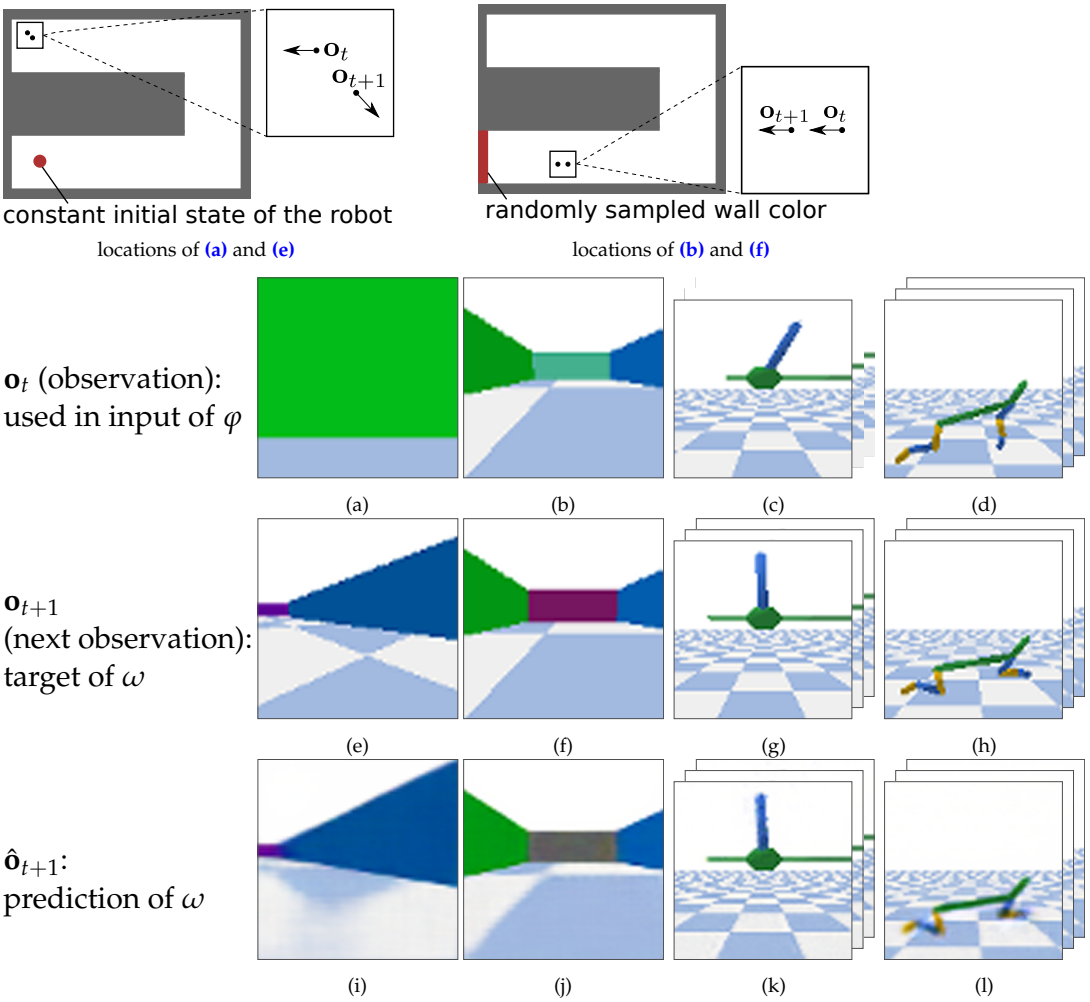


Figure 11. One of the most complex transitions in each of the test datasets (defined in Section 5.4.2.6) of environment from left to right: TurtleBot Maze, TurtleBot Maze w/ distractor, InvertedPendulum and HalfCheetah. The top line shows: (left) the locations of **(a)** and **(e)**; (right) the locations of **(b)** and **(f)**. In the bottom line, **(i)**-**(l)** show the corresponding $\mathbf{o}_{t+1}$ predictions with XSRL. In **(j)** XSRL ignores the random wall color because it predicts a neutral gray color instead. For InvertedPendulum and HalfCheetah environments, as the action is repeated four times, $\mathbf{o}_t$ corresponds to the three consecutive images $[\mathbf{I}_{t'-2}, \mathbf{I}_{t'-1}, \mathbf{I}_{t'}]$ (as explained in Section 5.4.2.5).

of 1,000 time steps, XSRL exploration reaches the farthest transitions much faster than XSRL-MaxEnt, while XSRL-random almost never does. Moreover, the trajectories generated by XSRL and XSRL-MaxEnt policies shown in Fig. 12 are not attracted to transitions with the distractor related to randomly sampled wall color near which the agent is initialized.

Fig. 13 shows that XSRL policies accelerate the probability that agents reach the other end of the maze in 500 time steps during pretraining. Specifically, with XSRL-random, agents can almost never reach the other end of the maze in only 500 time steps. Unlike the previous one, the exploration with XSRL and XSRL-MaxEnt use the visited state as input, which could lead an agent to always prefer transitions with a distractor (as analyzed by Burda et al. [2018]). However,

