ISTITUTO ITALIANO DI TECNOLOGIA

ADVANCED ROBOTICS

UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Efficient Reinforcement Learning for Robotics Visual Policies with Sim-to-Real Transfer of Decoupled Architectures

Carlo Rizzardo

Thesis submitted for the degree of *Doctor of Philosophy* (35° cycle)
January 2024

Darwin G. Caldwell — Supervisor
Paolo Massobrio — Head of the PhD Program

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Carlo Rizzardo
January 2024

Abstract

Learning-based approaches have brought great advances to robotics in recent years. Reinforcement Learning (RL) methods have shown to be capable of handling highly uncertain and complex tasks such as manipulation, locomotion, and visuomotor control, achieving extraordinary results. Furthermore these methods have brought forward the possibility of developing end-to-end methods that perform robot control directly from visual observations, removing the need for custom perception systems. The drawback of these methodology however is that such approaches often require vast amounts of training data. This scarce sample efficiency is a critical obstacle for the training of real-world robotics tasks, where collecting such massive amounts of data can be exceedingly expensive, if not impossible.

This thesis focuses on methods for reducing the overall data requirements of robotic visual policies, by employing sample efficient methods and performing sim-to-real transfer without introducing impractical computational needs.

Along the progress of this thesis, I construct a model-free architecture for learning visual tasks structured around a Soft Actor Critic agent and a learned model of the Partially Observable Markov Decision Process (POMDP) underlying the task.

At first, I concentrate on a simple implementation of this decoupled architecture, and show how such an architecture is more efficient than traditional fully end-to-end techniques. The learned POMDP model, based on a Variational formulation, learns to extract low dimensional representations from the input images, while biasing the representations toward containing information relevant to the task dynamics. I show that the learning of the feature extractor via an unsupervised learning objective improves sample efficiency in comparison to simply using the reinforcement learning reward signal, while the learning of task dynamics brings a beneficial effect on sample efficiency and asymptotic performance.

After this analysis I focus on the domain transfer capabilities of the method, to show its effectiveness for sim-to-real transfer. The decoupled nature of the method, with separate vision and RL modules, allows for independent transfer of policy and feature extractor. I show how domain transfer can be performed by only finetuning the vision section and

keeping the policy unchanged. The fundamental challenge in this approach is maintaining the latent representation expressed by the feature extractor compatible with the policy input.

I show how the presence of the dynamics model can act as a constraint to maintain the compatibility between policy and feature extractor by experimenting on a real and simulated table-top object pushing scenario. I then progressively explore more complex variations of this scenario and improve the architecture to support more complex tasks, and to relax the requirements it poses to achieve successful transfer.

In its final design the architecture demonstrates to be capable of performing sim-to-real transfer of the object-pushing task with remarkable efficiency. In the most simple case, it requires just a couple of hours of real-world experience, plus a couple of hours of training in simulation. Thus solving in four hours a task that would require multiple days to train directly in the real.

While the method has been evaluated on a simple experimental task, the architecture is not task-specific, and could be applied to vastly different problems. This work aims at building efficient architectures and defining effective and flexible sim-to-real transfer techniques. The availability of such techniques is crucial to the widespread diffusion of RL-based robotics, which has the potential of scaling to extremely complex tasks, advancing the practical capabilities of real-world robotic systems.

Table of contents

List of figures

List of tables

Part I

Introduction

Chapter 1

Introduction

1.1 Reinforcement Learning for Robotics

Traditionally, robotics systems have been structured around precise system models and perception pipelines designed for specific tasks. The different components of robotics systems have usually been kept separate and modular, allowing the separate development of each part and the focusing of engineering efforts on each component separately. Perception and motion control systems have usually been kept modularized, and manually adapted to specific tasks depending on the application needs. Following this framework, each of these components - vision, sensing, actuation, control - has to be precisely defined and tuned for the task at hand, for the environment in which the system is deployed. As tasks become more complex, and robotics systems become more general, precise construction of these modules grows more and more onerous, time-consuming and expensive, to the point of becoming impractical. For example, in the case of locomotion and manipulation tasks, proper interaction between robots and environment is essential, but at the same time the modeling of the complex contact dynamics involved in these tasks is highly uncertain and noisy. Properly handling tasks such as these requires considerable engineering work, and at the same time motion control and planning can become computationally expensive at runtime, when respecting the tight timings of online control is essential.

In this context, end-to-end reinforcement learning approaches have a strong appeal, as they completely bypass these problematics by learning models and perceptual systems from data. Model-free RL approaches do not require an explicit system model at all, as they directly infer control actions from state inputs. Model-based RL approaches still have an explicit model in their architecture, but automatically learn it from data, without the need for extensive manual design and tuning. In end-to-end RL methods, the perception pipeline

is also not manually designed for a specific task. Instead, it can be learned from data using highly generic architectures in a task-agnostic manner.

This mostly task-agnostic formulation of RL methods, together with the proven effectiveness of state-of-the-art methods such as Proximal Policy Optimization (Schulman et al., 2017) or Soft Actor-Critic (Haarnoja et al., 2018a,b), makes them a potentially groundbreaking tool for robotics. However, some issues have to be surpassed to make these methods readily applicable in real-world robotics.

RL methods usually require vast amounts of experience data to be trained. While acquiring such data in simulated environments can be relatively fast and inexpensive, collecting real-world data is costly. RL methods can require days, months, years, in extreme cases even thousands of years of experience data. Clearly, collecting these amount of data in the real world can be not only expensive, but potentially impossible. Also, during the first stages of training, reinforcement learning policies can behave in unexpected and unsafe ways, potentially damaging the environment or the robot itself.

This issues can be tackled with algorithmic and architectural improvements that increase sample efficiency in general, or by devising techniques to transfer to the real world models trained in simulation with large amounts of synthetic data.

In this thesis I explore these two directions while focusing on visual tasks, tasks in which the RL agent learns a policy for a control problem that operates directly from visual inputs. These tasks, while being particularly interesting for robotics applications, are also generally those for which RL methods require the most data.

1.2 Decoupling and Transferring RL Architectures

Standard RL algorithms such as DQN Mnih et al. (2015), PPO Schulman et al. (2017), or SAC (Haarnoja et al. (2018a), Haarnoja et al. (2018b)) have huge data requirements, especially for vision-based tasks. Such tasks have traditionally been solved by directly utilizing image observations in an end-to-end manner, the same way as tasks with low-dimensional observations are handled, directly learning to interpret visual observations from the reward signal. In this thesis I define a reinforcement learning architecture in which control policy training and visual feature extraction are decoupled. The feature extractor is learned as part of a full model of the environment based on a variational formulation capable of predicting observations and rewards. The control policy is trained as a Soft Actor-Critic agent that acts on the latent representation defined by the aforementioned model. This architecture is introduced in chapter 3 and then refined in chapter 5. This RL formulation is considerably

more sample efficient than standard algorithms, as the vision section of the network is trained from a representation learning signal, instead of just using the task reward, which may not convey much information to support visual understanding.

The decoupled nature of the architecture also brings advantages for the sim-to-real transfer of trained agents. The policy component and the perceptual component of the network are completely separate, and as such they can be transferred independently, as long as their shared state representation remains compatible. In this work I focus on the possibility of transferring the agent by only adapting the perception section of the architecture, in the assumption of performing transfer only across domains with limited differences in the task dynamics. This particular setup permits to completely avoid the use of RL in the real domain, consequently, the amount of real-world data is strongly reduced in comparison to the data required for training the task from scratch. Chapter 4 introduces one first strategy to perform this transfer while maintaining the compatibility between the components of the architecture, chapter 5 refines this methodology to improve performance and support a wider variety of tasks.

1.2.1 Experimental Evaluation

Across chapters 4 and 5, the domain transfer capabilities of the method were evaluated taking as an exemplary case a tabletop object pushing problem, in which a Franka-Emika Panda robotic arm is tasked with pushing a plastic cube to a predetermined destination. This task was chosen as it is fairly simple and manageable, but at the same time presents difficulties that make it a suitable ground for evaluating model-free reinforcement learning methods. Non-prehensile manipulation, and specifically object pushing, is a particularly challenging task for classic control methods due to the indeterminacy brought by friction forces, both between the manipulated object and the ground and between object and robot. Modeling such interactions precisely is challenging, identifying friction characteristics is a complex problem in itself and minute errors in the modeling have large impacts in the motion of the manipulated objects. Instead, model-free robot learning approaches such as the one proposed in this work handle these problematics implicitly without requiring careful explicit modeling of the system and can consequently solve this task effectively and reliably. To explore the transfer capabilities with different degrees of difficulty sim-to-sim experiments with different variations of the scenario were performed, starting from simple alterations to the colors of the scene and finishing with radical changes in the camera point of view. To ensure the

actual effectiveness of the method in a real-world scenario, sim-to-real experiments were also performed, physically implementing the scenario.

As will be shown first in chapter 4 and then in chapter 5, the method showed remarkable domain transfer capabilities, withstanding strong variations in the input observations, while requiring very limited amounts of real-world data and maintaining good performance. In its latest implementation the method is capable of learning a policy for an object pushing task defined with a non-shaped reward in just over two hours or real time, while collecting over three days of experience in simulation. Then, it can transfer the learned policy to the real in just two hours.

Chapter 2

State of the Art

Summary

This chapter will present an overview of the state of the art for reinforcement learning methods, in particular aiming at methods that tackle vision-based tasks and that are suitable for robotics applications.

In the first section I present an overview of the history of the field, starting from the earliest days of reinforcement learning, and arriving to mention some of the latest results of its application to robotics.

Then, sections 2.2 and 2.3 will present first the basic theory and "classic" reinforcement learning algorithms, and then the deep reinforcement learning methods that have been proposed in the latest years with the advent of deep learning methods.

Section 2.4 will then discuss sim-to-real , the transfer of learned policies form simulation to reality, and current methodology or tackling the problem.

Finally, section 2.5.1 briefly discusses decoupled and model-based RL methods, which train policies using learned representations and models of the environment, either by just exploiting representation learning methods for faster training of feature extractors or using learned dynamics models for planning and generating synthetic data.

2.1 A Brief History of Reinforcement Learning

Reinforcement learning has been achieving ever more impressive results in the past few years, becoming one of the most promising directions in the field of machine learning, and opening the doors to new solutions for unsolved robotics problems. It has shown striking capabilities

and proved capable to solve extremely complex tasks, both in simulated environments (Hafner et al., 2023; Silver et al., 2016; Vinyals et al., 2019) and in the real world (OpenAI et al., 2019; Rudin et al., 2021). Practical applications of reinforcement learning have only started becoming widespread in the last decade, as the capabilities of learning methods have scaled up and as new methods have shown to be capable of succeeding in areas where traditional methodologies fail, solving tasks that before could only be performed with human control. Such an explosion in results and applications has come thanks to new theoretical achievements and to the support of new computational hardware, but reinforcement learning itself has a long history.

The codification of the original idea itself of learning to interact with an environment by trial and error, which is the core idea of reinforcement learning, can be dated back to ethology and psychology works from the late 19th century from authors such as Alexander Bain, Conway Lloyd Morgan, Edward Thorndike (Sutton and Barto, 2018). The term 'reinforcement' itself, denoting the strengthening of a behaviour due to a positive stimulus, can be rooted back to the works of Pavlov in the 1920s. Thoughts of applying this logic to computer systems appear already in the works of Turing in 1948, when he describes a design of a "pleasure-pain system", which would memorize received positive and negative stimuli and subsequently use this knowledge to choose which actions to take.

The first foundational works proposing computational methods based on these concepts come from the 1950s, when *Dynamic Programming* was first proposed as a methodology for solving discrete stochastic optimal control problems. In 1957 Richard Bellman proposed Markov Decision Processes (MDPs) as a formalization for optimal control problems (Bellman, 1957) and introduced the *Value Iteration* algorithm as a way to derive policies to solve them. Soon after, in 1960, *Policy Iteration* was also proposed (Howard, 1960). These works and the subsequent developments in the following decades set the foundations for dynamic programming and the specific kind of optimal control that reinforcement learning focuses on. These works at the time were not regarded as learning methods, but could very well be considered as such in today's terminology.

A crucial development came with the introduction of *TD-Learning* in the 1980s in the works by Sutton, Barto and Anderson (Barto et al., 1983). The concept of TD-Learning, initially proposed with the TD(0) (Witten, 1977) and TD(λ) (Sutton, 1988) methods, revolves around the core idea of bootstrapping the learning of state-value functions for MDPs by using the current value estimate of temporally successive steps. This idea has been greatly successful and influential in Reinforcement Learning, as it supported the development of

more efficient methods and is one of the fundamental ingredients of modern value-based algorithms.

In 1989 Chris Watkins proposed Q-Learning (Watkins, 1989), combining dynamic programming and temporal-difference learning into an off-policy value-based algorithm. It differed from previous value-based approches in the fact that it did not try to estimate a state-value function $V(s)$, but as state-action-value function $Q(s,a)$, that would indicate the "value" of choosing a certain action when in a certain state. Due to its flexibility, its ability to use off-policy data and its general effectiveness, it became one of the most influential methods in reinforcement learning. In particular, its simple off-policy strategy made it possible to use data collected from any policy, separating the policy used for collecting data from the solving policy being optimized, it even supports the use of data collected by demonstrations, and it opened to the possibility of using replay buffers to store large amounts of experience data (Lin, 1992).

Up to this point in time the indisputably prevalent approach to solving reinforcement learning problems was value-based methods, in which some sort of state-action value function is learned to determine a solving policy. An alternative approach was introduced in 1992 with REINFORCE (Williams, 1992). The idea behind the method was to directly optimize an approximate policy via the gradient of the reward (Sutton et al., 1999). Methods of this kind are slower than value-based methods but they overcome some of their limits. They are very stable, and they can naturally learn stochastic and continuous policies, instead of only discrete deterministic ones.

A family of methods that soon caught traction, thanks to its capability to combine together the different advantages of value-based and policy-based approaches is that of *actor-critic* algorithms. Actor-critic methods have been originally conceived well before policy-gradient approaches, with works such as that of Barto et al. (1983). However, they really started to shine with the advent of powerful neural network function approximators and in combination with modern value-based methods such as Q-Learning (Degris et al., 2012).

Reinforcement Learning has been occasionally coupled with function approximation methods and neural networks since its early days, but effective policy learning techniques involving neural network function approximation started to be proposed only in the 1980s with the popularization of back-propagation by Rumelhart et al. (1985). An actor-critic method using neural network function approximation to balance and inverted pendulum was proposed by Barto et al. (1983), a gradient-based version of TD(0) was proposed by Sutton (1988), then REINFORCE in 1992 used a neural network to parametrize policies, and again in 1992 Tesauro proposed TD-Gammon (Tesauro et al., 1995), which was one of the first

methods demonstrating impressive human-level playing capabilities in a real human game. Around the same years Rummery and Niranjan (1994) proposed an approach that used neural network function approximators to learn Q-functions. In the subsequent years formalization for the use approximate methods were defined, introducing fitted value iteration and proofs of convergence bounds (Gordon, 1995, 1999, 2000).

In the early 2000s the advances in the methodology and theory of reinforcement learning together with the availability of progressively more powerful computing resources started to open new possibilities.

One of the most crucial theoretical advances in this period was the definition of Natural Gradient methods. First proposed by Kakade (2001), the natural policy gradient is an enhancement on the vanilla policy gradient of REINFORCE that better directs policy improvements leading to smoother and faster convergence. This development will be crucial in the definition of TRPO (Schulman et al., 2015) and then PPO (Schulman et al., 2017), one of today's most effective and used RL algorithms.

In these years numerous practical applications of Reinforcement Learning methods in robotics started to appear, showcasing the potential of these techniques, and demonstrating their effectiveness in solving tasks characterized by high modeling uncertainty and noise. An early notable work is that of Gullapalli et al. (1994) that solved peg-in-hole and ball-balancing tasks by using REINFORCE to train a neural-network-approximated stochastic policy.

An influential series of articles were those of Peters and Schaal. They used the Natural Actor Critic algorithm and solved progressively more complex tasks, starting from simple control problems and arriving to demonstrate a real baseball-swinging robot (Peters and Schaal, 2006, 2008; Peters et al., 2003, 2005).

Other notable works in the same years are those of Bagnell and Schneider (2001), in which a model-based RL approach is used to control an autonomous helicopter, that of Kohl and Stone (2004) that optimizes the parameters for a quadruped locomotion policy, that of Tedrake et al. (2005) that effectively learns a linear policy for a simple small bipedal robot, or that of Guenter et al. (2007), which employs natural actor critic to handle unmodeled variations and obstacles in an imitation learning pipeline. A continuation of the works of Peters was *PoWER* (Kober and Peters, 2008), which proposed to tackle complex exploration problems using a learned exploration policy. The method was demonstrated on a real-world pendulum swing-up task and, most notably, on a ball-in-cup task formulated with a challenging reward that was based just on the final position of the ball.

In these same year, a work that proved to be crucial for the history of reinforcement learning was proposed: Neural-Fitted Q-Learning (NQF), by Riedmiller (2005). It proposed

to learn a neural network approximation of the Q-function of Q-Learning, using a replay buffer to efficiently reuse off-policy data. These ideas, ten years later, will be some of the fundamental building blocks of DQN, one of the most influential and groundbreaking algorithms in reinforcement learning.

DQN (*Deep Q-Networks*), proposed by Mnih et al. (2013), was one of the first methods to show how reinforcement learning could reach human-level performance in a wide variety of tasks, even while using visual inputs instead of carefully crafted low-dimensional observations. It managed to reach human-level performance in 49 of the benchmarks of the *Arcade Learning Environment* (Bellemare et al., 2013), a collection of famous arcade games the likes of Breakout or Space Invaders, and did so using always the same architecture and hyperparameters (Mnih et al., 2015). The method combined together several crucial components: a neural-network approximation of the Q-value function trained via Q-Learning, an extensive replay buffer for storing vast amounts of off-policy experience, *Double Q-Learning* (Hasselt, 2010) to reduce instability, and a CNN feature extraction network. The results achieved on the Arcade Learning Environment were unprecedented for a completely end-to-end model-free method.

The most crucial limitation of DQN was that it could only handle small discrete action spaces. A method that first effectively overcame this limit is Deep Deterministic Policy Gradient (DDPG), proposed in the works of Silver and Lillicrap (Lillicrap et al., 2015; Silver et al., 2014). DDPG combined the innovations brought by DQN with a novel off-policy deterministic policy gradient approach, creating an efficient and completely off-policy actor-critic architecture capable of handling continuous action spaces. DDPG showed to be capable of solving complex control tasks, such as controlling a simulated gripper via torque, running with a bidimensional "cheetah" simulation, or driving in a racing simulator, all both only using visual input or only vector information. DDPG however was still a very brittle method, strongly sensitive to hyperparameters, unstable, and not particularly effective at exploring complex tasks. These limitations are being progressively surpassed by new methodologies, one of todays most effective methods, Soft Actor-Critic (SAC) (Haarnoja et al., 2018a,b) proposes to overcome this instability and improve the agent's exploration of the environment by using a stochastic off-policy actor-critic method based on a maximum entropy formulation. Because of it's generality, sample efficiency, stability and effective exploration characteristics, it is one of the most effective and used reinforcement learning methods of today. Section 2.3.3 will cover more in detail this method. Other works that built upon DDPG are TD3 (Fujimoto et al., 2018) and DP4G (Barth-Maron et al., 2018).

In these same year, parallel to the progress done in the off-policy realm by DDPG and then SAC, important advances were also being done with on-policy methodologies. Trust Region Policy Optimization (TRPO) was proposed by Schulman et al. (2015), by taking forward the ideas of Natural Actor Critic with new formalizations, more generality, and strongly improved capabilities and performance. Then, the ideas from TRPO were then taken forward to create a more practical algorithm, Proximal Policy Optimization (PPO) (Schulman et al., 2017). When compared with off-policy methods such as Soft Actor Critic, PPO generally provides more stability, but, due to its on-policy nature, this comes at the expense of greater data requirements, especially in tasks that require more articulated exploration. Because of these characteristic on-policy algorithms are often employed by exploiting highly parallelized simulation, like was originally done by A3C (Mnih et al., 2016), and was then recently demonstrated with great effectiveness on a real robotic task by Rudin et al. (2021), which managed to train a quadruped locomotion policy capable of navigating real-world complex and articulated terrain.

Up to now our discussion focused on model-free approaches, in which the algorithm does not try to create a model of the task it is trying solve, but instead directly defines a policy that from a state observations chooses which action to take. An alternative to this approach is the family of model-based methods, in which a model of the system dynamics is learned from data and used to support the policy training. A learned dynamics model can be used for example to generate synthetic data, or to plan trajectories.

One early work proposing a general formalization for model-based reinforcement learning is DYNA (Sutton (1991). It proposed the training of a dynamics model from data collected online via supervised learning, it hypothesized to use such a model for training a Q-learning policy with synthetic data, but also considered its use for planning.

These initial ideas were progressively brought forward in numerous work, refining the methods for the estimation of the dynamics model, and the techniques for planning or policy improvement. Early examples of the use of a learned model are the already mentioned work of Bagnell and Schneider (2001), in which a simple learned model is used to train a small control policy capable of controlling an autonomous helicopter, or the method proposed by Abbeel et al. (2006), that devises a training procedure for model optimization and uses policy gradient to train a policy from both real and generated data together. Later, Deisenroth and Rasmussen (2011) showed how the use of a stochastic learned model allows to train in just tens of seconds effective policies for tasks such a double-pendulum swing-up or the riding of a unicycle.

The growth of deep learning methods and general computational capabilities opened new possibilities also for model-based techniques, allowing the approximation of more complex models and the use of visual inputs. Wahlström et al. (2015) demonstrated the use of autoencoder architectures to learn latent low-dimensional dynamics models just from visual observations, and employed Model Predictive Control (MPC) to effectively solve simple control tasks such as balancing an inverted pendulum. The use of complex learned neural-network dynamics model was instead investigated by Chua et al. (2018), demonstrating how the use of simple feedforward networks can be coupled with MPC to solve task with accuracy on par to model-free methods, while requiring considerably less experience data. This methodology was then brought forward by Nagabandi et al. (2019), that showed how such methods can efficiently solve high-dimensional robotics tasks, displaying a striking demonstration of a 24 degrees-of-freedom hand rotating two balls around each other in its palm. The combination of model-based reinforcement learning and visual tasks was instead brought forward by the development of PlaNet (Hafner et al., 2019) and then Dreamer Hafner et al. (2020, 2021, 2023). PlaNet proposed an architecture for dynamics modeling integrating deterministic and stochastic components, combining it with MPC to perform planning. Dreamer integrated the PlaNet architecture into a model-free approach, managing to solve increasingly more complex tasks, up to controlling an agent within the free-world sandboxing videogame Minecraft.

Throughout its history reinforcement learning methods have been applied to more and more complex problems. From the application to well-defined and constrained environments such as tabletop games with backgammon with TD-Gammon, methods progressively matured to extremely complex, noisy and uncertain tasks. Especially in the last decade the advances in reinforcement learning techniques translated in more and more impressive results in the field of robotics. Most of the latest works we discussed (DDPG, SAC, TRPO, PPO) included in their justification benchmark results on control tasks, like those included in OpenAI Gym (Brockman et al., 2016) or the DeepMind Control Suite (Tassa et al., 2018), achieving progressively higher sample efficiency, stability and accuracy on problems ranging from simple control to whole-body humanoid 3D locomotion, both from vector observation and directly from images.

At the same time real world practical results evolved from control demonstration like those of the early papers of Peters, Schaal and Kober in the early 2000s to progressively more articulated and autonomous policies. An exemplary work is that of Levine et al. (2016), which used a Guided Policy Search algorithm to train an end-to-end policy capable of controlling a robot arm via torques directly from visual input, solving real-world tasks such as hang a

coat hanger, use a hammer or screw a bottle cap. The core novel highlight of this work was showing how end-to-end visual RL policies are an effective way to solve real-world robotics problems, a change of paradigm with respect to most previous works.

The growth of computational capabilities also allowed the implementation of ever more complex architectures and training methods. The availability of extreme amounts of simulated data permitted the development of methods such as that of Akkaya et al. (2019), in which a 24 degrees-of-freedom hand is trained to manipulate a Rubik's cube using up to 13 thousand years of simulated experience, achieving a remarkable resilience to perturbations, occlusions and physical variations.

The results of recent years showed how reinforcement learning methods have started to mature enough to be capable of solving real-world applications, for example works on locomotion such as those of Rudin et al. (2022, 2021), Jin et al. (2023) and Hoeller et al. (2023) showed how effective and resilient locomotion policies capable of navigating challenging terrains can quickly be trained exploiting modern hardware.

2.2 Reinforcement Learning

2.2.1 Markov Decision Processes

The core fundamental formalization for Reinforcement Learning revolves around the concept of Markov Decision Processes (MDPs). A Markov Decision Process is an extremely simple but general and effective formulation for a system in which an *agent* interacts with an *environment*. On an intuitive level, a Markov Decision Process represents possible situations within the environment as *states*, from each state the agent can choose an *action* to perform, depending on this action the agent will move to a subsequent state according to the system dynamics, and will receive a reward. Such a formalization is simple, intuitive, but extremely flexible. Depending on the choice of possible states, actions and dynamics, it can adapt to represent very simple tasks or the most complex systems.

