THE UNIVERSITY
of EDINBURGH

Generalisation in Deep Reinforcement Learning with Multiple Tasks and Domains

Chenyang Zhao

Doctor of Philosophy

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

2020

Abstract

A long standing vision of robotics research is to build autonomous systems that can adapt to unforeseen environmental perturbations and learn a set of tasks progressively. Reinforcement learning (RL) has shown great success in a variety of robot control tasks because of recent advances in hardware and learning techniques. To further fulfil this long term goal, generalisation of RL arises as a demanding research topic as it allows learning agents to extract knowledge from past experience and transfer to new situations. This covers generalisation against sampling noise to avoid overfitting, generalisation against environmental changes to avoid domain shift, and generalisation over different but related tasks to achieve lifelong knowledge transfer. This thesis investigates these challenges in the context of RL, with a main focus on cross-domain and cross-task generalisation.

We first address the problem of generalisation across domains. With a focus on continuous control tasks, we characterise the sources of uncertainty that may cause generalisation challenges in Deep RL, and provide a new benchmark and thorough empirical evaluation of generalisation challenges for state of the art Deep RL methods. In particular, we show that, if generalisation is the goal, then the common practice of evaluating algorithms based on their training performance leads to the wrong conclusions about algorithm choice. Moreover, we evaluate several techniques for improving generalisation and draw conclusions about the most robust techniques to date.

From the evaluation, we can see that learning from multiple domains improves generalisation performance across domains. However, aggregating gradient information from different domains may make learning unstable. In the second work, we propose to update the policy to minimise the sum of distances to the new policies learned in each domain in every iteration, measured by Kullback-Leibler (KL) divergence of output (action) distributions. We show that our method improves both the training asymptotic reward and testing policy robustness against domain shifts in a variety of control tasks.

We finally investigate generalisation across different classes of control tasks. In particular, we introduce a class of neural network controllers that can realise four distinct tasks: reaching, object throwing, casting, and ball-in-cup. By factorising the weights of the neural network, transferable latent skills are exacted which enable acceleration of learning in cross-task transfer. With a suitable curriculum, this allows us to learn challenging dexterous control tasks like ball-in-cup from scratch with only reinforcement learning.

Lay Summary

A long term goal for the robotics and artificial intelligence research community is to build intelligent systems that can generalise what they have experienced and learned to various new contexts. In this thesis, we intend to address two major challenges towards this goal: (1) can robots generalise prior knowledge to function in new situations, and (2) can robots generalise prior knowledge in performing new tasks. For example, like humans, a robot agent that has learned how to walk on concrete road should also be able to walk on the beach, despite the changes in the surface conditions between the two contexts. Moreover, it should be able to transfer its knowledge acquired during walking to and apply to related tasks, such as running.

We start with evaluating well-established learning algorithms in terms of their performances in generalisation with different situations. From the empirical results, we propose a new method to improve algorithm's generalisation to new situations. Our method shows robustness in agent's learning under unseen situations, such as walking with different wind speeds. Lastly, we improve agent's learning in new related task, e.g. ball-in-cup, from existed knowledge e.g. object throwing.

Acknowledgements

I would like to express my gratitude to my supervisor Professor Timothy M. Hospedales. His insights and intuitions constantly inspire me to explore and exploit more in research. Working with him has been truly an honour and a pleasure in the past years.

Thanks to colleagues in DREAM project, especially my collaborators, Olivier Sigaud from Sorbonne Université and Freek Stulp from German Aerospace Center (DLR). It's always encouraging to continue working in this topic with your sincere help and late night discussions. I would also like to thank my colleagues in Machine Intelligence Group, and IPAB. It's lucky to being around such a group of smart and friendly cohort.

Finally, thank you all to my parents, family, friends and especially Lucy for being there for me and support me throughout the years.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Chenyang Zhao*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Generalisation in Machine Learning

Generalisation plays a crucial role in the learning process, allowing intelligent individuals to recognise the similarities in knowledge acquired in one circumstance, and to transfer knowledge onto new situations [Kolb and Whishaw, 2001]. With the long term vision of machine learning research to build self-learning and self-adaptive artificial intelligence systems, generalisation problem is always a research direction of vital importance. Considering a robot learning to throw a ball to a target basket as an example, current generalisation problems can be further broken down to three categories given the differences between training and testing conditions: (1) generalisation within domain and task, where the robot can hit the target multiple trials with the same configuration, (2) generalisation across domains, where the robot can perform robustly against unforeseen perturbation, e.g., wind conditions, and (3) generalisation across tasks, where the robot transfer the knowledge to a new task, e.g., catching a ball.

Most machine learning algorithms are trained with a finite set of training data. With the assumption that training and testing data are randomly sampled from the same distribution, the goal is that the model learned with training data can also perform well on predicting outcomes given unseen testing data. However, in practice, the model could fit too closely to training data and therefore fail in generalising to testing scenarios. This problem is commonly addressed as *overfitting* in machine learning. To avoid overfitting to training data, generic techniques include regularisation, such as dropout [Srivastava et al., 2014] and L_2 regularisation [Bishop, 2006].

Different from overfitting problems, the distribution of data in testing conditions could be different from training, which results in a domain shift problem [Sugiyama

and Storkey, 2007]. For example, a robotic perception system well trained with indoor data may fail badly when deployed in outdoor scenarios due to changes in weather conditions. From this example we can see that, rather than focusing on training conditions only, it is essential to be able to generalise knowledge across different domains. One method to tackle this problem is so-called *domain adaptation*. It assumes that a small amount of labelled data or large amount of unlabelled data in target domain is available which helps the learned model to adapt specifically to target domain [Csurka, 2017a; Yang and Hospedales, 2017b; Finn et al., 2017]. In the case where no data in target domain is available, the objective is to improve the performance of a held-out target domain, without adaptation. *Domain generalisation* addresses this problem by designing algorithms such that learned policies can extract generalised knowledge across training domains and perform more robustly against domain shifts [Muandet et al., 2013; Li et al., 2018a].

Generalisation may not only occur within a task, but also across tasks with different objectives. An example setup would be a robot perception system which learns to recognise and manipulate a cube on the table, and then changes the task to recognise and push a round-shaped button. Cross-task generalisation aims to enable a system benefit from knowledge learned from previous tasks, through feature extractor for example, rather than learn from a tabula rasa. Given different ultimate goals, generalisation across tasks can be further broken down into two categories: *multi-task learning*, where agents co-learn a set of tasks simultaneously and optimise performance across all tasks through some shared knowledge; and *transfer learning*, where the focus is to abstract generalised knowledge from one or a few source tasks and to use the knowledge to improve performance in target tasks. Comparing to multi-task learning, learning performance in source tasks is irrelevant in the context of transfer learning. The scheme of transfer learning has been used in various applications, including computer vision [Yosinski et al., 2014; Yang and Hospedales, 2017b], natural language processing [Ruder, 2017; Ruder et al., 2017], etc.

Table 1.1 summaries generalisation problems in different settings and notable corresponding techniques. In this thesis, we focus on (1) generalisation across domains, especially when data in target domains is unavailable for adaptation, i.e. the *domain generalisation* settings; and (2) generalisation across tasks, where agents use knowledge learned from source tasks to bootstrap learning in target tasks.

Within Task and Domain (Overfitting)	Across Domains (Domain Shift)	Across Tasks (Knowledge Sharing)
Regularisation	Domain Adaptation	Multi-task Learning
	Domain Generalisation	Transfer Learning

Table 1.1: A summary of generalisation problems and techniques in different scales.

1.2 Generalisation in Reinforcement Learning

In Reinforcement Learning (RL) problems, agents learn to take a sequence of actions so as to maximise the cumulative reward. Different from classic dynamic programming methods, which require as input a mathematical model of task specification, RL methods generally learn without any prior understanding of tasks but explore the world by interacting with environment. Due to this generality, RL has been studied intensively in the field of robot control, especially when the exact model for specific task is complex and unavailable, e.g., inverted helicopter task [Ng et al., 2006] and ball-in-cup task [Kober and Peters, 2009]. With the recent emergence of deep learning techniques and growth of computational power, using deep neural networks as function approximators has achieved great success in solving many simulated control problems such as locomotion tasks [Schulman et al., 2015a, 2017; Lillicrap et al., 2015].

Despite the recent successes, the mainstream of RL research focuses on optimising the training return objective function and evaluating its performance over training conditions only, rather than its generalisation characteristics. In this thesis, we focus on investigating generalisation in RL, more specifically, domain generalisation and knowledge transfer among tasks.

1.2.1 Formalising Domains and Tasks in RL

Before introducing generalisation in RL, we start with formalising the problem of RL and distinguishing the terms of *domain* and *task* in the context of this thesis.

1.2.1.1 Markov Decision Process

The basic setting of RL is to learn a policy for making decisions from interactions. The learner and policy together is called the *agent*, and what the agent interacts with is called the *environment*. As in Figure 1.1, at each time step, the agent makes decision of an action given a state, and as a consequence of the action, the environment rolls out

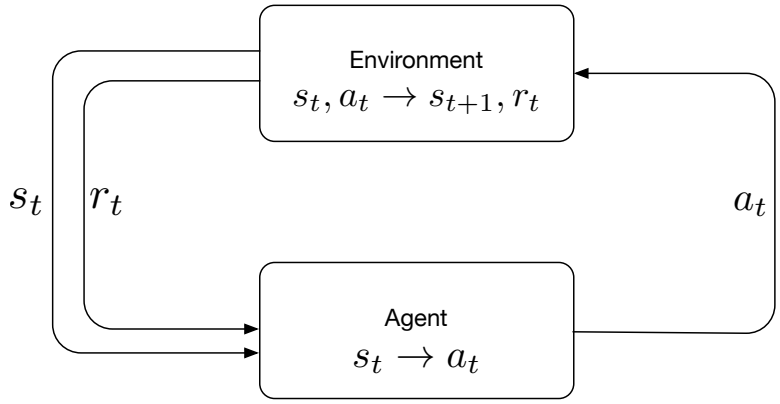


Figure 1.1: Block diagram of agent-environment interaction in reinforcement learning.

the state in next time step and provides a corresponding reward.

The environment is typically formulated as a Markov Decision Process (MDP). A MDP is a mathematical object that describes a stochastic environment. It is usually denoted as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where

- $\mathcal{S}$, a state space that contains a set of states.
- $\mathcal{A}$, an action space that contains a set of actions.
- $P(s_{t+1}|s_t, a_t)$, a transition model that describes the probability distribution of state at next time step $t+1$ given state s and action a at current time step t.
- $R(s_t, a_t, s_{t+1})$, a reward function that gives immediate reward for each tuple of (s_t, a_t, s_{t+1}).
- γ, a discount factor between 0 and 1.

In some settings, the MDP could be expanded with an initial state probability distribution $P_0(s_0)$.

1.2.1.2 Formalising Domains and Tasks

In the context of supervised learning, the problem is described with a combination of domain and task. Pan and Yang [2010] defined a domain $\mathcal{D}$ with two components: a feature space $\mathcal{X}$ and a marginal probability distribution of features $P(X)$, and a task $\mathcal{T}$ with two components: a label space $\mathcal{Y}$ and an objective predictive function $f(\cdot)$. The function $f(\cdot)$ can be used to predict the corresponding label, $f(x)$, of a new instance x. From a probabilistic viewpoint, $f(x)$ can be written as $P(y|x)$. In summary, a domain describes the input data, and a task describes the supervision signal. Usually, transferring to new task requires labelled data for new task, while transferring to new domain could be achieved without labelled data by sharing domain-agnostic knowledge.

Similarly, we describe the problem of an MDP with a domain that shows the distribution of data and a task that supervises the learning. More specifically, we define a domain with the components that influence the rolling-out of a trajectory $\tau = \langle (s_0, a_0); (s_1, a_1); ... \rangle$, including the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition model P, and the initial state probability distribution P_0; and define a task with the components that change the supervision, including the reward function R. For example, consider a robot arm manipulation task, changing the weight of an arm component (changing P) or a breakdown of the second joint such that it cannot bend counter-clockwise (changing $\mathcal{A}$) is considered as a domain shift, and transferring from picking up an object to pushing an object (changing R) is considered as a change in task.

1.2.2 Generalisation Across Domains

As a long stand vision in developing intelligence systems, we would like our RL agents to perform robustly against potential domain shifts. However, RL algorithms are often at risk of overfitting to training domains – learning policies overly specific to their training environment and failing to generalise to new conditions. It is especially problematic for agents with high capacity models, such as deep neural networks. Due to the relatively greater difficulty of obtaining a good solution to the RL training problem in the first place, evaluating for generalisation to novel conditions through such train/test splits is not common practice in Deep RL. Correspondingly, mainstream Deep RL algorithm research focuses on optimising the training condition well, rather than developing models that generalise well to novel conditions. Nevertheless, now that Deep RL training is increasingly successful, it is timely to move focus onto models' generalisation properties. Achieving generalisation is crucial if Deep RL should move out of the the lab and solve real-world problems where noise and uncertainty are intrinsic, and novel conditions will certainly be encountered [Sunderhauf et al., 2018].

Aside from first principle interests in the ability of our agents to succeed in diverse and novel environments, there is particular demand for generalisation in the context of the *reality gap* in robotics. Despite continuing improvements in algorithmic sample efficiency, Deep RL requires a large number of environmental interaction samples for learning complex tasks without prior knowledge. For this reason, the majority of Deep RL training is done in simulation, which is moreover usually deterministic or implemented with pre-defined noise. However, it is generally impossible to accurately model the real-world environments with simulators, i.e., the domain shifts between

training domains (simulated environments) and testing domains (real-world application) are generally inevitable. Such unknown domain shifts should be taken into consideration when deploying learned policy to real world. In the literature of robotics, deploying policies from such simulated training environments to potential real-world deployment is known as crossing the reality gap, and is well known to be difficult [Koos et al., 2013], thus providing an important motivation for studying generalisation across domains.