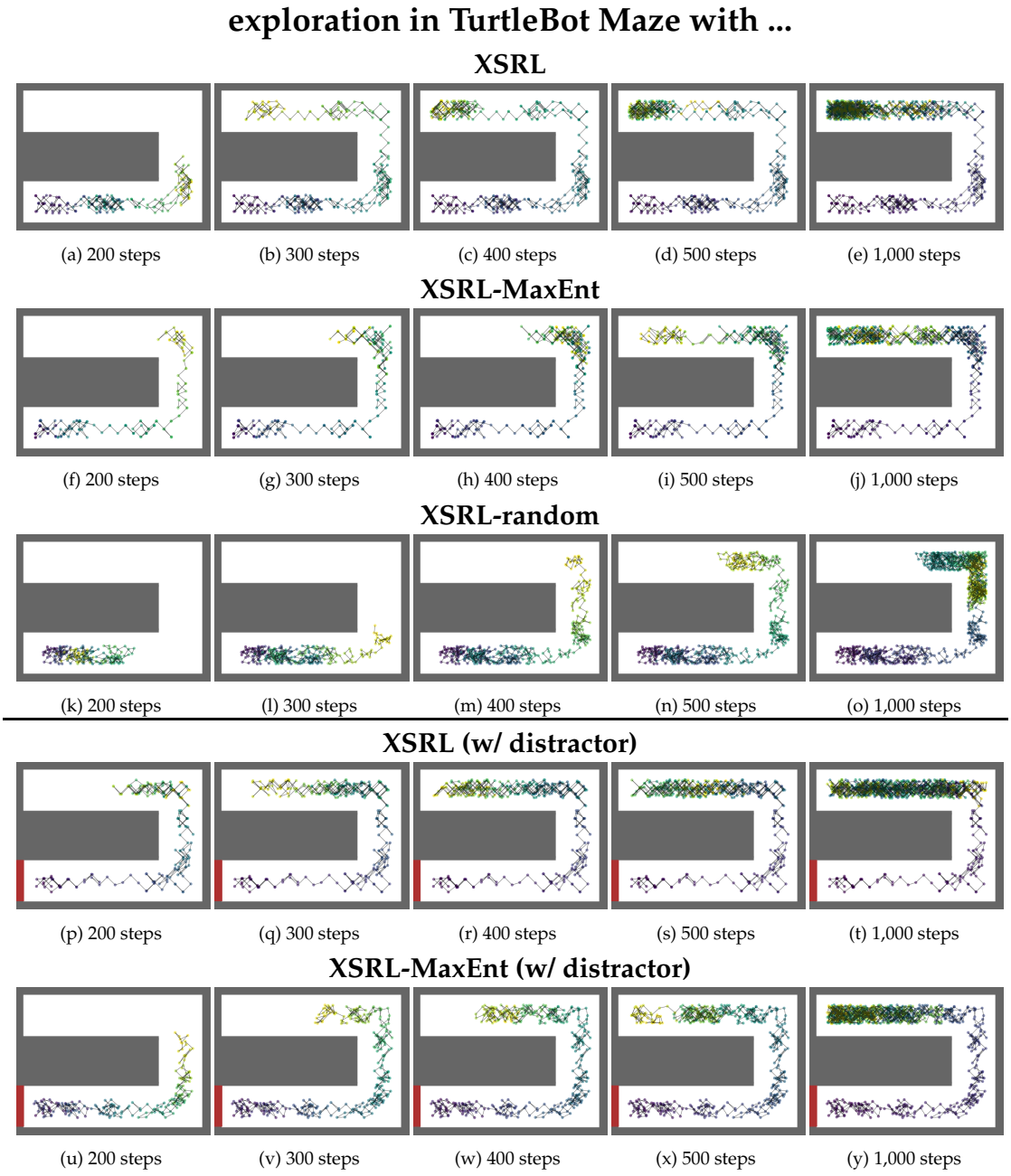
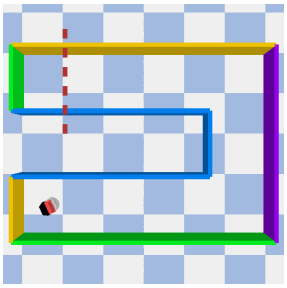
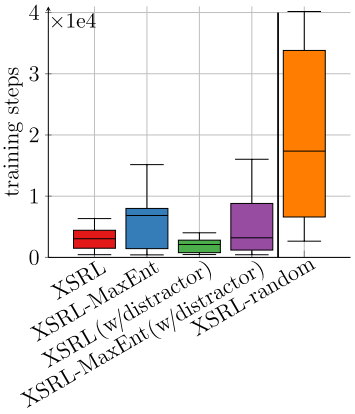


Figure 12. Visualization of the TurtleBot Maze exploration, where the color spectrum of the robot's position goes from purple (start of the episode) to yellow (end of the episode). The results obtained in the regular TurtleBot Maze environment correspond to: **(a)-(e)** XSRL, **(f)-(j)** XSRL-MaxEnt, **(k)-(o)** XSRL-random. The results obtained in TurtleBot Maze with a randomly sampled wall color correspond to: **(p)-(t)** XSRL, **(u)-(y)** XSRL-MaxEnt.

we observe no difference in performance with a distractor in TurtleBot Maze, which also confirms criterion (ii). Moreover, with and without a distractor, XSRL exploration reaches the end of the maze almost twice as fast as with XSRL-MaxEnt. These results thus confirm that XSRL discovery policies are successful in guiding agents quickly to diverse and learnable transitions, which is criterion (iii).



(a) TurtleBot Maze



(b)

Method	training steps
XSRL	$\mathbf{3,261 \pm 1,889}$
XSRL-MaxEnt	$6,593 \pm 5,281$
XSRL (w/ distractor)	$\mathbf{2,458 \pm 1,744}$
XSRL-MaxEnt (w/ distractor)	$6,129 \pm 5,804$
XSRL-random	$26,095 \pm 24,144$

(c)

Figure 13. **(a)** Shows the top view of the TurtleBot Maze where the robot's position corresponds to the constant initial state. **(b)**, **(c)** Number of training steps before an agent crosses the dotted red line during XSRL training (mean $\pm$ standard deviation over 10 runs; the lower the better). Our XSRL exploration strategy outperforms XSRL-MaxEnt, while XSRL-random provides an upper bound.

Given the previous qualitative and quantitative results, we interpret the performances of the exporation with XSRL and XSRL-MaxEnt as follows: XSRL discovery policies allow to visit the farthest transitions quickly because they tend to produce larger k-step learning progress bonuses of φ. They also visit more transitions that cause directional changes because they tend to produce larger prediction errors of a trained inverse model. On the contrary, XSRL-MaxEnt policies follow various paths without preferring the transitions farthest from the constant initial state (i.e. to the other end of the maze), even though the prediction errors of the next observation on these transitions are larger.

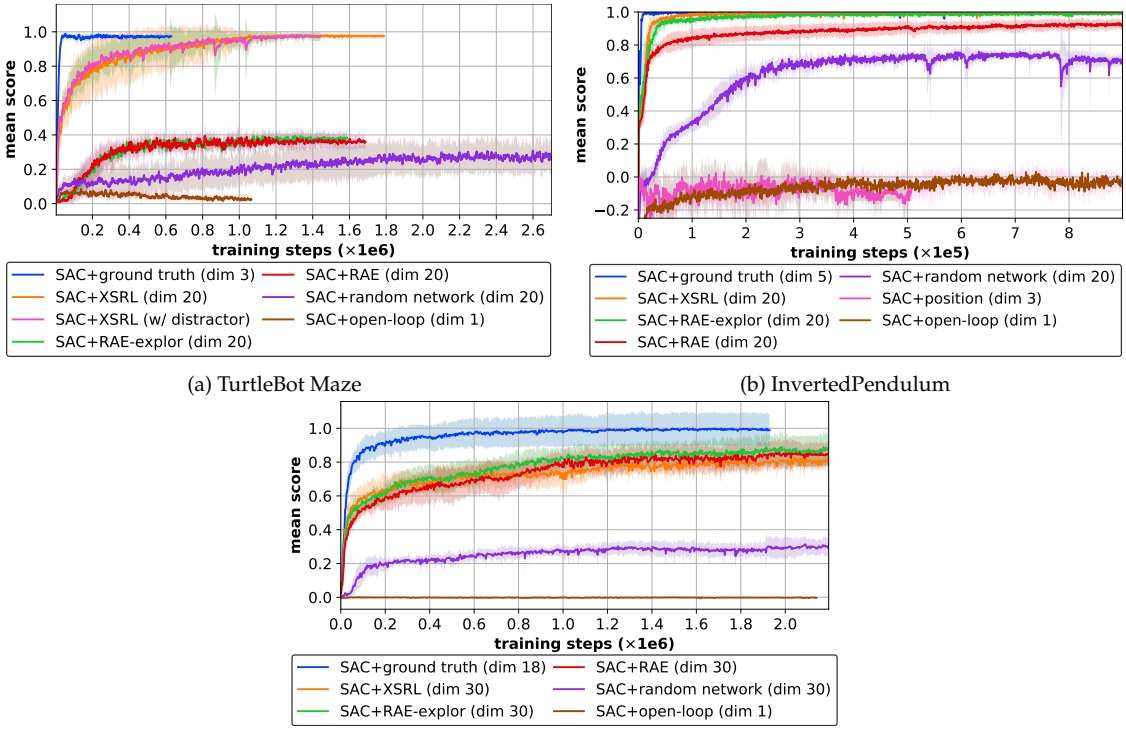
The video available here `https://youtu.be/IbGa-TC7wek`, shows a comparative evaluation between XSRL exploration (left) and random exploration (right) in each of the three environments. It highlights that discovery policies learned by XSRL allow: in TurtleBot Maze to quickly visit transitions far from the initial state position (as shown in the previous results); in InvertedPendulum to move the pendulum upwards while it is initialized downwards with zero velocity; in HalfCheetah to keep the robot constantly moving. We can see that for random exploration: in TurtleBot Maze the robot moves little away from its initial state; in InvertedPendulum the pendulum is never upwards; in HalfCheetah it is complicated for the robot to stay constantly in motion since it ends up stuck in a lying position.