In a more formal manner, an MDP defines the interaction of an agent with an environment as a temporally discrete succession of states $S_t \in \mathcal{S}$ that are visited by the agent at times $t = 1, 2, 3, \ldots$. At each timestep the agent performs an action $a_t \in \mathcal{A}$, and moves to state S_{t+1} according to the stochastic environment dynamics $p(s_{t+1}|s_t, a_t)$. Once the agent moves to the new state, it is informed of the state S_{t+1} and receives a reward r_t, according to a probability

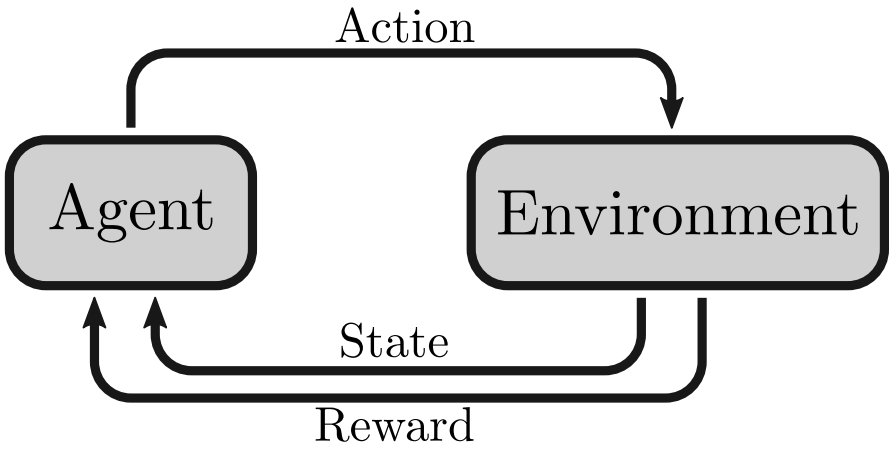


Figure 2.1 Agent-environment interaction in a Markov Decision Process.

density $p(r_t|s_t,a_t)$ [1]. Overall, the 4-tuple $(\mathcal{S},\mathcal{A},p,r)$ composed of the state space $\mathcal{S}$, the action space $\mathcal{A}$, the state dynamics and the reward density, defines the Markov decision process for a task.

In this formulation, the objective of reinforcement learning algorithms is to find how the agent can choose the best actions, so to maximize the total reward it collects. Not just in the current steps, but in all future steps. The method by which the agent selects actions is referred to as a *policy*.

Formally, we define a (stochastic) policy as the probability density $\pi(a_t|s_t)$ which assigns the probabilities of selecting actions a_t, conditioned on the current state s_t. A Reinforcement Learning method will have the objective of finding an optimal policy π^* that maximizes the total return G_t over episodes:

$$G_t \doteq \sum_{i=t}^{T} R_t \tag{2.1}$$

Or, in the more general case of continuing tasks which do not have a defined duration T, the discounted return:

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k} \tag{2.2}$$

[1] In practice the reward is often defined directly as a deterministic function $r : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of s_t, s_{t+1} and a_t

Being the policy and the state dynamics stochastic, the objective is to optimize the *expected* return, for every starting state. We define the expected return for a policy, starting from state s_0, as:

$$E_\pi[G_t|s_0] \tag{2.3}$$

A concept that is crucial in Reinforcement Learning formulations is that of *value function*. In an MDP, the value function $v_\pi(s)$ for a policy π at a state s is the expected return when starting from state s, and following the policy π.

$$v_\pi(s) \doteq E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s\right] \tag{2.4}$$

The value function expresses the "goodness" of a policy when starting from a specific state, an its maximization for all possible states or at least for a set of possible starting states is, in general, the objective of Reinforcement Learning algorithms. A policy π' can be can be said to be better than a policy π if $v_{\pi'}(s) > v_\pi(s)$ for all states $s \in \mathcal{S}$, and it is always true that there exists a policy π^* that is better than all other policies. π^* is said to be the *optimal policy* for the task.

2.2.2 Value Iteration

The original work by Bellman (1957) that proposed the MDP formulation already introduced the *value iteration* algorithm for finding a policy capable of solving the MDP problem. The value iteration algorithm is a dynamic programming approach built on the Bellman optimality equation, which states an optimal value function v_* must satisfy the following:

$$\begin{aligned} &v_*(s) = \max_a E\left[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a\right] \\ &\text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \end{aligned} \tag{2.5}$$

A policy with value function v_* is an optimal policy, from any state it selects the action which leads to the best possible total return.

In the x tabular case, that is if state and action spaces are finite and discrete, and if the transition dynamics are known, the optimal value function v_* can easily be used to construct an optimal deterministic policy π_* by selecting the possible action according to the value function at each step.

Algorithm 1 Value Iteration

Inputs:
ε: Accuracy threshold.

Initialize V(s) arbitrarily for all $s \in$ S except for $V(terminal) = 0$
repeat
    $\Delta := 0$
    for $s \in \mathcal{S}$ **do**
        $v := V(s)$
        $V(s) := \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
        $\Delta := \max(\Delta, |v - V(s)|)$
until $\Delta < \varepsilon$
Output $\pi(s) = \text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

$$\pi_*(s) = \underset{a}{\text{argmax}} \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \tag{2.6}$$

Following this logic the problem of approximating the optimal policy reduces to the one of approximating the value function. A first solution to this problem is the *value iteration* algorithm, which was already proposed in the original work by Bellman (1957), which introduced the MDP formulation. The value iteration algorithm iteratively evaluates and updates a value function and its related policy. Algorithm 1 presents the value iteration algorithm.

Value iteration can effectively produce a policy for solving generic MDPs, and is the foundation of a large part of the most important RL methods, even today. However, it poses requirements that are often not satisfied in practice. It requires the problem to be tabular, meaning that action and state spaces are discrete and finite, and it requires knowledge of the transition dynamics to update the value function and compute policy outputs. The second requirement is the most critical one, as it prevents the practical solution even of simple tasks when transition dynamics are not known, and, in the case in which the system dynamics are known precisely, other methodologies such as optimal control are often a preferable approach. This requirement can be surpassed by using sampled data and Montecarlo techniques. Combining experience sampling and value iteration leads to TD-learning, another fundamental piece of RL theory, critical to the definition of today's most effective methods.

2.2.3 Temporal Difference Learning

The critical step in value iteration that introduces the requirement of knowing the state-action transition dynamics is the policy evaluation step:

$$\begin{aligned} V(s) &:= \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma V(s')] \\ &:= \max_a E_{s',r|s,a}[r+\gamma V(s')] \end{aligned} \tag{2.7}$$

To remove this requirement Temporal-Difference Learning combines Montecarlo sampling-based estimation with the bootstrapping of the value function update of value iteration, using the previous approximation to perform the update. One-step TD value estimation (TD(0)), can be performed by using sampled transitions (s,a,r,s') composed of state, action, reward and next state. In this case the TD update is formulated as follows:

$$V(s) := V(s) + \alpha \left[T + \gamma V(S') - V(S)\right] \tag{2.8}$$

Such a formulation does not require knowledge of the transition dynamics of the system, and can be used with simple (s,a,r,s') sampled tuples, not requiring the computation of episodes returns, which is impractical for long episodes and impossible for continual tasks. As such, the method can be applied to a wide class of tasks, allowing to learn policies just from data.

2.2.4 Q-Learning

Temporal Difference learning per se is only a value evaluation method, it only learns the value function, however its combination with a policy learning approach is extremely effective. Integrating TD-Learning with value iteration and approximating Q values instead of just the value function, gives rise to Q-Learning, one of the most effective and successful algorithms of reinforcement learning history.

The Q-Learning algorithm, proposed by Watkins (1989) presented in algorithm 2, was the first to combine the most crucial features that make RL such a general and effective approach:

- It is model-free, it does not require a model of the system to be constructed manually.
- It can be trained from data collected online, no prior data collection is needed.
- It is off-policy, meaning it can be trained on data collected by any policy.

Algorithm 2 Q-Learning

Initialize Q(s,a) arbitrarily for all $s \in$ S and $a \in$ A except for $Q(terminal, \cdot) = 0$
Let π_Q be a policy derived from the Q-function Q
for each episode **do**
    $s :=$ current state
    for each step in the episode **do**
        $a := \pi_Q(s)$
        Take action a and observe the reward r and the next state s'
        $Q(s,a) := Q(s,a) + \alpha\left[R + \gamma \max_a Q(s',a) - Q(s,a)\right]$
        $s := s'$
Output π_Q

- It does not require full episode trajectories at each training iteration.

These characteristics make it an extremely versatile method, as it can be applied directly on a task without significant prior engineering of the problem, and it is remarkably data-efficient. Furthermore, in the tabular case, it has been proven to converge with probability 1 to the optimal policy (Watkins, 1989).

The main limitation of Q-Learning however is that it only supports finite discrete action spaces, due to the maximization performed in the Q-function update, and it cannot scale to large or continuous state spaces while maintaining its proven tabular formulation. The first issue has been tackled throughout the literature by using policy gradient methods and combining them with value based approaches, creating the family of Actor-Critic RL algorithms. These approaches are briefly discussed in the next section. The problem of handling vast state-action space has instead being tackled by moving from exact tabular function representations to function approximation, eventually performed with deep neural networks, leading to Deep Reinforcement Learning techniques. This is discussed in section 2.3

2.2.5 Policy Gradient and Actor-Critic Methods

The value-based methods seen up to now have always been limited to discrete action spaces. This results from the maximization operation performed on the action space during policy inference. For example, in the case of Q-Learning, the maximization of the Q-function performed to select the best action is feasible for small discrete action spaces but it becomes progressively more computationally expensive as the action space grows.

Policy gradient methods offer a different approach to solving the MDP formulation. Instead of learning a value function to then derive a policy from it, they directly approximate the policy from experience data. This can be done by defining a performance metric for a policy, which can then be maximized to determine the appropriate policy parametrization. In practice such a metric can be defined to be the value function for the policy:

$$J(\theta) = v_{\pi_\theta}(s_0) \tag{2.9}$$

The theoretical backing for this family of algorithm comes form the policy gradient theorem (Sutton et al., 1999), which states that the gradient of this performance metric can be expressed without knowledge of the state dynamics as follows:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) \tag{2.10}$$

Where μ is the state distribution under the policy π_θ. This is a notable result, as it shows how the direction of the gradient of policy performance can be determined without knowledge of the system dynamics, in a model-free manner.

The first method of this kind to be proposed was REINFORCE (Williams, 1992), which performed stochastic gradient ascent of policy performance by approximating the gradient with data collected with the optimized policy π_θ. The outer sum of the policy gradient theorem can be seen as an expectation over states encountered by the policy, thus it can be approximated with on-policy collected data. The same can then be performed on the action summation, re-formulating its inner expression, obtaining the following gradient estimate:

$$\nabla J(\theta) = E_{\pi_\theta}[G_t \frac{\nabla \pi_\theta(a_t, s_t)}{\pi_\theta(a_t|s_t)}] \tag{2.11}$$

With G_t being the total episode return starting from time t, and a_t and s_t being the sampled state and action at time t.

Policy gradient methods such as REINFORCE tend to have a high variance and be fairly sample inefficient, due to their Montecarlo formulation. Also, the formulation relying on full episode returns makes them difficult to apply in practice. To surpass these limits the policy gradient formulation can be integrated with the temporal difference value-based methods discussed in the previous sections. This combination gives birth to the greatly successful family of Actor-Critic algorithms, which encompasses also extremely successful methods of today such as SAC, PPO and DDPG.

In its most simple formulation Actor-Critic changes the policy gradient definition of equation 2.11 to use a TD estimate in place of the episode return G_t. To do so a value function is learned simultaneously to the policy. The gradient estimate takes the following form:

$$\nabla J(\theta) = E_{\pi_\theta}\left[\left(r_{t+1} + \gamma\hat{v}_\phi(s_{t+1}) - \hat{v}_\phi(s_t)\right)\frac{\nabla\pi_\theta(a_t, s_t)}{\pi_\theta(a_t|s_t)}\right] \tag{2.12}$$

Where the s_t, a_t, s_{t+1} and r_t can be easily collected as on-policy experience data. The resulting algorithm is extremely flexible and general, being able to handle continuous action spaces, define stochastic policy, and learn online. Refined versions of it, such as Natural Actor-Critic (Kakade, 2001), have also shown to be extremely effective in practice. However, this formulation still suffers some limitations. One is that it is an on-policy method, consequently it can be fairly sample inefficient as data needs to be re-collected at each iteration. This limit can be mitigated by using importance sampling, but this solution can often result to be convoluted and impractical. The other limit is that tasks with large state and action spaces can still be problematic if highly complex function approximation methods are not used. We will see in the next section how these limits can be surpassed, while scaling up the function approximation techniques to use neural networks and deep learning techniques.

2.3 Deep Reinforcement Learning

In the 2000s and especially in the 2010s machine learning methods based on deep neural network architectures started to take the spotlight as the most effective methodology in a wide variety of field, from computer vision, to natural language processing and speech recognition. This success was due at the same time to more and more refined methodologies and the availability of exponentially more capable computing resources, which made supervised learning techniques more and more effective.

Starting from the mid 2010s this, together with continuing algorithmical advances, gave rise to more and more effective methods capable of handling more complex problems, in terms of observation complexity, action dimensionality and articulation of the credit-assignment problem. The family of the methods that combine reinforcement learning and deep learning has taken the name of *Deep Reinforcement Learning*.

2.3.1 DQN

The term "deep reinforcement learning" was coined and popularized by the work of Mnih et al. (2013), which introduced the Deep Q-Networks (DQN) method. DQN combined

several components used in reinforcement and supervised learning in the decade previous to its conception in a way that allowed it to display never-before seen capabilities. The theoretical core formulation is that of Q-Learning, DQN uses a variation the Q-Learning algorithm in which the value function is approximated via a deep convolutional network, capable of understanding visual input. This was combined with the use of an extensive replay buffer (Lin, 1992) to efficiently reuse off-policy data. A similar formulation had already been explored in Neural-Fitted Q-Learning (Riedmiller, 2005), but DQN brought this together with the use of stochastic gradient descent for the optimization of the value function, which made the training considerably more efficient, making it practical to enlarge the value function approximator capacity and tackle more complex problems. Furthermore, one last ingredient was the use of Double Q-Learning (Hasselt, 2010), which made the training stable and reliable.

The exceptionality of DQN came from the fact that it showed to be capable of solving complex tasks directly from visual input, without requiring extensive tuning of its hyperparameters. The two works in which the method was originally explored, Mnih et al. (2013, 2015), evaluated the method on the Atari Learning Environment (Bellemare et al., 2013) and showed how the method could reach and surpass human capabilities in playing the vast majority of the games, all without performing task-specific hyperparameter tuning.

2.3.2 DDPG

DQN showed how the combination of reinforcement learning approaches with deep neural architectures has a great potential and can solve complex tasks in and end-to-end fashion. However, DQN is a purely value-based method, it follows the Q-Learning algorithm formulation, and as such it is still limited to tasks characterized by small discrete action spaces, due to the maximization over actions present in its policy formulation and in the update rule of algorithm 2. This prevents its direct application to continuous-control tasks and the definition of proper stochastic policies, which in some cases can be the optimal solution.

The solution to this limitation is the use of actor-critic approaches, derived from the initial formulation described in section 2.2.5. Many approaches have been proposed to make actor-critic an effective approach that is stable, data-efficient and general.

One direction has been the use of on-policy data through progressively more stable and efficient methods, starting from the Natural Policy Gradient of Kakade (2001), then the more performant Trust Region Policy Optimization (TRPO) by Schulman et al. (2015) and finally the more efficient Proximal Policy Optimization (PPO) of Schulman et al. (2017). These

methodologies have shown to be extremely effective and stable. And when combined with deep neural architectures they have shown to be capable to support highly complex tasks and the use of raw high-dimensional observations. However, they are inherently on-policy methods and as such are naturally quite sample-inefficient. While off-policy data can be used via importance sampling, this can result cumbersome and complex in practice.

Another direction has been the formulation of inherently off-policy actor-critic methods. One first effective methodology has been Deterministic Policy Gradient, first introduced by Silver et al. (2014). Deterministic policy gradient introduced a formulation for the performance gradient of deterministic policies, which supports the use of off-policy data without the use of importance sampling, and consequently without requiring the use of the collector policy during gradient updates. This greatly simplified the implementation and use of actor-critic off-policy methods, making them applicable to a much wider range of problems.

The original policy gradient theorem was formulated on the initial hypothesis of optimizing the performance of trained policy over its own state distribution:

$$J(\pi_\theta) = v_{\pi_\theta}(s_0) = E_{s\sim\mu_{\pi_\theta}}[R(s,\pi_\theta)] \tag{2.13}$$

Where $R(s,\pi_\theta)$ is the (discounted) total episode reward from state s using policy π_θ and μ_{π_θ} is the policy state distribution.

The crucial result of policy gradient, as seen in equation 2.10, was that the performance gradient of a stochastic policy can be expressed without the use of the environment dynamics. In the case of continuous state-action spaces this can be formulated as follows:

$$\nabla J(\pi_\theta) = \int_{\mathcal{S}} \mu_{\pi_\theta}(s) \int_{\mathcal{A}} q_{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) da ds \tag{2.14}$$

The on-policy nature of policy gradient derives from this formulation, as the gradient depends on the on-policy state distribution μ_{π_θ}. Approaches such as that of Degris et al. (2012) proposed to deal with off-policy data by altering the performance objective into using the state distribution μ_β of the collector policy β in place of μ_{π_θ}.

$$J_\beta(\pi_\theta) = E_{s\sim\mu_\beta}[R(s,\pi_\theta)] \tag{2.15}$$

This formulation is effective in defining a policy gradient on off-policy data, but it results in the use of importance sampling for the approximation of the inner integral of equation 2.14. The deterministic policy gradient, instead, changes the initial assumption by using a deterministic policy, which completely removes the action integral in equation 2.14. The

resulting policy gradient does not depend on the behaviour policy, and can be approximated with samples from the behaviour policy experience:

$$\begin{aligned}\nabla J_\beta(\pi_\theta) &\simeq \int_{\mathcal{S}} \mu_\beta(s) \nabla_a q_{\pi_\theta}(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) ds \\ &= E_{s\sim\mu_\beta}\left[\nabla_a q_{\pi_\theta}(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)\right]\end{aligned} \tag{2.16}$$

The combination of this policy update with a Q-Learning based approximation for the Q-function q_{π_θ} results in an off-policy actor-critic method. This formulation was the first to introduce an off-policy policy gradient method that did not use importance sampling and was practical to use and implement. The subsequent work by Lillicrap et al. (2015) combined this formulation with deep neural network approximators into the Deep Deterministic Policy Gradient (DDPG) algorithm, and showed how the same concepts introduced by DQN, such as the use of vast replay buffers, could be applied to continuous control tasks. DDPG showed to be capable of solving simulated tasks involving simple control task, but also complex locomotion and manipulation problem, all this both while using low-dimensional vector information and by using raw image inputs.

2.3.3 SAC

DDPG showed how an off-policy actor critic method can scale to highly complex continuous control problems problems with great effectiveness, while using either low-dimensional vector observations of the system or directly from high-dimensional observations such as images. It proved how this can be done with considerable sample efficiency, by reusing collected data in an off-policy fashion, utilizing replay buffers in the same manner as it was done by DQN.

However, in practice DDPG showed to be highly sensible to its hyperparameters. This is a critical limitation, as the requirement of per-task meticulous hyperparameter tuning can become a major obstacle in the solution of problems that require large amounts of data and training time.

Also, the formulation of DDPG does not explicitly aim at guaranteeing good exploratory behaviour. In practice exploration with DDPG must be incentivized by introducing a randomization in the collector policy. This in practice often does not result in good exploration, and the randomization intensity is one more tricky hyperparameter to tune.

Soft Actor-Critic, proposed by Haarnoja et al. (2018a,b), tackles these issue by changing the problem formulation from the optimization of a deterministic policy that maximizes total

reward, to the optimization of a stochastic policy that maximizes an objective composed of a total reward term and a policy entropy term.

$$J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)\sim\mu_\pi} \left[r(s_t,a_t) + \alpha \mathcal{H} \left(\pi(\cdot|s_t) \right) \right] \tag{2.17}$$

Where $\mathcal{H}$ is the entropy and α is a temperature weight. This objective differs from the standard reinforcement learning objective, as it is not anymore just an optimization of policy return, the presence of the entropy term explicitly pushes toward having a policy that behaves as randomly as possible while solving the task.

While following this formulation, the Q-function can be optimized via a soft Bellman update:

$$Q(s_t,a_t) := r(s_t,a_t) + \gamma E_{s_{t+1}\sim p(\cdot|s_t,a_t), a\sim\pi(s_t)} \left[Q'(s_{t+1},a) - \alpha log \pi(a|s_{t+1}) \right] \tag{2.18}$$

Where Q' is a Q function delayed as in double Q-Learning.

Soft Actor-Critic the combines this soft Q-Learning update with a novel policy gradient, which directly uses the learned Q function to learn an approximate policy. This is done by optimizing the policy toward a distribution derived from the Q-function:

$$\begin{aligned} \pi_{new}(s_t|\cdot) :&= \underset{\pi\in\Pi}{\operatorname{argmin}} D_{KL} \left(\pi \middle|\middle| \frac{e^{\alpha^{-1} Q_{\pi_{old}}(s_t,\cdot)}}{Z_{\pi_{old}}(s_t)} \right) \\ &= \underset{\pi\in\Pi}{\operatorname{argmin}} \left(-E_{a\sim\pi} \left[Q_{\pi_{old}}(s_t,a) \right] - \mathcal{H} \left(\pi(s_t|\cdot) \right) \right) \end{aligned} \tag{2.19}$$

This objective simultaneously optimizes the policy for maximizing the Q-value and for having high entropy. The iterated application of the policy evaluation step of equation 2.18 and the policy improvement step of equation 2.19 was proven to converge to the optimal policy in the tabular case. Thanks to the maximum-entropy formulation and the policy gradient based on the minimization of the Q-value, Soft Actor-Critic manages to optimize a stochastic actor-critic policy completely from off-policy data, without requiring the use of importance sampling. This makes the algorithm extremely general, stable, and sample-efficient. Also, the use of stochastic policies allows the correct handling of problems in which non-deterministic policies may be the optimal solution. Furthermore, the entropy maximization is helpful for exploration, as the policy is encouraged to try out more varied solving strategies.

As in DQN and DDPG, Soft Actor-Critic uses neural network function approximators to handle large continuous state-action domains and high-dimensional observations. Same as it was for DQN and DDPG, the proven convergence guarantees are lost, but the experimental evaluation of the method showed the method to be extremely effective in practice. When compared to on-policy methods such as Proximal Policy Optimization, SAC results considerably more data-efficient, and in comparison to other off-policy algorithms, such as DDPG and TD3, it shows to be noticeably more stable. Also, the better exploration properties together with the enhanced stability allow the method to solve highly complex control tasks where previous off-policy algorithms failed, such as the control the simulated 21-degrees-of-freedom humanoid of Duan et al. (2016).

2.4 From Simulation to Reality

Despite the advances in sample efficiency made by foundational methods such as Soft Actor-Critic, learning effective real-world control policies via reinforcement learning remains challenging. Even simplified simulated control tasks such as those of OpenAI Gym (Brockman et al., 2016) or the DeepMind Control Suite (Tassa et al., 2018) still require millions of experience steps to be solved, equivalent to days of experience time. This when having access to the ground truth system state data, when using image observations these data requirements grow even more. Also, solving equivalent tasks in the real necessitates even greater lengths of time due to the increased complexity of the task, the unavoidable partial-observability of the system state, and the time spent in managing and reorganizing the experimental setup at each episode. Furthermore, in the first stages of learning, trained policies often exhibit unsafe behaviours which could damage the real-world robotic hardware and the environment surrounding it. For example, training quadruped or humanoid locomotion policies from scratch without some underlying safety mechanism, like it is done in simulated benchmarks, is infeasible in the real world.

For these reasons the use of simulation remains a invaluable tool in the training of robotic policies. Being able to deploy in the real an agent trained in simulation, with limited amounts of real-world training, would greatly reduce the impact of these issues, allowing for easier development and application of learning-based robotic systems. However, transferring learned models from simulation to reality, and, more in general, between domains, is non-trivial. Numerous methodologies have been proposed to tackle this problem, in the context of policy learning, in vision and in machine learning in general. The problem of transferring learned models from simulation to reality is generally referred to as "Sim-to-Real".

2.4.1 Sim-to-Real

Sim-to-real RL methods exploit simulation software to efficiently train policies in virtual reproductions of the real environment, and then transfer the learned policy to the real-world domain by overcoming the reality gap, the discrepancy between simulation and reality.

The advantage of using simulation is first of all the possibility of generating vast amounts of experience much more rapidly than it would be possible in the real world. This can be achieved by simulating faster than real-time and by parallelizing multiple simulated environments. Gorila Nair et al. (2015), A2C and A3C Mnih et al. (2016) showed how parallelizing experience collection leads to substantial improvements in training time. More recently, Rudin et al. (2021) exploited modern hardware and simulation software to massively parallelize an environment for quadruped locomotion training, achieving in just 20 minutes a PPO gait policy capable of successfully controlling a real robot on complex terrains, further developments on this have been shown in the works of Rudin et al. (2022), Jin et al. (2023) and Hoeller et al. (2023).

Beyond just generating huge amounts of data, simulation software can also support training strategies that would be impossible in the real world. Pinto et al. (2018) shows how it is possible to speed-up training considerably by using simulator state knowledge during training, and transferred a policy trained in such a way to the real world, where this knowledge is unavailable.

The core issue with simulation training is the reality gap, the discrepancy between the characteristics of the simulated environment and those of the real one. These differences can be in the dynamics of the environment, due to inaccuracies in the physics simulation, in the observations the agent makes, due to imprecisions in the visual rendering or in the sensory input in general, or simply in the behavior of robotic components, which may be implemented differently in simulation and reality. Advances in realistic simulation software (Makoviychuk et al., 2021; NVIDIA, 2020; Unity, 2020)) are progressively narrowing the reality gap, but performing sim-to-real transfer remains complex as constructing simulations that closely match the real world remains a challenging task that requires considerable engineering work.

The problem of sim-to-real transfer can be generalized to the concept of domain transfer, where a method is trained in a source domain and then applied, and potentially adapted, to a target domain. In the case of sim-to-real the source domain is a simulation and the target is the real world, but the same concepts can be applied to sim-to-sim or real-to-real transfers in which some characteristics are altered between source and target scenarios. This terminology

can be applied in the same way to any learning method, being it reinforcement learning, supervised learning or representation learning.

Numerous strategies have been proposed to perform domain transfer and overcome the transfer gap. In general, we can distinguish between two families of techniques: those that aim at obtaining a policy capable of operating in both the real and the simulation without using real-world data, and those that use real-world data for adapting a model learned in simulation to the real domain. We refer to these latter ones as *domain adaptation* methods.

2.4.2 Domain Adaptation

Domain adaptation tackles the domain transfer problem by adapting a source-trained method to the target domain by using data collected in the target domain.