This has motivated a fruitful line of research targeting simulation to reality transfer, or *sim2real*. Based on whether data from target domain is available during training, the problem can be further divided into two categories: *domain adaptation* and *domain generalisation*. Domain adaptation assumes that small amount of data in target domain is available. Tzeng et al. [2015] and Gupta et al. [2017] encouraged the agents to learn similar embeddings for samples from different domains; Rusu et al. [2016b] used the progressive net to transfer learned policies from simulation to reality; Christiano et al. [2016] trained an inverse dynamics model from real world data to help transfer from simulation to a real robot. On the other hand, domain generalisation problem requires learned agents to perform well in target environments without access to target domain data. Corresponding methods usually generate a set of domains randomly and assume discrepancies between the source and target domains are modelled as variability in the set of domains. These methods are also called *domain randomisation* methods. The models trained with randomised domains then are directly applied to real world scenarios without adaptations. Successful applications include object manipulation [Tobin et al., 2017], in-hand manipulation [Andrychowicz et al., 2020; Akkaya et al., 2019], etc. One of the major limitations is that this usually requires a significant amount of manual tuning and a tight iteration loop between randomisation design in simulation and validation on a robot. In the thesis, we mainly cover the problem of *domain generalisation* in Chapter 3 and 4.

For RL control agents, there are several different challenges that can arise. The first is generalisation from a deterministic training environment to a noisy and uncertain testing environment, for example in the form of real sensor and actuator noise. Secondly, assuming we correctly model environmental variability in our training simulation, there is the question of whether an agent learns to generalise to future conditions drawn from the same distribution, or overfits to its specific training experiences [Zhang et al., 2018b]. Finally, there is the subtle but important point that no matter the effort applied to modelling environmental conditions and variability in simulated training, it

is generally impossible to predict and accurately model the environmental conditions and variability an agent might encounter in the real world [Koos et al., 2013]. Therefore an important way to think about model generalisation is not only robustness to overfitting per-se, but generalisation under some level of *domain shift*. In supervised learning, domain shift refers to changes in the data distribution which we would like a predictive model to be robust to, for example the type of camera in visual object recognition [Csurka, 2017b]. The corresponding notion in RL is that we would like our policy's success to be invariant to nuisance changes in the environment [Cully et al., 2015]. These could span both noise, for example sensor, actuator, and environmental noise; and variability, for example camera type, initial state of an agent, or mass of an objects being manipulated.

With the advances in Deep RL, the subject of generalisation against domain shifts has gained momentum recently [Zhang et al., 2018a; Packer et al., 2018; Zhang et al., 2018b]. In Chapter 3, we aim to study generalisation in Deep RL for continuous control, with a particular focus on robustness to domain shift between training and testing. We provide a thorough characterisation of domain shifts challenges in continuous RL and an evaluation of several state of the art Deep RL methods in terms of their generalisation properties. Moreover, we evaluate several existing techniques that might improve generalisation across domains, including different policy architecture designs, regularisation, etc.

To step further from evaluating to improving, in Chapter 4, we propose a novel framework for simultaneous learning multiple domains based on stochastic policy gradient methods. More specifically, we propose to average the policies in the probability space of action distributions given gradient information from multiple domains. We evaluate the proposed method in two continuous control benchmarking tasks from OpenAI Gym [Brockman et al., 2016]: *Walker* and *Hopper*. We show that this method for aggregating experience from multiple domains during training leads to both better asymptotic training performance as well as better robustness to testing in new domains.

1.2.3 Generalisation Across Tasks

Another important aspect of generalisation is to extract task-agnostic knowledge that can be shared across tasks, so as to improve sample efficiency of learning. Towards the goal of building autonomous robots, sharing and transferring knowledge across tasks is especially useful, because learning an individual challenging robot control skill usually

requires prohibitive amounts of training experience. With prior knowledge learned from different but related tasks, the agent is able to significantly bootstrap the learning for novel tasks and reduce the amount of total experience required.

The desire to increase autonomy in this way has motivated extensive work into generalising skills. For example, multi-task learning addresses sharing information across multiple skills [Deisenroth et al., 2014; Parisotto et al., 2016] and contextual (or parameterised) policies build skills that generalise across variants [Stulp et al., 2013; Kupcsik et al., 2013] within one family of tasks. Considering the task of throwing to different target locations [Kupcsik et al., 2013] as an example, the agent takes as inputs both the observation and the target position, which encodes the task variants. Other applications include moving a puck to different target positions [Kupcsik et al., 2017], going via different waypoints to avoid various obstacles [Stulp et al., 2013], etc. Most of these studies however, address generalisation of skills that are relatively simple parametric variants of each other.

In Chapter 5, we are inspired by the vision of lifelong learning [Thrun, 1996a; Ruvolo and Eaton, 2013b]. That is, the idea that new tasks should get progressively easier to learn as a wider and deeper set of prior tasks are mastered by a learner with the ability to extract task-agnostic generalisations from experience. With this in mind, we go beyond existing contextual policy work and explore transfer learning across a heterogeneous set of dynamic control tasks that do not lie in a simple parameterised family. In our setting, a robot starts with knowledge of a set of source task variants (e.g., throwing type tasks). The aim is then for it to master a different category of target task (e.g., catching type task) autonomously through RL. We explore four different categories of tasks: reaching to a target position, throwing at a target [Deisenroth et al., 2014; Kober et al., 2010], casting at a target [Kober and Peters, 2010], and ball-in-cup [Stulp et al., 2014]. Some of these are typically difficult to learn directly with RL, so to learn one task autonomously based only on past experience of another, the robot must abstract and transfer task-agnostic generalisations. To achieve this, we define a class of neural network controllers loosely inspired by the dynamic movement primitives (DMPs) [Schaal et al., 2005] commonly used to solve these dynamic tasks. We first multi-task learn a *set* of multiple source tasks from a given family (e.g., throwing objects of various weights to various locations), and the controllers for these correspond to a stack of such neural networks. Then we *factorise* this set of tasks to obtain transferrable latent skills. Finally, by using these latent skills as a basis to construct a policy network, we are able to learn a set of target tasks (such as ball-in-cup with var-

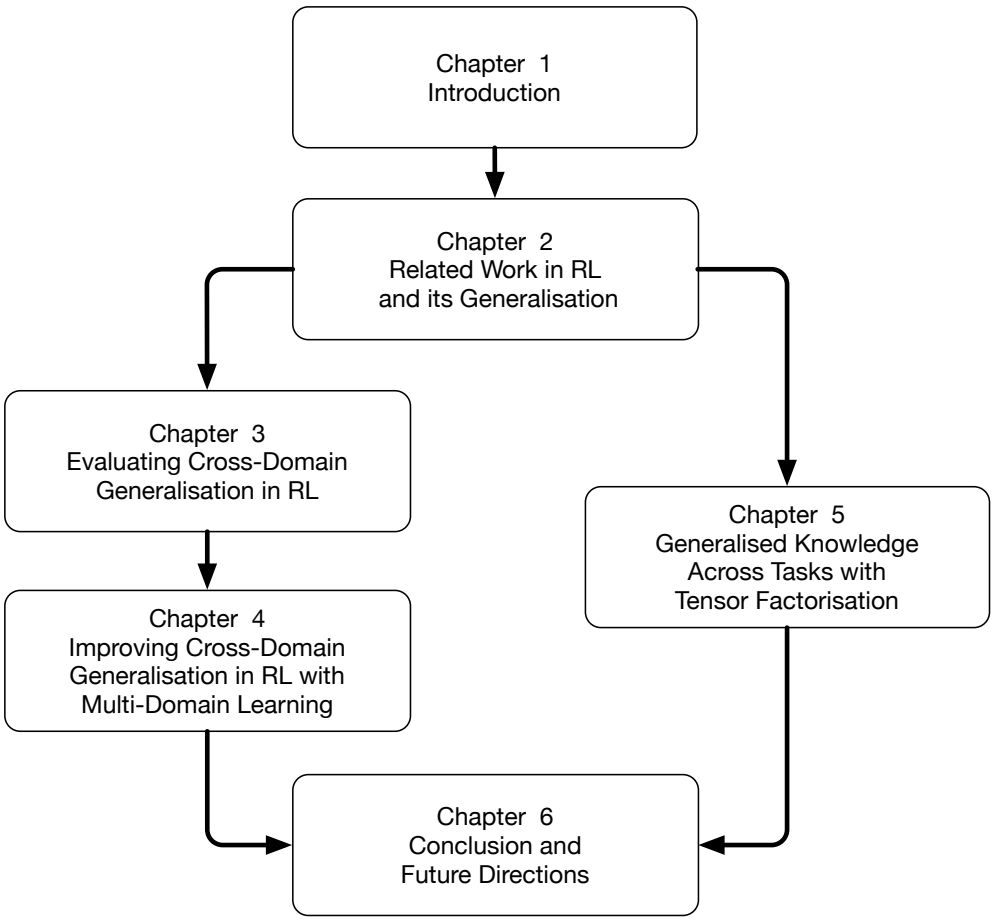


Figure 1.2: An illustration of the outline of this thesis and relation of the chapters.

ious string lengths) autonomously, without demonstration. Uniquely, our tensor-based transfer framework enables simultaneous discovery and sharing of latent skills across *both* task categories and actuators [Luck et al., 2014].

1.3 Thesis Outline

In this last section of introduction, we briefly outline the structure of this thesis and connections between remaining chapters (Figure 1.2). Overall, this thesis addresses the problem of generalisation in RL, including both cross-domain settings (Chapter. 3, Chapter. 4) and cross-task settings (Chapter. 5).

Chapter 2 provides an overall literature review on the prior work on generalisation in RL. First, we introduce the basic concepts and notable methods in RL. We next review the recent work that benchmark state of the art Deep RL algorithms, with a particular focus on evaluating generalisation. And finally, we review the techniques to-

wards improving generalisation in two categories: generalisation for multiple domains and generalisation for multiple tasks.

Chapter 3 first characterises the problem of generalisation across domains in learning continuous control tasks. Next, we provide a throughout evaluation of several state of the art Deep RL methods and other techniques in terms of their generalisation performance. The experimental results raise the concern that common practice of picking algorithms and architectures based on training performance may lead to the wrong choice in terms of generalisation performance. We then show a selection of techniques that improve generalisation across domains, as a starting point for future work. This chapter corresponds to a preprint paper, which is under review for JMLR [Zhao et al., 2019].

Inspired by one of the conclusions in previous chapter that training with multiple domains helps to learn more generalised policies, in **Chapter 4**, we propose a novel method for simultaneously learning with multiple domains based on distances between policies in statistical manifold. We show that the proposed method provides a more efficient and robust learning compared to using the average gradient in euclidean space. Moreover, the learned policies generalise significantly better against domain shifts.

In **Chapter 5**, we move from cross-domain generalisation to cross-task generalisation. We propose a novel method to abstract task-agnostic knowledge from multiple tasks and transfer the knowledge for learning novel tasks. We further show that with a designed curriculum of tasks, the agent is able to learn progressively from simple to complex tasks. This chapter corresponds to a published paper in IJCAI 17' [Zhao et al., 2017].

Finally, **Chapter 6** concludes the thesis, summarising the commonalities of the problems we address and the major contributions of this thesis, and discussing the future research directions.

Chapter 2

Related Work

In this chapter, some fundamental concepts in reinforcement learning (RL) and the related algorithms used throughout this thesis will be introduced in Section 2.1. In Section 2.2, we review different evaluations of Deep RL methods, with a particular focus on their consideration of the generalisation properties. Finally, we discuss the related work towards improving the generalisation in RL, in both cross-domain setting (Section 2.3) and cross-task setting (Section 2.4).

2.1 Background in Reinforcement Learning

Consider a continuous control task described by a MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. The general objective for RL is to find the optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximises the expected return,

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\tau \big[R(\tau) \big] \tag{2.1}$$

$$= \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\tau \big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \big], \tag{2.2}$$

where the trajectory τ consists of a sequence of state-action pairs. Note that different from the immediate reward $R(s_t, a_t, s_{t+1})$, $R(\tau)$ describes the cumulative reward of a trajectory: $R(\tau) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$.

There are two primary model-free methods in RL: value-based vs. policy-based. Value-based methods aim to build a value function which estimates how good it is for the agent to be in a state (denoted by $V^\pi(s)$) or a state-action pair (denoted by $Q^\pi(s,a)$). The policy is subsequently defined as choosing the actions that generates the highest values. Value-based methods have achieved great success in recent years,

solving various discrete tasks, such as Atari games [Mnih et al., 2015; Hessel et al., 2018]. However, these methods does not scale well for challenges encountered in robot continuous control problems. Value-based methods usually estimates value functions iteratively by bootstrapping. This leads to instabilities when combined with function approximation due to the recursive use of their own value estimates and the difficulty of maximising over actions.

Policy-based methods, on the other hand, are usually used to solve continuous control tasks [Deisenroth et al., 2013; Sigaud and Stulp, 2019]. In contrast to value-based methods, policy-based methods do not rely on estimation of state values, instead they parameterise the policy π with a vector of parameters θ and search for the optimal policy in the space of policy parameters. Given different update strategies, these methods can be divided into two categories: *evolutionary methods*, which is episode-based, gradient-free and black-box optimisation, and *policy gradient methods*, which is step- and gradient-based optimisation.

2.1.1 Evolutionary Methods

Based on the idea of mutation and selection in evolution, evolutionary methods aim to optimise the objective function without explicitly estimating the policy gradient or building a model of objective function. Instead, these methods sample policies from policy parameter space and move towards policy parameters of higher performance. We here introduce covariance matrix adaptation evolution strategy (CMA-ES), one of the most notable methods of evolutionary methods.

The general framework of evolution strategies is that at each iteration, the agent generates a population of policies around the current policy with certain noise and retains the individuals with the best performance for next iteration. The simple evolution strategy uses a fixed Gaussian noise for exploration. However, in practice, there are times when we want to explore more and increase the standard deviation of our search space, and also there are times when we are confident we are close to a good optima and just want to fine tune the solution. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Kern et al., 2004] take top k-percentile of individuals in terms of their performance, and updates both the centre and covariance matrix of search space for next generation. A dynamic visualisations and more technical details can be found in [Ha, 2017].