In addition to these qualitative and quantitative comparisons, the better performance of XSRL exploration is also confirmed by the quantitative evaluation of the prediction error measure displayed in Fig. 10. This measure reaches its lowest value with XSRL exploration, followed by XSRL-MaxEnt and finally XSRL-random which is by far the worst strategy.

Apart from the comparative study of our XSRL exploration, we observe that an effective exploration improves the generalization performance of RAE models. Indeed, the quantitative evaluation of the observation reconstruction (see Fig. 10) shows a smaller error on the test dataset with RAE-explor (which is trained with an efficient exploration as explained in Section 5.4.1) than with RAE. Thus, training on diverse transitions improves the generalization performance of observation reconstruction with RAE.

Qualitative and quantitative performance differences with respect to exploration strategies show the advantage of visiting quickly diverse transitions during state embedding pretraining to obtain better generalization performance over new transitions. However, as we see below, it is only with the XSRL prediction error measure that this generalization performance is sure to translate into good transfer performance with a new RL task.

5.5.2 *XSRL Representations Transfer*



(a) TurtleBot Maze

(b) InvertedPendulum

(c) HalfCheetah

Figure 14. Learning curves of the episode returns averaged over 10 episodes (mean in lines and standard deviation in shaded areas over 10 runs; the higher the better) of SAC with different state representation strategies on three different continuous control tasks. The XSRL pretrained representations are the only one to perform well in three of the environments, while ground truth and open-loop provide an upper and lower bound respectively. A video showing the corresponding optimal policies can be found at https://youtu.be/XpRcU75i-iQ.

In this section, we show quantitative evaluations to validate whether state estimators pretrained with our algorithm (XSRL) provide advantageous inputs to RL systems for solving three unseen control tasks (which is an instance of criterion (iv) defined in Section 5.4). In particular, we use the deep RL algorithm SAC (Soft Actor-Critic) [Haarnoja et al., 2018b] which has shown promising results on the standard continuous control tasks InvertedPendulum and HalfCheetah. Throughout these experiments, all parameters of the pretrained state embeddings (with XSRL and RAE) are kept fixed: only the actor and critic neural networks of SAC are trained. We performed 10 runs with a random seed just like [Henderson et al., 2018, Yarats et al., 2019], resulting in 10 different trained policies for each of the representation strategies. For each state embedding pretraining approach (XSRL and RAE) and for the random network, we used 5 different models trained with a random seed, from which 2 SAC runs with a random seed are executed. In addition, unlike ground truth, open-loop and position baselines, they transform visual observations into compact state representations of 20 dimensions for TurtleBot Maze and InvertedPendulum, and 30 dimensions[3] for HalfCheetah.

Fig. 14 shows the learning curves of the episode returns averaged over 10 episodes across 10 runs. After training, we measured the episode returns averaged over 100 episodes for the 10 trained policies, which are displayed in Table 7. For clarity, we normalized all episode returns between the average SAC+ground truth performance and that of SAC+open-loop, except for the task with TurtleBot Maze as this is evaluated with the probability to reach the goal (from a random initial configuration) in 100 time steps or less. Indeed, SAC+ground truth is an upper bound because it has easy access to the agent's proprioceptive information, and SAC+open-loop is a lower bound because it corresponds to a blind agent. These results show that only XSRL state representations perform well in all three RL problems, unlike the other state representation baselines.

Fig. 14(b) shows that the position baseline does not allow SAC to learn a good policy on the InvertedPendulum task. This confirms that InvertedPendulum and HalfCheetah tasks require information from the positions and velocities of the agent's joints to verify Markovian state transitions which are only related to the local consistency of the environment. On InvertedPendulum, SAC+XSRL and SAC+RAE-explor reach same performance as SAC+ground truth, SAC+RAE follows right after, and even lower follows SAC+random network which reaches a mean score of 0.74. On HalfCheetah, we obtain in decreasing order of performance: SAC+RAE-explor, SAC+RAE, SAC+XSRL, SAC+random network. While the first three performances are similar with a score of about 0.85, SAC+random network is well below with about 0.3.

In TurtleBot Maze, none of the state representation strategies other than XSRL were successful on the navigation task. Specifically, SAC+XSRL reaches the same

3 The numbers of 20 and 30 dimensions were empirically selected to account for the trade-off between sample efficiency and final performance (i.e. between computation time and the optimal policy performance).

Table 7. Episode returns after convergence of the curves in Fig. 14 averaged over 100 episodes (mean ± standard deviation over 10 runs; the higher the better).

Mean score	TurtleBot Maze	InvertedPendulum	HalfCheetah
SAC+XSRL	**0.98 ± 0.02**	**1 ± 0**	0.82 ± 0.03
SAC+RAE-explor	0.34 ± 0.04	0.99 ± 0	**0.87 ± 0.09**
SAC+RAE	0.34 ± 0.06	0.93 ± 0.03	0.85 ± 0.08
SAC+random network	0.27 ± 0.1	0.74 ± 0.02	0.31 ± 0.05
SAC+ground truth	0.98 ± 0.02	1 ± 0	1 ± 0.1
SAC+open-loop	0.04 ± 0.03	0 ± 0.06	0 ± 0

performance as SAC+ground truth; SAC+RAE-explor and SAC+RAE reach the same score of 0.34, while SAC+random network is just below with a score of 0.27 but has much slower convergence. This shows that only the XSRL representations verify the local consistency and topology of the environment to retrieve the orientation and position of an agent. As explained earlier in Section 5.3, these properties of the representations are required to make state transitions Markovian due to perceptual aliasing [Cadena et al., 2016]. Furthermore, as shown by Fig. 14(a), the performance of SAC+XSRL is the same in TurtleBot Maze with a distractor. This confirms that by memorizing the information useful for predicting the next observation and filtering out all the other information, the XSRL representations contain the information in a way that can control the robot.

According to these results, it is more difficult to learn state embeddings that encode the environment properties independent of an agent (i.e. topology) than those that depend on him (i.e. local consistency). Overall, these quantitative evaluations show that pretrained state estimators with XSRL provide advantageous inputs to solve unseen control tasks with SAC algorithm, which is an instance of criterion (iv).

5.6 DISCUSSION

Experimental results show that the proposed XSRL algorithm builds state representations that are the only ones to perform well on the three unseen RL tasks. We see the link between the generalization performance of XSRL with respect to the next observation prediction (see Table 10) and the transfer performance of its pretrained state estimator (φ) to a new RL system (see Table 7). More specifically, the good prediction performance on a test dataset of state embeddings pretrained with XSRL guarantees their good transfer performance to new RL tasks.