The most simple domain adaptation approach is to just perform policy finetuning in the real, the same way it is often done in supervised learning settings. The policy is first trained in simulation, then the agent is transferred to the real and the training continues in the until satisfactory performance is achieved. This strategy is simple, intuitive and can be very effective, however, it often still requires considerable amounts of real-world experience, and it is not guaranteed the robot will behave properly and safely when first transferred to the real domain. Also, if proper precautions are not taken, there is no guarantee that the transferred models will be able to maintain the useful knowledge acquired in simulation and only correct for the domain differences. If this knowledge is not maintained, the advantage of performing sim-to-real is lost, as the training essentially restarts from scratch. In general, the interpretation of different environment characteristics performed by a neural network model is not represented in the neural network weights in a modular and interpretable fashion, and it may not be possible to adapt each characteristic individually.

Other methods explicitly target the domain gap problem by matching the output of feature extractors between the simulated domain and the real domain, creating feature extractors that are invariant to the switch between simulated and real-world inputs. This can be achieved via different approaches. Some methods try to train feature extractors for the two domains while keeping the distributions of the two resulting feature representations similar, with losses based on distribution distance metrics such as Maximum Mean Discrepancy (MMD) (Tzeng et al., 2014), MK-MMD (Long et al., 2015) or others (Sun and Saenko, 2016). Others try to keep the feature representations of samples from the two domains close via Adversarial approaches. A discriminator network is trained to classify feature vectors between the two domains, the feature extractor is then optimized to generate indistinguishable representations

(Ganin and Lempitsky, 2015; Tzeng et al., 2015, 2017). Alternatively, other techniques take inspiration from style transfer methods and directly convert target-domain samples into source-domain samples or samples from a third "canonical" domain (Bousmalis et al., 2018, 2017; Hoffman et al., 2018; James et al., 2019). Other methods attempt to identify corresponding samples from source and target domain and then force the representations of these corresponding samples to be similar. Gupta et al. (2017) does so by assuming samples from corresponding timesteps in RL episodes should be similar, Tzeng et al. (2016) first identifies weakly paired samples and improves on this with an adversarial approach.

Even if some of these methods work well for vision tasks, they generally do not apply effectively to policy learning, in particular in the case of difficult exploration problems. The aforementioned approaches either require target data to be available while performing the original source domain training or they train the encoder with offline data. This is problematic, as in difficult exploration problems collecting fully representative data before completely training the policy may be impractical or impossible. In simple tasks it may be possible to collect human-generated demonstrations, but in more complex tasks, for example locomotion, collecting demonstrations is not trivial.

2.5 Domain Randomization

A sim-to-real approach that does not require target-domain data is *Domain Randomization*. The core idea of the method is to randomize visual (Tobin et al., 2017) and physical (Peng et al., 2017) characteristics of the simulated environment, so that once the agent is transferred in the real world it can interpret the new domain as just another random variation. This is approach can be seen as enlarging the source domain distribution so to include the target domain in it, effectively reducing the size of the domain gap. A survey on domain randomization approaches for robotics is provided by Muratore et al. (2022).

In the context of policy learning, these methods can be applied to both visual and state-based tasks, and have been extremely successful, being able to solve even extremely complex visuomotor control problems while maintaining strong robustness to occulusions and perturbations. The idea of randomizing simulator parameters to improve policy robustness in robotics tasks has been present in research for a while, Wang et al. (2010) already proposed to use randomization for improving walking controllers more than a decade ago. The combination and popularization of this approach with deep neural architectures for robotic control can be traced back to the work of Tobin et al. (2017), which was the first method to demonstrate how randomization of visual characteristics could be extremely effective

in performing zero-shot transfer of robotics systems from simulation to reality. Peng et al. (2017) instead applied the approach to the training of model-free reinforcement learning policies. Then, one of the most impressive results achieved with domain randomization was the work of OpenAI et al. (2019), which trained a policy for in-hand manipulation of a Rubik cube, managing to achieve unprecedented manipulation ability. This last work displayed how this methodology, if scaled to use colossal amounts of data (13 thousand years of simulated experience) can achieve extraordinary results.

As exemplified by this last example, as tasks get more complex, they require progressively greater amounts of simulation data, long training times, and vast computational capabilities. To reduce these issues, various methods have been proposed to constrain the amount of randomization to just what is necessary. Ramos et al. (2019), Possas et al. (2020) and Muratore et al. (2021) achieve this by identifying simulator parameter distributions via Likelihood-free Inference. Heiden et al. (2021) instead shows how it is possible to use differentiable simulators to identify possible simulator parameters from real data.

2.5.1 Decoupled RL Architectures and Model-based RL

As we previously discussed, the possibility of directly using high-dimensional sensor observations as inputs to RL agents is extremely valuable for robotics applications, where the system state is often inferred from a combination of image and vector readings. However, solving tasks of this kind with standard reinforcement learning algorithms generally requires particularly great amounts of experience data and long trainings.

When dealing with image observations, the reinforcement learning methods we discussed up to now have traditionally been used in a monolithic manner, in which the agent directly uses images as input, in the same manner as it would do if it had direct access to the true system state. This formulation is derived from the use of an MDP model, where the agent has direct access to the state of the system. This assumption is generally not satisfied in robotics, where the system is better characterizes as a Partially Observable Markov Decision Process (POMDP), where the agent only has access to environment observations, which may not contain enough information to determine the exact system state.

To tackle the sample efficiency problem on vision tasks, various methods have been proposed that employ a POMDP formulation and learn an RL policy separately from a perceptual model that maps observations to state estimates. This perceptual model can be learned by employing representation learning approaches, that use unsupervised methodologies based for example on reconstruction or contrastive losses.

One example of these kind of approaches is SAC+AE, by Yarats et al. (2021), which uses an autoencoder architecture to learn a low-dimensional latent space from images, and simultaneously uses the resulting latent vector as inputs for a Soft Actor-Critic policy.

A similar but more articulated architecture is Stochastic Latent Actor Critic (SLAC), by Lee et al. (2020). Here a POMDP model of the system is learned with a variational formulation, including models for image encoding, decoding and for dynamics prediction in the latent space. The state of the system is determined using the current image observation and previous latent representations. A SAC policy is trained concurrently to the model using the learned state representation.

A work that follows a different direction for the learning state representations is CURL, of Laskin et al. (2020), which instead uses a contrastive loss to learn a latent space for the image observations.

Beyond just learning a representation of the input observations some architectures showed how it is possible to learn a full model of the system, modeling the perceptual channel and the environment dynamics. A learned model of the dynamics can be used for generating synthetic data, which can be used to support the training of the RL policy, or it can be used by the agent to perform planning. Examples of methods that use the dynamics model for planning are PDDM (Nagabandi et al., 2019) and PlaNet Hafner et al. (2019).

The first, PDDM, learns the dynamics model as a simple architecture composed of an ensemble of fully-connected MLP networks. These neural dynamics models are then used with an non-linear MPC controller, which predicts future trajectories with the learned model and uses a reward-weighted cross-entropy method (CEM) optimizer to select actions. The method is shown to be quite effective, being able to learn complex simulated manipulation tasks like rotating a valve with a three-fingered gripper, reorienting objects while holding them with a 24-degrees-of-freedom hand, handwriting with a robotic hand, and, noticeably, controlling a real-world 24-dof hand to rotate two balls around each other in its palm. All of this with very remarkable sample efficiency, just 3 hours of training for the real-world task.

While being very effective, PDDM operates on low-dimensional vector observations, and consequently relies on custom perception systems when applied in the real. PlaNet (Hafner et al., 2019) is instead a learned planning-based end-to-end method, it uses directly image observations to perform planning. This is done by learning a multi-step variational model of the system that combines a stochastic and deterministic dynamics formulation, and includes observation encoders for inferring latent representations from images and decoders for reconstructing images from latent vectors. This learned mode is then used with an MPC planner based on the cross-entropy method. The method was evaluated on manipulation

and locomotion tasks from the DeepMind control suite Tassa et al. (2018), and proved to be stable, precise and sample efficient.

The work on the system modeling of PlaNet was then taken forward in Dreamer (Hafner et al., 2020, 2021, 2023). Dreamer uses the dynamics model of PlaNet to interpret image observations and trains an Actor-Critic RL policy with latent trajectories generated with the learned dynamics model, exploiting at the same time the differentiable learned dynamics model to backpropagate analytic gradients from the trajectories into the RL policy. This formulation allows the agent to be trained with huge amounts of simulated data, while maintaining the high generality of model-free methods. Across its several iterations Dreamer showed to be extremely effective on a wide variety of tasks, from the control tasks of the DeepMind control suite, to Atari games, up to a complex 3D game like Minecraft.

Part II

Training and Transferring Decoupled RL Architectures for Robotics

Chapter 3

Accelerating RL by Modeling Task Dynamics

Summary

As we discussed in the previous chapters, vision-based Reinforcement Learning methods have a great potential in robotics. They can be applied directly to robotics tasks and solve them end-to-end, dealing with everything from perception to control. As such, they remove the need for the development of custom perception systems, precisely targeted to the task at hand. They also overcome the need for precise modeling of the system. Model-free methods forgo the use of a system model altogether, while model-based methods learn the model from data.

However, to effectively apply vision-based Reinforcement Learning techniques to robotics it is crucial to have a sample efficient architecture, that does not require vast amounts of real-world data to be trained. It is clearly impossible to collect thousand of years of experience data, but it is also impractical to collect even just days of data. Even if it is possible to collect days of data it is impractical to do so repeatedly, every time the tasks conditions change in an unexpected manner, the task requirements change, or, in the case of research, when the methodology itself is being experimented upon.

This chapter focuses on the starting point of our work, the development of an efficient architecture for learning vision-based robotics tasks. We will see how I focused on a decoupled architecture, using an unsupervised learning objective to support the training of the vision section of the agent, and a Soft Actor-Critic policy to learn a control policy. We will see how the inclusion of a dynamics model in our architecture enhances its asymptotic performance and sample efficiency.

3.1 Decoupling Feature Extraction

Traditional Reinforcement Learning methods usually tackle vision-based tasks in an end-to-end manner, they assume to have direct access to the state of the system at the current time-step, as in a standard MDP formulation. From this assumption they can optimize approximate value functions $v(s_t)$ or polices $\pi(a_t|s_t)$ that are directly conditioned on the state.

However, in real-world settings, such an assumption is usually never satisfied. This is particularly true in robotics, where the state of the system can generally only be inferred from indirect, noisy and partial sensory information. In a robotics scenario, the state of the system could be composed of joint positions and velocities, object positions and orientations, or even the state of complex or deformable objects. This state may only be observable via motor encoders, cameras, torques and tactile sensors, giving to the agent a flow of information that requires to be processed to be properly utilized.

Such a system can be modeled as a *Partially Observable Markov Decision Process* (POMDP). In contrast to a MDP, a POMDP assumes that the agent may only observe the state of the system s_t through partial observations. Observations may have a different format and structure than the underlying states $s_t \in \mathcal{S}$, as such, we define a separate *observation space* containing all the possible observations, such that $o_t \in \mathcal{O}$ for all observations o_t. The relation between states and observation is given by the density $o(o_t|s_s) = P[O = o_t|S = s]$, which defines the agent's perceptual channel, which introduces noise and potentially loses information. This formulation can be formalized defining the POMDP as a 7-tuple $(S, A, p, r, \gamma, O, v)$. The first five terms represent respectively the state space, the action space, the state transition probability density $p(s_{t+1}|s_t, a_t)$, the reward function $r(s_t, a_t)$ and the discount factor. The last two terms represent the observation space and observation density $o(o_t|s_t) = P[o_t = o|s_t = s]$ which defines our sensory channel. This system formalization is represented in figure 3.1.

In the POMDP formulation, when RL methods like DQN, DDPG, SAC or PPO are employed end-to-end they learn a policy $\pi(o)$, that directly selects actions based on observations. This policy is learned only through the reward signal, which usually does not directly convey information pertaining to visual understanding.

To improve the sample efficiency of such architectures we can separate the policy optimization from the training of a feature extractor, dedicated to inferring the current state from observations. Such a feature extractor can be trained efficiently via an unsupervised representation learning objective.

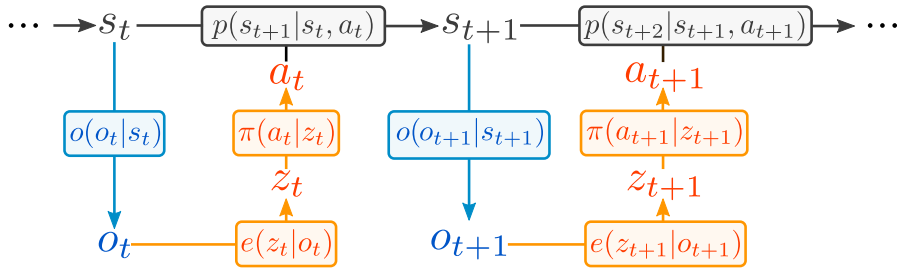


Figure 3.1 POMDP formulation. In orange the agent, in blue the sensory channel, in black the underlying Markov Decision Process.

In our implementation, we opted for the utilization of a *Variational Autoencoder* (VAE) architecture. We made this choice due to several reasons, including the VAE's ability to ensure the smoothness of the latent space, its capability to constrain the latent space distribution to approximate normality, its well-established effectiveness in prior research, and its seamless integration into the POMDP formulation.

We define our VAE architecture as a stochastic encoder $e_\theta(z_t|o_t)$ that maps observations $o_t \in O$ to latent representations $z_t \in \mathcal{Z}$ and a deterministic decoder $d_\theta(z_t)$ that performs the opposite transformation. Following the POMDP formulation we can see the variational autoencoder effectively models both directions of the perceptual channel. The posterior represents the feature extractor $e_\theta(z_t|o_t)$. The decoder instead is an approximation of the state-to-observation perceptual channel $o(o_t|s_t)$. The z_t produced by the encoder is a representation of the state of our system, we refer to this as a *latent representation*, and the set of its possible values $\mathcal{Z}$ as a *latent space*. We can choose the shape of this latent representation by tuning the structure of encoder and decoder, the most practical choice is to make it a low-dimensional vector, having $\mathcal{Z} = \mathbb{R}^k$, and choosing the latent size k depending on the degrees of freedom of the system.

Encoder and decoder are both parametrized as neural networks, their specific architecture being defined consequently of the kind of observations given by the task. In our experimental setups, in the most general case, observations are mixture of visual and vector observations, consequently encoder and decoder are composed of a vector section and an image section. Vector encoding and decoding is performed with simple MLPs, with hidden LeakyReLU activations and output activation defined based on the task at hand. Image encoding and decoding has instead been done in most of our experiments with simple convolutional networks and resize-convolution layers (Odena et al., 2016). In all our implementations the stochastic encoder has been defined to produce a parametrization of a multivariate diagonal

normal distribution. In practical terms this means the encoder produces in output the mean and the standard deviation of the distribution, $\mu_\theta(o_t)$ and $\sigma_\theta(o_t)$. Within the VAE architecture these values are than sampled to generate stochastic outputs, employing the reparametrization trick. Once the architecture is set, the variational autoencoder can be trained using the standard variational evidence-lower-bound loss:

$$\begin{aligned} \mathcal{L}_{VAE}(\theta; o_t) &= D_{KL}(e_\theta(z_t|o_t)||\mathcal{N}(\mathbf{0}, I_k)) + \alpha MSE(\hat{o}_t; o_t) \\ \text{With} z_t &\sim e_\theta(o_t) \end{aligned} \tag{3.1}$$

Once the autoencoder is successfully trained, any RL method can be used to optimize a policy $\pi(a_t|z_t)$ which uses latent vectors as inputs, in practice providing to the agent preprocessed low-dimensional observations.

3.2 A Fully Decoupled Agent

With this formulation, any RL algorithm can be used to train a control policy from the latent representations. In this thesis I will focus on the use of Soft Actor-Critic (SAC). This choice was made as SAC presents a series of characteristics that are particularly suited for visual tasks and for potential real-world deployment.

SAC is an off-policy stochastic actor-critic approach. Its off-policy nature enables the reuse of data collected during the training, resulting in a remarkable sample efficiency. This

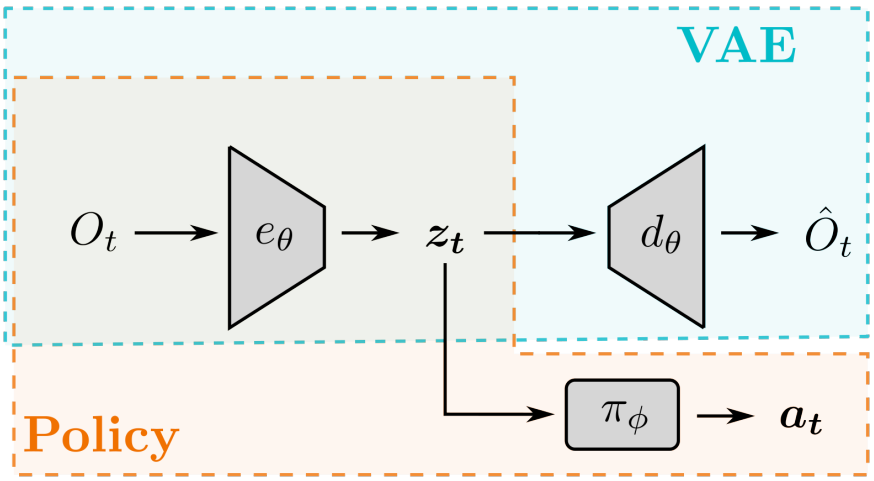


Figure 3.2 VAE-SAC architecture.

proves particularly valuable for complex and articulated visual robotic tasks, that would otherwise demand extensive amounts of experience data. The actor-critic nature of the method, instead, is essential for supporting continuous action spaces, a critical requirement for robotic control application. The incorporation of stochasticity in the policy is equally crucial, as it enables the maximum entropy formulation of the algorithm, which in turns brings stability and enhances the method's efficacy in exploring complex tasks. The formulation of Soft Actor-Critic was discussed in detail in section 2.3.3.

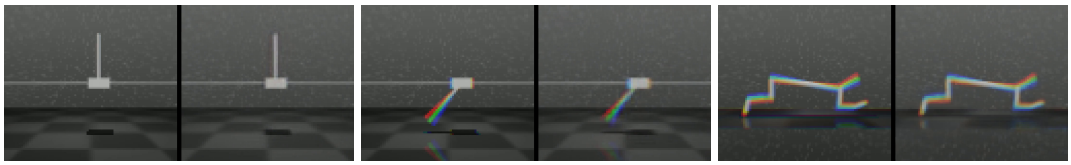
The use of different RL architectures would bring different advantages and disadvantages. One method that is often used in robotics applications is *Proximal Policy Optimization* (PPO). It is known for its stability and its proven capability to solve complex tasks reliably, however, it is an on-policy method, as such it cannot reuse data from previous training iterations and requires the collection vast amounts of experience. Many works (e.g. Rudin et al. (2021)) overcome this limit by employing highly parallelized simulation to collect new data after every policy optimization step. While this is a valid strategy, it is clearly impractical to follow in the real, and becomes more and more unsustainable as task complexity increases.

Algorithm 3 VAE-SAC training

Inputs:
N: number of training episodes
K_{VAE}: number of VAE gradient steps per episode
K_{SAC}: number of SAC training steps per episode

Let π_ϕ be a Soft Actor-Critic policy parameterized by ϕ
Let θ be the parameters for the VAE encoder e_θ and decoder d_θ
Randomly initialize θ and ϕ
for N episodes **do**
    Collect one episode into a dataset $\mathcal{D}$ according to policy π_ϕ
    for K_{VAE} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of observations o_t
        $g = \nabla \mathcal{L}_{VAE}(\theta; \mathcal{B})$
        $\theta = ADAM(\theta, g)$
    for K_{SAC} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of tuples (o_t, a_t, r_t, o_{t+1})
        $\mathcal{B}_e = \{(e_\theta(o_t), a_t, r_t, e_\theta(o_{t+1}))$ for all (o_t, a_t, r_t, o_{t+1}) in $\mathcal{B}\}$
        $\phi = SAC\_UPDATE(\phi, \mathcal{B}_e)$

Figure 3.3 Training procedure for VAE-SAC. VAE weights are updated with ADAM Kingma and Ba (2015) and the policy is trained with SAC_UPDATE as defined in Haarnoja et al. (2018b) as discussed in 2.3.3.



(a) Cartpole - Balance (b) Cartpole - Swingup (c) Cheetah - Run

Figure 3.4 Tasks from the DeepMind Control Suite. For each task is shown an example frame and its reconstruction through the VAE-SAC architecture. The observations include frame-stacked images. The red, green, and blue channels of the image are three temporally subsequent images of the scene, the movement of the links can be inferred from their relative displacement in the frames. It is possible to see how the pole in the balance task is not moving, while in the swingup task it is moving at a high velocity.

We refer to the combination of a Variational Autoencoder and a Soft Actor-Critic policy as VAE-SAC. We train the whole architecture with data collected online, as the policy training proceeds and the agent progressively explores the environment. Both parts of the architecture can be trained on off-policy data, consequently we can use a single experience replay buffer to sample data for the training of both policy and latent extractor. To maintain a strict separation between feature extraction and policy modules, we train the respective networks completely separately, without propagating gradients from the policy into the encoder. Figure 3 presents the pseudocode for the VAE-SAC training procedure.

By itself this simple architecture is already capable of solving challenging control tasks, such as those in the *DeepMind Control Suite*. Figure 3.4 shows some examples of such control tasks, with the observation reconstruction produced by the autoencoder architecture. While such a reconstruction is not used by the policy, it is a useful way to qualitatively understand the representational capability of the architecture. Effective reconstruction of the input observations is indicative of an effective latent representation. Quantitative results are presented in section 3.4.

3.3 Predicting the Dynamics

The VAE-SAC architecture that we defined up to now uses the variational autoencoder to learn an approximation of the state-observation mapping of the task's POMDP. However, the learned latent representation $z_t = e_\theta(o_t)$ is only optimized to contain the information necessary to reconstruct the observation o_t, in the form of $\hat{o}_t = d_\theta(z_t)$. In general, a reconstruction error loss on the observation may not be a strong learning signal for learning features relevant to control.

Some important features, even if inferable from the observations, may not be strongly reflected in the reconstruction loss value. An example of this can be seen in highly dynamic tasks in which the observations are temporally stacked images. A situation that is common in benchmark control tasks such as those of the DeepMind control suite, where it is common practice to use as observations images composed of three or more subsequent frames, so to include information concerning the speed of moving bodies in the image. From a reconstruction perspective, precisely reconstructing the difference between the three frames is not particularly critical. However, it is of fundamental importance for a control policy to know the velocity of the objects within the image.

To overcome this limit, the architecture can be extended to model also the dynamics of the system. By modeling the state-observation mapping, the dynamics and the reward function we can effectively train a model of the whole POMDP system, a world model of our environment. By including the dynamics and reward functions in the architecture the learned latent representation is driven toward including all the features necessary to perform dynamics prediction and reward computation. As these are the features that determine the task evolution and objective, they are also features valuable as control policy inputs.

To model the task dynamics the architecture was modified to include a deterministic dynamics predictor $f_\theta(z_t, a_t)$ in the variational autoencoder bottleneck, and the decoder, instead of producing observations o_t was made to predict observations o_{t+1}. With this simple changes the architecture performs a one-step observation prediction instead of simple observation reconstruction. An approximation of the reward function was also included in the

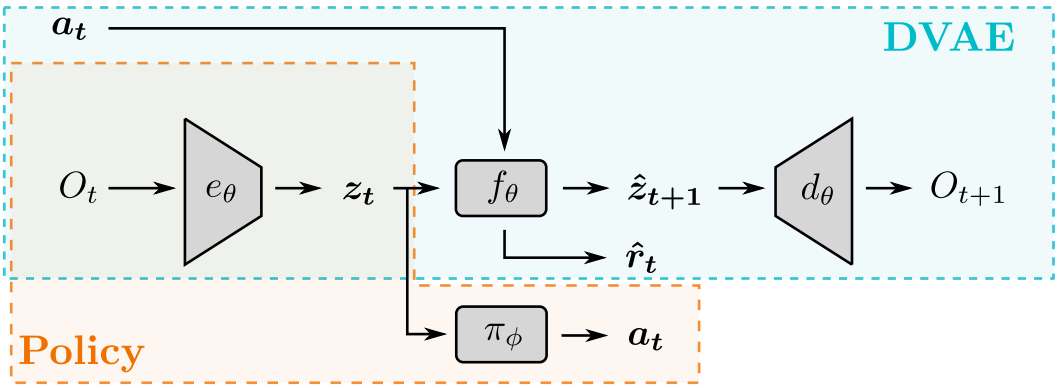


Figure 3.5 DVAE-SAC architecture. In blue the DVAE representation learner. In orange the end-to-end policy composed of the DVAE encoder and the policy network, in our case implemented with a SAC agent. For graphical simplicity the reward network of the DVAE is omitted and incorporated in the dynamics predictor f_θ.

architecture, in the form of a learned function $r(z_t, a_t, z_{t+1})$. We will refer to this architecture as DVAE (Dynamics-VAE).

The architecture is trained with a variational loss, similarly to a standard variational autoencoder, but reconstructing o_{t+1} and r_t instead of just o_t, and introducing an action input for the dynamics prediction. Consequently, the architecture is trained with batches of tuples (o_t, a_t, o_{t+1}, r_t) instead of batches of observations o_t.

$$\begin{aligned}&\mathcal{L}_{DVAE}(\theta; o_t, a_t, o_{t+1}, r_t) = D_{KL}(e_\theta(z_t|o_t)||\mathcal{N}(\mathbf{0}, I_k)) + \alpha MSE(\hat{o}_{t+1}, o_{t+1}) + \beta MSE(\hat{r}_t, r_t)\\ &\quad \text{With: } \hat{o}_t = d_\theta(f_\theta(z_t, a_t)))\\ &\qquad\quad z_t \sim e_\theta(o_t)\\ &\qquad\quad \alpha \text{ and } \beta \text{ being hyperparameters}\end{aligned} \tag{3.2}$$

The dynamics predictor was implemented as a residual MLP network, $f_\theta(z_t) = f'_\theta(z_t) + z_t$. The weight of the f_θ network were initialized to have an initial output close to zero, so to have the dynamics initialized to be close to the identity function, a reasonable initialization for the dynamics. The reward network was also parametrized as an MLP.

The training procedure is almost identical to that of VAE-SAC, presented in the previous section. Only the latent extractor loss is changed, together with the sampling of its training batches. Figure 4 presents the updated training procedure.

In practice, the evaluation of the architecture through standard benchmarks showed that the inclusion of the dynamics and reward prediction has a beneficial effect both on sample efficiency and asymptotic policy performance.