CMA-ES, as a black-box optimisation approach, is simple to use and implement

as it does not require backpropagating gradients and is indifference to the distribution of rewards (sparse or dense) [Salimans et al., 2017]. However, it does not scale well in high dimensional problems. One of the reasons is that CMA-ES often has low sample efficiency as it discards most of the experience in one iteration, especially when the environment is noisy, several rollouts are needed to evaluate an individual policy [Deisenroth et al., 2013]. Moreover, as a rollout has to terminate for the return to be calculated in evolutionary methods, these methods can only be used in episodic MDPs. Whereas value based methods also works for non-episodic MDPs, i.e., MDPs with infinite horizons.

2.1.2 Policy Gradient Methods

Policy gradient methods rely upon estimating the gradients of expected return with respect to policy parameters. One of the earliest policy gradient methods was called REINFORCE, proposed by Williams [1992], where the gradient is estimated with a stochastic policy. The agent collects one or more trajectories with current policy and uses the cumulative reward to update the policy parameters. A few problems that REINFORCE suffer from include (1) high variance in gradient estimation as cumulative rewards can deviate from each other at great degree, and (2) sample efficiency as multiple episodes need to be rolled out at each update iteration.

One way to lower the variance in gradient estimation is to subtract the cumulative reward with a baseline. Intuitively, this will result in smaller gradient steps and therefore stabilise the learning process. One common approach is to use an actor-critic architecture, which consists of an *actor* and a *critic* [Konda and Tsitsiklis, 2000; Sutton et al., 2000]. The actor refers to the policy and the critic refers the the estimation of a value function (e.g., Q-value function). During policy update, the critic can be used as baselines and help stabilise learning. Most of recent popular RL algorithms belongs to the family of actor-critic methods, such as TRPO [Schulman et al., 2015a], PPO [Schulman et al., 2017], A3C [Mnih et al., 2016], DDPG [Lillicrap et al., 2015], Soft Actor-Critic [Haarnoja et al., 2017b], etc.

On the other hand, one approach to increase the sample efficiency is to reuse the past experience collected from previous iterations. Algorithms such as TRPO and PPO, only use data generated from current policy. These methods are called *on-policy* methods. To the contrast, in *off-policy* methods such as DDPG, history data generated with policies in previous iterations are also used to update current policy. The main

difference between on-policy and off-policy methods is that if the policy to optimise and the policy to generate data is the same. In general, off-policy methods have a better sample efficiency because of reusing data from previous iterations, but are more unstable in learning process because that additional bias is introduced. As a midpoint, several algorithms learn the critic off-policy, but learn the actor on-policy [Haarnoja et al., 2017b; Gu et al., 2016]. Consequently, such methods find a better balance between stable learning process and sample efficiency.

Next in this subsection, we will introduce a few prominent policy gradient algorithms in more details.

2.1.2.1 REINFORCE

Let's first consider a stochastic policy. A stochastic policy outputs a distribution of possible actions from which an action is randomly sampled at each time step. The policy is defined by $\pi_\theta(a|s) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\pi_\theta(a|s)$ denotes the probability of choosing an action a given a state s.

Stochastic policy gradient methods work by constructing the objective function as

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[R(\tau)\big], \tag{2.3}$$

and iteratively estimating its derivative with respect to policy parameters. As no analytic expression is available, the gradients are usually estimated with Monte Carlo methods, i.e., randomly sampling data and take the average. Given different representations of policies, this estimation can be constructed in various ways.

The REINFORCE algorithm [Williams, 1992] takes the idea of likelihood ratio methods and estimated the gradient as

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_\tau \nabla_\theta p_\theta(\tau) R(\tau) d\tau && (2.4)\\
&= \int_\tau p(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d\tau && (2.5)\\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\nabla_\theta \log p_\theta(\tau) R(\tau)\big]. && (2.6)
\end{aligned}$$

Given that the probability of a trajectory $p(\tau) = p(s_0) \prod_{t=0}^{N} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$, the gradient estimation in Eq. (2.6) can be rewritten as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\big(\sum_{t=0}^{N} \nabla_\theta \log \pi_\theta(a_t|s_t)\big) R(\tau)\big]. \tag{2.7}$$

Though the gradient estimation is unbiased, Monte Carlo estimation introduces inherent high variability in log probability and cumulative reward, because each trajectories

during training can deviate from each other at great degrees. Consequently, the high variability will result in large variances in gradient estimation. To reduce the variance, there are several different variants of baselines for estimating the gradients. They generally have the following form [Schulman et al., 2015b]

$$\nabla_\theta J(\theta) = \mathbb{E}\big[\sum_{t=0}^{N} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\big], \tag{2.8}$$

where notable choices of Ψ_t includes

- $\sum_{t'=t}^{N} R_{t'}$: cumulative reward following (s_t, a_t),
- $Q_\theta(s_t, a_t)$: state-action value function,
- $A_\theta(s_t, a_t) = Q_\theta(s_t, a_t) - V_\theta(s_t)$: advantage function.

Finally, the parameters θ are updated through gradient ascent with a learning rate α.

2.1.2.2 Natural Policy Gradient

As variants of stochastic gradient ascent methods, policy gradient methods intend to take small steps of updates while optimising the objective functions. REINFORCE uses the euclidean metric, i.e., $||\Delta\theta||_2$, to measure the step size of updates. However, this ad hoc choice of measurement is not necessarily appropriate due to the nonlinearities of policies [Kakade, 2002]. An alternative is to measure the closeness with statistical distance (e.g., Kullback-Leibler (KL) divergence) between the distributions of the trajectories generated by the current policy and the updated policy. Natural policy gradient [Kakade, 2002; Peters and Schaal, 2008] takes a linear approximation of $J(\theta)$ and a quadratic approximation of KL divergence $D_{\mathrm{KL}}(p_\theta(\tau)||p_{\theta+\Delta\theta}(\tau))$, resulting in the following optimisation problem

$$\underset{\Delta\theta}{\text{maximise}} \quad \Delta\theta^T \nabla_\theta J(\theta), \tag{2.9}$$

$$\text{subject to} \quad \Delta\theta^T I(\theta) \Delta\theta \leq \delta, \tag{2.10}$$

where $I(\theta) = \mathbb{E}\big[\nabla_\theta \log \pi_\theta(\cdot|s_t) \nabla_\theta \log \pi_\theta(\cdot|s_t)^T\big]$ is known as the Fisher information matrix. The natural gradient is given by

$$\Delta\theta \propto I(\theta)^{-1} \nabla_\theta J(\theta). \tag{2.11}$$

2.1.2.3 Trust Region Policy Optimisation

To control the gradient step size, conservative policy iteration is proposed with explicit lower bounds on the improvement of objective function [Kakade and Langford, 2002]. Bulit on this lower bound, Trust Region Policy Optimisation (TRPO) defines the surrogate loss $L(\theta)$ with likelihood ratio between the updated policy and the current policy. Moreover, it applies a hard constraint on changes in policies measured with expected KL divergence between action distributions generated by each of the two policies. The optimisation problem is defined below

$$\underset{\Delta\theta}{\text{maximise}} \quad \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\big[r_t(\theta)A_{\pi_\theta}(s,a)\big], \tag{2.12}$$

$$\text{subject to} \quad \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\big[D_{\mathrm{KL}}(\pi_\theta(\cdot|s)||\pi_{\theta+\Delta\theta}(\cdot|s))\big] \leq \delta, \tag{2.13}$$

where $r_t(\theta) = \frac{\pi_{\theta+\Delta\theta}(a|s)}{\pi_\theta(a|s)}$. This leads to a solution that $\Delta\theta = I(\theta)^{-1}L'(\theta)$, . In practice, this solution is estimated with conjugate gradient algorithm, and an additional line search is used to ensure the satisfaction of the KL divergence constraint.

2.1.2.4 Proximal Policy Optimisation

One major downside of TRPO is that it involves the problem of computing a second-order derivative matrix (Hessian matrix) and its inverse when constraining on KL divergences. To avoid this, Proximal Policy Optimisation (PPO) modifies the surrogate loss in Eq. (2.12) and formalises a soft constraint on the size of update step with clipped objective:

$$\underset{\Delta\theta}{\text{maximise}} \quad \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\big[\min\big(r_t(\theta)A_{\pi_\theta}(s,a), \text{clip}(r_t(\theta),1-\varepsilon,1+\varepsilon)A_{\pi_\theta}(s,a)\big)\big], \tag{2.14}$$

where $r_t(\theta) = \frac{\pi_{\theta+\Delta\theta}(a|s)}{\pi_\theta(a|s)}$, is the probability ratio between the updated policy and the current policy. With the ratio measuring how different two policies are, the clipped objective effectively discourages any large policy update that moves the ratio out of the interval $[1-\varepsilon, 1+\varepsilon]$.

By replacing the hard constraints, the objective can be optimised with first-order optimiser like Stochastic Gradient Descent (SGD) methods.

2.1.2.5 Deep Deterministic Policy Gradient

Policy gradient methods were first proposed for solving continuous tasks with stochastic policies. Silver et al. [2014] showed that these methods can be extended to deterministic policies. Several variants of deterministic policy gradient algorithms have

been proposed to solve complex dynamics tasks [Lillicrap et al., 2015; Fujimoto et al., 2018].

A deterministic policy, denoted by $\pi_\theta(s)$, generates an action given a state: $\pi_\theta(s) : \mathcal{S} \to \mathcal{A}$. Different with stochastic policies, where exploration is implicitly integrated in the policies, deterministic policies require additional noise during training for the sake for exploration.

Deterministic policy gradient methods are based on generalised policy iteration: interleaving policy evaluation and policy improvement. The policy evaluation step estimates state-action value $Q^\pi(s,a)$ with, for example, temporal-difference learning, and the policy improvement step updates the policy π_θ in the direction of gradient of Q. More specifically, applying the chain rule, the gradient is estimated as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi_\theta(s,a)|_{a=\pi_\theta(s)}\right]. \tag{2.15}$$

Combining the idea of deterministic policy gradient and function approximators modeled by deep neural networks, Deep Deterministic Policy Gradient (DDPG) proposed the practical method to solve large-scale dynamic problems with deep neural networks. The implementation includes several tricks, such as soft target updating, experience replay buffer, etc. [Lillicrap et al., 2015].

Compared to stochastic policy gradient algorithms like TRPO and PPO, deterministic policy gradient methods usually achieve higher sample efficiency, as they reuse past experience through replay buffer. However, these methods are shown to be very sensitive to the choices of hyperparameters in practice and require a lot of tuning to get them to converge.

2.2 Benchmarking Generalisation in Reinforcement Learning

The growing community of RL research has benefitted tremendously from efforts on standardised environment models and benchmarks, such as OpenAI Gym bechmarking set [Brockman et al., 2016] and Deepmind Control Suite [Tassa et al., 2018]. Building on these, a variety of Deep RL algorithms for continuous control were implemented and compared to the most notable algorithms based on *training return* in Duan et al. [2016a]. However, because of the nature of RL problems settings, these results have high variance, leading to concerns about reproducibility of conclusions and dependence on specific choice of training seeds [Henderson et al., 2018].

Recent work has also noted the overfitting risk in this standard practice of evaluation by training return. Zhang et al. [2018b] studied overfitting of Deep RL in discrete maze tasks. Learning agent should learn to navigate through the maze from different initial positions. During training, a finite set of initial positions are sampled, while testing environments are generated with the same maze configuration but different initial positions as training. Deep RL algorithms were shown to consistently suffer from overfitting to training configurations and memorise training scenarios. Similarily, Zhang et al. [2018a] formalised overfitting in continuous control problems. By splitting random seeds for training and testing environments, the performance of generalisation properties is quantitatively evaluated through the generalisation error – the difference between average return in training and testing environments:

$$E_{G,RL} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} R(s_t, a_t|\text{seed}_i^{tr}) - \frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{T} R(s_t, a_t|\text{seed}_i^{te}), \tag{2.16}$$

where N is the number of training random seeds and M is the number of testing random seeds. A high generalisation error shows that the learned model captures the residual variation induced by training seeds, and therefore, is overfitted. The experimental results suggested that learned model may overfit to the specific set of training seeds and fail to generalise in testing, especially when the number of training seeds is small.

Overfitting happens when learned policies fail to generalise from training to testing data although they are drawn from the same underlying distribution. In contrast to overfitting, domain-shift challenge happens when a model is trained in one domain and is required to perform in a target domain with different statistics. This difference usually leads to a significant drop in agent's performance. In real-world application of Deep RL-trained models, this domain shift challenge is unavoidable: training in simulation will always mismatch reality due to the reality gap [Koos et al., 2013] of modelling errors and the unpredictability of the unconstrained real world. Zhang et al. [2018a] studied the some limited facets of domain-shift by adding unforeseen noise to observations or initial states during testing compared to the training simulation. Similarly, Packer et al. [2018] studied performance under train-test domain shift by modifying environmental parameters such as robot mass and length to generate new domains. Other than the randomly sampled noise (e.g. Gaussian, uniform noise), in the context of discrete control, Huang et al. [2017] also discussed generalisation of RL against adversarially designed noise targeting the policies.

2.3 Towards Cross-Domain Generalisation

Since domain shifts between simulation and real world are inevitable, cross-domain generalisation is crucially important in learning robot control. To formalise this problem, we denote a domain by d_ξ, where ξ is the configuration of domain d_ξ. We call the domain in which the agent is trained as *source* domain $d_{\xi_{\text{tr}}}$ (e.g., simulatior) and the target domain for intended transfer as *target* domain $d_{\xi_{\text{te}}}$ (e.g., real world). We assume that the agent has no access to target domain during training and is not equipped to update its policy during operational use, i.e., no domain adaptation will occur during testing. The key objective here is to train a policy which is insensitive to domain shifts (compared to source domain) and, therefore, performs well in target domains. Following this section, we review some related work towards improving cross-domain generalisation.