As we have seen, the generalization performance on the test dataset strongly depends on the exploration efficiency (see Fig. 10), which is better with XSRL than with its two ablations. Moreover, our exploration allows agents to reach transitions far away from their initial states and much faster than the policy entropy maximization and random strategies (see Fig. 13). Hence the link between fast exploration of various transitions and good generalization.

However, the strategy of maximizing the policy entropy works well for learning policies with extrinsic rewards in the RL context. In particular, Eysenbach and Levine [2021] show in a navigation task with obstacles that it allows to increase the exploration diversity of an agent and thus to more efficiently bypass obstacles than with a random exploration where he would remain blocked.

In contrast to XSRL, our results showed that the good reconstruction performance of state embeddings pretrained with RAE did not always guarantee good transfer to RL tasks. Specifically, no matter how much RAE models minimize the reconstruction error on the test dataset of TurtleBot Maze, RL systems will perform poorly with inputs from this type of representation.

Finally, we analyze the results of SAC+XSRL across the three different continuous control tasks. While in TurtleBot Maze and InvertedPendulum SAC+XSRL obtains performances equal to those of SAC+ground truth, in HalfCheetah its performance is lower (0.82) and close to SAC+RAE (0.85). Moreover, this poorer performance is accompanied by a failure during the pretrainings of XSRL models, since they obtain abnormally high prediction errors on the training datasets. Indeed, comparing this prediction performance on the most similar environment InvertedPendulum (because it does not present perceptual aliasing) we have in Table 10: 4.2 on InvertedPendulum, 27.6 on HalfCheetah. Furthermore, as we have already discussed, XSRL exploration allows in HalfCheetah to keep the robot in constant motion, whereas random exploration cannot because the robot quickly gets stuck in a lying position[4]. Therefore, the lower performance of SAC+XSRL in the HalfCheetah task is likely due to poor modeling of the transition model, as the result of the next observation prediction in Fig. 11(l) tends to confirm. Indeed, it shows that there are errors in the next observation prediction related to the positions of the robot joints, which proves that the state transition estimator has not been well modeled. This could be explained by the high complexity of the transitions generated by HalfCheetah using a torque control. Moreover, due to the high number of degrees of freedom (six), the properties of its dynamics are much more unstable than those of InvertedPendulum (only two degrees of freedom), and far more than those of TurtleBot Maze, the latter using a velocity control.

While there are sources of prediction errors that φ fails to reduce, φ forces compact state representations to discard information related to aleatoric uncertainty (as we explained in Section 5.3.1). Achieving robustness to random distractors is a common goal of the state estimation techniques initiated by Kalman [1960b]. However, some random distractors can sometimes be useful in specific control tasks. For example, in a task where an agent must follow a behavior only when a lamp that randomly changes color is green. Since this information is random, it will be unpredictable to XSRL and thus filtered by its state representations. To deploy the state transition estimator to this specific task, it is therefore necessary to concatenate to output of φ a vector representing the color of the lamp

4 See the video available at https://youtu.be/IbGa-TC7wek

(either handcrafted or learned) so that a RL algorithm can properly exploit this information.

Overall, experimental results have highlighted the main advantage of XSRL in learning state embeddings which are able to verify both the local consistency of the environment and its topology. While the state-of-the-art RAE method succeeds perfectly in encoding the former, it fails completely in encoding the latter, as shown by the RL results with the TurtleBot Maze task (see Table 7). By evaluating XSRL with one transfer scenario where it outperforms a state-of-the-art method, and with another scenario where XSRL comes on par with it, we follow the experimental recommendations of Henderson et al. [2018] to prove that our new XSRL method matters for RL transfer scenarios.

5.7 CONCLUSION

We have presented a SRL algorithm (XSRL) for pretraining state representations from next observation prediction errors by extending the understudied discovery faculty we realized with a new exploration strategy, that trains discovery policies to sample the most diverse and learnable unknown transitions. Our experiments show that XSRL exploration provides fast maze traversal compared to traditional random policy and policy entropy maximization strategies. Moreover, our comparative evaluation on unseen RL tasks confirmed the transfer efficiency of the pretrained XSRL models. One of the most striking results is the superiority of XSRL representations over autoencoder ones, which is obviously due to better representational properties since the underlying data manifold (i.e. agent and environment degrees of freedom) is constrained to follow Markovian transitions with respect to the prediction objective. Furthermore, these results illustrate the importance of the overlooked discovery faculty and suggest that an exploration strategy in the SRL framework can lead to better state representation pretraining approaches.

6

CONCLUSION

6.1 CONCLUSION

The initial goal of this thesis is to propose new pretraining approaches of state representations without a priori knowledge of the task, the ground truth state of the controlled system, or the environment. In this way, we can apply them in any rewardless environment in order to solve unknown tasks with reinforcement learning (RL). To this end, we have studied these approaches in the setting of controlled systems defined by a simulated environment with continuous actions and raw sensory observations. In this setting, we have extended the major impetus of state representation learning (SRL) literature through a yet understudied axis: discovery.

6.1.1 *General Contributions*

With our goal of unsupervised learning and discovery of an a priori unknown environment in mind, we devised two new state estimation methods for transfer to a wide range of new control tasks. Both methods use a different discovery strategy by leveraging on agents experience.

In Chapter 4, we introduced SRLfD (State Representation Learning from Demonstration) which leverages experience from several expert controllers solving various tasks. These are oracle policies assumed to be learned from sensors in a laboratory setting (e.g. with motion capture) in order to generate agents experience through multiple demonstration trajectories. The main idea is to learn a state embedding from more realistic observations (in particular images) by imitation learning of multiple oracle policies solving a different task. Here the discovery to solve the underfitting/overfitting tradeoff generalizes in the task space in the batch setting (i.e. with a limited dataset). Therefore, training and validation *tasks* are used here, whereas in supervised learning, training and validation *labeled data points* are normally used. In the same way that in supervised learning a large and varied labeled dataset improves the generalization performance on a task, here a large and varied set of tasks improves the generalization performance of state representations across multiple tasks. In this way, SRLfD is intertwined with the meta-learning idea which learns to generalize across tasks rather than through task-specific training examples [Hospedales et al., 2020]. However, in the context of inaccessible ground truth states, these approaches typically learn input embeddings in an end-to-end deep RL (DRL) approach like Parisotto et al. [2015]. Instead, our idea is that, in this context, it is much easier to learn state embeddings

in a self-supervised learning manner. We conducted a comparative evaluation of classical methods such as Kalman filtering, where tracker learning systems show better performance using SRLfD representations which were pretrained across several instances of the same tracking task. Furthermore, our results show the superiority of pretrained SRLfD models over the end-to-end approach in two different settings of compact state inaccessibility. In the first large-scale goal reaching task, because it is more difficult for an end-to-end DRL system to learn from scratch input embeddings that retrieve the two torque-controlled positions and velocities of the robotic arm from visual observations. In the second ballistic projectile tracking task, because it is more difficult for a tracking learning system to learn from scratch input embeddings that retrieve the velocity of the projectile from its position measurements.