In the next section we present a comparison on standard benchmark environments of the two approaches, with the inclusion of the dynamics predictor and without.

3.4 Experimental Results

Here is presented an experimental analysis on the impact of the inclusion of the dynamics prediction in the DVAE-SAC architecture. The method was evaluated on selected benchmark environments provided by the *DeepMind Control Suite* (Tassa et al. (2018)).

In figure 3.7 it is possible to observe the performance of the method on the cartpole-balance, cartpole-swingup and cheetah-run tasks. These tasks were selected as they span different difficulties among the tasks of the DeepMind Control Suite.

Algorithm 4 DVAE-SAC Training

Inputs:
N: number of training episodes
K_{DVAE}: number of DVAE gradient steps per episode
K_{SAC}: number of SAC training steps per episode

Let π_ϕ be a Soft Actor-Critic policy parameterized by ϕ
Let θ be the parameters for the DVAE encoder e_θ, decoder d_θ, and dynamics predictor f_θ
Randomly initialize θ and ϕ
for N episodes **do**
    Collect one episode into a dataset $\mathcal{D}$ according to policy π_ϕ
    for K_{DVAE} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of tuples (o_t, a_t, o_{t+1}, r_t)
        $g = \nabla \mathcal{L}_{DVAE}(\theta; \mathcal{B})$
        $\theta = ADAM(\theta, g)$
    for K_{SAC} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of tuples (o_t, a_t, r_t, o_{t+1})
        $\mathcal{B}_e = \{(e_\theta(o_t), a_t, r_t, e_\theta(o_{t+1}))$ for all (o_t, a_t, r_t, o_{t+1}) in $\mathcal{B}\}$
        $\phi = SAC\_UPDATE(\phi, \mathcal{B}_e)$

Figure 3.6 Training procedure for DVAE-SAC. VAE weights are updated with ADAM Kingma and Ba (2015) and the policy is trained with SAC_UPDATE as defined in Haarnoja et al. (2018b).

The cartpole-balance task is the most simple of the three. In this task the agent is tasked with balancing an inverted pendulum attached to a cart that can move left and right on a rail. The scene is observed via 84x84 pixel images, 3 subsequent images are provided to the agent. We can see in figure 3.7a how the inclusion of the dynamics prediction has an important impact on the performance of the agent in this task. The VAE-SAC architecture, which does not include the dynamics prediction, completely fails at solving the task, while the DVAE-SAC architecture succeeds at balancing the pole although with some instability. For reference, a reward of 1000 corresponds to the pole being upright for the whole duration of the episode, with zero total reward instead the pole is always hanging below the cart.

In figure 3.7b instead we can see the results for the swingup task. This task presents the same scenario as the cartpole-balance task, but in the initial configuration the pole is not in the upright position. From the figure we can see that the DVAE-SAC method manages to achieve a slightly higher reward than VAE-SAC, however it does not succeed in balancing the pole. Qualitatively observing how DVAE-SAC behaved it was possible to see that it learns to

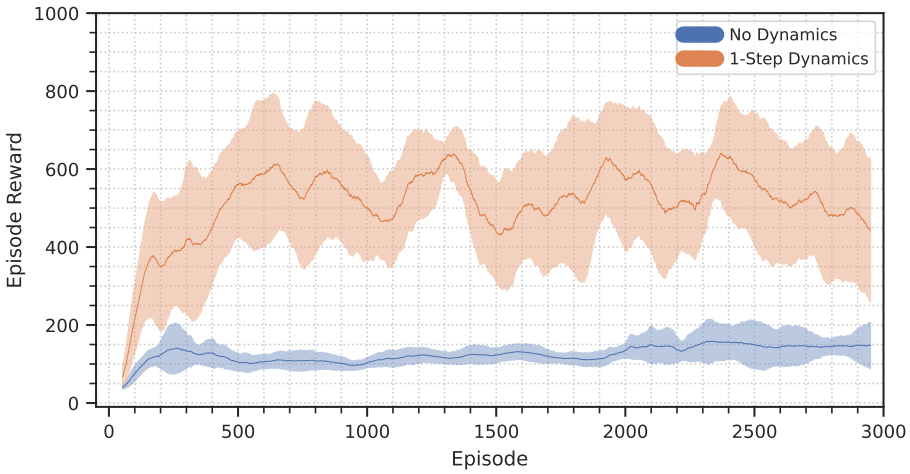
rotate the pole around in circles instead of balancing it upright. We can explain this failure of the method from the fact that for successfully swinging up the pole and then stopping it in the correct position it is necessary to estimate the bodies' velocities with good precision.

Finally, figure 3.7c presents the performance data for the cheetah-run task. In this task the agent must learn to control a planar cheetah robot so that it runs forward. The agent controls the torques on the 6 actuated joints of the robot. Again, the scene is observed via 84x84 pixel images, and the 3-frame frame stacking is used. This task is the most complex of the three, as it presents a higher state and control dimensionality, and the dynamics of the task are particularly complex due to the contact dynamics between the cheetah and the ground. However, both VAE-SAC and DVAE-SAC learn a policy capable of making the robot run forward, although not at high velocity. The presence of the dynamics again shows to bring improved performance. In figure 3.8 it is possible to observe a frame sequence obtained from a training with DVAE-SAC, it is possible to see how the agent successfully runs forward and how the image predictions correctly match the future frames.

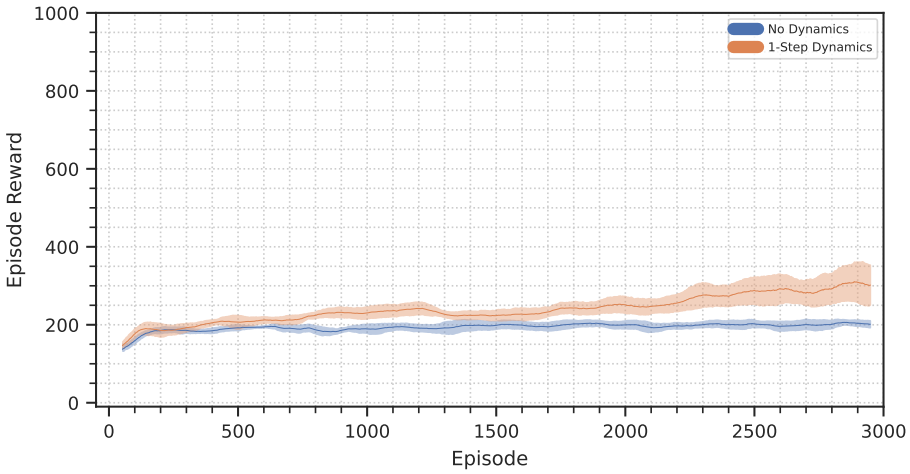
3.5 Conclusions

This chapter showed how a simple decoupled architecture can be constructed to efficiently learn vision-based tasks with reinforcement learning. Simply learning a state representation from images using a variational autoencoder can already be an effective method, but it was shown how the inclusion of a dynamics predictor in the architecture can considerably improve performance. The evaluation on DeepMind Control Suite tasks showed how the inclusion of the dynamics in general improves performance, and can enable the solution of tasks that the simpler architecture without dynamics does not manage to resolve.

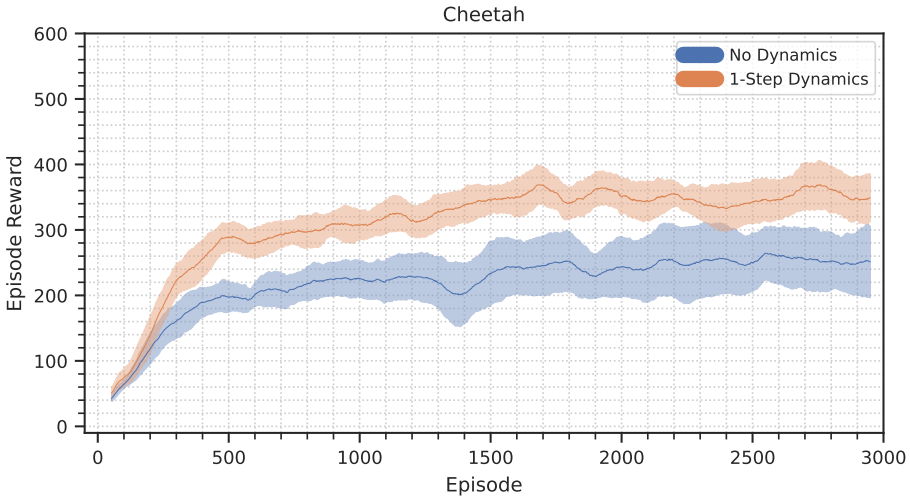
Still, the architecture cannot solve tasks that require a good precision in identifying dynamical features, as can be concluded from the failure at solving the cartpole-swingup task. However the intuition that the inclusion of a dynamics predictor can improve policy performance proved to be correct. In chapter 5 I will present an extension to this architecture that goes forward on this intuition and performs multi-step observation prediction, further improving performance, and successfully solving the swingup task.



(a) *Cartpole - Balance* task



(b) *Cartpole - Swingup* task



(c) *Cheetah - Run* task

Figure 3.7 Comparison of VAE-SAC and DVAE-SAC performance on the DeepMind Control Suite cheetah_run, cartpole_balance_sparse, and cartpole_swingup tasks.

Figure 3.8 Frames from an episode of the cheetah_run task. At each timestep we show the input observation and the predicted next observation.

Chapter 4

Sim-to-Real: Domain Transfer via Latent Prediction

Summary

The DVAE-SAC architecture developed in the previous chapter showed to be capable of solving complex vision-based control tasks and effectively improving sample efficiency when compared to standard end-to-end methods such as SAC or PPO. However, for practical robotics applications, the amount of experience required for training such policies can still be an insurmountable obstacle. Furthermore, the training of such control agents in the real world is impractical, due to the unsafe behaviour of untrained policies in the early stages of learning.

One potential solution to both sample efficiency issues and training safety is the use of simulated environments, followed by a transfer of the trained policy to the real world domain. However, the discrepancy in visual and physical characteristics between reality and simulation, namely the sim-to-real gap, often significantly reduces the real-world performance of policies trained within a simulator. This chapter proposes a sim-to-real technique that makes use of the decoupled nature of DVAE-SAC to independently perform the transfer of feature extractor and control policy. The presence of the dynamics model in the DVAE architecture will prove to be crucial to the effectiveness of the transfer, as it can be exploited as a constraint on the latent representation when finetuning the feature extractor on real-world data.

We show how the DVAE-SAC architecture can support the transfer of a trained agent from simulation to reality without retraining or finetuning the control policy, but using real-world data only for adapting the feature extractor section of the network. By avoiding the training

of the control policy in the target domain we overcome the need to apply Reinforcement Learning on real-world data, instead, we only focus on the unsupervised training of the feature extractor, considerably reducing real-world experience collection requirements. The method will be evaluated on sim-to-sim and sim-to-real transfer of a policy for table-top robotic object pushing. We demonstrate how the method is capable of adapting to considerable variations in the task observations, such as changes in point-of-view, colors, and lighting, all while substantially reducing the training time with respect to policies trained directly in the real.

4.1 Efficiently Transferring RL Policies

As we discussed in section 2.4, sim-to-real is a essential methodology in robotics. The development of robotics systems often happens initially in simulation for a variety of practical reasons, such as cost, safety and flexibility in exploring alternative approaches. In the setting of robot learning the use of simulation also opens the door to the use of vast amounts of data that would be impossible to collect in the real world. Even when using sample-efficient methods such as the one developed in chapter 3, as the task complexity increases the amount of data required to train effective policies can become prohibitive. This is particularly true for applications that require the reinforcement learning agent to explore a vast state space, to learn to handle all the possible variations in a task, and to explore areas of the state space that may be accessible only after considerable policy training. An example of this is the tabletop object-pushing task that will be analyzed in toward the end of this chapter. Despite its relative simplicity, such a task already presents perception and exploration complexities that make the task impractical to train in reality, as solving it with an architecture such as DVAE-SAC can require more than a day of experience data.

Section 2.4 discussed several approaches for sim-to-real and domain transfer, each of them with different advantages and disadvantages. Some adaptation works that aim at making feature extractors invariable to domain changes require target domain data during source domain training (e.g. Gupta et al. (2017)), some even requiring paired observations between source and target domain observations (Isola et al. (2017)). Other methods which tackle the unpaired case may not have good characteristics for online training of complex tasks. Some, such as for example the work of Tzeng et al. (2016), assume the availability of an observation task that properly covers the observation space of the task, an assumption that may not be verified for complex exploration problems. Other works (e.g. Gupta et al. (2017)) make strong assumptions on the structure of the task to determine parings between corresponding

source-target observations. Others, like the approach of Bousmalis et al. (2018), assume the variations between real and simulated observations to be limited to simple low-level alterations. Most of these issues are surpassed by domain randomization approaches, however domain randomization generally brings and increase in data requirements, and selecting appropriate randomizations may not be trivial for all tasks.

In this chapter I aim at performing the transfer of vision-based control policies without knowledge of the target domain during source training, without using source data during target-domain adaptation, while being sample efficient in simulation and reality, and supporting complex tasks and difficult exploration problems. The combination of these characteristics is highly desirable in practical robotic applications, as the method can maintain a high generality while supporting complex use cases, not requiring long training times, not demanding the collection of big amounts of data, and not necessitating for manual data annotation.

4.2 Transferring Decoupled Policies

In the solution of the most complex reinforcement learning problems the need for vast amounts of data arises from the exploration challenges of the problem. Tasks that requires the exploration of articulated system dynamics will require great amounts of data for the agent to discover which states offer the highest reward and which state-trajectories allow to reach them. This is a complexity that is not present generally in supervised learning or representation learning, where the whole data distribution is usually assumed to be easily accessible in the training set.

The use of fully decoupled reinforcement learning methods such as the DVAE-SAC method developed in the previous chapter separates the policy learning from the learning of state representations that map observations to latent state. In the setting of sim-to-real this property can be exploited to transfer the agent's latent extractor and its control policy independently of each other. Assuming the same behaviours learned in simulation can be applied in reality, it is then possible to directly transfer of the complex policy behaviours learned through reinforcement learning, only adapting the mapping between observations and latent states and greatly reducing real-world data requirements.

4.2.1 Problem formulation

Following this intuition, the objective is to transfer the learned state representation, learning a new mapping between the new observation space and the latent states. As in chapter 3, we formalize our setting as a *Partially Observable Markov Decision Process* (POMDP), defined by the 7-tuple $(S, A, p, r, \gamma, O, v)$. With the 7 terms representing the state space, the action space, the state transition probability density, the reward function, the discount factor, the observation space and the observation density. The system is represented in figure 4.1.

The overall objective of the agent is to learn a policy $\pi(a_t|o_t)$ that maximizes the expected discounted total sum of rewards $R(\pi) = E_{s_{0:T}}(\sum_{t=0}^{T} \gamma^t r(s_t, a_t))$. However, in the sim-to-real setting we assume the agent to be already optimized for the source domain. We want to use this preexisting knowledge to solve the target domain with a high sample efficiency.

In the DVAE-SAC from the previous chapter, the agent is subdivided in four components:

- The **control policy** π_ϕ, which selects actions based on observed latent states z_t
- The **encoder network** e_θ, which extracts latent states z_t from the input observations o_t
- The **latent dynamics predictor** f_θ, which predicts latent states z_{t+1} and rewards r_t from latent states z_t and actions a_t
- The **decoder network** d_θ, which can reconstruct predicted observations $\hat{o}_{t+1}$ from latent states z_{t+1}

The overall end-to-end policy is composed of the combination of the encoder network and the policy, which can be chained to obtain the stochastic policy $\pi_\phi(a_t|e_\theta(z_t|o_t))$, effectively using the encoder as a feature extractor. However, of the four components of the agent, only the control policy π_ϕ is trained via reinforcement learning, the encoder, decoder, and dynamics predictor are trained via the variational representation learning objective. This decoupling of the architecture completely separates the action-selection logic of the policy from the observation understanding, as no gradients from the RL objective are propagated into the encoder network.

From the sim-to-real and domain transfer perspective this allows to conceptually separate the transfer of the observation representation from that of the control policy. Tackling these two aspects of the transfer separately can be particularly advantageous in practice. Depending on the task, the size of these two domain gaps can be very different. At the extreme, the same identical task can be represented with different observations, by using different sensors, changing environmental conditions such as the lighting or for example changing the position

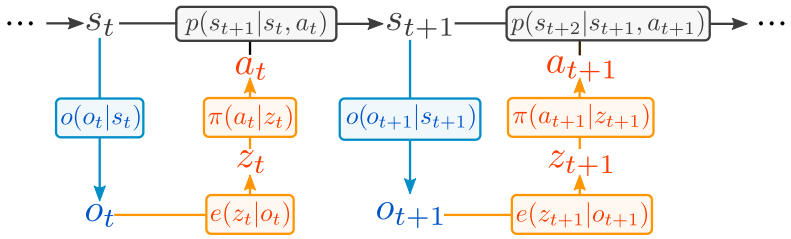


Figure 4.1 POMDP formulation of the problem. In orange the policy we implement, in blue the sensory channel, in black the underlying Markov Decision Process.

of cameras. In the opposite way, different tasks could be observed via the same perceptual system.

The method proposed in this chapter focuses on the former case, in which we assume the transfer gap in the task dynamics to be negligible. Such an assumption can be limiting in the case of highly dynamical tasks, but can still be satisfied for a wide range of transfer scenarios. Sim-to-sim and real-to-real can satisfy this assumption exactly, as the underlying task is indeed the same. But also, non highly-dynamical tasks can present a small gap in the dynamics, as the physical properties of the system become less relevant to the environment evolution. An example of such a problem, which we will tackle later in the chapter, is an object pushing task, in the case for example of high frictional damping and a position controlled robot. Also, this approach does not exclude the use of domain randomization, which can still be used on the physical properties of the task to reduce the effective size of the transfer gap.

4.3 Decoupled Transfer of DVAE-SAC

Using the DVAE-SAC architecture we can transfer policy and feature extractor through domains independently of each other. In case the transfer gap on task dynamics is small, it is possible to only adapt the DVAE observation model, and keep the policy as-is. However, this cannot be done simply be retraining or finetuning the DVAE, as it is crucial to maintain the compatibility between policy and encoder network. The RL policy within the agent is trained to understand as input the latent representation learned by the DVAE, if this representation was to change, a fixed control policy would not be able to adapt, it would fail to correctly interpret its input and ultimately become unable to solve the task. Consequently, when finetuning the observation model and using a fixed policy it is crucial to avoid alterations in the representation, preventing changes to the state-to-latent-state mapping.

In this chapter we tackle this representation drift problem by using the dynamics predictor of the DVAE architecture as a constraint. When we transfer the control policy as-is, we are assuming the dynamics of the task are not significantly changing between the source and target domain. This assumption also allows us to also transfer the dynamics predictor network f_θ of the DVAE without adaptation.

More than being just transferred without adaptation, the source-domain dynamics predictor can be used as a constraint to reduce the representational drift. By transferring the DVAE architecture to the target domain and finetuning only the encoder and decoder sections of the network while keeping the dynamics predictor frozen, we can obtain a latent representation for the target domain that is compatible with the dynamics predictor of the source domain. The following sections will show experimentally how constraining the latent representation to be compatible with the source dynamics can be enough to maintain compatibility with the control policy, and effectively solve the task in the target domain.

Algorithm 5 DVAE-SAC Transfer

Inputs:
N: number of training episodes
K_{DVAE}: number of DVAE gradient steps per episode

Let π_ϕ be a Soft Actor-Critic policy parameterized by ϕ
Let θ_e, θ_d, θ_f be respectively the parameters for the DVAE encoder e_{θ_e}, decoder d_{θ_d}, and dynamics predictor f_{θ_f}
Initialize θ_e, θ_d, θ_f and ϕ with the pretrained source-domain network weights
for N episodes **do**
    Collect one episode into a dataset $\mathcal{D}$ according to policy π_ϕ
    for K_{DVAE} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of tuples (o_t, a_t, o_{t+1}, r_t)
        $g = \nabla \mathcal{L}_{DVAE}(\{\theta_e, \theta_d\}; \mathcal{B})$
        $\theta = ADAM(\{\theta_e, \theta_d\}, g)$

Figure 4.2 Domain transfer procedure for DVAE-SAC. DVAE encoder and decoder weights are updated with ADAM (Kingma and Ba (2015)).

4.4 Evaluating the Transfer

To demonstrate the method's effectiveness, its performance was evaluated on sim-to-sim and sim-to-real scenarios. Assessing the method requires selecting a task that poses challenging

Observation space:	$[0,1]^{128\times128\times3} \times [-1,1]^2$
Action space:	$[-1,1]^2$

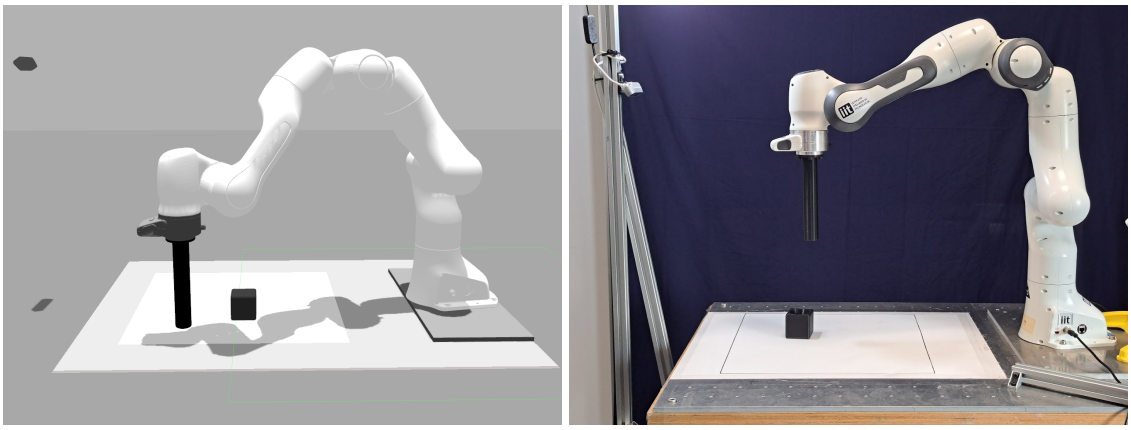
Table 4.1 Object pushing environment observation and action spaces. The action space is normalized to $[-1,1]$, but corresponds to a displacement of maximum 2.5cm in x and y.

transfer aspects yet remains manageable, is suited for a real-world construction, and allows for repeated execution with minimal human intervention. Also, to evaluate the transfer performance of the method we need to have clearly identifiable characteristics of the task, that can be altered in simulation or in the real.

Following these requirements the experimentation focused on a robotic table-top object pushing task. This task, despite being relatively simple, already presents complex aspects from the point of view of policy learning and control. Depending on the reward function formulation, it can present difficult exploration challenges. And if solved with traditional control approaches, it would require contact and friction modeling, which is often non-trivial, together with the development of a visual perception system for identifying the pose of the pushed object. At the same time, by using a position-controlled setup, it can be easily constructed in the real, without the risk of robot collisions and always maintaining the manipulated object in a bounded area.

The specific scenario we focus on is the one of a 7-DOF Franka Emika Panda robotic arm tasked with pushing a 6cm cube to a target position. The robot arm is controlled in cartesian space, the end-effector moves only horizontally within a 45cm square workspace located in front of the robot itself. Each episode is initialized with a random cube position and a random end-effector position. The target cube position is kept constant across episodes. The agent controls the robot specifying a displacement in the bidimensional workspace of the end-effector, resulting in a continuous 2D action space. The environment is observed through a camera placed on the opposite side of the table with respect to the robot arm, which produces RGB images with 128x128 pixels resolution. In addition to the images the agent also has access to proprioceptive information from the robot, in the form of the 2D position of the end-effector tip. Figure 4.3 shows the simulated and real scenarios, figure 4.5 displays an example of a successful episode.

Each episode lasts 40 steps. Once the cube reaches the target position, within a 5cm tolerance, the episode is considered successful, but it is not interrupted until the 40 steps timeout is reached.



(a) Simulated scenario (b) Real-world scenario

Figure 4.3 Simulated (4.3a) and real (4.3b) object pushing setups. The camera used to collect the input images is visible in the top left in both pictures.

To speed-up the learning and ease experimentation the reward was shaped to give a strong and directed learning signal. The reward function was defined as the composition of 3 terms, one to encourage the end-effector tip to stay close to the cube, one for the cube to stay close to the goal, one for the cube to be moved in any direction. We define them as follows, where $r(p_c, p_t)$ is the overall reward, $r_c(p_c, p_g)$ is the cube-goal term, $r_t(p_t, p_c)$ the tip-cube term, $r_d(p_c, p'_c)$ the cube displacement term, and $r_b(p_c, p_g)$ is a further bonus given when the cube is within d meters from the target. The r_c and r_t functions are defined as linear ramps, with value 0 at 40cm from the target and respectively 100 and 50 at the target.

$$r(p_c, p_t) = 0.1 \left(r_c(p_c, p_t) + r_b(p_c, p_g) + r_t(p_t, p_c) + r_d(p_c, p'_c) \right)$$

With:

$$\begin{aligned} r_c(p_c, p_g) &= 100 \ \frac{0.4 - ||p_c - p_g||}{0.4} \\ r_t(p_t, p_c) &= 50 \ \ \frac{0.4 - ||p_c - p_t||}{0.4} \\ r_d(p_c, p'_c) &= 2000||p_c - p'_c|| \\ r_b(p_c, p_g) &= \begin{cases} 200^{\frac{d - ||p_c - p_g||}{d}} & \text{if } ||p_c - p_g|| \leq d \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4.1}$$

This experimental scenario was implemented both in the real world and in a *Gazebo* simulation (Koenig and Howard, 2004). To maintain consistency between the two setups

and minimize implementation discrepancies, both systems were developed utilizing the same Robot Operating System (ROS) (Quigley et al., 2009) tools. The robot arm motion was controlled with *MoveIt* (Coleman et al., 2014), planning linear end-effector trajectories according to the actions selected by the agent. In the real, the image observations are captured using an *Intel® RealSense™* camera positioned in front of the robot, and processed to obtain white-balanced 128x128 RGB images. In the simulation the images are rendered by a camera with the same intrinsic parameters as the real one, and are then processed through the same pipeline. The computation of the reward function during training requires knowledge of the position of the cube. In simulation, this information is obtained directly from the simulator. In the real-world scenario a tracking system was implemented using a second camera and a simple pointcloud-based pipeline that computes the 2D position of the cube. In simulation the reset of the environment performed at the start of every episode, which involves the position of the cube and end-effector at their respective random initial positions, is performed by teleporting the cube and moving the arm via Moveit. In the real, the same is achieved by moving the cube with the end effector itself, utilizing a matching hole fabricated in the top of the cube, and then moving the arm to its required pose.