2.3.1 Network Architecture

Recent work typically uses a generic multilayer perceptrons (MLPs) for policy representation. However, [Rajeswaran et al., 2017] suggested this default choice of policy architecture might not be ideal for control problems.They compared generic MLPs with simple policy parameterisations, including linear policies and Radius Basis Functions (RBF) policies and showed that simple architectures can achieve better generalisation properties with no significant lose in learning performance. This encouraged the pursuit of better network architecture designs.

Srouji et al. [2018] introduced a modified MLP, called Structured Control Net (SCN). Inspired by nonlinear control theory [Isidori, 2013] and ResNet [He et al., 2016], SCN splits the generic multilayer perceptron (MLP) into a nonlinear planning stream and a linear control stream. Intuitively, the nonlinear stream if for forward-looking and global control; and the linear stream is for stabilising the local dynamics around the residual of global control. These two streams are then combined additively into the final action. Their results showed the learned policies performed more robust against both state noise and action noise.

2.3.2 Regularisation

Regularisation has been playing a significant role in supervised learning. These regularisation techniques are less often employed in RL, presumably because the data are

usually not separated as training and testing set. Cobbe et al. [2018] deployed several regularisation techniques which are commonly used in supervised learning to RL problems, including L_2 regulariser, dropout and data augmentation. In a discrete problem setting, it is shown that such techniques could also improve RL agents with better generalisation properties.

Other than these conventional regularisation techniques, RL with stochastic policy is commonly regularised with entropy of the output distribution [Schulman et al., 2017; Haarnoja et al., 2017b]. As discussed by [Ziebart, 2010], maximum entropy policies are more robust in the face of model and estimation errors. More recently, robots are able to learn robust quadrupedal locomotion skills with entropy regularisers. [Haarnoja et al., 2017b].

2.3.3 Multi-Domain Learning

A more popular framework for bridging the gap between source and target domains is to train with a diverse set of domains. The common characteristic of such approaches is the perturbation on the parameters that determine the domain configuration. Different perturbations include, but not limited to, observation and actuation noise [Jakobi et al., 1995; Andrychowicz et al., 2020], initial states [Wang et al., 2010], dynamic features (e.g., mass of robots, friction coefficients, control frequency, etc.) [Andrychowicz et al., 2020; Peng et al., 2018; Mordatch et al., 2015; Rajeswaran et al., 2016].

Let us denote a distribution of training domain configurations as $p(\xi)$. The general training objective in Eq. (2.3) becomes

$$J(\theta, p(\xi)) = \mathbb{E}_{\xi \sim p(\xi)} J(\theta, d_\xi) \tag{2.17}$$

$$= \mathbb{E}_{\xi \sim p(\xi)} \left[\mathbb{E}_{\tau \sim p_\theta(\tau | d_\xi)} \left[R(\tau) \right] \right]. \tag{2.18}$$

With a proper choice of source domain set, discrepancies between the source and target domains can be modelled as variability among the source domains [Peng et al., 2018]. Moreover, as suggested by Haarnoja et al. [2017a], learning with uncertainties helps to improve exploration and thus discover more robust behaviour. However, defining the distribution of training domains is not trivial. The learning can be hard to converge if the training set of domains is too diverse; whereas the learned model may fail in generalising against testing domain shifts if the training set is too tight. In practice, this requires a significant amount of manual tuning and iteration loops between domain distribution design in training and validation on testing.

2.3.3.1 Risk Sensitive Reinforcement Learning

Optimising Eq. (2.18) allows us to learn the best policy in terms of expectation of performance over domains in the source domain distribution. However, this may not lead to the best policy in terms of robustness. The policy may be failing in some domains while still achieving locally optimal average performance over source distribution, due to high variability among domains. To explicitly seek for a robust policy, an alternative is to optimise for the conditional value at risk [Rajeswaran et al., 2016; Tamar et al., 2015], which is a lower bound of previous objective in Eq. (2.18):

$$\underset{\theta,y}{\text{maximise}} \quad \mathbb{E}_{\xi\in\mathcal{F}(\theta,y)}\big[J(\theta,d_\xi)\big], \tag{2.19}$$

$$\text{subject to} \quad P(J(\theta,d_\xi)\leq y)\leq \varepsilon, \tag{2.20}$$

where $\mathcal{F}(\theta,y)=\{\xi|J(\theta,d_\xi)\leq y\}$ is the subset of Ξ that produce returns worse than y. This can be interpreted as maximising the expectation return for the worst ε-percentile of domains from the source distribution. In practice, at each iteration, the agent samples K different domains and rollouts one trajectory for each domain. The worst k ($k<K$) trajectories are used for optimising policy and the rest are discarded. This will lead to a solution the perform well in all domains if converged. In practice, however, it is hard to converge because the gradients are biased and the variances of gradient estimation are high due to constantly switching between domains.

2.3.3.2 Adversarial Perturbations

The idea of optimising policy for the worst case has also motivated adding adversarial disturbances to the training process [Mandlekar et al., 2017; Pinto et al., 2017]. Different from previous methods, where the agent randomly samples multiple domains and selects the worst cases, the adversarial methods generate noise specifically targeting at the policy. Adversarially Robust Policy Learning (ARPL) proposed physically plausible perturbations by randomly deciding when to add a rescaled gradient of the expected return [Mandlekar et al., 2017]. On the other hand, Robust Adversarial Reinforcement Learning (RARL) introduced a second agent whose goal is to hinder the first agent from fulfilling its task. Both agents are trained simultaneously as a zero-sum game. In general, adversarial approaches could provide generalised policies. However, training with adversarial is still challenging without any further restrictions.

2.4 Towards Cross-Task Generalisation

The vision of learning-based robot control is for robots to learn new skills autonomously. Direct application of RL, however, requires prohibitive amounts of training while risking physical damage to robots. This has motivated a fruitful line of research into *multi-task learning* (MTL) and *transfer learning* (TL). MTL and TL aim to improve and accelerate learning by sharing knowledge across different tasks, in a multi- and uni-directional way respectively. One of their earliest applications to RL was in classic pole balance problem. By transferring knowledge from other similar systems, the agent learned a policy for a novel system much faster [Selfridge et al., 1985]. Since then, TL has been widely applied to accelerate RL [Taylor and Stone, 2009]. In the meanwhile, MTL has been oftern used jointly to optimise multiple RL tasks such as the inverted pendulum problems with various masses, various lengths [Lazaric and Ghavamzadeh, 2010], or stacking various numbers of blocks [Deisenroth et al., 2014].

As summaried in [Taylor and Stone, 2009], the type of generalised knowledge can be primarily characterised by its specificity, including experience instances ($\langle s_t, a_t, r_t, s_{t+1} \rangle$ tuples), an action-value funtion Q, a policy π, etc. In this section, we will review recent work in cross-task generalisation problems, with a particular focus in policies sharing. We split them into three categories following [Yang, 2017]: hard parameter sharing, additive sharing model, and multiplicative sharing model.

2.4.1 Hard Parameter Sharing

Consider a set of task $\mathcal{T} = \{T_1, T_2, ..., T_N\}$ and the corresponding policies $\Theta = \{\theta_1, \theta_2, ..., \theta_N\}$, the MTL objective is

$$\underset{\Theta}{\text{maximise}} \frac{1}{N} \sum_{i=1}^{N} J(\theta_i, T_i), \tag{2.21}$$

where $J(\theta_i, T_i)$ is the expected return in task T_i. One common setting is to share knowledge by sharing parts of policy parameters. Each policy parameter set θ_i is split into a task-agnostic set $\theta_{(0)}$ and a task-specific set $\phi_{(i)}$. For example, as illustrated in Figure 2.1, if each policy θ_i represents an MLP network, $\theta_{(0)}$ is usually chosen as the feature extractor layers and $\phi_{(i)}$ is the output layer. The optimisation problem in Eq. (2.21) becomes

$$\underset{\theta_{(0)}, \Phi}{\text{maximise}} \frac{1}{N} \sum_{i=1}^{N} J(\mathcal{C}(\theta_{(0)}, \phi_{(i)}), T_i), \tag{2.22}$$

where $\Phi = \{\phi_1, \phi_2, ..., \phi_N\}$ is the set of all task-specific parameters and $\mathcal{C}(\cdot)$ is a concatenation function.

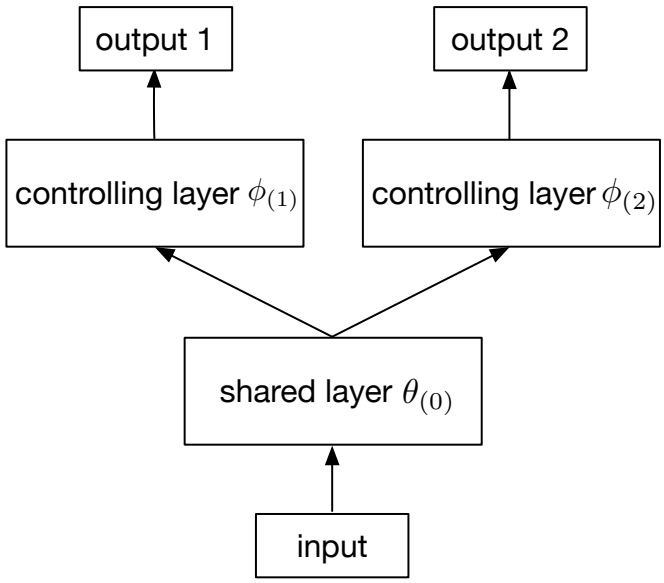


Figure 2.1: An example of hard parameter sharing architecture with two tasks. Policies are modelled as MLPs with one hidden layer, where the first layer is hard shared and the second layer is task-specific.

Learning multiple tasks together is extremely challenging even with separate control layers in Deep RL, due to interference between the different policies, different reward scaling, and the inherent instability of learning value functions [Rusu et al., 2015]. In the context of learning discrete tasks, the approach of learning each task individually then distilling into a student policy is used instead [Rusu et al., 2015; Parisotto et al., 2015]. The student policy is trained using supervised learning with samples generated by different teachers. The student is able to outperform the teachers, even though it is not explicitly trained in the environments. Teh et al. [2017] extends this idea of distillation to an online learning process. The agent follows an alternating maximization procedure over training policies individually and distilling into a shared policy.

In order to transfer to a target task, the feature extractor layer in target task policy is initialised as $\theta_{(0)}$ and the controlling layer is randomly initialised. The agent then fine-tunes the policy in the target task. Given a sequence of tasks, the agent is able to acquire different skills by alternating between fine-tuning in the task and distilling into a shared policy [Berseth et al., 2018].

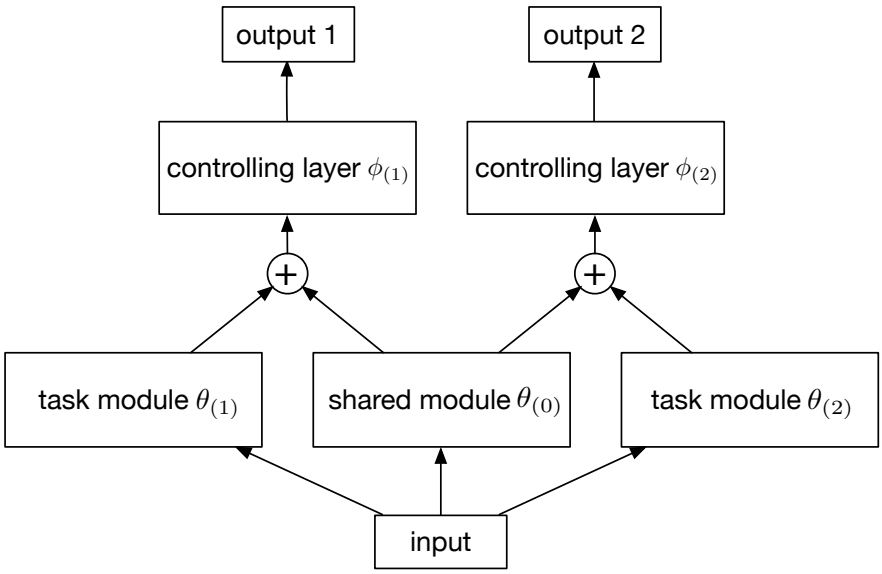


Figure 2.2: An example of additive sharing architecture with one shared module and one task module each task. The + node represents a summation operation.

2.4.2 Additive Sharing Model

The policies do not necessarily share all parameters in early layers. One alternative of sharing strategy is to add up the shared parameters $\theta_{(0)}$ and task-specific parameters $\theta_{(i)}$. Combining with a separate controlling layer, the policy is represented as $\theta_i = \mathcal{C}(\theta_{(0)} + \theta_{(i)}, \phi_{(i)})$. An example of this sharing architecture is available in Figure 2.2.

One example use of this architecture is progressive networks [Rusu et al., 2016a]. Given a new task, progressive network adds a new column of policy of network, and the transfer is enabled via lateral connections to features of previously learned columns. Silimarly in PathNet, each task policy is represented as the summation of multiple modular neural networks at each layer [Fernando et al., 2017]. However, PathNets do not explicitly separate modules as task-agnostic and task-specific ones, but instead provide a sufficient pool of modules and have the agent decide whether to reuse existing modules or to add new modules to the pool by evolutionary strategies.