In Chapter 5, we introduced XSRL (eXploratory State Representation Learning) which leverages experience from discovery policies exploring unknown transitions. The main idea is to learn a state embedding with a twofold training procedure. In the first one, the representation model is trained as a state transition estimator simultaneously with a next observation predictor. In the second one concerns the exploration to solve the underfitting/overfitting tradeoff. To this end, we chose here to generalize in the space of environment transitions in the online setting (i.e. with an unlimited dataset if we assume an unlimited pretraining time). This involves training discovery policies to maximize self-computed rewards from different degrees of uncertainty: (i) with prediction errors of an inverse model (related to action uncertainty dependent on the controllability of learned state transitions), (ii) with k-step learning progress bonuses of the representation model (related to action uncertainty dependent on the learnability of state transitions). These rewards are used to train policies by supervised learning, whereas in other works on exploration they are typically maximized by RL [Burda et al., 2018]. Just as in RL, a large number of varied control strategies improves the generalization performance of learned policies, here a large number of diverse transitions improves the transferability of learned state models to unseen tasks. Moreover, in the context of inaccessible compact states, the proposed exploration strategies typically follow an end-to-end DRL approach (where input embeddings are learned from scratch) like [Sekar et al., 2020]. Instead, our idea is that in this context it is much more efficient to pretrain unsupervised state representations using our discovery policies. We performed a comparative evaluation of exploration strategies with the random strategy and the policy entropy maximization ones, among which the XSRL exploration strategy shows better performance in exploring diverse transitions much faster. Furthermore, our results show the superiority of the pretrained XSRL models over the state-of-the-art RAE (Regularized Autoencoder [Ghosh et al., 2019]) approach in a navigation task with partial observability. Indeed, it is more difficult for RAE to learn representations that retrieve the position and orientation of the robot in a U-shaped maze from a first-person camera.

We have seen through these two methods new ways to take advantage of the SRL framework. First, thanks to its unsupervised learning approach to exploit huge datasets in the context of a simulated environment without reward. Second, thanks to its applicability to controlled systems to develop the discovery faculty which is still little studied in this context. In particular, both SRLfD and XSRL benefit from an exploration strategy that automatically generates training examples to solve the underfitting/overfitting tradeoff. In doing so, the training procedure circumvents the usual assumption that the training data distribution is close to the test distribution in order to generalize to multiple tasks [Marcus, 2018]. Our results showed that solving the underfitting/overfitting tradeoff with respect to each of the two learning heuristics in our method during state representation pretraining corresponded to good transfer performance on the unknown control tasks. Overall, our experiments have shown that these "divide and conquer" transfer learning techniques exhibit clear advantages over more traditional task-specific approaches, such as the end-to-end DRL algorithms that learn input embeddings from scratch, whether in terms of computational cost, sampling efficiency or final performance.

6.2 DISCUSSION

State Representation Learning Evaluation The SRL problem is inherently ill-posed because it has no unique solution. Thus, without a unique objective, it is very difficult to evaluate the performance of SRL algorithms. The most widely used evaluation measure is the performance obtained with pretrained SRL models on various RL applications [Lesort et al., 2018]. To perform such a rigorous evaluation protocol, we used the recommandations of Henderson et al. [2018]. However, as previously observed by Lesort et al. [2018], such evaluations are time consuming and may not be applicable to real-world applications. This is one of the reasons why we limited the number of representation baselines we used in our comparative evaluations, especially those of state representations learned from scratch with end-to-end DRL, as they belong to much more computationally and memory intensive learning regimes. For our XSRL method, we also evaluated the performance as it is typically done in the supervised learning paradigm, i.e. with respect to the error measure used in the objective function on a test dataset. We showed a correlation between this error measure and that of RL applications using XSRL pretrained models. In contrast, autoencoder pretrained representations did not yield such a correlation, showing the inefficiency of the observation reconstruction objective in learning state representations for control tasks. In addition to this error measure, physics-based measures could be used for future evaluations in a context without knowledge of the ground truth state. For example, the physical priors proposed by Jonschkowski and Brock [2015] may be adapted into evaluation measures to analyze which physical properties are learned by state representations.

Simulation to Reality Representation Transfer We have studied one phase of transfer learning: pretraining state representations in a simulated environment without reward for unknown tasks. Extending our SRL methods to real-world control applications would give impetus to the main goal of this thesis: pretraining state representations for controlled systems to give them the ability to autonomously learn new tasks with RL algorithms in particular. However, the cost of data acquisition is high in the real world robotics, so the data-intensive unsupervised learning paradigm may not work in this setting. Thus, in order to properly train our two SRL algorithms, we need to pretrain the state models in a simulation where data can be generated indefinitely. Hence, in order to adapt the pretrained representations to better grasp the real-world information, another transfer learning phase is required. To do this, we could extend previous work that bridges the gap between simulation and reality [Tercan et al., 2018].

Continual Representation Learning Our work has similarities to that of continual learning (a.k.a. lifelong learning) [Caselles-Dupré et al., 2018]. However, this field focuses on sequential learning of different tasks in a typically changing environment [Lesort et al., 2020]. In contrast, SRL focuses on better structuring state representations for new learning tasks. Nevertheless, the SRL framework studied in this thesis could provide an effective way to address the complex phenomenon of catastrophic forgetting peculiar to this field, i.e. which experience from the previous task should be memorized. Indeed, sequential training of a state embedding to continuously adapt to a changing environment is an effective way to memorize useful concepts from the raw data, independent of the current task, which should help the controlled system make action predictions. More precisely, thanks to this memorization of states, catastrophic forgetting would only concern control strategies to be retained in order to combine them for learning more difficult tasks.

6.2.1 *Generalization and State Representation Properties*

The generalization performance of the state representations clearly depends on the pretraining time and available data, the complexity of the environment and the robot (i.e. the controlled system), the exploration strategy, and the properties of state representations discussed below: their ability to be usable by machines, their ability to capture the physical properties of the controlled system, and their size.

Machine Interpretable Representations This thesis has shown that SRL pretraining methods can retrieve the manifold underlying agents experience into a machine-readable vector (i.e. a state representation) to provide better inputs to new learning systems than raw data. Throughout our various results, we have acknowledged that numerical representations interpretable by machines

are different from visual observations interpretable by humans. Indeed, while humans can easily retrieve from the perception of raw images the underlying degrees of freedom of a controlled system, machines need a powerful mathematical tool in order to translate images into numerically usable inputs in the form of a multidimensional vector which, conversely, would be unreadable for humans. Among the most important properties for the representations to be machine interpretable are obviously the compactness property to mitigate the curse of dimensionality, independence from a specific task, the possibility to achieve better results than with end-to-end trained input embeddings, and to discard distractors [Locatello et al., 2018, Lesort et al., 2018]. Our empirical results give us promising hope in the direction of SRL and deep learning to provide such representations. As proven by our various experiments, SRLfD and XSRL methods learn state estimation models which retrieve the low-level and high-level concepts corresponding to physical properties. This amounts to converting the a priori non-numerical abstract concepts of the raw data into machine-interpretable numerical representations. More generally, unsupervised deep learning provides us with an effective mathematical tool to automatically learn input embeddings from huge datasets. The deep network we used was introduced by LeCun et al. [1995]. The effectiveness of this not very recent mathematical tool comes from the fact that it allows learning systems to rely on the global structure of the manifold underlying the data rather than just the local properties of the data as it is the case for more traditional embedding learning methods [Bengio et al., 2007]. This is why our SRL methods can so effectively represent manifold information (i.e. the intrinsic degrees of freedom) in state embeddings. Furthermore, our experiments have shown that random networks do not extract interpretable representations for new learning systems, which confirms that constraints must be imposed on the manifold by a training procedure. More precisely, a proper information arrangement in the state representations is a necessary condition for their generalization to new tasks.