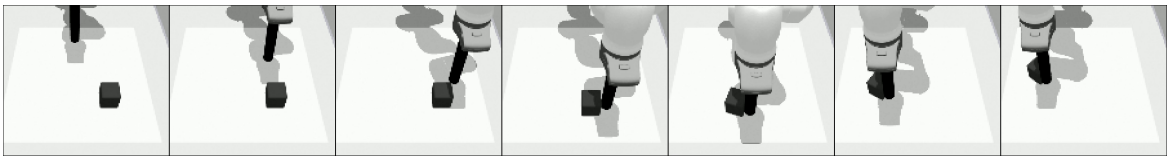
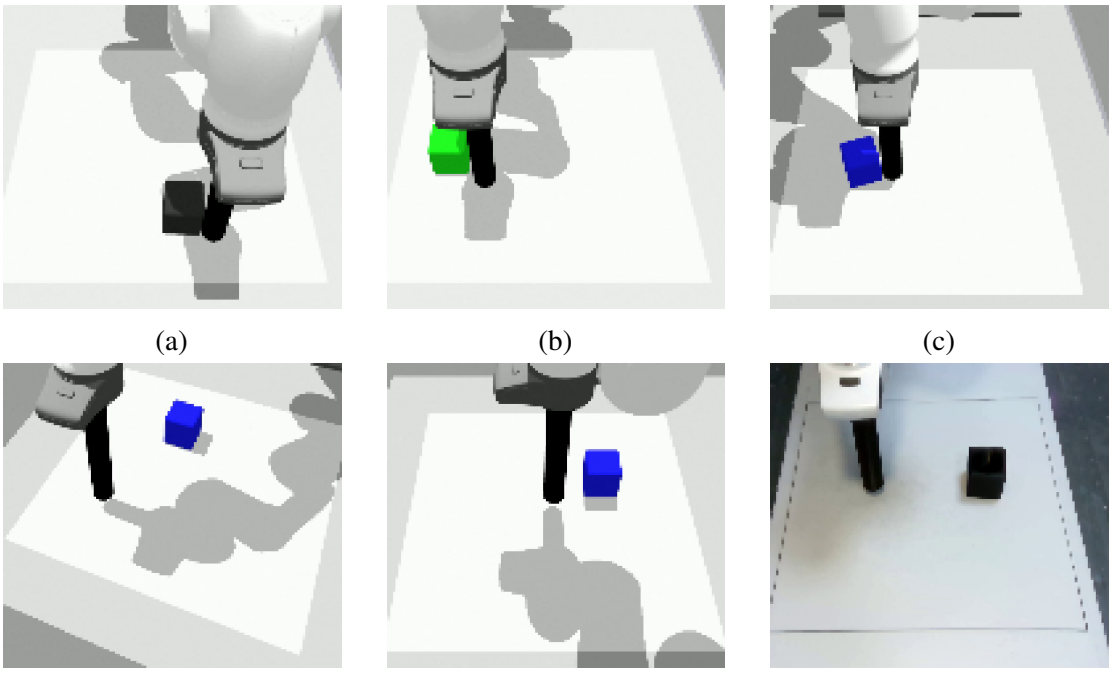


Figure 4.5 Example of one successful episode in the simulated setup. The image shows the observed images for steps 0, 4, 8, 12, 16, 20, 24.

4.4.1 Sim-to-sim

The core objective of the methodology discussed up to now is to perform policy transfer from simulation to reality. However, before discussing the sim-to-real transfer we analyze a series of sim-to-sim tasks. Performing the transfer between different simulated scenarios gives possibility of precisely controlling which variations are present across the transfer. This allows to craft different transfer scenarios of varying difficulty, enlarging or shrinking the size of the transfer gap, so to define a controlled and well-defined experimental setup.

The sim-to-sim evaluation was performed on a series of four scenarios of increasing difficulty. The scenarios are constructed by maintaining a fixed source domain and defining



(a) (b) (c)

(d) (e) (f)

Figure 4.7 Examples of the transfer scenarios: in 4.7a the source scenario, in 4.7b a minimal-gap scenario, in 4.7c a small-gap scenario, in 4.7d a medium-gap scenario, in 4.7e a large-gap scenario, in 4.7f the real-world scenario.

four sets of target domains, in which we alter characteristics such as the cube color, the lighting, and the camera pose. We will refer to the different transfer gaps as *Minimal Gap*, *Small Gap*, *Medium Gap*, and *Large Gap*.

In this section we discuss the different setups and the quantitative results achieved in each of them. The results are also reported in table 4.3 and figure 4.9. For comparison, figure 4.9b reports the performance achieved training from scratch in the source simulation scenario. Table 4.2 summarizes the characteristics of the different scenarios. Figure 4.7 presents example images for the different transfer scenarios.

Minimal Gap

In this scenario we only change the color of the manipulated cube. While in the source domain the cube is black, we define eight target scenarios with eight different colors: red, green, blue, yellow, cyan, magenta and 2 grades of gray.

With this setup, depending on the selected color, the policy has an initial performance that varies between 10 and 95 percent, and the method consistently reaches a 90% success

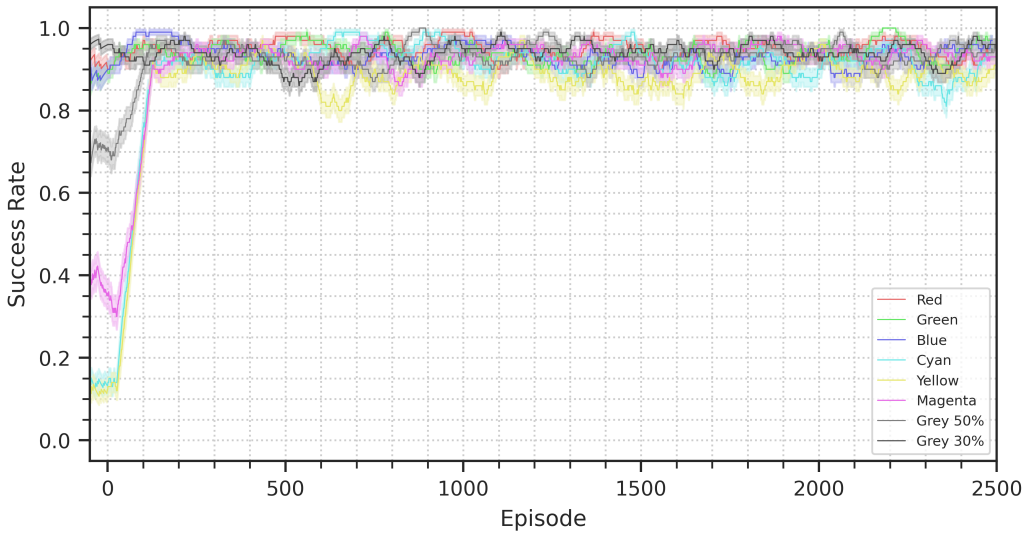


Figure 4.8 Success rate progression in the simulated minimal-gap scenario. Each run, corresponding to a different cube color and seed, is represented individually. Each line represents the success rate in the 100-episode window preceding the current episode, the background bands represent the 95% confidence interval for the average.

rate in just 110 episodes of finetuning, and then maintains a performance oscillating between 92 and 95 percent (see figure 4.8).

Small Gap

In this setup we introduce variations also in the camera pose and in the light direction. We vary the pose by translating the camera left, right, up or down of 5cm, whithout changing its orientation. We vary the light direction from the vertical axis of the source domain to four possible axes with a 30 degrees inclination either toward the left, right front or back. We set the cube color to be red across all variations of this scenario.

With these setup the agent performance is initially around a 5% success rate, which is comparable to a random policy. Performing the encoder and decoder finetuning the agent consistently reaches an 80% success rate in about 210 episodes and a 90% success rate in about 950. Consistently reaching an 88% success rate considerably sooner, by episode 500. The results are reported in figure 4.9.

Medium Gap

To further widen the transfer gap in this scenario we increment the magnitude of the camera pose change. We move the camera of 20cm instead of 5, and we alter its orientation to

Scenario	Cube Color	Light Direction	Camera Position	Camera Orientation
Original (Sim)	Black	Vertical	\	\
S2S - Minimal Gap	10 colors	Vertical	Unchanged	Unchanged
S2S - Small Gap	Red	30°: Left,Right,Back,Front	∼5cm offset	Unchanged
S2S - Medium Gap	Red	30°: Left,Right,Back,Front	∼20cm offset	Toward Center
S2S - Large Gap	Red	30° Left	∼70cm offset	90° Yaw
S2R - Minimal Gap	Black	Multiple sources, diffused	∼5cm offset	Minimal

Table 4.2 Variations from the source domain across the different experimental scenarios

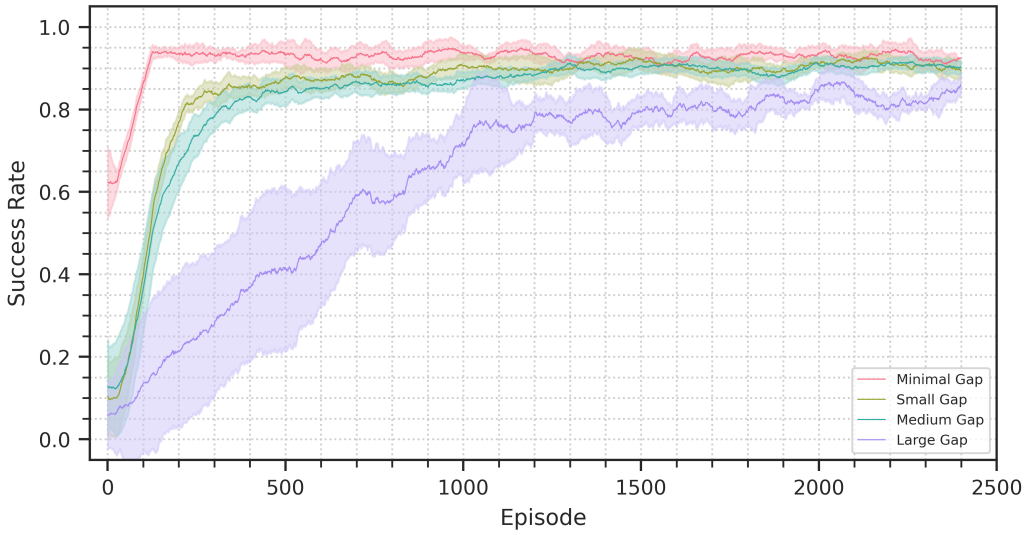
maintain the manipulation area in the field of view. We maintain the cube color as red in all tasks and vary the light direction in the same way as in the small-gap scenario.

The training performance is qualitatively equivalent to that of the small-gap scenario, reaching 80% success rate in 320 episodes and 90% in 1250. Also in this case a performance just under 90% is reached sooner, reaching 85% at episode 750.

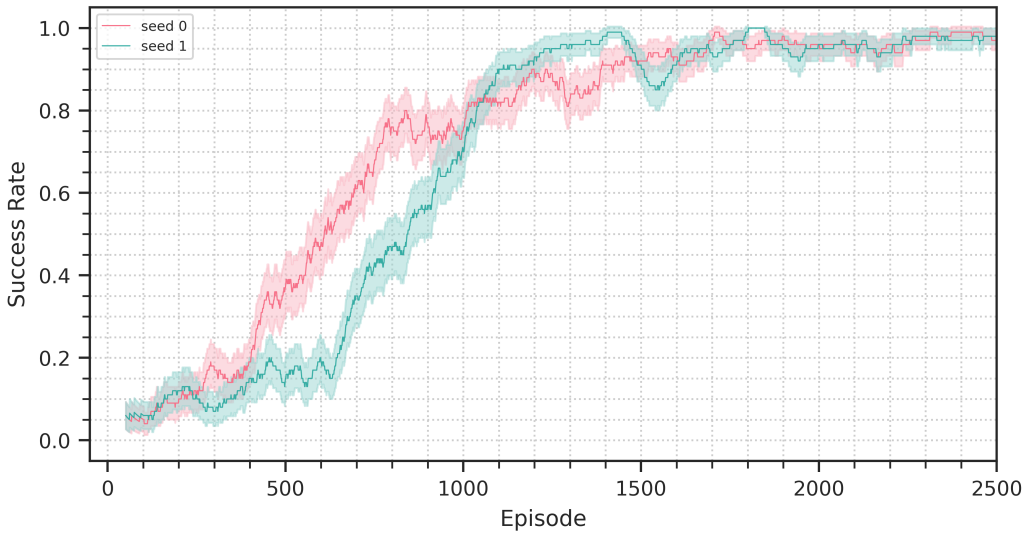
Large Gap

In the hardest sim-to-sim scenario we completely change to camera point of view, while still altering cube color and light direction. The camera is moved so that it faces the manipulation area from the side instead of the front, with a 90 degree point of view change.

In this scenario, which goes beyond what is just a sim-to-real transfer problem, the agent training requires about as much time as is required to train the agent from scratch. However, notably, it succeeds in overcoming the transfer and adjust its latent representation to match the policy. About 1200 episodes are required to reach an 80% success rate, and a maximum performance of 86% is reached by episode 2000, performing worse than the source training in asymptotic behaviour.



(a) Training progress across the different simulated scenarios.



(b) From-scratch training progress in the simulated scenarios.

Figure 4.9 Success rate progress for the simulation experiments. Figure 4.9a displays the results for all the scenarios in an aggregated form, the plots show the average performance across seeds on a 100 episode window. For the minimal, small, medium and large scenarios the number of random seeds was respectively 8, 12, 16, 4. The shaded area represents the 95% confidence interval on the success rate mean Figure 4.9b shows the progression for the training from scratch performed in simulation. Two random seeds are being shown. The solid lines represents the success rate in the 100-episode window preceding the current episode, again the background bands represent the 95% confidence interval.

4.4.2 Sim-to-real

After the sim-to-sim evaluation the performance of the method was investigated on a sim-to-real transfer scenario. In the real, the method was only evaluated on "minimal-gap" transfer, in which we minimize the environment differences by not intentionally introducing variations and trying to replicate the simulated scenario for what is possible. Despite this, the transfer still presents considerable differences from the simulation, small changes in the camera pose are present, and the lighting and textures of the environment are more varied and complex. Also, changes in the physical characteristics of the task are inevitably present, for example in the contact and friction dynamics, or in the inertia of sthe bodies. For a reference of the visual differences, figure 4.12 shows the camera view and decoder reconstruction in the simulated and real scenarios.

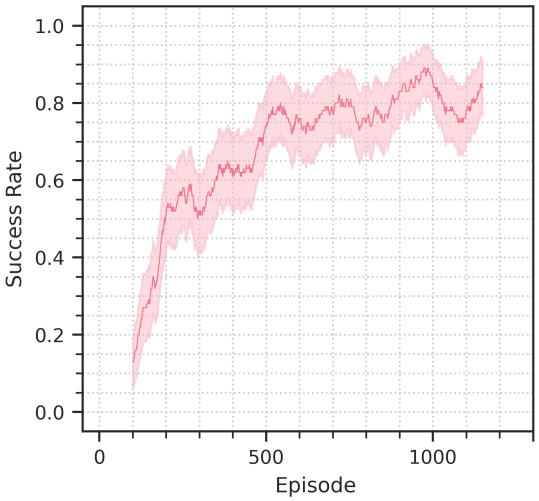
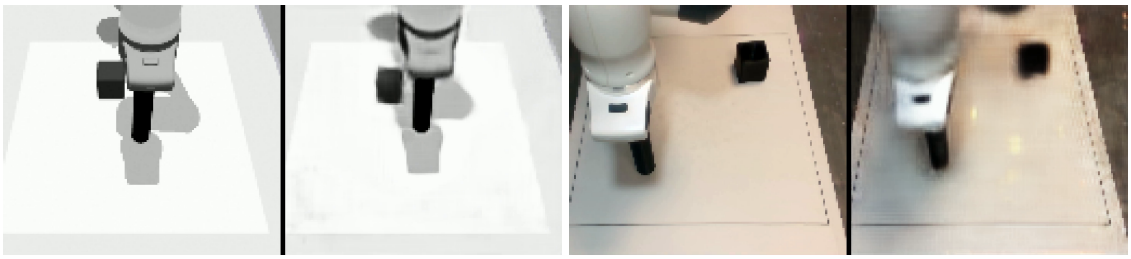


Figure 4.11 Success rate progression in the sim-to-real experiment. The solid line represents the success rate in the 100-episode window preceding the current episode, the background bands represent the corresponding 95% confidence interval.

In addition to these differences, the reward function computation is also different. It relies on the estimated cube position to calculate the distances between the cube and the goal and the cube and the end-effector. This estimation inevitably introduces errors, delays, and noise, a characteristic that is not present in simulation.

In this transfer scenario the initial success rate achieved by the policy is about 10%. The DVAE finetuning brings the success rate 80% in about 550 episodes, corresponding to 5 hours of experience, a performance of 90% is achieved in 990 episodes, 10 hours of experience data.

When compared to the training from scratch in simulation, the sim-to-real finetuning is considerably faster in the first stages of learning, achieving 80% success rate in about half the time, however reaching 90% requires almost as much time as the source training. It must be noted however that a training from scratch in the real may require considerably more time than in simulation, due to the higher complexity of the sensory inputs. Also, differently form the sim-to-sim scenarios, in this case there are alteration in the physical properties of the



(a) Input and prediction in simulation (b) Input and prediction in the real world

Figure 4.12 Input image and predicted image in the simulated (left) and real (right) setups

Scenario	Success %	Init. Succ. %	T.T. 80%	T.T. 90%
Sim. From Scratch	98%	5%	1020 Eps.	1180 Eps.
S2S - Minimal Gap	95%	50%	90 Eps.	110 Eps.
S2S - Small Gap	92%	5%	210 Eps.	950 Eps.
S2S - Medium Gap	92%	5%	320 Eps.	1250 Eps.
S2S - Large Gap	85%	5%	1200 Eps.	\
S2R - Minimal Gap	92%	5%	550 Eps.	990 Eps.

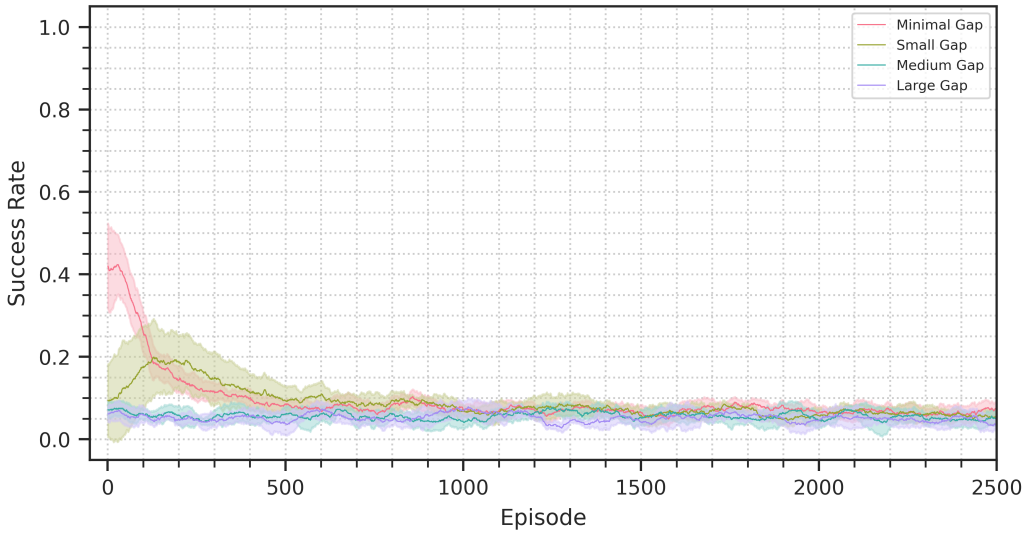
Table 4.3 DVAE-SAC results on the four sim-to-sim (S2S) and the sim-to-real (S2R) scenarios. Columns indicate respectively: the best achieved success rate, the initial success rate in the target domain (i.e. zero-shot transfer performance), the number of episodes required to reach 80% success rate, the number of episodes required to reach 90% success rate.

system, which are not being directly tackled by the agent finetuning. No attempt was made to do so, but these difference could be tackled by performing a final policy finetuning, or by using domain randomization on the physical parameters of the simulated source scenario.

4.4.3 Sim-to-sim with VAE-SAC

To explore the importance of the dynamics predictor constraining for domain transfer we evaluated the performance of a VAE-SAC agent on the sim-to-sim scenarios. As discussed in section 3.2, the VAE-SAC agent is a version of our architecture in which the dynamics predictor is not present. Figure 4.13a shows the achieved transfer performance. As expected the domain transfer fails, as there is no constraint to keep the latent representation compatible with the control policy. Even in the minimal-gap scenarios, where the zero-shot performance

performance is not zero, the success rate rapidly descends to performance comparable to that of a random policy. In the small-gap scenario we can see the performance initially rising, but then decaying too to negligible success rates.



(a) Success rate progress for the sim-to-sim VAE-SAC experiments. The plots show the average performance across seeds on a 100 episode window. The shaded area represent the 95% confidence intervals for the means.

4.5 Conclusions

Throughout this chapter I presented a methodology for transferring a DVAE-SAC decoupled visual policy across domains, between different simulated scenarios and across the reality gap. As was discussed, an architecture such as DVAE-SAC has the advantage of being extremely sample efficient in its source training while maintaining a high generality and the capability to solve tasks end-to-end, from raw observations to control actions. The possibility to transfer such an architecture allows to deploy in the real world complex policies with small amounts of real-world data.

The sim-to-sim experiments of section 4.4.1, focused on the object pushing task, showed how the method can perform domain transfer across increasingly difficult scenarios. The most simple scenarios only present simple variations in the visual observations, such as changes in the hue of the manipulated object. The most complex ones involve radical changes on the point of view of the camera, together with changes in lighting and hue. The method proved to be extremely effective at surpassing the smaller domain gaps, requiring up to ten

times less data than a from-scratch training. It also showed to be capable of solving the most difficult transfer tasks, albeit with progressively increasing data requirements. The experiments of intermediate difficulty, in which the camera is moved resulting in point of view changes up to 20 degrees, are still solved with remarkable sample efficiency gains, requiring up to four time less data than the from-scratch training. The hardest task, in which the camera point of view is changed of 90 degrees, is still solved by the method, but requires roughly the same amount of data as training the agent from scratch. However, in this harder task, the domain transfer gap is already beyond what could be a typical sim-to-real gap.

The method also manages to solve the sim-to-real transfer task. It exhibits a fast initial adaptation, reaching a success rate of roughly 50% in less than half the time required to do so in the source training, however it then slowly settles toward an asymptotic performance worse than the source training one. This can be explained on one side with the higher complexity of the visual observations in the real, where more complex lighting and textures are present. At the same time, it must be considered that the transfer of the method does not account for differences in the physical and dynamical characteristics of the task. The SAC policy is transferred without any adaptation, and the dynamics model is not being trained. Such differences are inevitably present when transferring between simulation and reality, due to modeling errors and inaccuracies. The friction coefficients and inertias of the various bodies in the scene were not tuned carefully to match the real scenario. Such limitation can be tackled with different approaches, which can be applied together with the current transfer strategy. The first is to perform an accurate modeling of the system and employ system identification techniques to identify proper model parameters. The second is to perform policy finetuning in the real, after a period of adaptation of the representation. The third is to use domain randomization on the physical parameters of the system.

In addition to these considerations it is also important to note that another potential source of error is noise in the reward function, which is present in the real world setup due to the cube detection pipeline, but is not present in simulation. This error directly affects the representation constraining as the reward prediction is performed by the frozen dynamics model, which is used to constrain the latent representation.

Despite these limits, the transfer strategy proposed in this chapter shows how it is possible to perfom domain transfer of decoupled RL policies by separately adapting policy and feature extraction. The experimental analysis focused on adaptation of the feature extraction section of the agent, showing how it is possible to adapt a reinforcement learning policy without using reinforcement learning in the target domain. This characteristic has the potential to

greatly reduce data requirements for the training of complex policies, as adaptation data requirements are not directly dependant on task complexity.

As we will see in the next chapter the transfer strategy proposed up to now comes with strong requirements on the definition of the task. Most importantly, the presence of the highly-structured reward function used in the pushing task proved to be essential to the effective transfer of the policy. The next chapter will show how this requirement can be removed, allowing to tackle more complex tasks and improving the generality of the approach.

Chapter 5

Solving and Transferring Complex Tasks

Summary

The DVAE-SAC architecture developed in chapter 3 showed to be capable of efficiently solving complex visual-based control tasks, and the transfer methodology introduced in chapter 4 showed how this same architecture can be used to transfer learned policies between domains, across sizable sim-to-sim gaps and from simulation to reality.

However, the DVAE-SAC method can struggle to find satisfactory solutions in tasks where it is challenging to extract from visual observations the information relevant to the system's dynamics. Also, the transfer strategy proposed in the previous chapter in practice heavily relies on the presence of shaped rewards. Defining informative shaped rewards can be non-trivial for complex tasks, and computing such rewards in the real can require the development of complex perception systems, introducing considerable engineering work.

This chapter will extend the architecture to overcome these limitations. Section 5.1 discusses how predicting multi-step trajectories instead of simply generating observations one timestep into the future can greatly improve the method's stability and performance, especially in highly dynamical tasks. Then, section 5.3 will explore how domain transfer can be improved by introducing an explicit *consistency loss* to keep the target-domain latent representation similar to the one trained in the source domain. Experimental results will show how this improvement greatly enhances adaptation speed and permits to surpass sizable domain transfer gaps even with scarcely informative almost-sparse rewards.