2.4.3 Multiplicative Sharing Model

Another assumption of knowledge sharing is that the policies (task parameters) lie in a low dimensional subspace [Argyriou et al., 2008; Kumar and Daume III, 2012]. Specifically, each policy θ_i is represented as an inner product of the shared latent knowledge L and a task-specific weight vector $s_{(i)}$. An example of the sharing architecture is il-

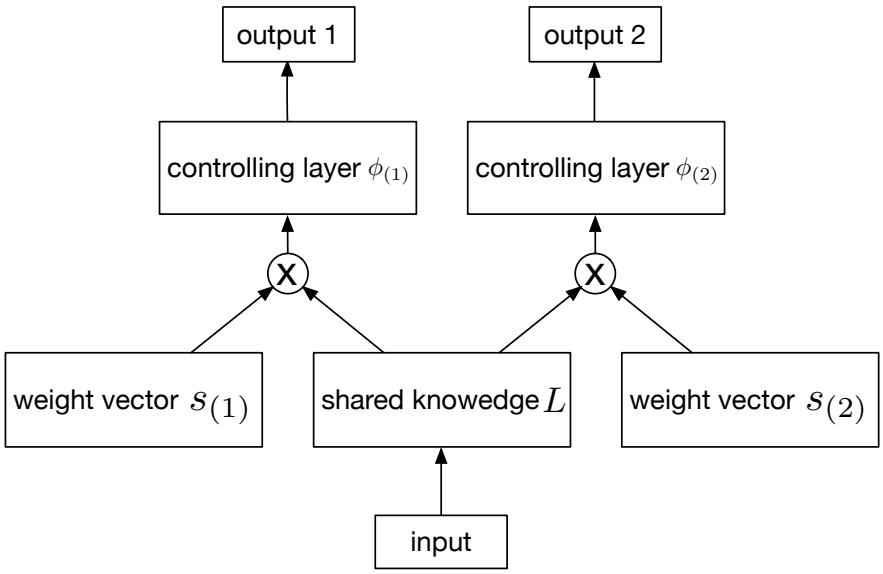


Figure 2.3: An example of multiplicative sharing architecture with a shared knowledge base L and task-specific weight vectors $s_{(1,2)}$. The X node represents an inner product operation.

lustrated in Figure 2.3. We stack all weight vectors $s_{(i)}$ to form a matrix S. This results in the optimisation problem:

$$\underset{L,S}{\text{maximise}} \frac{1}{N} \sum_{i=1}^{N} J(\mathcal{C}(S_{\cdot,i}L, \phi_{(i)}), T_i), \tag{2.23}$$

where $S_{\cdot,i}$ denotes the i-th row vector in S.

PG-ELLA [Ammar et al., 2014] adapts the successful GO-MTL algorithm [Kumar and Daume III, 2012] from supervised learning to the policy gradient based RL setting, with linear policy representations. Furthermore, our work in Chapter 5 [Zhao et al., 2017] extends it to complex nonlinear dynamics tasks with designed RBF networks. Especially, instead of using separate controlling layers, the entire policies are sharing information through inner products of shared knowledge and task weight vectors.

2.5 Summary

In this chapter, we first reviewed the background and several well-established algorithms in RL, with a focus on continuous domains. Then we discussed related work in the following aspects: current literature that evaluating and benchmarking RL algorithms, and efforts towards cross-domain and cross-task generalisation.

Chapter 3

Generalisation Across Domains in Reinforcement Learning

In this chapter, we first present a thorough characterisation of the generalisation challenges, especially in overfitting and domain shift in continuous control. To quantify these issues empirically, we contribute a comprehensive benchmark for measuring Deep Reinforcement Learning (RL) generalisation performance with several OpenAI Gym tasks [Brockman et al., 2016] and evaluate several popular algorithms. Existing Deep RL algorithms are generally shown to be vulnerable to domain shifts between training and testing conditions. Correspondingly, the standard practice of picking algorithms and architectures based on training performance leads to the wrong choice in terms of generalisation performance. We further evaluate several techniques that might improve generalisation as a starting point for future work.

3.1 Characterising Generalisation in RL

We first start by giving a thorough characterisation of the within-task generalisation challenges that can arise for RL agents.

3.1.1 Sources of Uncertainty and Variability

Consider a classic robot system as illustrated in Figure 3.1. The agent π provides the controller and the environmental transition model can be broken into an actuation module g, a sensor module f and a dynamical module h. In model-free RL, the agent can only access the observation and reward, but not the environmental modules. In real

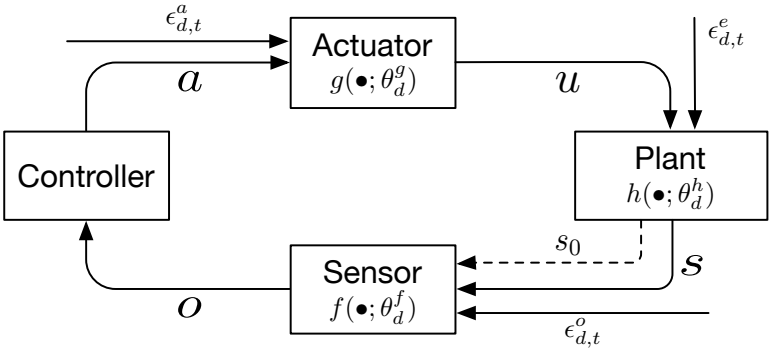


Figure 3.1: Block diagram of a classic control system. f represents the observation function, g represents the actuation function, h represents the transition function, and ε represents the noise that enters the system due to different types of uncertainties.

world applications, each of the three modules may contain uncertainties due to noise. Moreover, each may exhibit contextual variability that creates potential *domain shifts* between distinct encounters with the environment, including between training and testing. To unpack the distinction between uncertainties and disturbances in rollouts (i.e. noise), and variations that induce systematic shifts in domains, we further specify the environmental MDP corresponding to Figure 3.1 with more details as Eq. (3.1), (3.2).

$$p_d(s_0)s_{d,0} \qquad \sim \mathcal{N}(\mu^{s_0} + \theta^{s_0}_d, \sigma^{s_0}_d) \tag{3.1}$$

$$p_d(s_{t+1}|s_t, a_t) \begin{cases} o_{d,t} & \sim \mathcal{N}(f(s_{d,t}; \theta^f_d), \sigma^o_d) \\ u_{d,t} & \sim \mathcal{N}(g(a_{d,t}; \theta^g_d), \sigma^u_d) \\ s_{d,t+1} & \sim \mathcal{N}(h(s_{d,t}, u_{d,t}; \theta^h_d), \sigma^s_d) \end{cases} \tag{3.2}$$

In this model of the MDP, noise is introduced at three time-scales. The observations o, commands u and next states s as perturbed at *each time step*, as indicated by the $_t$ subscript. They are perturbed by Gaussian noise with variances σ^o, σ^u, and σ^s, respectively. The initial state s_o is sampled *once per episode* from a Gaussian with variance σ^{s_0}, before the episode starts. The combined function parameters $\theta^{s_0,f,g,h}$ and the corresponding variances $\sigma^{s_0,o,u,s}$ define the MDP. We summarise these parameters as $\Theta = \{\theta^{s_0,f,g,h}, \sigma^{s_0,o,u,s}\}$. Switching to an environment with different set of Θ implies a domain switch. As discussed in Section 3.1.2, these can also be sampled from a distribution *before learning*. These parameters then stay fixed during learning, which is why they are indexed with domain $_d$. With this formalisation, we summarise several related work in Table 3.1.

To illustrate these terms by way of example: A change in the mass of an object to be manipulated (or of the robot itself) in the environment or friction constant would correspond to a change in transition function parameter θ_d^h. Stochasticity in the outcomes of transition model due to environmental noise such as changing wind condition is determined by σ_d^s. Changes in the observation function f via parameters θ_d^f correspond to events such as a change of camera when doing vision-driven control. Meanwhile proprioceptive noise is generated with variance σ_d^o. Changes in the actuation function g's parameter θ_d^g could correspond to wear in a motor or increased joint friction reducing the obtained forces. Noise generated internally by motor while actuating actions are sampled with variance σ_d^u. In general, we would like our agents to be robust to as much of these variations and noise as possible.

3.1.2 Generalisation Across MDP Distributions

With this formalisation in mind, we can understand the goal of generalisation as robustness to a potential *distribution* of both environments $p(\Theta)$ and samples from those environments. That is, insensitivity to both environmental parameters Θ, and noise samples. This is in contrast to commonly used deterministic simulations ($\sigma = 0$), without environmental variability (Θ constant).

Given a fixed set of environmental parameters Θ, the corresponding transition model and initial state distribution are denoted $\mathcal{P}^\Theta$ and $\mathcal{P}_0^\Theta$. We denote $\eta_\Theta(\pi)$ as the expected return given a set of environmental parameters Θ. If the environmental parameters are varying across trials, we denote $\eta_{p(\Theta)}(\pi)$ as the expected return under the distribution of environment parameters $p(\Theta)$:

$$\eta_\Theta(\pi) = \mathbb{E}_{\tau\sim(\pi,\mathcal{P}^\Theta,\mathcal{P}_0^\Theta)}\left[R(\tau)\right] \tag{3.3}$$

$$\eta_{p(\Theta)}(\pi) = \mathbb{E}_{\Theta\sim p(\Theta)}\left[\eta_\Theta(\pi)\right]. \tag{3.4}$$

We would like our agents to solve a distribution over (non-deterministic $\sigma^{(o,u,s)} > 0$) environments in Eq. (3.4), rather than the conventional RL criterion in Eq. (2.2). That is for a given trial, we would expect to sample an environment once $\Theta \sim \mathcal{N}(\Theta_0, \Sigma)$, and then at each time-step sample noise $\varepsilon \sim \mathcal{N}(0, \sigma)$. We want agents to perform well over both this long-time scale variability, and short-time scale uncertainty.

Furthermore, by evaluating on training return, standard RL practice implicitly assumes that the simulated training domain models the testing domain perfectly: $\Theta_{tr} = \Theta_{te}$ or $p_{tr}(\Theta) = p_{te}(\Theta)$. While this assumption can hold for some tasks like Atari

Study	Training Setting	Testing Setting
Common Evaluation Practice	$s_{0,tr} \sim \mathcal{N}(\cdot, \sigma)$	$s_{0,te} = s_{0,tr}$
Zhang et al. [2018b]	$s_{0,tr} \sim \mathcal{N}(\cdot, \sigma)$	$s_{0,te} \sim \mathcal{N}(\cdot, \sigma)$
Zhang et al. [2018a]	$s_{0,tr} \sim \mathcal{N}(\cdot, \sigma_1)$	$s_{0,te} \sim \mathcal{N}(\cdot, \sigma_1)$ $s_{0,te} \sim \mathcal{N}(\cdot, \sigma_2)$
	$s_{0,tr} \sim \mathcal{N}(\cdot, \sigma_1)$, $o_{tr} \sim \mathcal{N}(\cdot, 0)$	$s_{0,te} \sim \mathcal{N}(\cdot, \sigma_1)$, $o_{te} \sim \mathcal{N}(\cdot, \sigma_2)$
Packer et al. [2018]	$s_{0,tr} \sim \mathcal{N}(\cdot, \sigma)$, $\theta^h_{tr} = \theta_0$	$s_{0,te} \sim \mathcal{N}(\cdot, \sigma)$, $\theta^h_{te} \sim \mathcal{U}(\theta_0 - \sigma_1, \theta_0 + \sigma_1)$ $s_{0,te} \sim \mathcal{N}(\cdot, \sigma)$, $\theta^h_{te} \sim \mathcal{U}(\theta_0 - \sigma_2, \theta_0 + \sigma_2)$

Table 3.1: Comparison of common evaluation practice Deep RL literature and recent work that evaluate generalisation performance. $\{\cdot\}_{tr}$ are samples generated with random training seeds, $\{\cdot\}_{te}$ are samples generated with random testing seeds.

games, creating a sufficiently accurate simulated model is challenging for dynamic tasks [Koos et al., 2013], and is generally impossible if the testing domain is the unconstrained real-world. Therefore, an important quantity of interest to measure is how trained models generalise to encounters with a certain degree of domain shift between environments ($\Theta_{tr} \neq \Theta_{te}$ or $p_{tr}(\Theta) \neq p_{te}(\Theta)$). Therefore, besides training performance, we should monitor the robustness of our models via the quantity:

$$\eta_{p_{te}(\Theta)}(\pi^*) \mid \{\pi^* = \underset{\pi}{\operatorname{argmax}}\, \eta_{p_{tr}(\Theta)}(\pi),\ p_{te}(\Theta) \neq p_{tr}(\Theta)\}. \tag{3.5}$$

That is, the performance of the model π^* trained on Θ_{tr} or $p_{tr}(\Theta)$; when tested on Θ_{te} or $p_{te}(\Theta)$. This view encompasses robustness to changes in distribution of starting condition [Zhang et al., 2018b] and maps [Cobbe et al., 2018], training with deterministic observations $\rightarrow$ non-deterministic testing [Zhang et al., 2018a] (but also includes action and environmental noise), and extrapolation in environmental parameters such as object mass [Packer et al., 2018] that will arise in the practice due to the reality gap [Koos et al., 2013]. Given the inability to exactly control or simulate the distribution of real-world environmental encounters, the model robustness quantified above should be a consideration in our development of new methods, and our choice of algorithms and architectures in practice.

Since the vast majority of existing work does not explicitly separate training and

testing phases, in the following sections we introduce a set of benchmarks with clear train/test distinctions. Based on these, we systematically measure the generalisation of several popular algorithms under the diverse variations.

3.2 Experimental Design

We design a benchmark of generalisation – testing rather than training performance. We cover both generalisation across seeds when the simulation is non-deterministic ($\sigma^{(o,u,s)} > 0$) in observation, actuation and process; and particularly focus on robustness to environment parameter variation, i.e., domain-shift $\Theta_{tr} \neq \Theta_{te}$ or $p_{tr}(\Theta) \neq p_{te}(\Theta)$.

3.2.1 Training Algorithms and Architectures

We study several model-free policy gradient based Deep RL algorithms with OpenAI baseline implementations [Dhariwal et al., 2017] including Trust Region Policy Optimisation (TRPO) [Schulman et al., 2015a], Proximal Policy Optimisation (PPO) [Schulman et al., 2017] and Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015]. In addition to basic Deep RL algorithms, we also consider several modifications of the baseline algorithms and architectures that may improve generalisation of learned policies. The detailed hyper

Entropy Regulariser Policies with higher entropy may be more robust to uncertain dynamics [Ziebart, 2010]. We consider two different methods that encourage learning higher-entropy policies. One is to add an entropy regulariser to the convenctional PPO training objective [Schulman et al., 2017], denoted with suffix *-Ent*. The other is Soft Actor-Critic, which also include an entropy term to training objective [Haarnoja et al., 2017b], denoted as *SAC*.