Physical Representations We have seen through our results that theoretically inspired methods such as the variational autoencoder [Kingma and Ba, 2014] and others [Higgins et al., 2017, Ghosh et al., 2019] have not pretrained interpretable state representations for RL systems. We believe that notions such as disentanglement, on which these approaches are based, may be too general to pretrain good state representations [Bengio, 2012]. While disentanglement is necessary to retrieve the manifold underlying the data in a low-dimensional embedding, SRL algorithms clearly need more constraints on the manifold structure to learn state representations that contain physical properties [de Bruin et al., 2018].

We believe that research efforts to push creativity towards designing new learning heuristics based on physical principles may lead to new promising SRL methods for enabling machines to leverage huge datasets to retrieve abstract concepts in their own numerical representations. In this regard, we follow the

recommendation to privilege creativity over theory proposed in the presentation talk of Bottou and Bousquet [2018] at the 2018 NeurIPS Test of Time award for the winning paper [Bottou and Bousquet, 2008]. Imitating expert behaviors as with SRLfD, or predicting pixel changes in the next observation as with XSRL, are two new examples of learning objectives that intuitively let learning systems build representations which verify physical laws. Furthermore, both our methods constrain the state representation to guarantee Markovian transitions with respect to their learning heuristics so that the state representations retrieve more abstract physical concepts, such as the structure of the environment or its topology, while discarding unnecessary information to effectively reduce data dimensions. In particular, such pretrained state representations in environments where an observation is not sufficient to determine the actual state, i.e. where the partially observable Markov decision process (POMDP) holds, have clear advantages over state embeddings trained from scratch for RL algorithms. Indeed, in our context a policy does not need to depend on previous time steps to predict actions, making the exploration/exploitation tradeoff easier to solve in order to learn a policy with good generalization performance. Another form of knowledge on the controlled system different from physical knowledge that we have not studied in this thesis are the causal relationships that would allow representations to better retrieve logical concepts for example [Pearl, 2009, Lake et al., 2016, Peters et al., 2017].

State Dimension In this thesis, we have carefully selected the hyperparameters of neural networks, one of the most crucial of which is the dimension of the state representation. Choosing them carefully may be sufficient to generalize a SRL algorithm to different applications. Indeed, XSRL worked well with the same hyperparameters in different large-scale controlled systems (i.e. with visual observations and continuous actions in simulated environments). However, it may be useful to define in advance the dimension of the state representation without having to resort to a cross-validation, thus consolidating the goal of this thesis in a context where not only the task of interest is unknown but also the environment.

Ideally, the optimal number of dimensions in the state vector should allow the learned embedding to extract the degrees of freedom of the controlled system, while retaining past information to have Markovian transitions. Thus in theory, the state vector requires at least the number of degrees of freedom plus additional dimensions to retain information from past interactions when partial observability holds. Since in the simplest context of a fully observable environment, an agent's past experience is not useful, the state vector should have the same number of dimensions as the number of degrees of freedom. Therefore, we can assume that there exists a class of state representation mappings whose output is in bijection with the degrees of freedom. If this bijection exists, then we can

arbitrarily approximate such a mapping with a neural network according to the universal approximation theorem [Cybenko, 1989].

However, we observed empirical contradictions, that it is not bad to have more dimensions than the number of degrees of freedom. We studied this phenomenon with SRLfD, as we did not have enough computational time during the XSRL experiments. In the ballistic projectile tracking task, the ground truth state corresponds to a 4D vector composed of the 2D cartesian positions and velocities of a projectile. Our results showed that this task performs better with ground truth than with pretrained 4D SRLfD representations from position measurements, while it becomes better with at least six dimensions in the representations. In the RL application of the goal reaching task, the ground truth state corresponds to a 4D vector composed of the two torque angles and velocities of a robotic arm. Our results showed that at least 16 dimensions in the pretrained SRLfD representations are required for success in this application. These results show that the theoretical minimum dimension was not a sufficient condition for good generalization performance of state representations on new learning tasks. Instead, an optimal state dimension of the representation would be slightly larger than the theoretical minimum, in order to resolve the tradeoff between the curse of dimensionality and the generalization performance. We acknowledge that analyzing this optimal dimension to resolve the tradeoff between the curse of dimensionality and the generalization performance of state representations is challenging and requires further work.

Our hypothesis is that even if, according to the universal approximation theorem [Cybenko, 1989], there exists a neural network to learn a mapping between the observation and the state representation that is in bijection with the degrees of freedom of the controlled system, in practice it may be too complex and require too many parameters hence the risk of overfitting. In other words, this mapping may exist but is not smooth enough to be well approximated, i.e. without any continuity with respect to derivatives of different orders [Pinkus, 1999]. It is obviously easier to have the smoothness property in higher dimension if we want to verify the injectivity property between the learned representation and the degrees of freedom.

For example, what would be a good representation of angle θ? It is expected that a representation in the form $(\cos(\theta), \sin(\theta))$ would be simpler to learn than a direct representation of θ given the discontinuities it contains in $-\pi, \pi$. This means that by adding a dimension, it is not an additional information that has been added, but that the representation has been improved because it is now more regular. Thus, there may exist simpler mappings of the data to a larger state vector which is in injection with the degrees of freedom. In other words, using a redundant state vector (i.e. with more dimensions than the ground truth state), allows us to learn a state estimation model with a less complex deep network, thus effectively overcoming the overfitting problem.

While we thought, as did the literature [Lesort et al., 2018, Nuzzo, 2020], that learning a lower-dimensional state representation would circumvent the overfitting problem, we were wrong. These misconceptions have their origin in the confusion between the number of parameters in the hidden layers of a deep network with those in the state representation. Many works have studied the underfitting/overfitting tradeoff in terms of the former in order to analyze its impact on the generalization capacity of deep networks [Maennel et al., 2018, Volhejn and Lampert, 2021]. On the contrary, to our knowledge, none has studied this tradeoff with respect to the dimension of the neural network output.

We have assumed that a state estimation model benefits from having more dimensions in its output, but we could conversely assume that a RL system benefits from having more dimensions in its input. Again, we have not yet found any work in the DRL literature that studies the influence of the generalization performance of a policy with respect to the dimension of a pretrained state representation. We also believe here that a few more dimensions in the input representation would simplify the mapping to be learned between state and action, and thus more easily resolve the exploration/exploitation tradeoff to learn a policy that generalizes better. More literally, we believe that the dimensionality of the state representations should be large enough to give flexibility to a SRL algorithm at pretraining time, and to a RL algorithm at policy learning time.

From a broader viewpoint in the field of learning low-dimensional input embeddings for controlled systems, we have found that much more research effort is devoted to end-to-end DRL applications where agents autonomously strive to discover an optimal policy by solving the difficult exploration/exploitation tradeoff. In this thesis, we have studied unsupervised pretraining of state representations where agents autonomously strive to discover an optimal state estimator by solving the difficult underfitting/overfitting tradeoff.