Section 5.2 instead explores how DVAE-SAC can scale to solve complex tasks. The focus will be especially on tasks that present hard exploration challenges, as these are the ones where reinforcement learning can display its most interesting and distinctive capabilities. To explore this kind of tasks with the DVAE-SAC architecture, I will focus on extensions of

the object-pushing task. It will be shown how the method can solve the pushing task even when using sparse rewards, or rewards that are not fully informative of the task. Also, the performance of the method on a more complex pushing task is explored, a "gate" will be introduced in the center of the manipulation area, which the robot must learn to open before successfully bringing the cube to the destination. We will see how the agent can learn to solve this task if provided enough data, even without explicitly rewarding the interaction with the gate.

5.1 Improving Performance: Trajectory Prediction

The intuition behind the introduction of the dynamics predictor of the DVAE-SAC architecture in chapter 3 was that its presence would push the autoencoder architecture toward including in the latent representations the features that are necessary for dynamics prediction. Such features may not be clearly visible in the image observations and may not have a strong impact in image reconstruction by itself. However reconstructing the next observation instead of the current one can make the impact of such features bigger on the reconstruction error, thus pushing the encoder into including these features in the latent vectors. This idea proved to be correct, as moving from simple observation construction to observation prediction provably improves performance. However, predicting just one observation into the future may not always be enough to generate a strong learning signal toward representing these features. For example, in the case of slowly moving objects the difference between reconstructing the current frame or the subsequent one may be negligible. This can have an impact on the quality of the learned representation, degrading the overall performance of the agent.

To strengthen this weakness the architecture can be extended to predict over multiple steps. To do so, the dynamics predictor can be used recursively to produce latent trajectories, and from these latent trajectories it is possible to predict full observation trajectories. To train this architecture optimization will be performed at the same time for observation and latent prediction. The details of the implementation are covered in the next section.

As we will see soon, this proves to greatly improve the performance of the method, increasing stability, asymptotic performance and sample efficiency when compared to 1-step prediction. Even allowing to solve tasks for which the previous methodology failed, such as for example the cartpole-swingup scenario. The next section will discuss the implementation of this new architecture, while section 5.1.2 will present the experimental results achieved with this new implementation.

5.1.1 Implementation

As in the previous chapters the DVAE architecture is composed of three components:

- The **encoder network** $e_\theta : O \to \mathcal{Z}$, which extracts latent states z_t from the input observations o_t
- The **latent dynamics predictor** $f_\theta : (\mathcal{Z} \times \mathcal{A}) \to (\mathcal{Z} \times \mathbb{R})$, which predicts latent states z_{t+1} and rewards r_t from latent states z_t and actions a_t
- The **decoder network** $d_\theta : \mathcal{Z} \to O$, which can reconstruct predicted observations $\hat{o}_{t+1}$ from latent states z_{t+1}

This basic structure of the architecture is maintained also when performing multi-step trajectory prediction. However, the way the architecture is used during training is considerably different. To predict trajectories the dynamics predictor f_θ can be used recursively to generate latent trajectories $\{\tilde{z}_t, \hat{z}_{t+1}, \hat{z}_{t+2}, ..., \hat{z}_{t+T}\}$, where we indicate with $\hat{z}_i$ the predicted latent vectors, and with $\tilde{z}_i$ the latent vectors obtained by directly encoding observations as $e_\theta(o_i)$. To recursively generate predicted trajectories with the 1-step dynamics predictor we only need an initial latent vector z_t and a sequence of actions $a_{t\ldots t+T-1}$:

$$\begin{aligned} &\hat{z}_{t+1} \approx f_\theta(\tilde{z}_t, a_t) \\ &\hat{z}_{t+2} \approx f_\theta(\hat{z}_{t+1}, a_{t+1}) \\ &\ldots \\ &\hat{z}_{t+T} \approx f_\theta(\hat{z}_{t+T-1}, a_{t+T-1}) \end{aligned} \tag{5.1}$$

In order to appropriately train the architecture we employ three separate losses: an observation prediction loss, a latent prediction loss and a reward prediction loss. The observation prediction loss is based on a mean square error loss applied between predicted and ground truth observation trajectories. The latent prediction loss is based on a mean square error loss between the recursively predicted latent trajectories and the latent trajectories obtained by encoding ground truth observation trajectories. The reward prediction loss is based on the mean square error between predicted and ground truth rewards. Formally, these losses are defined as follows:

$$\begin{aligned}
\mathcal{L}_{obs}(\theta; o_{0...T}, a_{0...T}, r_{0...T}) &= \sum_{i=0}^{T-1} \alpha_{obs,i} ||\hat{o}_i - o_i||_2^2 \\
\mathcal{L}_{rew}(\theta; o_{0...T}, a_{0...T}, r_{0...T}) &= \sum_{i=0}^{T-1} \alpha_{rew,i} ||\hat{r}_i - r_i||_2^2 \\
\mathcal{L}_{lat}(\theta; o_{0...T}, a_{0...T}, r_{0...T}) &= \sum_{i=1}^{T-1} \alpha_{lat,i} ||\hat{z}_i - \tilde{z}_i||_2^2
\end{aligned} \tag{5.2}$$

With:

$\hat{o}_j = d_\theta(\hat{z}_j)$ for all j

$\tilde{z}_j = e_\theta(o_j)$ for all j

$\hat{z}_{j+1}, \hat{r}_{j+1} = f_\theta(z_j, a_j)$

$\alpha_{obs,i}, \alpha_{rew,i}, \alpha_{lat,i}$ being hyperparameters

$o_{0...T}, a_{0...T}, r_{0...T}$ being ground truth observations, actions and rewards

The overall loss of the architecture combines the new latent, observation and reward losses with the KL-divergence loss of the variational formulation, that is only applied to the initial latent vector $\tilde{z}_t$.

$$\begin{aligned}
\mathcal{L}_{DVAE_t}(\theta; o_{0...T}, a_{0...T}, r_{0...T}) = &\lambda_{kl} D_{KL}(e_\theta(\tilde{z}_t|o_t)||\mathcal{N}(\mathbf{0}, I_k)) + \\
&\lambda_{rew}\mathcal{L}_{rew}(\theta; o_{0...T}, a_{0...T}, r_{0...T}) + \\
&\lambda_{obs}\mathcal{L}_{obs}(\theta; o_{0...T}, a_{0...T}, r_{0...T}) + \\
&\lambda_{lat}\mathcal{L}_{lat}(\theta; o_{0...T}, a_{0...T}, r_{0...T})
\end{aligned} \tag{5.3}$$

This formulation introduces several hyperparameters. The λ_{obs}, λ_{lat} and λ_{rew} coefficients control the importance given respectively to the reconstructions, the latent prediction and to the reward prediction, while the $\alpha_{obs,i}$, $\alpha_{lat,i}$ and $\alpha_{rew,i}$ weights can be used to balance the importance of predictions farther into the future or closer in time. In practice, the predictions farther into the future are generally inherently harder to approximate, thus it may be needed to reduce their weight, however it is also desirable for the architecture to not fully concentrate on the first reconstruction $\hat{o}_0$, which does not use the dynamics predictor. Throughout the experimentation the $\alpha_{\cdot,t}$ weights for steps $t > 0$ were set according to a discount factor: $\alpha_{\cdot,t} = \gamma^{t-1}$. The factor for step zero, for which no prediction is performed, was kept at a lower value (e.g. 0.2 or 0.1).

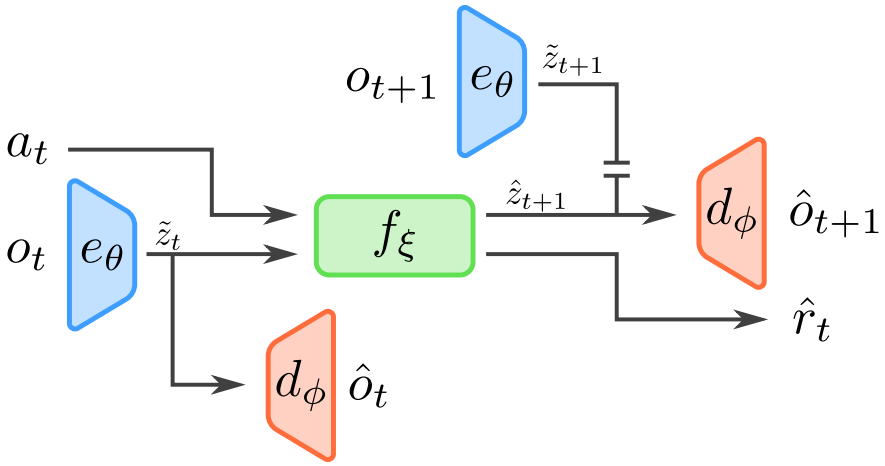


Figure 5.1 Multistep DVAE-SAC architecture in the case of 1-step trajectories. The o_t and o_{t+1} observations are encoded by the same encoder e_θ into the latent vectors $\tilde{z}_t$ and $\tilde{z}_{t+1}$, the dynamics predictor f_ξ predicts the latent vector $\hat{z}_{t+1}$ and the reward r_t from the encoded latent vector $\tilde{z}_t$ and the action a. The latent vectors $\tilde{z}_t$ and $\hat{z}_{t+1}$ are the decoded in the reconstructed observations $\hat{o}_t$ and $\hat{o}_{t+1}$.

While the original DVAE architecture was trained on single transitions (o_t, a_t, o_{t+1}, r_t), this new iteration of the architecture is trained on observation-action-reward trajectories. The computation of each loss sample requires the use of a trajectory of observations, coupled with the corresponding action trajectory and the resulting rewards.

Like before, this experience data can be collected online and stored in a buffer, from which the trajectories can be sampled to optimize via stochastic gradient descent. As the computation of each sample of the loss sample requires the computation of multiple encoded z_i vectors and decoded o_i observations, encoder and decoder are re-utilized multiple times during each forward step.

Figure 5.1 shows a representation of the architecture in the case of a 1-step trajectory. Already in the 1-step case the use of the architecture has changed with respect to chapter 3. Previously, the decoder was only used on z_{t+1} and the encoder was only used on o_t, now instead the decoder is also applied to the latent vector z_t to produce the reconstructed observation $\hat{o}_t$, and the encoder is also used on the observation o_{t+1} to produce the latent vector $\tilde{z}_{t+1}$. The reconstructed observation $\hat{o}_t$ is used together with the predicted observation $\hat{o}_{t+1}$ to compute the reconstruction loss $\mathcal{L}_{obs}$, while the predicted latent vectors $\hat{z}_{t+1}$ and $\tilde{z}_{t+1}$ are used to compute the latent loss $\mathcal{L}_{lat}$. The reconstruction of o_t and the encoding of o_{t+1}, which were not performed before, push z_t and z_{t+1} into using the same representation, as the two vectors now have to be compatible with the same encoder and decoder pair. In

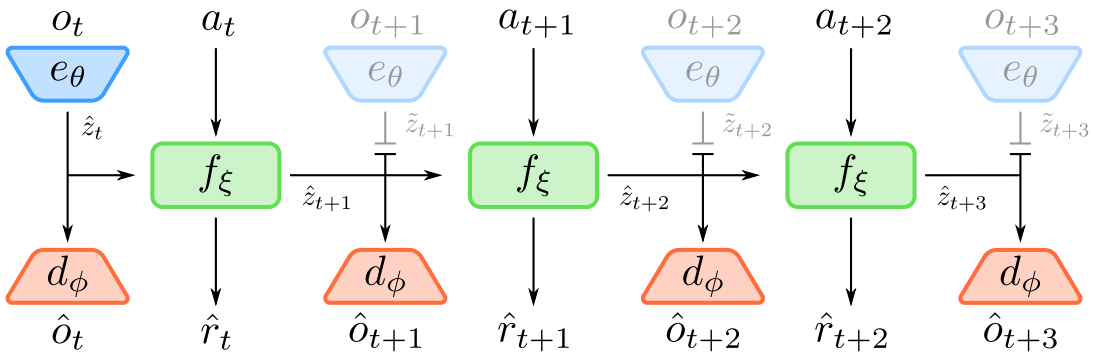


Figure 5.2 DVAE architecture unrolled over a 3-step trajectory. The first observation and the action trajectory are used to produce the latent trajectory and the corresponding predicted observations and rewards. The ground truth observation trajectory is used to compute the reference latent trajectory for the latent loss computation.

the previous implementation of the architecture the latent representation at the output of the encoder and at the input of the decoder could have been completely different, nothing constrained them to be compatible. Using the same latent representation in the input and output of the dynamics predictor is a first necessary step toward being able to use the dynamic network recursively to predict trajectories. If the representation used in the input and out of the dynamics predictor was same, it could already be expected that the dynamics predictor would perform adequately when used recursively to predict trajectories, even if trained just on 1-step transitions, however this is not true in practice and training over trajectories of multiple is necessary to obtain good prediction over longer horizons. The number of steps used for the training can be considered an hyperparameter, and decided depending on the characteristics of the task. Algorithm 6 presents the proposed training procedure.

Figure 5.2 shows the architecture unrolled over a 3-step trajectory. The observation o_t and the action trajectory $\{a_t, a_{t+1}, a_{t+2}\}$ are given as input, as output we obtain the latent trajectory $\{\tilde{z}_t, \hat{z}_{t+1}, \hat{z}_{t+2}, \hat{z}_{t+3}\}$, the reconstructed observations $\{\hat{o}_t, \hat{o}_{t+1}, \hat{o}_{t+2}, \hat{o}_{t+3}\}$, and the predicted rewards $\{\hat{r}_t, \hat{r}_{t+1}, \hat{r}_{t+2}\}$. The full observation trajectory $\{o_{t+1}, o_{t+2}, o_{t+3}\}$ is used to produce the reference latent trajectory $\{\tilde{z}_{t+1}, \tilde{z}_{t+2}, \tilde{z}_{t+3}\}$, which is only used to compute the latent loss $\mathcal{L}_{lat}$.

5.1.2 Experiments

The impact of the multi-step dynamics prediction on the performance of the overall agent was quantified on selected tasks from the DeepMind Control Suite. We evaluate on the

Algorithm 6 DVAE-SAC Multi-Step Training

Inputs:
N: number of training episodes
M: number of episodes collected for each training iteration
T: trajectory length used for dynamics training
K_{DVAE}: number of DVAE gradient steps per episode
K_{SAC}: number of SAC training steps per episode

Let π_ϕ be a Soft Actor-Critic policy parameterized by ϕ
Let θ be the parameters for the DVAE encoder e_θ, decoder d_θ, and dynamics predictor f_θ
Randomly initialize θ and ϕ
for N episodes **do**
    Collect M episodes into a dataset $\mathcal{D}$ according to policy π_ϕ
    for K_{DVAE} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of trajectories $(a_{0...T}, o_{0...T}, r_{0...T})$
        $g = \nabla \mathcal{L}_{DVAE_t}(\theta; \mathcal{B})$
        $\theta = ADAM(\theta, g)$
    for K_{SAC} gradient steps **do**
        Sample from $\mathcal{D}$ one batch $\mathcal{B}$ of tuples (o_t, a_t, r_t, o_{t+1})
        $\mathcal{B}_e = \{(e_\theta(o_t), a_t, r_t, e_\theta(o_{t+1}))$ for all (o_t, a_t, r_t, o_{t+1}) in $\mathcal{B}\}$
        $\phi = SAC\_UPDATE(\phi, \mathcal{B}_e)$

Figure 5.3 Multi-step training procedure for DVAE-SAC. DVAE weights are updated with ADAM Kingma and Ba (2015) and the policy is trained with SAC_UPDATE as defined in Haarnoja et al. (2018b).

cartpole-balance, cartpole-swingup and cheetah-run environments as in chapter 3. We use these three setups as benchmark as they each present different challenges.

The cartpole-balance task is the simplest of the three, but even in its simplicity extracting dynamics-relevant features from its visual observations can be challenging. The task consists of balancing a pole on a moving cart, consequently the evolution of the dynamics is heavily sensitive to small changes in the velocity and position of the two bodies. This scenario uses frame-stacked image observations, so it possible to infer the velocities of the bodies directly from an image, however they can only be discerned from small details in the image observations. As long as the pole is upright it is relatively easy to maintain the balance, but once this equilibrium is broken it is not trivial to recover. To successfully solve the task it is necessary for the agent to obtain a good estimation of the velocities of cart and pole. In the plots in figure 5.4 it is possible to see the performance of the agent in the case in which no

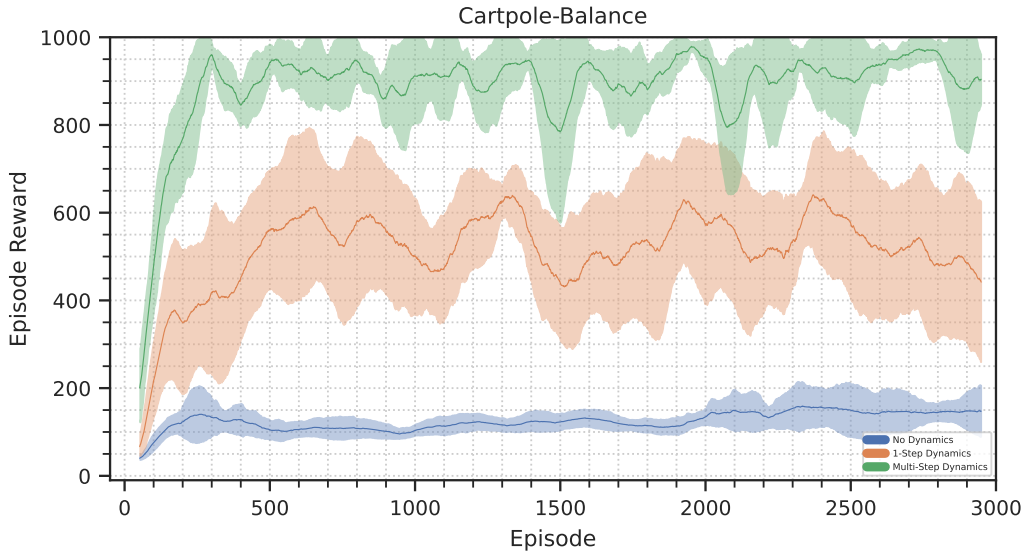


Figure 5.4 Episode return progress during training of the cartpole-balance task. The three curve show the performance for the no-prediction VAE-SAC, for the 1-step prediction DVAE-SAC and for the 5-step prediction DVAE-SAC. Each curve displays the average performance over a 100-episode window and 8 separate runs with different random seeds.

dynamics prediction is used, in the case in which 1-step prediction is performed and in the case of multi-step prediction. In this latter case the dynamics prediction was performed on trajectories of 5 steps. Episodes are 10 seconds long, with a control rate of 25Hz, a reward of 1000 is equivalent to the pole being upright for the whole episode duration.

The swing-up task presents the same difficulties we have just discussed for the carpole-swingup, but it starts from a non-balanced position. Consequently the agent cannot learn to just balance the pole, it has to learn to swing it up. This requires even more precise inference of the cart and pole velocities, as the pole must be accelerated and then stopped precisely at the upright position. The versions of the architecture that do not use the dynamics or only use 1-step prediction fail at solving this task. From observing the policy behaviour it was possible to see how they converged on a local minima, a simpler policy that continuously rotated the pole around, achieving a reward higher than random, but not actually balancing the pole. Only considerably reducing the control frequency considerably below the chosen default of 12.5Hz allows these policies to succeed, although in a considerably brittle fashion and only after long training times. An explanation for this is that lower control frequency result in greater differences between the frames of the frame-stacked observations, permitting an easier inference of the velocities. The multi-step architecture instead succeeds at bringing the pole upright and balancing it at a 12.5Hz control frequency. Like in the balancing task,

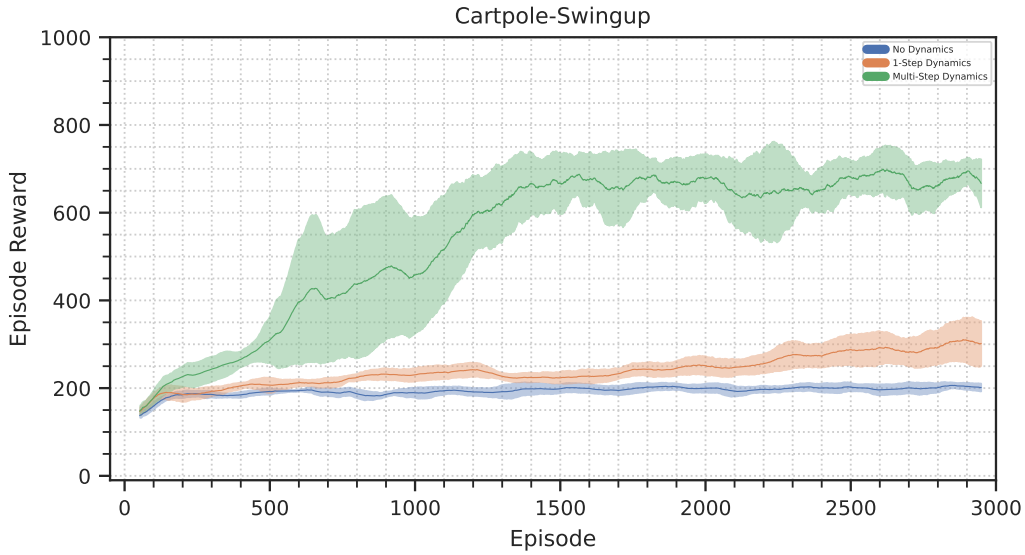


Figure 5.5 Achieved episode return during training of the swing-up task. The three curve show the performance for the no-prediction VAE-SAC, for the 1-step prediction DVAE-SAC and for the 5-step prediction DVAE-SAC. Each curve displays the average performance over a 100-episode window and 8 separate runs with different random seeds.

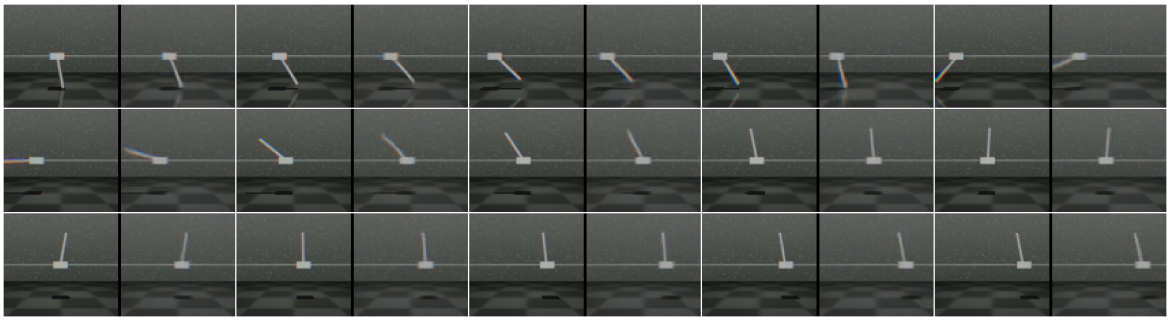
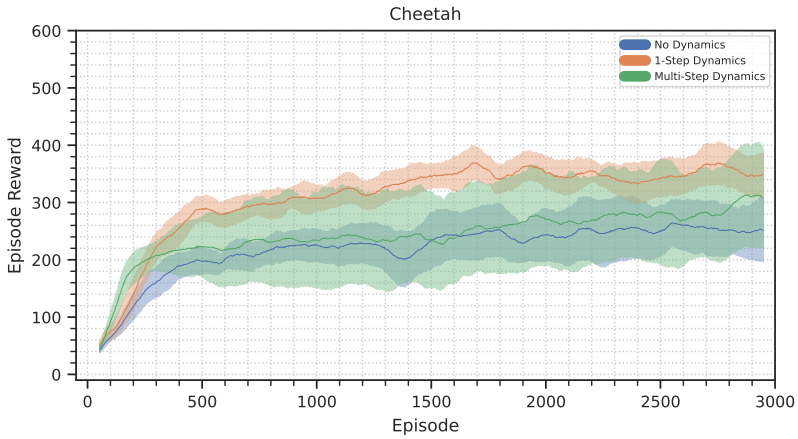


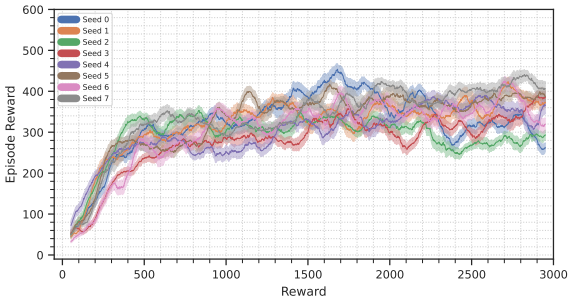
Figure 5.6 Frames depicting the behaviour of a policy successfully trained on the swing-up task. Each pair of images shows the input observation on the left and the 1-step observation prediction on the right. This specific sequence is from the training of a 5-step prediction DVAE-SAC policy.

trajectories of 5 steps were used for the model training. Figure 5.5 presents a comparison of the performance of the three methods, in figure 5.6 we can instead see an example of the trained policy.

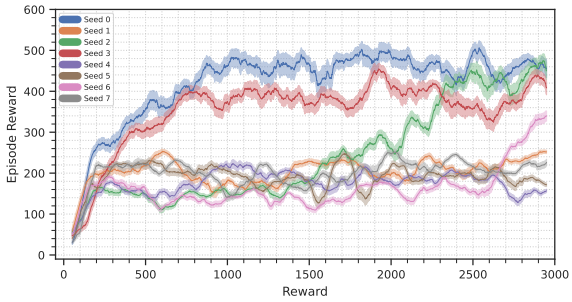
The last task that was examined is the cheetah task. The cheetah task is the most complex of the three Deepmind Control Suite tasks taken for analysis. The dynamics of the task are considerably more complex, as contact dynamics are involved in the robot-ground interaction, and in addition to this the degrees of freedom and the action dimensionality are increased.



(a) Average performance achieved with no dynamics, 1-step dynamics and multis-sep dynamics (5 steps).



(b) 1-step prediction



(c) 5-step prediction

Figure 5.7 Cheetah trainings with 1-step and 5-step prediction, seed are shown individually.

The robot is controlled on 6 joints via torque, and in addition to these degrees of freedom, the observation model must also infer for the 2D position and velocity of the robot. From the plot in figure 5.7a, we can see how, in the case of the cheetah the multistep prediction does not seem to bring a clear advantage. Average performance across runs is actually worse than the 1-step case, however, in figure 5.7b and 5.7c it is possible to see how in some cases the multi-step architecture actually manages to match and surpass the 1-step dynamics performance, but it often fails to learn the task and stops in a local minimum. Qualitatively observing the agent behaviour this local minimum it can be seen that it corresponds to the robot hopping forward instead of running.