Structured Control Net Inspired by classic control theory, Structured Control Net (SCN) splits a Deep RL policy into a linear module and a nonlinear residual module and shows improved robustness against noise [Srouji et al., 2018]. We train SCNs with PPO, denoted *PPO-SCN*.

Architecture A standard continuous control policy architecture is a multilayer perceptrons (MLP) with two 64 unit hidden layers. To investigate the influence of network size on generalisation performance, we use a smaller MLP with two 16 unit hidden layers. The smaller network is indicated as with suffix *-16*, such as *PPO-16*, *SCN-16*.

Adversarial Attacks Assist Learning Several attempts have been made to improve policy robustness with the assistance of adversaries [Pattanaik et al., 2018; Pinto et al., 2017]. We follow 'adversarially robust policy learning' (ARPL) [Mandlekar et al., 2017] where adversarial noise maximises the norm of output actions $\delta_t = \varepsilon \nabla_s ||\pi_\theta(s_t)||$. To minimise the interference in the simulation platform, adversaries only attack in the observation space.

Multi-Domain Learning Training agents on multiple domains is a simple strategy to improve generalisation over environment changes [Tobin et al., 2017]. Ensemble Policy Optimisation (EPOpt) further addresses the risk of optimising average performance over all source domains and instead optimises the expected return of the worst ε-percentile of source domains [Rajeswaran et al., 2016]. To simulate variability in domains, we generate a distribution of domains controlled by a parameter Σ. At each training rollout, we sample a new domain from the distribution $\Theta \sim \mathcal{N}(\Theta_0, \Sigma)$. In this case we only sample dynamics parameters $\theta^h \in \Theta$. We evaluate both optimising average return of all source domains and worst ε-percentile of source domains, denote the training settings *-MDL*, *-EPOpt* respectively.

Meta Learning Model-Agnostic Meta Learning (MAML) [Finn et al., 2017] and Reptile [Nichol et al., 2018] are simple meta-learning algorithms that train a fast-to-adapt model based on a set of training task/domains. Different from MDL, they do not simply optimise for the joint performance across all training domains, but instead split them into a meta-train and meta-test set and optimise for fast adaptation from meta-train to meta-test. Li et al. [2018b, 2019] suggest that such meta-learning pipelines could in multi-domain learning to improve cross-domain generalisation performance.

In this paper we propose simple adaptations of MAML and Reptile to improving domain generalisation in a multi-domain learning scenario. Specifically, we regard domains as tasks in MAML/Reptile and proceed by iteratively: Forking the main policy to independently perform training on a single batch of data sampled from each domain, collecting gradients for each domain after training, and then updating the main policy given all the gradients. Unlike MAML and Reptile, once trained we then use the main policy directly for testing without any updating. Detailed training algorithms are described in Algs. 1 and 2. The intuition is that this procedure optimizes to find a single central policy that is very 'close' to the optimal specific policy of each training domain. A good solution to this is a single policy that is good for all domains, and this is exactly what is required for our desired domain-agnostic policy. We denote the

Algorithm 1 MAML-DG

Input: a distribution of domains $p(\Theta)$, initial policy parameters π, step size hyper-parameters α, β

while not converged **do**

Sample a batch of domains $\Theta_i \sim p(\Theta)$

for all domain Θ_i **do**

Sample trajectories $D_i = \{(x_0, s_0, r_0), ...\}$ with policy π in domain Θ_i

Update domain policy with sampled data D_i: $\pi_i = \pi + \alpha \nabla_\pi \eta_{\Theta_i}(\pi)$

Sample trajectories $D'_i = \{(x'_0, s'_0, r'_0), ...\}$ with policy π_i in domain Θ_i

end for

Update policy $\pi \leftarrow \pi + \beta \nabla_\pi \sum_i \eta_{\Theta_i}(\pi_i)$ using each D'_i

end while

Output: a domain-general policy π

training settings -*MAML-DG* and -*Reptile-DG* respectively.

3.2.2 Environments and Evaluation

We experiment on several MuJoCo simulated [Todorov et al., 2012] environments in OpenAI Gym [Brockman et al., 2016] as summarised in Table 3.2. To explore robustness to environmental parameter variation, we modify various environmental dynamics parameters θ^h as shown in Table 3.2. For example, in *Walker2d*, we modify robot mass, friction and gravity coefficients, and apply constant horizontal force as wind.

For each training setting (task, algorithm/architecture, train-environment), we train 12 policies with different random seeds (seeds are selected arbitrarily). For each condition (task, algorithm/architecture, test-environment) we evaluate by averaging over 20 different testing rollouts. We assume constant sensor and actuation module context parameters $\theta_{tr}^{(f,g)} = \theta_{te}^{(f,g)}$, and the same initial state distribution but different random seeds between training and testing $p_{tr}(s_0) = p_{te}(s_0)$. Our evaluation focuses on robustness against various noise scales in observation, action and environmental parameter space $\sigma_d^o, \sigma_d^u, \sigma_d^s$, as well as systematic shifts in the environmental parameters θ_d^h. Gaussian noise is directly added to outputs of dynamic plant and agent policies as observation and action noise. For environmental parameter noise, at each time step, a set of environmental parameter (e.g. wind condition) is sampled and simulation is modified accordingly. In contrast, for systematic shifts, environmental parameters are

Algorithm 2 Reptile-DG

Input: a distribution of domains $p(\Theta)$, initial policy parameters π, step size hyper-parameters α, β

while not converged **do**

Sample a batch of domains $\Theta_i \sim p(\Theta)$

$\pi' = \pi$

for all domain Θ_i **do**

Sample trajectories $D_i = \{(x_0, s_0, r_0), ...\}$ with policy π' in domain Θ_i

Update policy with sampled data D_i: $\pi' \leftarrow \pi' + \alpha \nabla_{\pi'} \eta_{\Theta_i}(\pi')$

end for

Update policy with Adam optimiser $\pi \leftarrow \pi + \beta(\pi' - \pi)$

end while

Output: a domain-general policy π

sampled at the start of each trial and remain constants within the trial. Four testing settings are denoted with *Obs, Act, Env, Dom* respectively.

We use three evaluation metrics including *Testing return* (Eq. (3.3)) $\eta_{\Theta_{te}}(\pi)$ where possibly $\Theta_{te} \neq \Theta_{tr}$ and *Expected testing return* $\eta_{p_{te}(\Theta)}(\pi)$ (Eq. (3.4)) where possibly $p_{te}(\Theta) \neq p_{tr}(\Theta)$. Finally, as an aggregate measure of performance given that we may not know the strength of noise or variability in the testing domain, we also compute the *Area Under Curve (AUC)* of testing return with respect to scale of the underlying Gaussian distribution $[\sigma_1, \sigma_2, ..., \sigma_N]$:

$$\text{AUC}(\pi) = \sum_{n=1}^{N} \Delta\sigma \eta_{\sigma_n}(\pi). \tag{3.6}$$

where σ could be both noise $\sigma^{(o,u,s)}$ and domain shift Σ, and $\Delta\sigma$ is the step size of varying scales, $\Delta\sigma = \sigma_2 - \sigma_1$.

3.3 Experimental Results and Discussion

3.3.1 Is training return a valid metric for performing model selection?

In Deep RL research, training return is the standard evaluation metric for comparing learning algorithms and architectures [Duan et al., 2016a; Henderson et al., 2018]. In

Task Name	Environment Factors
InvertedPendulum-v2	m_{cart}, m_{pole}
InvertedDoublePendulum-v2	m_{cart}, m_{pole_1}, m_{pole_2}
Walker2d-v2	m_{body}, c_{wind}, c_{friction}, g
Hopper-v2	m_{body}, c_{wind}, c_{friction}, g
HalfCheetah-v2	m_{body}, c_{friction}, g

Table 3.2: Summary of evaluation environments and their environment factors that are included to generate shifts in transition model.

learning continuous control problems, such as MuJoCo simulated OpenAI gym tasks, algorithms such as TRPO and PPO now achieve impressive training return; but do the resulting policies generalise to novel contexts at testing time? Given that ultimately we should care about testing return, this practice is based on the strong assumption that there is no overfitting and all distributions are identical during training and testing. However, we know that overfitting could occur, and modeling errors between training and testing in the real woisy world are unavoidable [Sunderhauf et al., 2018; Koos et al., 2013]. Therefore training performance may not reflect testing performance, and it is important to ask what is the implication of this evaluation practice on the algorithms and architectures we determine to be 'winners'.

As an illustration, we first show how generalisation performance evolves during training. Figure 3.2 shows the testing AUC score and training return as a function of PPO training iterations in the *Walker2d-v2* environment. Training and testing performance initially improve in tandem, but overfitting occurs as learning continues.

Before going into details on specific algorithms and benchmarks, we summarise the effect of overfitting on algorithm choice in Figure 3.3. Specifically, we fit a Pareto frontier to the testing AUC score vs training return of a selection of methods described in Section 3.2.1, and evaluated in more detail later in Sections 3.3.2-3.3.4. Each algorithm is represented with the learned policy with best training performance among multiple random seeds. Similar results are obtained if using average performance across seeds. Figures 3.3(a), 3.3(b) show illustrative curves for *Walker2D* with Action Noise and *HalfCheetah* with Observation Noise. The Pareto frontiers illustrate that it is hard to achieve good training and testing performance simultaneously. We further compute the correlation between testing AUC and training return for each task under each noise type in Figure 3.3(c). (Here the algorithm variants in Fig 3.3(a) are the elements being

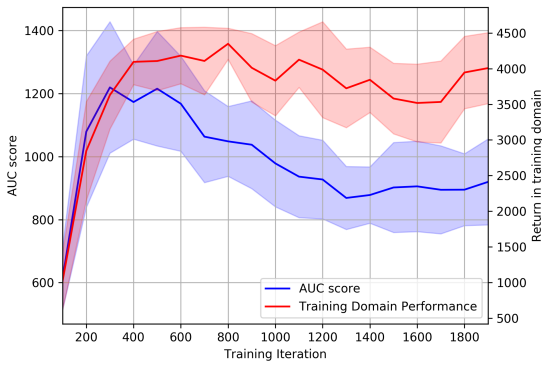
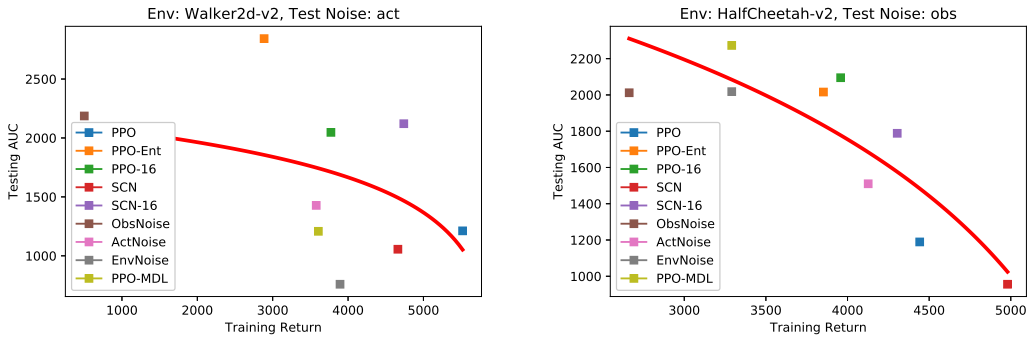


Figure 3.2: Overfitting occurs in Deep RL. Comparison of testing AUC score and training return during PPO learning of *Walker2d-v2* with non-deterministic observations. Each iteration consists of 2048 steps in simulation.

correlated). Most environments and noise types have clear negative correlation. Thus if we follow standard practice of evaluating algorithms based on training performance, we will often pick the least robust algorithm with worst generalisation. Thus, *training return is not a valid metric for model selection*. If generalisation is of interest, as it should be, then evaluations should use generalisation metrics such as the benchmarks proposed here.

3.3.2 How do standard continuous controllers generalise under different amounts of noise and domain-shift?

As we conclude above, the common practice of reporting training return does not measure the generalisation performance. We next investigate in detail how each training algorithm performs in terms of generalisation under different sources of domain changes. We analyse this question for observation-, action-, and environment-noise and domain shifts. We consider four popular Deep RL algorithms: PPO, TRPO, DDPG and SAC. Note that SAC is trained to optimise a modified objective function with an entropy regulariser, which has implications for its robustness. Figure 3.4 shows the results for *Walker2d* as an example environment. The results for all environments are shown in Figure 3.7 and summarised as AUCs in Table 3.4. Figures 3.4(a-c) show performance degradation of these standard policies as increasing observation, action, and environmental noise are added at testing. We can see that policies are relatively sensitive to observation noise compared to the other types. In terms of domain-shift rather than noise, Figures 3.4(e-h) show that the expected return $\eta_{\Theta_{te}}$ of standard models de-



(a) Walker2D - Action

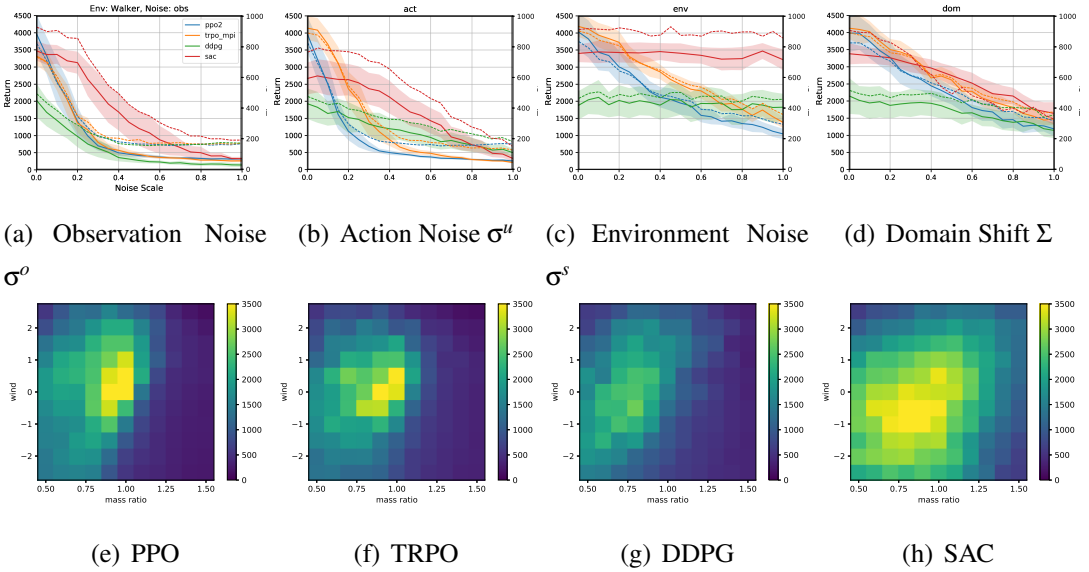
(b) HalfCheetah - Observation

	OBS.	ACT.	ENV.	DOM.
WALKER	-0.760	-0.722	-0.457	-0.424
HOPPER	-0.203	-0.346	-0.473	-0.415
HALFCHEETAH	-0.946	-0.843	-0.046	-0.477
PENDULUM	-0.132	-0.584	-0.665	-0.630
D-PENDULUM	-0.033	-0.732	0.025	0.324

(c) Correlation Coef. between training returns and testing AUC

Figure 3.3: Training return does not reflect generalisation performance. (a,b): Clear Pareto frontiers exist in testing AUC vs training return across algorithms (dots). (c): Testing AUC and training return are generally negatively correlated.

grades rapidly as example environmental parameters (mass ratio, wind direction) are changed at testing, with the degradation rate depending on the factor being modified (e.g., greater mass-sensitivity than wind). Figure 3.4(d) summarizes the domain-shift performance as an average over increasing shift in all walker parameters (Table 3.2: mass, wind, friction, gravity). In this particular environment, SAC is usually the most robust algorithm. However overall, there is not a consistent winner in algorithm robustness across all environments as seen in Table 3.4. For example, PPO performs better in *InveretedPendulum* and *InvertedDoublePendulum*. In the following sections, we focus our further investigation on PPO as it is currently the closest to a widely-used industry-standard – thus maximising the relevance of our subsequent results – and is easy to integrate with the other modifications we will explore.