BIBLIOGRAPHY

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

P. Abbeel. *Apprenticeship learning and reinforcement learning with application to robotic control*. Stanford University, 2008.

P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

J. Achiam and S. Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm. Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems*, pages 8766–8779, 2019.

F. Anselmi, J. Z. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, and T. Poggio. Unsupervised learning of invariant representations with low sample complexity: the magic of sensory cortex or a new framework for machine learning? 2014.

K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017. URL http://arxiv.org/abs/1708.05866.

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. Data-efficient learning of feedback policies from image pixels using deep dynamical models. *arXiv preprint arXiv:1510.02173*, 2015.

T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part i. *IEEE robotics & automation magazine*, 13(2):108–117, 2006a.

T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006b.

G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

R. Bellman. *Adaptive Control Processes - A Guided Tour*, volume 2045 of *Princeton Legacy Library*. Princeton University Press, 1961. ISBN 978-1-4008-7466-8. doi: 10.1515/9781400874668. URL https://doi.org/10.1515/9781400874668.

R. E. Bellman et al. Dynamic programming. *Cambridge Studies in Speech Science and Communication. Princeton University Press, Princeton*, 1957.

Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013a.

Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai. Better mixing via deep representations. In *International conference on machine learning*, pages 552–560, 2013b.

D. P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005. ISBN 1886529264. URL https://www.worldcat.org/oclc/314894080.

D. P. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

W. Böhmer, S. Grünewälder, Y. Shen, M. Musial, and K. Obermayer. Construction of approximation spaces for reinforcement learning. *The Journal of Machine Learning Research*, 14(1):2067–2118, 2013.

W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning

agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, 29 (4):353–362, 2015.

L. Bottou. How big data changes statistical machine learning. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1–1. IEEE, 2015.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

L. Bottou and O. Bousquet. Neurips - test of time award. https://www.facebook.com/watch/live/?v=271569366878864&ref=watch_permalink, 2018.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

S. Bubeck, R. Munos, and G. Stoltz. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*, pages 23–37. Springer, 2009.

Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

L. Busoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 2018.

C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32 (6):1309–1332, 2016.

R. Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pages 657–664, 1995.

H. Caselles-Dupré, M. Garcia-Ortiz, and D. Filliat. Continual state representation learning for reinforcement learning using generative replay. *arXiv preprint arXiv:1810.03880*, 2018.

L. Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL https://arxiv.org/abs/2002.05709.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

C. K. Chui and H. N. Mhaskar. Deep nets for local manifold learning. *Frontiers in Applied Mathematics and Statistics*, 4:12, 2018.

A. Cichocki, A. H. Phan, Q. Zhao, N. Lee, I. V. Oseledets, M. Sugiyama, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. *Found. Trends Mach. Learn.*, 9(6):431–673, 2017. doi: 10.1561/2200000067. URL https://doi.org/10.1561/2200000067.

N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on learning theory*, pages 698–728, 2016.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

E. Coumans, Y. Bai, and J. Hsu. Pybullet physics engine, 2018.

W. Curran, T. Brys, D. Aha, M. Taylor, and W. D. Smart. Dimensionality reduced reinforcement learning for assistive robots. In *Proc. of Artificial Intelligence for Human-Robot Interaction at AAAI Fall Symposium Series*, 2016.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw-Hill Higher Education New York, 2008.

T. de Bruin, J. Kober, K. Tuyls, and R. Babuška. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=BJtNZAFgg.

D. L. Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. C. Courville. Adversarially learned inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=B1ElR4cgg.

D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1577–1584. IEEE, 2018.

T. Eboli, J. Sun, and J. Ponce. End-to-end interpretable learning of non-blind image deblurring. *arXiv preprint arXiv:2007.01769*, 2020.

D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 201–208, 2010.

B. Eysenbach and S. Levine. Maximum entropy rl (provably) solves some robust rl problems. https://arxiv.org/pdf/2103.06257.pdf, 2021.

C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018. URL http://arxiv.org/abs/1811.12560.

A. Gaier and D. Ha. Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*, pages 5364–5378, 2019.

Z. Ghahramani. Unsupervised learning. In *Summer School on Machine Learning*, pages 72–112. Springer, 2003.

P. Ghosh, M. S. Sajjadi, A. Vergari, M. Black, and B. Schölkopf. From variational to deterministic autoencoders. *arXiv preprint arXiv:1903.12436*, 2019.

T. Glasmachers. Limits of end-to-end learning. In M. Zhang and Y. Noh, editors, *Proceedings of The 9th Asian Conference on Machine Learning, ACML 2017, Seoul, Korea, November 15-17, 2017*, volume 77 of *Proceedings of Machine Learning Research*, pages 17–32. PMLR, 2017. URL http://proceedings.mlr.press/v77/glasmachers17a.html.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018b. URL http://arxiv.org/abs/1812.05905.

J. Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

N. Hansen, Y. Sun, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. *CoRR*, abs/2007.04309, 2020. URL https://arxiv.org/abs/2007.04309.

M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, pages 1225–1234. PMLR, 2016.

K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

T. Hester and P. Stone. Intrinsically motivated model learning for a developing curious agent. In *2012 IEEE international conference on development and learning and epigenetic robotics (ICDL)*, pages 1–6. IEEE, 2012.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJ6yPD5xg.

A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2021.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

I. Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

R. Jonschkowski and O. Brock. Learning task-specific state representations by maximizing slowness and predictability. In *6th international workshop on evolutionary and reinforcement learning for autonomous robot systems (ERLARS)*, 2013.

R. Jonschkowski and O. Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

R. Jonschkowski, R. Hafner, J. Scholz, and M. Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *arXiv preprint arXiv:1705.09805*, 2017.

M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354, 1992.

R. E. Kalman. On the general theory of control systems. In *Proceedings First International Conference on Automatic Control, Moscow, USSR*, pages 481–492, 1960a.

R. E. Kalman. A new approach to linear filtering and prediction problems. 1960b.

R. E. Kalman et al. Contributions to the theory of optimal control. *Bol. soc. mat. mexicana*, 5(2):102–119, 1960.

A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4): 361–379, 2012.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

V. R. Kompella, M. D. Luciw, and J. Schmidhuber. Incremental slow feature analysis: Adaptive and episodic learning from high-dimensional input streams. *CoRR*, abs/1112.2113, 2011a. URL http://arxiv.org/abs/1112.2113.

V. R. Kompella, L. Pape, J. Masci, M. Frank, and J. Schmidhuber. Autoincsfa and vision-based developmental learning for humanoid robots. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 622–629. IEEE, 2011b.

I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *CoRR*, abs/2004.13649, 2020. URL https://arxiv.org/abs/2004.13649.

M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

B. Lake and M. Baroni. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. 2018.

B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2016.

S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

S. Lange, M. Riedmiller, and A. Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2012.

M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. *CoRR*, abs/2004.14990, 2020. URL https://arxiv.org/abs/2004.14990.

S. Lawrence, C. L. Giles, and A. C. Tsoi. What size neural network gives optimal generalization? convergence properties of backpropagation. Technical report, 1998.