A qualitative evaluation of the dynamics prediction can also be made from observing the predicted observation trajectories. Figure 5.8 presents a 10-step predicted trajectory from the fully trained cheetah task. We can observe how the predicted observations remain consistent with the ground truth for a several steps before degrading. In the case shown in the figure the

architecture was trained over 5-step trajectories, but we can see how the predictions remain consistent up to 8-9 steps in the future.

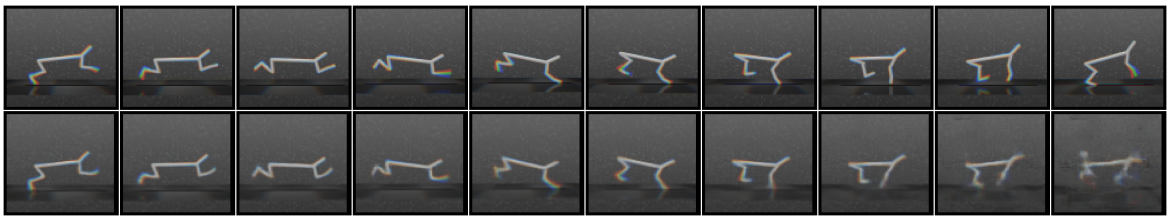


Figure 5.8 Observation trajectory prediction in the cheetah task. The first row of images displays a ground truth observation trajectory, the second row presents the observation trajectory predicted by the DVAE-SAC architecture using the first observation image and the sequence of actions. The model used to produce this output was trained over 5-step trajectories.

5.2 Solving Difficult Tasks

Before focusing on how to perform domain transfer of policies for complex tasks, this section focuses on how to train these tasks in the source domain in the first place.

The first objective is to train the DVAE-SAC agent without the use of a shaped reward, but only using a sparse or almost-sparse reward. A sparse reward is a reward that only rewards the accomplishment of a task goal, for example in the case of the object pushing a sparse reward could be a function which is always zero except when the cube is within the goal position tolerance.

$$r(p_c, p_g) = \begin{cases} 1 & \text{if } ||p_c - p_g|| \leq d \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

With p_c and p_g being respectively the cube position and the goal position, and d being the goal tolerance distance

By almost-sparse reward I refer to rewards that are not strongly shaped to guide the agent into how to solve the task, but are still not strictly sparse. In the case of the object pushing task an almost-sparse reward is for example a reward only based on the distance between cube and goal. The following reward function for example gives a value of 100 when the cube is exactly at the goal and decreases linearly as the cube gets further.

$$r(p_c, p_g) = 100\frac{1 - ||p_c - p_g||}{1} \tag{5.5}$$

With p_c and p_g being respectively the cube position and the goal position

Supporting the use of sparse rewards is an important requirement as they are an extremely general and simple paradigm for expressing task objectives. The use of sparse rewards completely avoids the reward-tuning process, which can be a laborious undertaking when tasks are complex. The use of shaped reward also relies on knowledge of the rewarded features, which in general cannot be assumed and may require the implementation of specific sensing systems.

However, sparse reward are inherently harder to learn from, the agent has to discover the rewarding states and is not guided in the behaviours it has to employ, this by itself constitutes a

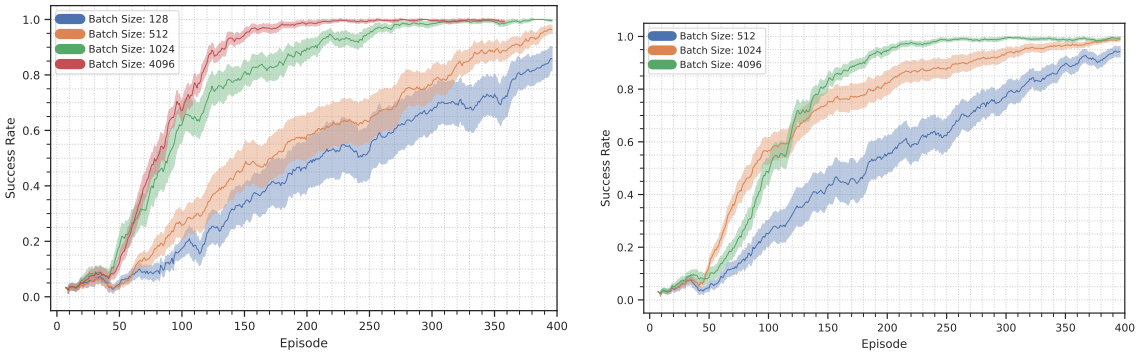
difficult exploration problem. The next two sections will discuss the two different approaches that were employed to tackle this problem with the DVAE-SAC architecture, the use of parallelized simulation, and the use of intrinsic rewards. The effectiveness of this approach will be evaluated on the object-pushing task already used in chapter 4. Section 5.2.3 discusses the use of the DVAE-SAC method to solve even more complex exploration problems, taking as an example an extension of the object pushing task in which the robot must move an obstacle before bringing the cube to the goal.

5.2.1 More Experience Data

The most immediate and simple way to improve exploration and solve sparse reward tasks is to use larger amounts of data and greater batch sizes in the learning of the RL policy. Intuitively the use of larger batches smoothes the highly irregular loss landscape induced by the sparse reward, allowing for the value and policy networks to properly converge. This is an effective approach, in particular in the case of "exploring starts", in which the initial states of the episodes already give good coverage of the state space. This because if rewarding states cannot be reached with random exploration from the episode starts then the agent will have no learning signal to follow, as it will always ever see zero returns.

In the case of the almost-sparse and strictly-sparse rewards this proved to be an effective approach for the object pushing task. To effectively collect data for the task a new simulated implementation of the task was developed. This new scenario was implemented using the PyBullet simulator and removing the ROS-based robotics pipeline used in chapter 4, only maintaining the end effector attached to a 2-DOF position-based actuation. This allowed to effectively parallelize the simulation, exploiting high core-counts CPUs it was possible to simulate either up to 64 environments simultaneously. The availability of these amounts of data allowed the use of big batch sizes, in my experiments batch sizes of 128 samples showed to already be sufficient to solve the task, but higher batch sizes improve stability and reduce training time, a batch size of 4096 showed to be a good compromise between stability, training speed, memory usage and computational load.

The use of a decoupled architecure such as DVAE-SAC proved to be particularly suited for solving visual tasks that require big batch sizes. The use of large batches is only needed for the training of the SAC policy. The vision section of the network, which accounts for the greater part of the parameters, is not optimized with large batches, considerably reducing memory requirements.



(a) Almost-sparse task

(b) Strictly-sparse task

Figure 5.9 Training progress with different batch sizes for the almost-sparse and strictly-sparse reward cases.

In figures 5.9a and 5.9b it is possible to see the learning curves for the sparse and almost-sparse case, with different batch sizes.

Using this approach it was possible to train solving policies in less than 3 hours of real-world time, while collecting approximately 5 days of experience data in simulation. The training also proved to be more reliable and stable than for the non-parallelized shaped-reward task of chapter 4. Figure 5.10 shows a representation of the policy performance as the training progresses.

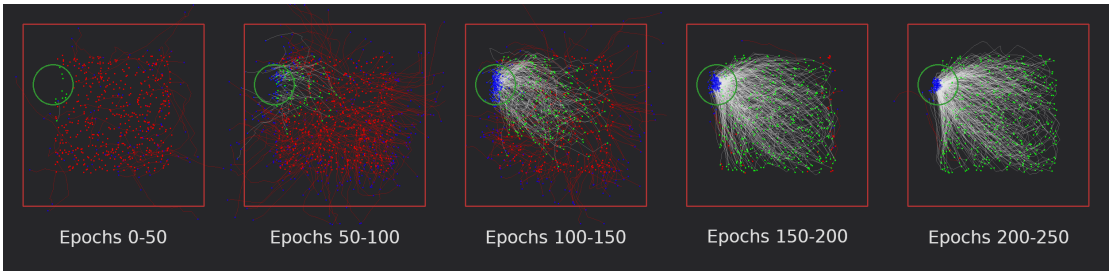


Figure 5.10 Policy behaviour evolution during training. The green circle and the red rectangle indicate respectively the goal area and the manipulation area. Green dots and red dots indicate respectively the initial positions of successful and failed cube trajectories. Blue dots indicate final cube positions. White lines indicate successful trajectories and red lines indicate failed trajectories. Each image shows 500 trajectories collected during training, each image covers 50 epochs, where at epoch 32 episodes are collected parallely.

5.2.2 Using Intrinsic Rewards

An alternative approach to solving the sparse reward problem is to introduce *intrinsic rewards*. Intrinsic rewards are rewards formulated to encourage exploration in a task agnostic manner. With a well-formulated intrinsic reward an agent could ideally learn to explore and cover the state-space of a problem without receiving any task-specific reward signal. To incentivize exploration of the state space an intrinsic reward function has to give greater rewards to states that have been seen less and smaller rewards to states that the agent has already encountered.

In problems with small and discrete state spaces this can achieved by counting how many times the agent visits a specific state. In continuous non-tabular setting this concept can be extended with pseudo-count formulation (Bellemare et al., 2016). However count and pseudo-count approaches are difficult to scale to complex tasks that require large amounts of experience. Alternative approaches use the generalization error of learned models as an estimator of state novelty. The work of Stadie et al. (2015), used the prediction error of a learned dynamics model to derive an indicator of the novelty states. As the agent visits a state-space neighbourhood more often, the dynamics model becomes more accurate in this region and its error decreases, consequently, states with low modeling error can be expected to not be novel, states with high modeling error can be assumed to be novel and are consequently rewarded. Burda et al. (2018) took this intuition forward realizing that the same logic could be applied to any model learned using states as inputs. Random Network Distillation (RND) was defined following this intuition, using the prediction error of a network trained to distill a static secondary randomly-initialized network.

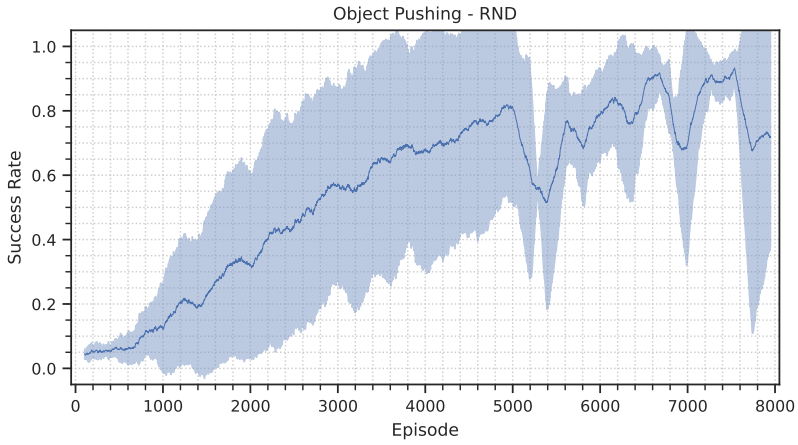


Figure 5.11 Success rate achieved during training using RND intrinsic exploration. The plot averages the success along a 200-episode window and across three runs with different random seeds. The shaded area represents a 95% confidence interval on the mean.

RND is an extremely simple and reportedly effective method to generate intrinsic rewards. As such it was combined with the DVAE-SAC architecture by applying it to the learned latent state representation. This showed to be very effective, allowing to solve the almost-sparse reward task without using huge amounts of data and a single environment. The combination of RND with the simulation parallelization approach of section 5.2.1 showed to improve performance and stability.

Figure 5.11 presents the learning curve for the training of the object-pushing task with RND intrinsic exploration, with a single training environment.

5.2.3 Training Difficult Exploration Tasks

To further explore the capabilities of the DVAE-SAC method when learning from scratch the method was evaluated on a complex exploration problem. The object pushing task was extended to include a "gate" in the manipulation area, an obstacle that the agent has to move to successfully push the cube to the goal. In practice this was implemented by placing rectangular box at the center of the manipulation area, connected to a prismatic link allowing it to slide left an right. With this configuration the manipulation area results to be divided in two parts by the gate, which can allow passage or not depending on its position. To handles were placed on the gate to allow the robot end effector to push it. As in the previous scenario, cube and end-effector initial positions are chosen randomly, as such at the beginning of the episode they an each be on either side of the gate. Figure 5.12 displays a frame sequence of the robot successfully opening the gate and pushing the cube to the goal. The reward function for this task was kept unaltered from the simpler object-pushing task of previous sections, no explicit reward for the gate opening was introduced.

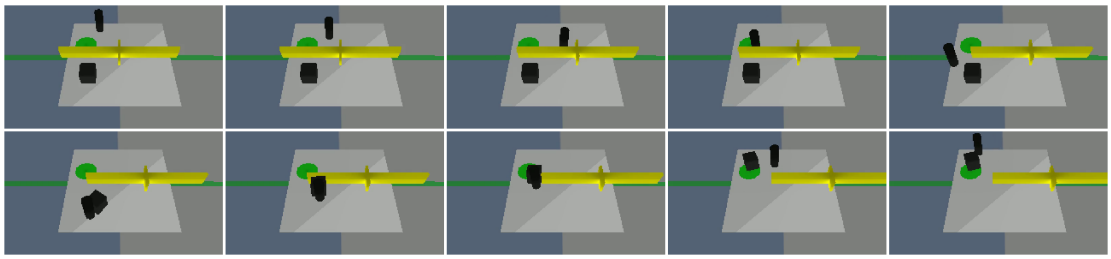


Figure 5.12 Frame sequence depicting a DVAE-SAC trained policy successfully opening the gate and bringing the cube to the goal.

This task is particularly complex from an exploration point of view, as the agent must learn to perform a complex interaction with the environment, which is not directly rewarded. The agent must explore the environment, learn that it has to push the cube, learn that it must

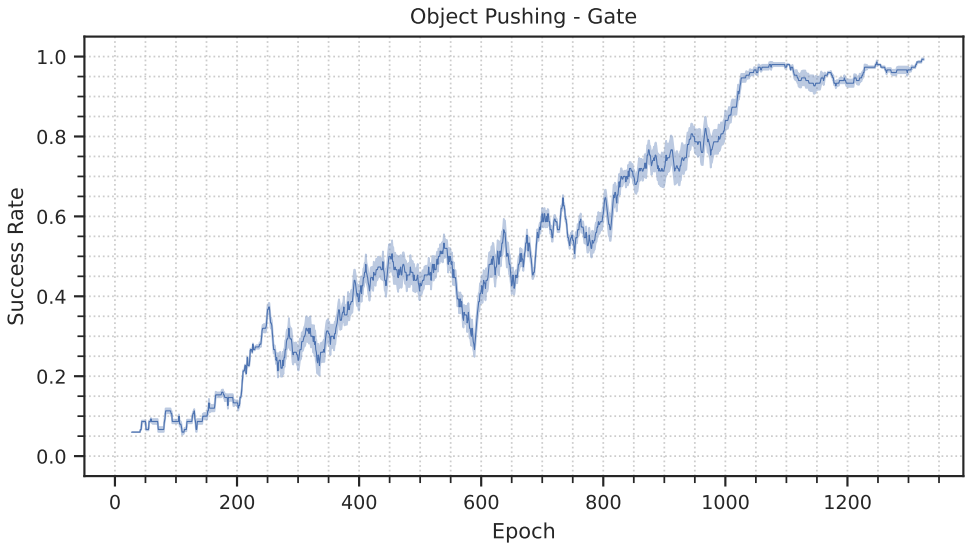


Figure 5.13 Success rate achieved during training on the gate task, average performance of three different trainings. Each epoch corresponds to the collection of 32 episodes of 100 steps. The plot averages the success along a 100-episode window, the shaded area represents a 95% confidence interval on the mean.

push it toward the goal, then it must learn that it can push the gate and that this allows to bring the cube toward the goal even if it is on the wrong side of the manipulation area. All this while using image observations.

The DVAE-SAC architecture proved to be capable of solving this problem by parallelizing experience collection and using RND intrinsic rewards. Figure 5.13 shows the training performance achieved using RND, batches of 2048 samples for the SAC policy training and 32 parallel environments. Figure 5.14, gives a qualitative idea of the policy behaviour by showing sampled cube trajectories occurred during training.

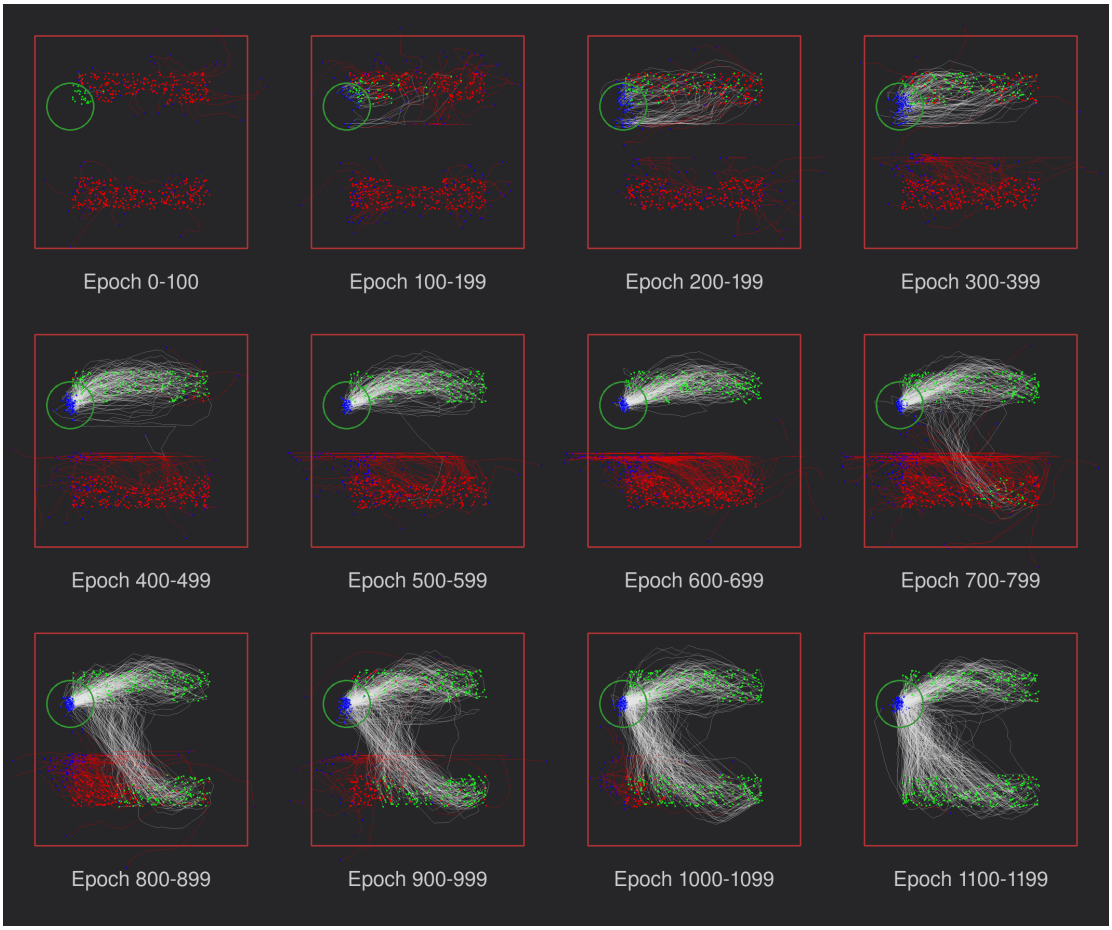


Figure 5.14 Policy behaviour evolution during training of the gate task. Each image displays 500 trajectories collected while training. The color scheme is the same as in figure 5.10.

5.3 Explicitly Constraining Decoupled Transfer

As mentioned in the introduction to this chapter, the approach to domain transfer proposed in chapter 4, while effective, relied on the presence of shaped rewards to maintain consistency in the latent representation and thus successfully performing domain transfer. This section proposes an approach for overcoming this limitation, making the methodology more general, and opening to the possibility of applying the proposed architecture to more varied tasks. The core idea of this methodology is to push the latent representation to be similar between target and source domain by using the source-domain DVAE model as a constrain during target domain training, in a similar fashion to cycle-consistency losses used in generative models (Zhu et al., 2017). As we will see, combining this approach with the latent prediction constraining of chapter 4 showed to be extremely effective. It allowed the architecture to transfer without the use of shaped reward, while at the same time considerably improving transfer sample efficiency and stability.

The next section presents the formulation and detail the implementation of this new methodology, while section 5.3.2 presents new experimental results.

5.3.1 Consistency Losses

As we discussed in the 4 chapter, the decoupled nature of the DVAE-SAC architecture allows to fully separate the section of the agent dedicated to interpreting observations, the latent extractor, from the section dedicated to selecting actions, the control policy. This permits to perform domain transfer of these two parts separately, potentially reducing considerably the real-world data required to deploy the agent in the real. For example, by performing the transfer adapting the latent extractor but keeping the control policy unchanged.

The difficulty when performing decoupled domain transfer is to maintain the compatibility between the representation produced by the latent extractor and control policy. If the representation produced by the latent extractor changes, the control policy will inevitably fail, as its input is not anymore in the form it had during training.

The methodology proposed in chapter 4 took as an hypothesis that the dynamics would not change much between the domains, and, relying on this fact, the dynamics were used as a constrain on the latent representation. In practice however, this formulation turns out to heavily rely on the presence of a shaped reward, strongly dependant on the important features of the state space. The presence of a strongly shaped reward, which does not change between target and source domain, acts as a supervision when performing transfer,

connecting matching states between the two domains. Once this supervision is removed by using a sparse or almost-sparse reward, the methodology of chapter 4 cannot correctly adapt anymore. In the case of the object-pushing task taken in consideration in chapter 4 this hypothesis was verified, as the reward depended on the cube and the end-effector positions, the two most important features of the state space. However, in the general case this may not be true.

To overcome this limit, and strengthen the general domain transfer capabilities of the method, the target domain latent representation can be more explicitly constrained toward being consistent with the one of the source domain. This can be accomplished by combining the source-domain and target-domain latent extractors to perform a conversion of observations between domains. On an intuitive level, this approach is based on the fact that, in the ideal case in which the two latent extractors for the two domains were to use the same latent representation, it would be possible to encode latent target-domain observations into latent vectors and decode these latent vectors into source domain observations using the source-domain decoder. By using target and source decoder and encoder in the opposite way it would also be possible to convert source-domain observation to the target domain. Formally this can be expressed as follows:

$$\begin{aligned} o_t &= d^*_{\theta_t}(e^*_{\theta_s}(o_s)) \triangleq c^*_{s\to t}(o_s) \\ o_s &= d^*_{\theta_s}(e^*_{\theta_t}(o_t)) \triangleq c^*_{t\to s}(o_t) \end{aligned} \tag{5.6}$$

With o_t and o_s being respectively corresponding target-domain and source-domain observations, $d^*_{\theta_t}$, $e^*_{\theta_t}$, $d^*_{\theta_s}$, $e^*_{\theta_s}$ being respectively ideal decoder/encoder pairs in the target and source domains, and $c^*_{s\to t}$ and $c^*_{t\to s}$ being newly defined conversion functions.

From this formulation we can see that, under these conditions, the composition $c^*_{s\to t} \circ c^*_{t\to s}$ of the conversion expressions should be an identity, as we are cycling back to the same observation domain. This fact can be used as a constrain on the latent space.

Following this reasoning we introduce a cycle consistency loss in the target domain training, reconstructing observations through the $c_{\theta,s\to t} \circ c_{\theta,t\to s}$ network [1]. In practice we want to train the target domain DVAE to match an already-trained source domain model. To do so, the weights of the source-domain DVAE can be frozen, so that the source-domain model parameters θ_s do not change, and consequently e_{θ_s} and d_{θ_s} remain fixed while the

[1]The conversion networks $c_{\theta,s\to t} \circ c_{\theta,t\to s}$ are indicated for brevity as being parameterized by θ. In actuality, to be more precise, they are both parameterized by θ_t and θ_s, the parameters respectively of the target-domain and source-domain DVAE architectures.

target domain architecture uses them. Considering a single target-domain observation sample o we can formally express this loss, which we refer to as the *target reconstruction consistency loss*, as follows:

$$\begin{aligned}\mathcal{L}_{cons,rec,trg}(\theta_t;o) &= MSE\left(o, c_{\theta,s\to t}(c_{\theta,t\to s}(o))\right)\\ &= MSE\left(o, d_{\theta_t}\left(e_{\theta_s}(d_{\theta_s}(e_{\theta_t}(o)))\right)\right)\end{aligned} \tag{5.7}$$

In practice, this has been used in conjunction with the multi-step architecture from section 5.1.1. As such, for every training sample this target reconstruction consistency loss has been applied to all the observations along the trajectory, resulting, in the case of trajectory of length T, in a per-sample loss defined as:

$$\mathcal{L}_{cons,rec,trg}(\theta_t;o_{0..T-1}) = \frac{1}{T}\sum_{i=0}^{T-1} MSE\left(o_i, c_{\theta,s\to t}(c_{\theta,t\to s}(o_i))\right) \tag{5.8}$$

This is the core idea of this adaptation method, however, using only this loss to constrain the latent space lets the target-domain latent extractor free to choose a latent representation that does not use the source-domain decoder and encoder appropriately. It could converge into using latent vectors that are outside of the distribution the source-domain decoder has been trained on, and is thus capable of decoding appropriately. This would result in the

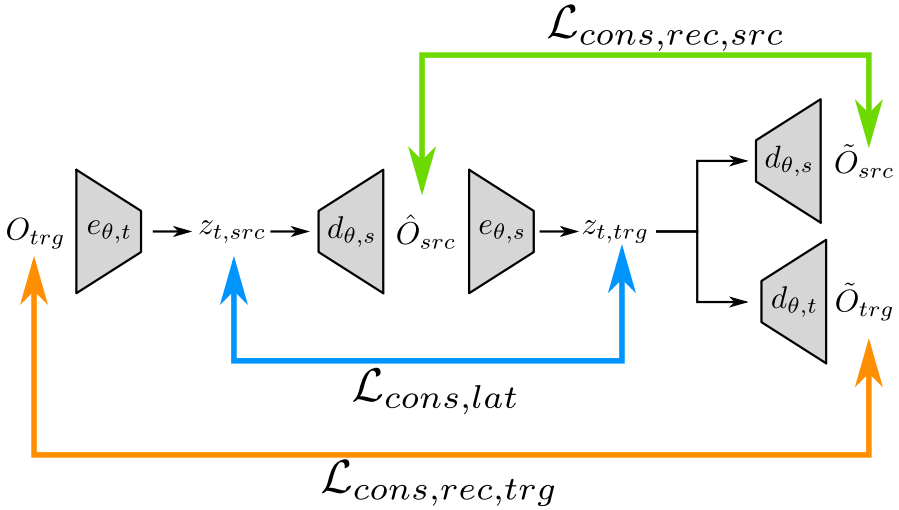


Figure 5.15 Graphical representation of the application of the consistency losses on the source and target domain encoder/decoder pairs.

source-domain decoder producing incorrect and uninformative converted observations, and consequently failing to provide a useful training signal. To prevent this, two additional losses can be introduced. The first, which we refer to as the *latent consistency loss* is aimed at maintaining the composition of source-domain decoder end encoder $d_{\theta_s} \circ e_{\theta_s}$ close to the identity function for the vectors in the latent distribution. In practice this mean that source-domain observations produced from target-domain latent vectors should be re-encoded to the same latent vector when using the source-domain encoder. The second one, named the *source reconstruction consistency loss*, aims instead at maintaining the composition $e_{\theta_s} \circ d_{\theta_s}$ close to an identity, which in practice means that observations converted from target to source domain should still be reconstructed properly when they go through the source-domain autoencoder pipeline. The two losses are also applied to observation trajectories, and can be expressed formally as follows:

$$\begin{aligned} \mathcal{L}_{cons,rec,src}(\theta_t; o_{0..T-1}) &= \frac{1}{T}\sum_{i=0}^{T-1} MSE\left(c_{\theta,t\to s}(o_i), d_{\theta_s}\left(e_{\theta_s}\left(c_{\theta,t\to s}(o_i)\right)\right)\right) \\ \mathcal{L}_{cons,lat}(\theta_t; o_{0..T-1}) &= \frac{1}{T}\sum_{i=0}^{T-1} MSE\left(e_{\theta_t}(o_i), e_{\theta_s}\left(c_{\theta,t\to s}(o_i)\right)\right) \end{aligned} \tag{5.9}$$

In figure 5.15 it is possible to see a graphical representation of the architecture indicating how the consistency losses are applied. One final detail that was found to be important is that to maintain an equivalence between source and target domain trainings, it is useful to apply the consistency losses also during the source domain. Using the source-domain model itself as the imitated architecture, enforcing consistency of the model with itself. This showed to be an important detail for the effectiveness of the method.