(a) Observation Noise σ^o (b) Action Noise σ^u (c) Environment Noise σ^s (d) Domain Shift Σ

(e) PPO (f) TRPO (g) DDPG (h) SAC

Figure 3.4: Generalization of standard continuous control policies for *Walker2d-v2*. Top: Performance with varying testing noise $\sigma^{(o,u,s)}$ and environment variation Σ scale. Bottom: Heatmaps illustrate policy performance over a grid of environmental domain-shifts. Each cell corresponds to a particular set of context parameters θ^h_{te} with training domain at $(0,0)$. Results are averaged over 12 random seeds. See Table 3.4 and Figure 3.7 for results summarised over all benchmark environments.

3.3.3 Does modelling noise and variability in training improve generalisation?

We saw above that performance degrades rapidly with noise. However, as discussed earlier, testing a deterministically trained policy in a stochastic environment can be seen as a form of domain-shift ($\sigma_{tr} = 0 \rightarrow \sigma_{te} > 0$). We therefore study if reducing this domain shift by adding noise and environment variation during training improves generalisation.

As a detailed example, we analyse PPO-trained *Walker2d* in Table 3.3. To reduce domain shift between training and testing, we train the policies in noisy environment ($\sigma_{tr} = 0.2$) to align with a testing condition, in comparison to training with default environment ($\sigma_{tr} = 0$). We compare both training performance $\eta_{\sigma_{tr}}$ (Table 3.3a) and testing return $\eta_{\sigma_{te}}$ under multiple testing noise levels σ_{te} (Table 3.3b). The experiment considers both i.i.d Gaussian noise and training domain randomisation in preparation for testing on novel domains (denoted 'Dom'). The expected testing return results (cf. Eq. (3.4)) are averaged over 12 training $\times$ 20 testing seeds.

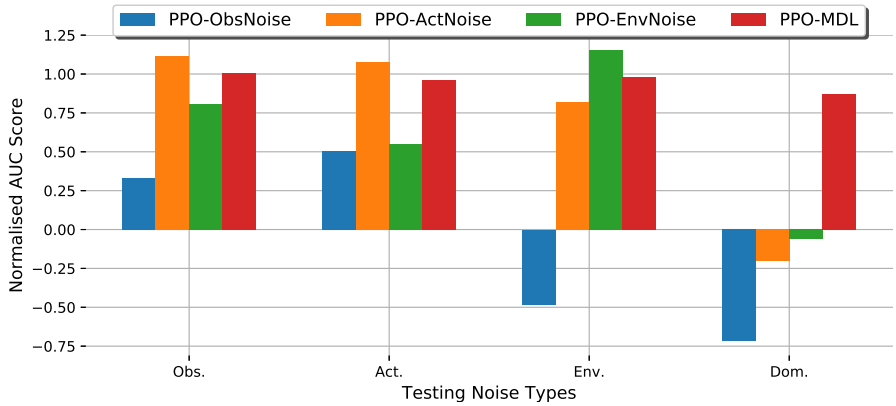
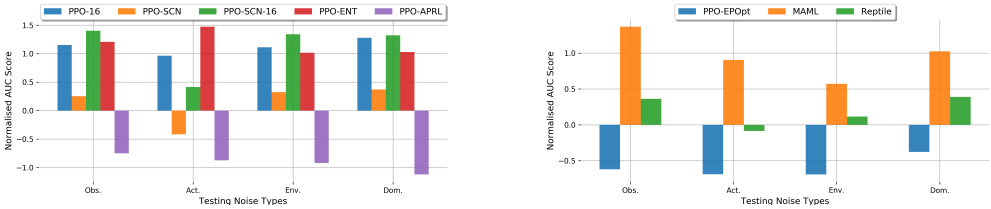


Figure 3.5: Change in normalised testing AUC score when training with more stochastic environments compared to training with a deterministic environment. Positive bars imply improvements in generalisation performance over vanilla PPO. Results are averaged over all five training environments. See Table 3.5 for a breakdown of results by benchmark environment.

From the results in Table 3.3, we can see that (i) in each case, except environmental noise, the (expected) training return is significantly lower when adding noise (Table 3.3a, compare cols), (ii) For observation and action noise, training with noise in preparation for testing with noise improves performance compared to the naive deterministic training (Table 3.3b, compare green numbers). (iii) However, there is not a clear benefit from removing the domain shift in this way if the testing scenario contains environment noise, or novel domains compared to training (Table 3.3b, compare red numbers). (iv) Finally, we evaluate the impact of a domain-shift $\sigma_{tr} = 0.2 \rightarrow \sigma_{te} = 0.4$ corresponding to mis-specified noise strength. In this case, we can see that while testing performance has generally degraded at $\sigma_{te} = 0.4$ compared to $\sigma_{te} = 0.2$, the degradation is ameliorated significantly (compared to deterministic training) in the case of action noise and observation noise.

Figure 3.5 shows the impact of training with multiple domains or different noise compared to training with deterministic environments, averaging over all five benchmark environments. Results are expressed as difference to vanilla PPO. Detailed results across all tasks under the aggregate testing AUC metric (Eq. (3.6)) are visible in Table 3.5. Overall the results show positive improved generalisation when training with noisy environments. Interestingly, there is some transferability across noise types. MDL training often improves robustness not only to novel domains at testing, but also i.i.d observation, action, and environment noise. Meanwhile, observation



(a) Training with one domain

(b) Training with multiple domains

Figure 3.6: Change in normalised testing AUC score using various training settings. (a) Impact of different single-domain training methods compared to vanilla PPO training. (b) Impact of different multiple-domain training methods compared to PPO-MDL training. Results are averaged over all five training environments. See Table 3.6 for a breakdown of results by benchmark environment.

noise training improves robustness to action noise and MDL testing in *HalfCheetah*. In summary, *modeling noise and environmental variability during training often helps to improve generalisation at testing*, but better methods are still necessary particularly if the uncertainty is mis-specified.

3.3.4 What existing techniques improve generalisation?

We next investigate if any of the robustness-promoting methods discussed in Section 3.2.1 improve generalisation performance. We build on PPO due to being easier to integrate with the modifications (unlike, e.g., TRPO), and its good stability and training efficiency.

Single Domain Learning We first consider the case where policies are trained in a single domain. Figure 3.6(a) illustrates the change in testing generalisation performance (normalised testing AUC score) using each training technique when compared to vanilla PPO, aggregated over all environments tested. Detailed expected testing AUC (Eq. (3.6)) results for all enviroments are summarised in Table 3.6. We can see that: (i) The smaller PPO-16 often surpasses the classic PPO architecture with 64 hidden units each layer in generalisation, and similarly for SCN vs SCN-16. (ii) Entropy-regularised PPO usually surpasses vanilla PPO, sometimes by a large margin. (iii) Adversarial PPO-APRL sometimes improves, but often also worsens vanilla PPO. (iv) The best performing model is either PPO-16, SCN-16, or PPO-Ent. Thus generalisation performance can be increased by reducing architecture size, or adding regularisers to reduce overfitting. However, there is no specific overfitting reduction

strategy that works consistently across environments and noise types.

Multi-Domain Learning As discussed before, aggregating the experience from multiple domains (PPO-MDL) helps improving the generalisation performance. Next, we investigate several methods that train on multiple domains and compare them to the PPO-MDL baseline, namely: PPO-EPOpt [Rajeswaran et al., 2016], and our proposed PPO-MAML-DG and PPO-Reptile-DG. Figure 3.6(b) shows the changes in testing generalisation performance aggregated over environments. The results for all environments are as AUCs in Table 3.7. We can see that: (i) EPOpt does not generalise well for unseen domains compared to PPO-MDL in most cases. (ii) Our adaptations of few-shot meta-learning methods (MAML and Reptile) to multi-domain training do succeed in learning more generalisable policies compared to vanilla MDL domain randomisation.

Overall, smaller architectures, SCN, and entropy-regularisation are promising strategies to improve generalisation given single-domain/deterministic training. Meanwhile, if multiple training domains are available, MDL domain randomisation and more advanced methods such as PPO-MAML-DG can also benefit. To summarise these most promising results, Figure 3.8 shows the generalisation curves for PPO-Ent, PPO-MDL and PPO-MAML-DG, where improvement over the PPO baseline is usually clear.

3.4 Summary

In this chapter, we contributed an analysis and set of benchmarks to investigate generalisation in deep RL. The analysis showed that standard algorithms and architectures generalise poorly in the face of noise and environmental shift. In particular, training and testing performance are often anti-correlated, so the standard practice of developing models with the aim of maximising training performance may be leading the community to produce less robust models. The results showed that different off-the-shelf algorithms better address different aspects of generalisation performance, and various enhanced training strategies can also improve aspects of generalisation. However there is currently no generally good solution to all facets of generalisation, and new algorithms are needed.

Table 3.3: Improving Walker2d-PPO generalization by training with noise. (a) Compares training return in deterministic ($\sigma_{tr} = 0.0$) and noisy ($\sigma_{tr} = 0.2$) conditions. (b) Compares the testing performance of different training conditions. 'MDL' and 'DOM' refer to training or testing on multiple domains respectively. For simplicity we overload σ to refer to Σ in the multi-domain setting, and use $\eta_\sigma(\pi)$ to indicate expected return $\eta_{p(\Theta;\Sigma)}(\pi)$.

(a) Training performance $\eta_{\sigma_{tr}}$ of deterministic vs. noisy training

TRAIN WITH $\sigma_{tr} = 0.0$	TRAIN WITH $\sigma_{tr} = 0.2$	
RETURN $\eta_{\sigma_{tr}}(\pi)$	NOISE TYPE	RETURN $\eta_{\sigma_{tr}}(\pi)$
3128.6 ± 402.3	OBS.	1424.0 ± 472.9
	ACT.	2694.8 ± 504.9
	ENV.	3261.3 ± 436.5
	MDL.	2445.3 ± 631.0

(b) Testing performance $\eta_{\sigma_{te}}$ of deterministic vs. noisy training

NOISE TYPE	σ_{te}	TRAIN WITH $\sigma_{tr} = 0.0$	TRAIN WITH $\sigma_{tr} = 0.2$
OBS.	0.0	3723.9± 321.3	3682.9±415.6
	0.2	2522.2 ±593.4	3036.1±576.3
	0.4	937.8 ± 308.9	1303.8±374.5
ACT.	0.0	3678.8±355.5	3596.5 ±406.3
	0.2	2096.3±570.4	3044.0±525.6
	0.4	919.7±249.5	1951.5±486.9
ENV.	0.0	3756.3±341.7	3761.8±415.1
	0.2	3133.5±465.9	3225.4±450.0
	0.4	2528.6±415.9	2660.1 ± 436.1
DOM.	0.0	3725.1±343.7	3417.9±549.2
	0.2	3228.7±408.6	2985.2±486.4
	0.4	2462.3±394.4	2468.2±341.9

Table 3.4: Generalisation performance of basic Deep RL learners in terms of AUC scores. Algorithms are trained on five standard deterministic environments, and tested with varying degrees of noise and domain-shift. Results are averaged over 12 training seeds $\times$ 20 testing seeds. For each testing condition (row), the results of the best-performing algorithm, as well as algorithms that are not significantly different are highlighted in boldface. Welch's t-test with $p < 0.05$ is used for significance testing.