H. Le, L. Vial, J. Frej, V. Segonne, M. Coavoux, B. Lecouteux, A. Allauzen, B. Crabbé, L. Besacier, and D. Schwab. Flaubert: Unsupervised language model pre-training for french. *arXiv preprint arXiv:1912.05372*, 2019.

Q. V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

P. H. Le-Khac, G. Healy, and A. F. Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 2020.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989a.

Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Y. LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 19:143–155, 1989b.

A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

R. Legenstein, N. Wilbert, and L. Wiskott. Reinforcement learning on slow features of high-dimensional input streams. *PLoS computational biology*, 6(8), 2010.

T. Lesort, M. Seurin, X. Li, N. D. Rodríguez, and D. Filliat. Unsupervised state representation learning with robotic priors: a robustness benchmark. *arXiv preprint arXiv:1709.05185*, 2017.

T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

Y. Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018. URL http://arxiv.org/abs/1810.06339.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

H. Liu, M. Long, J. Wang, and M. I. Jordan. Towards understanding the transferability of deep representations. *arXiv preprint arXiv:1909.12031*, 2019a.

X. Liu, F. Zhang, Z. Hou, Z. Wang, L. Mian, J. Zhang, and J. Tang. Self-supervised learning: Generative or contrastive. *arXiv preprint arXiv:2006.08218*, 1(2), 2020.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019b.

L. Ljung. Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, 24(1): 36–50, 1979.

F. Locatello, S. Bauer, M. Lucic, S. Gelly, B. Schölkopf, and O. Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in neural information processing systems*, pages 206–214, 2012.

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

H. Maennel, O. Bousquet, and S. Gelly. Gradient descent quantizes relu network features. *arXiv preprint arXiv:1803.08367*, 2018.

A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015. URL http://arxiv.org/abs/1511.05644.

G. Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

L. Martin, B. Muller, P. J. O. Suárez, Y. Dupont, L. Romary, É. V. de la Clergerie, D. Seddah, and B. Sagot. Camembert: a tasty french language model. *arXiv preprint arXiv:1911.03894*, 2019.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

A. Merckling, A. Coninx, L. Cressot, S. Doncieux, and N. Perrin. State representation learning from demonstration. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 304–315. Springer, 2020.

L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

T. M. Moerland, J. Broekens, and C. M. Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

M. Morik, D. Rastogi, R. Jonschkowski, and O. Brock. State representation learning with robotic priors for partially observable environments. In *IROS*, pages 6693–6699, 2019.

J. Munk, J. Kober, and R. Babuška. Learning state representation for deep actor-critic control. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 4667–4673. IEEE, 2016.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

H. Narayanan and S. Mitter. Sample complexity of testing the manifold hypothesis. In *Advances in neural information processing systems*, pages 1786–1794, 2010.

A. Y. Ng, H. J. Kim, M. I. Jordan, S. Sastry, and S. Ballianda. Autonomous helicopter flight via reinforcement learning. In *NIPS*, volume 16. Citeseer, 2003.

A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer, 2006.

A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4467–4477, 2017.

Q. Nguyen and M. Hein. Optimization landscape and expressivity of deep cnns. In *International conference on machine learning*, pages 3730–3739. PMLR, 2018.

F. Nuzzo. Unsupervised state representation pretraining in reinforcement learning applied to atari games, 2020.

C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

K. Ota, T. Oiki, D. K. Jha, T. Mariyama, and D. Nikovski. Can increasing input dimensionality improve deep reinforcement learning? *arXiv preprint arXiv:2003.01629*, 2020.

K. Ota, D. K. Jha, and A. Kanezaki. Training larger networks for deep reinforcement learning. *arXiv preprint arXiv:2102.07920*, 2021.

P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

D. Pathak, D. Gandhi, and A. Gupta. Self-supervised exploration via disagreement. *arXiv preprint arXiv:1906.04161*, 2019.

J. Pearl. *Causality*. Cambridge university press, 2009.

K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2 (11):559–572, 1901.

H. Penedones, D. Vincent, H. Maennel, S. Gelly, T. Mann, and A. Barreto. Temporal difference learning with neural networks-study of the leakage propagation problem. *arXiv preprint arXiv:1807.03064*, 2018.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

A. Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8(1):143–195, 1999.

L. Pinto and A. Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2161–2168. IEEE, 2017.

T. Poggio and Q. Liao. Theory i: Deep networks and the curse of dimensionality. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, 66(6), 2018.

T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

A. Raffin, A. Hill, K. R. Traoré, T. Lesort, N. Díaz-Rodríguez, and D. Filliat. Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics. *arXiv preprint arXiv:1901.08651*, 2019.

M. Ranzato, C. Poultney, S. Chopra, and Y. L. Cun. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

R. P. Rao. An optimal estimation approach to visual perception and learning. *Vision research*, 39(11):1963–1989, 1999.

M. Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14(3):693–714, 1993.

W. Samek, T. Wiegand, and K.-R. Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

A. Sax, J. O. Zhang, B. Emi, A. Zamir, S. Savarese, L. Guibas, and J. Malik. Learning to navigate using mid-level visual priors. *arXiv preprint arXiv:1912.11121*, 2019.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak. Planning to explore via self-supervised world models. *arXiv preprint arXiv:2005.05960*, 2020.

P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2017.

P. Shyam, W. Jaśkowski, and F. Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788. PMLR, 2019.

D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in neural information processing systems*, pages 1857–1865, 2016.

A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *CoRR*, abs/2004.04136, 2020. URL https://arxiv.org/abs/2004.04136.

I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez, et al. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855*, 2017.

A. Stooke, K. Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. https://openreview.net/forum?id=_SKUm2AJpvN, 2020.

N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5): 405–420, 2018.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

H. Tercan, A. Guajardo, J. Heinisch, T. Thiele, C. Hopmann, and T. Meisen. Transfer-learning: Bridging the gap between real and simulation data for machine learning in injection molding. *Procedia Cirp*, 72:185–190, 2018.

G. Tesauro. Practical issues in temporal difference learning. In *Advances in neural information processing systems*, pages 259–266, 1992.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

M. H. Ur Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, and S. U. Khan. Big data reduction methods: a survey. *Data Science and Engineering*, 1(4): 265–284, 2016.

L. Van Der Maaten, E. Postma, and J. Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3928–3934. IEEE, 2016.

M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec): 3371–3408, 2010.

V. Volhejn and C. Lampert. Does sgd implicitly optimize for smoothness? *Pattern Recognition*, 12544:246, 2021.

N. Wahlström, T. B. Schön, and M. P. Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

M. Wang, W. Fu, X. He, S. Hao, and X. Wu. A survey on large-scale machine learning. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

C. J. C. H. Watkins. Learning from delayed rewards. 1989.

M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

B. Widrow and M. E. Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

L. Wiskott and T. J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

H. S. Witsenhausen. Separation of estimation and control for discrete time systems. *Proceedings of the IEEE*, 59(11):1557–1566, 1971.

B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images. *CoRR*, abs/1910.01741, 2019. URL http://arxiv.org/abs/1910.01741.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13 (3):55–75, 2018.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

A. Zhan, P. Zhao, L. Pinto, P. Abbeel, and M. Laskin. A framework for efficient robotic manipulation. *arXiv preprint arXiv:2012.07975*, 2020.

A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017a.

R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1058–1067, 2017b.