5.3.2 Experiments

The architecture was evaluated on the robotic object pushing task already used in chapter 4. The analysis concentrated on the almost-sparse reward case, in which the reward depends only on the distance between the cube and the goal. The reward was defined as follows:

$$r(p_c, p_g) = 100\frac{0.5 - ||p_c - p_g||}{0.5} \tag{5.10}$$

With p_c and p_g being respectively the cube position and the goal position

The source training were performed on the simplified PyBullet scenario introduced in section 5.2, parallelizing the experience collection with 32 concurrent simulations. With this setup it was possible to train the image-based almost-sparse reward task to a 100% success rate in just under 3 hours.

Using this source training setup the domain transfer was evaluated targeting a series of scenarios implemented in PyBullet, Gazebo and constructed in the real. Two sets of tasks were defined for each scenario, the first being a minimal-gap case, in which the target characteristics are kept as similar as possible to the source domain, the second instead introducing camera pose changes equivalent to the medium-gap tasks of chapter 4, with the camera point of view rotating of about 25 degrees.

In all the scenarios the method was used by training on multi-step trajectories of a variable length sampled between 1 and 10 at each iteration, using 64x64 pixel grayscale images and transferring by using the consistency losses while also freezing the dynamics prediction network as was done in chapter 4. The combined use of consistency losses and dynamics freezing proved to be crucial for successful transfer.

In figure 5.16 it is possible to see how the use of the consistency losses results in fast and stable transfer even across difficult scenarios. The most simple transfers are solved in about 70 episodes, while the harder ones achieve good performance in about 150. When looking at

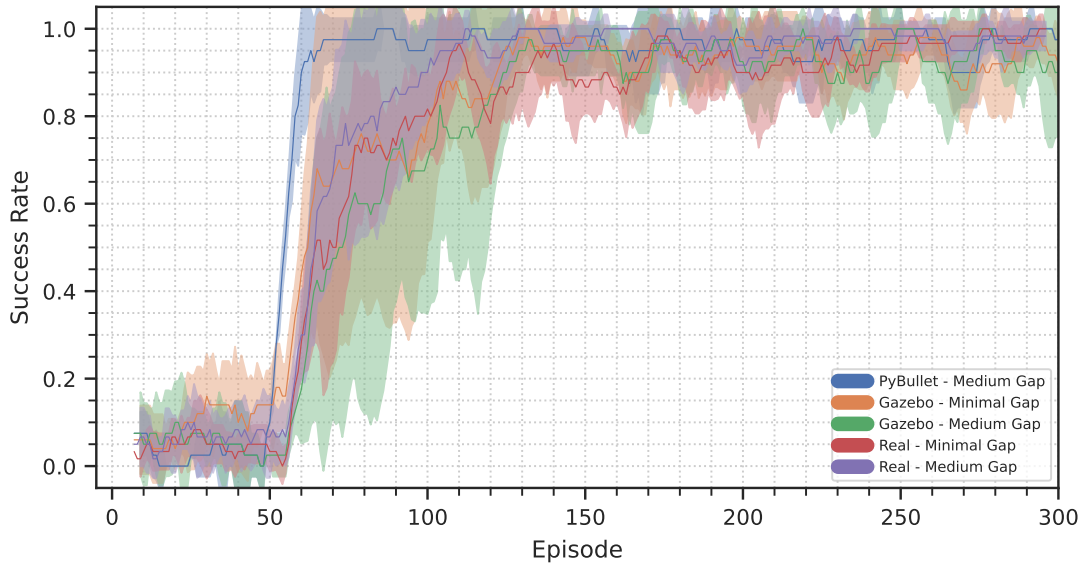


Figure 5.16 Sim-to-sim and sim-to-real transfer success rate progression. Each of the sim-to-sim plots is constructed from data collected with 4 different random seeds, the real-world plots use 5 random seeds each. Averages are computed across the different seeds and on 10-episode windows.

these results it is important to highlight that training actually starts at 50 episodes, after an initial period of random data collection.

Overall, even if tackling a harder setting than in chapter 4, the new domain transfer method shows to be faster and more stable, achieving better asymptotic performance. The real-world transfers with the old approach required more than 1000 episodes to complete, while now they can be performed in just over 100. While the experimental setup it is not directly comparable, also in the sim-to-sim transfers the performance is vastly improved (see figure 4.9).

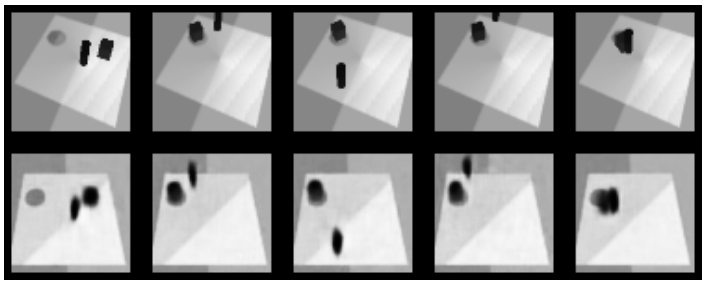
To complement the quantitative analysis, figure 5.17 provides also a qualitative outlook on the observation matching across domains, and on the kind of transfers that were performed. From these images it is possible to see how, even in the presence of strong alterations in the appearance of the observations the architecture is capable of converting the images between one domain and the other, reflecting a correct alignment between the latent spaces of target and source domain models.

5.4 Conclusions

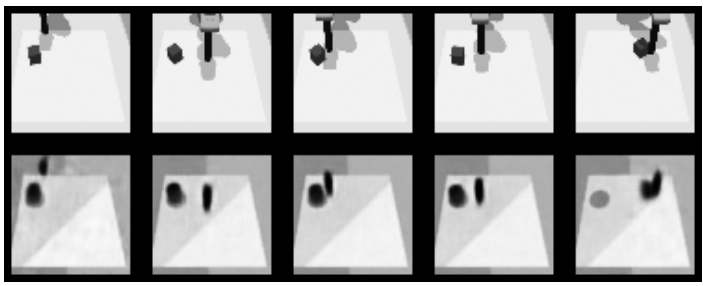
In this chapter I presented a series of enhancements to the methods proposed in chapters 3 and 4.

The first part of the chapter extended the DVAE architecture to predict observations along trajectories of multiple steps, instead of just predicting only one step forward. This reinforces on the intuition that the better policy learning performance that results from the inclusion of the dynamics predictor in the DVAE architecture comes from the necessity of including dynamics-relevant features in the latent vectors for effectively predicting observations. Predicting multiple steps forward requires a greater precision and, even more importantly, it permits the dynamics predictor to model evolutions in the observations that may not be clearly observable in 1-step transitions, such as for example low-velocity movements. This multi-step extension of the method is a superset of the methodology of chapter 3, as the old architecture can be recovered using 1-step trajectories and appropriately choosing loss weights. The multi-step architecture showed to be capable of learning policies for tasks that the original method could not solve, such as the cartpole-swingup task.

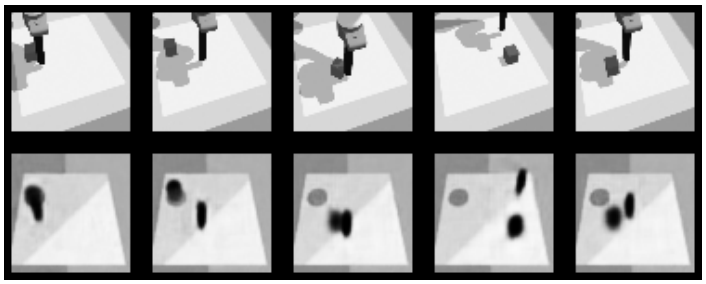
In section 5.2 I explored the capabilities of DVAE-SAC on difficult tasks, evaluating the method on sparse and almost-sparse reward versions of the object-pushing task of chapter 4, and on a difficult exploration task, in which the agent must learn to open a "gate" before



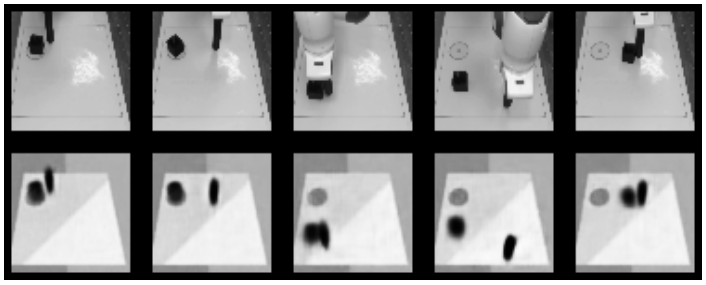
(a) PyBullet to PyBullet - Medium Gap



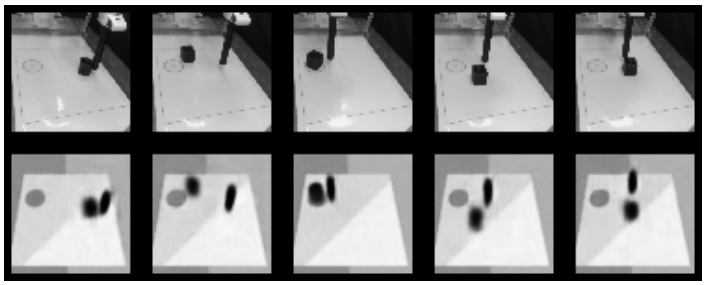
(b) PyBullet to Gazebo - Minimal Gap



(c) PyBullet to Gazebo - Medium Gap



(d) PyBullet to Real - Minimal Gap



(e) PyBullet to Real - Medium Gap

Figure 5.17 Sim-to-sim and sim-to-real frame conversion. In each figure the first row presents 5 randomly sampled target observations, the second row presents the conversion of each frame to the source domain via the DVAE architecture.

pushing the object to its target location. This last task is particularly complex, as the opening of the gate is not explicitly rewarded. The method proved to be able to solve all of these tasks if given enough data, and using intrinsic rewards, showing how the method can potentially scale to increasingly complex tasks.

Lastly, section 5.3 proposes a novel domain transfer approach for the DVAE architecture. This new strategy combines the encoder and decoder sections of the source-domain and target-domain models to convert observations between domain. Using this conversion method, cycle-consistency losses are imposed during the training to maintain compatibility in the latent space. The methodology is evaluated on sim-to-sim and sim-to-real transfers of the almost-sparse reward object-pushing task, showing greatly increased performance in comparison to the approach of chapter 4. The transfer performance is improved in both sample efficiency and asymptotic performance, succeeding at adapting through sim-to-sim and sim-to real tasks with just 2-3 hours of target experience data, while the source training required almost three days of simulated experience.

Chapter 6

Conclusions

Throughout this thesis' work I explored how Deep Reinforcement Learning methods can be utilized and adapted to serve robotics applications. The motivation for pursuing this direction came from the potential that reinforcement learning techniques have in robotics applications. In the years prior to the start of this thesis work, RL had shown to potentially have the capability of solving an extremely wide variety of tasks, being able to tackle even particularly complex problems in the fields of control, manipulation, and locomotion. This generality, came together with the possibility of tackling these problems end-to-end, directly using raw sensory inputs, such as images, to control robot hardware at a low level. These characteristics make RL an extremely promising tool for robotics. Having a reliable generic method for solving a wide variety of tasks, without the need of extensive engineering work for the definition of system models and perceptual pipelines, would allow to build robotics solutions much more easily and rapidly than current tools allow.

The issue with RL methods however is that they generally require vast amounts of data to be trained, and collecting data for real-world robotics tasks is difficult and expensive. Data requirements become too large, collection can become impractical, if not impossible. Furthermore Solving vision-based tasks, which are particularly interesting in robotics due to their generality, is especially data intensive with RL.

The goal of this thesis work was thus to explore and identify methods for reducing the amount of real-world data required for solving robotics tasks with reinforcement learning. This exploration followed two directions: the use of decoupled RL architectures to enhance visual-RL sample efficiency, and the adoption of sim-to-real techniques to reduce the use of real-world data.

Chapter 3 followed the first of these two directions and introduced the DVAE-SAC architecture. In DVAE-SAC I combined a Soft Actor-Critic agent with a representation learned based on a variational formulation task with learning a state representation from images. This representation learning approach is derived from variational autoencoders (VAE), it learns a latent representation using the evidence lower-bound loss, utilizing an image reconstruction error and a KL-divergence component. The novelty in this approach was that instead of applying this methodology just to reconstruct images, it is used to predict images one step into the future, introducing a latent dynamics predictor that uses the current latent state and action inputs to predict future latent states. While this dynamics predictor is not used neither for planning nor for generating synthetic data, its presence has a positive influence on the learning performance, potentially due to the better shaping of the latent state representation, which must contain dynamics-relevant features to be able to predict future observations. It was shown experimentally how this is true and how it results in better sample efficiency and asymptotic performance.

This approach was then improved in chapter 5, by using the dynamics predictor to predict multi-step trajectories. Predicting longer trajectories allows to better include in the latent representation features that may not be clearly identifiable by 1-step transitions, allowing for better precision and the solutions of tasks where the 1-step method failed.

Also in chapter 5, the capabilities of the DVAE-SAC method were investigated by exploring the solution of sparse reward tasks and difficult exploration problems. It was shown how the method can solve a sparse-reward object-pushing task and even an almost-sparse reward task in which the agent must learn to open a gate and then push a cube to a target location.

Chapter 4 introduced the use of sim-to-real by proposing a novel strategy for transferring across domains decoupled architectures such as DVAE-SAC. The particular architecture of DVAE-SAC decouples the learning of the agent's perceptual channel, which interprets observations into a latent representation, from the learning of the control policy. The two components are optimized concurrently, but, apart for the influence they have on their common data distribution, the training of the two is completely independent. The proposed method exploits this characteristic by performing the domain transfer of the two components separately. In exploring this methodology I concentrated on the possibility of transferring across domains by adapting the perceptual section of the agent while maintaining the control policy unchanged. This setting has the advantage of not requiring the use of RL methods in the target domain, however it requires the latent representation to remain consistent across the transfer. The core and novel idea of this approach was to use the dynamics model of

the DVAE architecture to constrain the latent state representation to remain compatible during transfer, consequently allowing the control policy to operate correctly in the target domain. This strategy shows how it is possible to transfer a decoupled architecture such as DVAE-SAC without using labeled data and without using source-domain data during the transfer. Experimentation on an object-pushing setup showed the methodology to be effective when perfoming finetuning of the source architecture to do the transfer and tackling a task with an informative shaped reward.

A substantial extension to this sim-to-real approach was proposed in chapter 5, introducing a novel strategy to further constrain the latent representation of the DVAE architecture while transferring across domains. This new strategy is based on a cycle consistency approach, and explicitly constrains latent vectors from the source-domain architecture to match those of the transferred target-domain model. Again, this is done without using any labeled data, and no source domain data is used during the transfer. This approach proved to be extremely effective, allowing to transfer tasks while using an almost-sparse reward with great data efficiency and asymptotic performance. This is a considerable enhancement from the previous approach, which relied on the presence of shaped rewards.

Overall, the contributions of this thesis work have been the following:

- The development of the DVAE-SAC architecture, from it's simplest definition in chapter 3 to its multi-step version of chapter 5. The architecture is shown to be capable of learning complex visual control tasks such as those of the DeepMind Control Suite, and even difficult exploration problems such as the gate opening an object pushing task of section 5.2.3.
- The dynamics-based domain transfer approach of chapter 4, which uses the dynamics predictor of the DVAE-SAC architecture to maintain the learned latent representation during domain transfer, without using source-domain data during transfer. This methodology was published as a journal article in *Frontiers in Robotics and AI* (Rizzardo et al., 2023)
- The cycle-consistency-based domain transfer approach presented in section 5.3, which allows to efficiently transfer the DVAE-SAC architecture across domains without the use of informative shaped rewards, without the use of any labeled data and without the use of source-domain data during target-domain transfer.

I believe these contribution to be a valuable addition to the current knowledge in the use of reinforcement learning method in robotics. While other decoupled RL methods have been

developed in these past years (Hafner et al. (2023), Lee et al. (2020), Yarats et al. (2021), Laskin et al. (2020)), the analysis of the performance of DVAE-SAC showed how the mere presence of a dynamics model in a representation learning architecture is beneficial to policy learning, and the sim-to-real approaches introduced for the same architecture showed how simulation data can be exploited in conjunction with decoupled RL architectures.

Notably, the main contribution of this thesis, the novel sim-to-real transfer methodologies of chapter 3 and section 5.3, showed how it is possible to transfer decoupled architectures by adapting latent space encoders and decoders, and transferring learned latent representations, allowing to adapt RL policies from simulation to reality without using reinforcement learning in the real.

References

Abbeel, P., Quigley, M., and Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. (2019). Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*.

Bagnell, J. A. and Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE.

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

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29.

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

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., et al. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE international conference on robotics and automation, ICRA 2018*, pages 4243–4250, Brisbane, Australia. IEEE.

Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., and Krishnan, D. (2017). Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition, CVPR 2017*, pages 3722–3731, Honolulu, Hawaii, USA.

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

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

Coleman, D., Sucan, I., Chitta, S., and Correll, N. (2014). Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. In *International Conference on Machine Learning*.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Ganin, Y. and Lempitsky, V. (2015). Unsupervised domain adaptation by backpropagation. In *International conference on machine learning, ICML 2015*, pages 1180–1189, Lille, France. PMLR.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Machine learning proceedings 1995*, pages 261–268. Elsevier.

Gordon, G. J. (1999). *Approximate solutions to Markov decision processes*. Carnegie Mellon University.

Gordon, G. J. (2000). Reinforcement learning with function approximation converges to a region. *Advances in neural information processing systems*, 13.

Guenter, F., Hersch, M., Calinon, S., and Billard, A. (2007). Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics*, 21(13):1521–1544.

Gullapalli, V., Franklin, J. A., and Benbrahim, H. (1994). Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine*, 14(1):13–24.

Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. (2017). Learning invariant feature spaces to transfer skills with reinforcement learning. In *International Conference on Learning Representations, ICLR 2017*, Toulon, France.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning, ICML 2018*, pages 1861–1870, Stockholm, Sweden. PMLR.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018b). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *International conference on machine learning, ICML 2019*, pages 2555–2565, Long Beach, California, USA. PMLR.

Hafner, D., Lillicrap, T. P., Ba, J., and Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. In *8th International Conference on Learning Representations, ICLR 2020*, Addis Ababa, Ethiopia. OpenReview.net.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. (2021). Mastering atari with discrete world models. In *9th International Conference on Learning Representations, ICLR 2021*, Virtual Event, Austria. OpenReview.net.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, 23.

Heiden, E., Millard, D., Coumans, E., Sheng, Y., and Sukhatme, G. S. (2021). Neuralsim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation, ICRA 2021*, pages 9474–9481, Xi'an, China. IEEE.

Hoeller, D., Rudin, N., Sako, D., and Hutter, M. (2023). Anymal parkour: Learning agile navigation for quadrupedal robots. *arXiv preprint arXiv:2306.14874*.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A., and Darrell, T. (2018). Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning, ICML 2018*, pages 1989–1998, Stockholm, Sweden. Pmlr.

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

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2019*, pages 12627–12637.

Jin, J., Zhang, C., Frey, J., Rudin, N., Mattamala, M., Cadena, C., and Hutter, M. (2023). Resilient legged local navigation: Learning to traverse with compromised perception end-to-end. *arXiv preprint arXiv:2310.03581*.

Kakade, S. M. (2001). A natural policy gradient. *Advances in neural information processing systems*, 14.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, California, USA.

Kober, J. and Peters, J. (2008). Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154, Sendai, Japan. IEEE.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE.

Laskin, M., Srinivas, A., and Abbeel, P. (2020). CURL: contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650, Virtual Event. PMLR.

Lee, A. X., Nagabandi, A., Abbeel, P., and Levine, S. (2020). Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems, NeurIPS 2020*, volume 33, pages 741–752, Virtual-only. Curran Associates, Inc.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

Long, M., Cao, Y., Wang, J., and Jordan, M. (2015). Learning transferable features with deep adaptation networks. In *International conference on machine learning, ICML 2015*, pages 97–105, Lille, France. PMLR.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., et al. (2021). Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning, ICML 2016*, pages 1928–1937. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop 2013*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Muratore, F., Gruner, T., Wiese, F., Belousov, B., Gienger, M., and Peters, J. (2021). Neural posterior domain randomization. In *Conference on Robot Learning, CoRL 2021*, London, UK.

Muratore, F., Ramos, F., Turk, G., Yu, W., Gienger, M., and Peters, J. (2022). Robot learning from randomized simulations: A review. *Frontiers in Robotics and AI*, page 31.

Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. (2019). Deep dynamics models for learning dexterous manipulation. In *3rd Annual Conference on Robot Learning, CoRL 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 1101–1112, Osaka, Japan. PMLR.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*. ICML 2015 Deep Learning Workshop.

NVIDIA (2020). Nvidia isaac sim. https://developer.nvidia.com/isaac-sim.

Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N. A., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. M. (2019). Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE.

Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural actor-critic. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*, pages 280–291. Springer.

Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. (2018). Asymmetric actor critic for image-based robot learning. In *Proceedings of Robotics: Science and Systems, R:SS 2018*, Pittsburgh, Pennsylvania.

Possas, R., Barcelos, L., Oliveira, R., Fox, D., and Ramos, F. (2020). Online bayessim for combined simulator parameter inference and policy improvement. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020*, pages 5445–5452. IEEE.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan.

Ramos, F., Possas, R., and Fox, D. (2019). Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators. In Bicchi, A., Kress-Gazit, H., and Hutchinson, S., editors, *Robotics: Science and Systems XV, R:SS 2019*, Freiburg im Breisgau, Germany.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*, pages 317–328. Springer.

Rizzardo, C., Chen, F., and Caldwell, D. (2023). Sim-to-real via latent prediction: Transferring visual non-prehensile manipulation policies. *Frontiers in Robotics and AI*, 9:1067502.

Rudin, N., Hoeller, D., Bjelonic, M., and Hutter, M. (2022). Advanced skills by learning locomotion and local navigation end-to-end. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2497–2503. IEEE.

Rudin, N., Hoeller, D., Reist, P., and Hutter, M. (2021). Learning to walk in minutes using massively parallel deep reinforcement learning. In *5th Annual Conference on Robot Learning, CoRL 2021*, London, UK.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1985). Learning internal representations by error propagation.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

Sun, B. and Saenko, K. (2016). Deep coral: Correlation alignment for deep domain adaptation. In Hua, G. and Jégou, H., editors, *Computer Vision – ECCV 2016 Workshops*, pages 443–450, Amsterdam, Netherlands. Springer International Publishing.

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

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Tedrake, R., Zhang, T. W., Seung, H. S., et al. (2005). Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, volume 95585, pages 1939–1412. Beijing.

Tesauro, G. et al. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30.

Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S., Saenko, K., and Darrell, T. (2016). *Adapting Deep Visuomotor Representations with Weak Pairwise Constraints*, pages 688–703. Springer International Publishing, San Francisco, California, USA.

Tzeng, E., Devin, C., Hoffman, J., Finn, C., Peng, X., Levine, S., Saenko, K., and Darrell, T. (2015). Towards adapting deep visuomotor representations from simulated to real environments. *CoRR*, arXiv preprint arXiv:1511.07111.

Tzeng, E., Hoffman, J., Saenko, K., and Darrell, T. (2017). Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Honolulu, Hawaii, USA.

Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darrell, T. (2014). Deep domain confusion: Maximizing for domain invariance. *CoRR*, arXiv preprint arXiv:1412.3474.

Unity (2020). Unity. https://unity.com/solutions/automotive-transportation-manufacturing/robotics.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wahlström, N., Schön, T. B., and Deisenroth, M. P. (2015). From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2010). Optimizing walking controllers for uncertain inputs and environments. *ACM Transactions on Graphics (TOG)*, 29(4):1–8.

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

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

Witten, I. H. (1977). An adaptive optimal controller for discrete-time markov environments. *Information and control*, 34(4):286–295.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., and Fergus, R. (2021). Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 10674–10681, Held Virtually. AAAI Press.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232.