ENV. NAME		PPO	TRPO	DDPG	SAC
WALKER2D	OBS.	913.6 ± 284.9	895.4 ± 229.3	544.3 ± 204.4	**1565.4 ± 544.9**
	ACT.	845.5 ± 251.4	1295.5 ± 310.2	1115.2 ± 531.1	**1725.5 ± 463.8**
	ENV.	2141.1 ± 558.5	2653.6 ± 402.3	1550.5 ± 671.4	**3391.6 ± 527.5**
	DOM.	2310.0 ± 596.7	**2648.4 ± 417.5**	1929.4 ± 839.8	**2779.8 ± 443.8**
HOPPER	OBS.	**592.6 ± 244.4**	**643.8 ± 107.2**	352.7 ± 245.5	502.7 ± 223.2
	ACT.	609.0 ± 167.6	683.0 ± 121.8	582.3 ± 163.9	**837.5 ± 219.8**
	ENV.	1241.6 ± 438.7	1638.7 ± 383.7	1042.2 ± 337.9	**1882.9 ± 824.3**
	DOM.	**1640.0 ± 507.4**	**1622.6 ± 374.2**	1073.5 ± 358.3	1186.1 ± 298.7
HALFCHEETAH	OBS.	1029.3 ± 136.0	1045.9 ± 444.6	1152.7 ± 361.0	**1538.3 ± 646.6**
	ACT.	675.4 ± 170.7	906.1 ± 530.3	943.6 ± 312.8	**1425.9 ± 692.3**
	ENV.	1989.0 ± 658.2	2322.0 ± 1235.3	1856.4 ± 481.0	**2526.0 ± 810.9**
	DOM.	1871.0 ± 583.5	2264.7 ± 1191.2	1963.9 ± 451.7	**3036.8 ± 1326.7**
PENDULUM	OBS.	139.5 ± 32.5	**182.8 ± 139.9**	94.9 ± 33.1	144.4 ± 36.7
	ACT.	362.7 ± 43.7	201.0 ± 112.8	393.8 ± 14.1	**625.8 ± 151.6**
	ENV.	**995.4 ± 8.0**	678.9 ± 371.3	247.2 ± 22.4	543.8 ± 147.5
	DOM.	**942.2 ± 177.6**	676.1 ± 370.9	609.2 ± 281.3	861.2 ± 121.2
D-PENDULUM	OBS.	**923.2 ± 61.5**	715.1 ± 215.0	300.8 ± 2.9	725.8 ± 282.3
	ACT.	**2110.7 ± 74.6**	1280.9 ± 286.4	438.5 ± 32.7	1953.9 ± 429.3
	ENV.	**7494.9 ± 1189.1**	3023.9 ± 753.5	950.2 ± 45.2	4285.7 ± 1571.9
	DOM.	**7103.8 ± 850.6**	6103.9 ± 1530.1	3470.3 ± 350.6	4866.7 ± 1286.0

Table 3.5: Generalisation performance of training with noisy and multiple environments in terms of AUC score. All settings are evaluated with 5 environments and 4 noise types in each environment. For all noisy training settings, gaussian noise with 0.2 standard deviation is used during training. All performance are averaged over 12 training seeds $\times$ 20 testing seeds. Results of the best-performing training setting on each testing condition, as well as the ones that are not significantly different, are highlighted in boldface. Welch's t-test with $p < 0.05$ is used for significant testing.

ENV. NAME	TEST	PPO	PPO-OBSNOISE	PPO-ACTNOISE	PPO-ENVNOISE	PPO-MDL
WALKER2D	OBS.	913.6 ± 284.9	$\mathbf{1455.9 \pm 388.9}$	1172.5 ± 204.3	849.1 ± 100.5	1114.4 ± 378.2
	ACT.	845.5 ± 251.4	$\mathbf{2078.3 \pm 522.8}$	1162.9 ± 199.1	779.8 ± 72.8	1082.1 ± 349.5
	ENV.	2141.1 ± 558.5	2301.8 ± 521.0	2982.5 ± 695.0	$\mathbf{3120.0 \pm 419.9}$	2232.5 ± 609.4
	DOM.	$\mathbf{2310.0 \pm 596.7}$	$\mathbf{2313.7 \pm 521.9}$	1959.5 ± 422.0	1790.6 ± 240.8	2232.9 ± 601.8
HOPPER	OBS.	592.6 ± 244.4	592.6 ± 332.5	$\mathbf{824.7 \pm 270.6}$	754.3 ± 241.1	733.6 ± 143.4
	ACT.	609.0 ± 167.6	717.8 ± 382.7	$\mathbf{926.1 \pm 189.6}$	711.3 ± 209.4	$\mathbf{905.8 \pm 143.4}$
	ENV.	1241.6 ± 438.7	1164.9 ± 804.3	1770.7 ± 646.1	$\mathbf{2025.1 \pm 520.7}$	1935.8 ± 428.4
	DOM.	1640.0 ± 507.4	1167.8 ± 813.6	1444.4 ± 463.9	1584.4 ± 478.1	$\mathbf{1939.7 \pm 425.0}$
HALFCHEETAH	OBS.	1029.3 ± 136.0	$\mathbf{1787.3 \pm 562.8}$	999.7 ± 233.9	1166.3 ± 409.7	1629.6 ± 585.5
	ACT.	675.4 ± 170.7	$\mathbf{1865.0 \pm 652.4}$	961.2 ± 351.7	1069.2 ± 524.1	1219.3 ± 502.2
	ENV.	1989.0 ± 658.2	3633.7 ± 1039.0	2137.4 ± 1122.5	2750.7 ± 1477.9	$\mathbf{3898.6 \pm 1590.5}$
	DOM.	1871.0 ± 583.5	$\mathbf{3588.5 \pm 1020.7}$	2108.2 ± 1131.0	2763.7 ± 1508.6	$\mathbf{3748.2 \pm 1550.7}$
PENDULUM	OBS.	139.5 ± 32.5	54.1 ± 25.0	$\mathbf{247.3 \pm 36.1}$	221.6 ± 34.3	162.3 ± 39.9
	ACT.	362.7 ± 43.7	250.8 ± 109.4	$\mathbf{453.6 \pm 16.8}$	429.5 ± 30.9	393.0 ± 16.2
	ENV.	995.4 ± 8.0	774.5 ± 382.4	981.5 ± 28.5	967.9 ± 40.5	$\mathbf{999.1 \pm 2.4}$
	DOM.	942.2 ± 177.6	775.0 ± 381.8	979.4 ± 26.4	981.6 ± 25.6	$\mathbf{999.3 \pm 1.8}$
D-PENDULUM	OBS.	923.2 ± 61.5	160.8 ± 13.9	1068.6 ± 81.5	$\mathbf{1297.4 \pm 74.0}$	1071.8 ± 87.1
	ACT.	2110.7 ± 74.6	301.2 ± 64.6	$\mathbf{2194.3 \pm 91.6}$	$\mathbf{2206.0 \pm 57.1}$	2161.5 ± 36.0
	ENV.	7494.9 ± 1189.1	312.8 ± 50.5	9263.3 ± 121.2	$\mathbf{9301.0 \pm 78.8}$	8183.7 ± 144.5
	DOM.	7103.8 ± 850.6	1178.9 ± 688.1	9107.6 ± 71.8	$\mathbf{9319.5 \pm 75.1}$	8174.2 ± 178.2

Table 3.6: Generalisation performance of different algorithms and architectures in terms of AUC score. Algorithms are trained on five standard deterministic environments, and tested with varying degrees of noise and domain-shift. All performance are averaged over 12 training seeds $\times$ 20 testing seeds. The results of the best-performing algorithm on each testing condition, as well as all algorithms that are not significantly different, are highlighted in boldface. Welch's t-test with $p < 0.05$ is used for significant testing.

ENV. NAME		PPO	PPO-16	PPO-SCN	PPO-SCN-16	PPO-ENT	PPO-APRL
WALKER2D	OBS.	913.6 ± 284.9	1243.8 ± 370.6	1035.6 ± 223.9	1700.5 ± 401.3	**2442.6 ± 723.6**	992.6 ± 243.9
	ACT.	845.5 ± 251.4	1181.6 ± 418.2	944.8 ± 299.1	1677.2 ± 464.6	**2843.9 ± 1075.6**	862.4 ± 185.7
	ENV.	2141.1 ± 558.5	2337.4 ± 633.7	2180.5 ± 458.9	3050.9 ± 805.8	**3383.3 ± 725.8**	2038.3 ± 489.0
	DOM.	2310.0 ± 596.7	2315.8 ± 618.4	2166.6 ± 447.6	**3039.5 ± 810.8**	**3271.8 ± 740.4**	1086.3 ± 323.3
HOPPER	OBS.	592.6 ± 244.4	936.5 ± 302.4	576.1 ± 175.9	**943.1 ± 244.7**	764.6 ± 96.6	624.0 ± 175.0
	ACT.	609.0 ± 167.6	994.1 ± 296.4	550.6 ± 124.0	**1060.9 ± 299.3**	890.8 ± 108.7	724.7 ± 181.7
	ENV.	1241.6 ± 438.7	1971.6 ± 545.3	1277.8 ± 314.8	**2109.0 ± 651.4**	1856.8 ± 240.1	1344.1 ± 320.2
	DOM.	1640.0 ± 507.4	1970.2 ± 543.4	1277.1 ± 317.2	**2111.3 ± 650.8**	1779.9 ± 259.1	1186.1 ± 298.7
HALFCHEETAH	OBS.	1029.3 ± 136.0	**1325.4 ± 357.4**	956.4 ± 59.2	1116.6 ± 287.9	**1382.3 ± 579.7**	942.2 ± 87.0
	ACT.	675.4 ± 170.7	1099.4 ± 399.3	771.2 ± 160.6	986.2 ± 412.9	**1316.9 ± 782.1**	685.5 ± 166.6
	ENV.	1989.0 ± 658.2	**3213.8 ± 1417.0**	2837.1 ± 937.6	3014.3 ± 1109.5	2835.7 ± 1570.0	2202.6 ± 927.7
	DOM.	1871.0 ± 583.5	**3035.4 ± 1309.8**	2478.3 ± 1129.5	2730.3 ± 1405.9	2677.5 ± 1454.6	2035.4 ± 756.4
PENDULUM	OBS.	139.5 ± 32.5	146.4 ± 39.8	148.1 ± 74.9	**177.4 ± 46.2**	121.6 ± 22.9	57.4 ± 58.4
	ACT.	362.7 ± 43.7	**389.4 ± 18.9**	287.4 ± 122.3	302.5 ± 80.1	363.6 ± 26.8	156.8 ± 165.1
	ENV.	995.4 ± 8.0	**998.7 ± 4.2**	820.6 ± 344.9	975.8 ± 24.7	**999.2 ± 3.6**	381.8 ± 401.4
	DOM.	942.2 ± 177.6	998.7 ± 4.0	820.5 ± 345.0	975.4 ± 25.2	**999.9 ± 0.3**	406.3 ± 430.1
D-PENDULUM	OBS.	923.2 ± 61.5	1014.1 ± 78.4	1051.3 ± 151.0	**1093.8 ± 121.6**	956.1 ± 66.9	754.1 ± 82.6
	ACT.	2110.7 ± 74.6	**2160.2 ± 44.2**	2016.9 ± 70.8	1944.1 ± 55.9	**2168.5 ± 66.3**	1917.0 ± 64.6
	ENV.	7494.9 ± 1189.1	**8036.7 ± 194.5**	**7997.1 ± 283.9**	7856.8 ± 601.9	5622.9 ± 929.1	3147.3 ± 1157.1
	DOM.	7103.8 ± 850.6	**8032.1 ± 188.8**	**8006.9 ± 278.6**	7844.4 ± 654.8	7436.7 ± 696.6	6667.0 ± 1233.4

Table 3.7: Generalisation performance of different multiple domain training methods in terms of AUC score. All settings are evaluated with 5 environments and 4 noise types in each environment. Training domain parameters are sampled from $\theta_d^h \sim \mathcal{N}(\cdot, 0.2)$ for all training settings except for PPO. All performance are averaged over 12 training seeds $\times$ 20 testing seeds. The results of best-performing algorithm in each testing condition, as well as all algorithms that are not significantly different, are highlighted in boldface. Welch's t-test with $p < 0.05$ is used for significant testing.

Env. Name		PPO	PPO-MDL	PPO-EPOpt	PPO-MAML-DG	PPO-Reptile-DG
Walker2d	obs.	913.6 ± 284.9	1114.4 ± 378.2	765.2 ± 322.2	1278.9 ± 427.7	**1550.4 ± 345.1**
	act.	845.5 ± 251.4	1082.1 ± 349.5	741.9 ± 218.8	1311.6 ± 489.0	**1466.2 ± 294.7**
	env.	2141.1 ± 558.5	2232.5 ± 609.4	2003.8 ± 810.9	**3872.3 ± 1241.7**	**3889.0 ± 467.4**
	dom.	2310.0 ± 596.7	2232.9 ± 601.8	1503.7 ± 456.8	**2842.2 ± 962.5**	**2793.9 ± 413.3**
Hopper	obs.	592.6 ± 244.4	733.6 ± 143.4	646.1 ± 88.1	779.1 ± 131.7	**818.4 ± 206.6**
	act.	609.0 ± 167.6	**905.8 ± 143.4**	584.8 ± 75.7	777.4 ± 71.9	**927.2 ± 205.4**
	env.	1241.6 ± 438.7	**1935.8 ± 428.4**	1101.6 ± 181.5	1594.9 ± 301.6	**2379.8 ± 835.2**
	dom.	1640.0 ± 507.4	**1939.7 ± 425.0**	1066.1 ± 216.9	1559.9 ± 220.3	**2048.8 ± 488.7**
HalfCheetah	obs.	1029.3 ± 136.0	**1629.6 ± 585.5**	1176.4 ± 466.7	**1673.8 ± 424.3**	1067.0 ± 73.2
	act.	675.4 ± 170.7	1219.3 ± 502.2	1074.6 ± 474.5	**1977.1 ± 662.1**	834.4 ± 287.8
	env.	1989.0 ± 658.2	3898.6 ± 1590.5	3042.7 ± 1770.9	**4249.5 ± 1537.5**	2565.6 ± 1154.7
	dom.	1871.0 ± 583.5	3748.2 ± 1550.7	3068.0 ± 1807.4	**4235.1 ± 1539.3**	2451.9 ± 1037.2
Pendulum	obs.	139.5 ± 32.5	162.3 ± 39.9	187.7 ± 37.7	**388.7 ± 40.9**	266.8 ± 39.5
	act.	362.7 ± 43.7	393.0 ± 16.2	377.9 ± 39.1	**485.4 ± 18.7**	405.7 ± 19.2
	env.	995.4 ± 8.0	**999.1 ± 2.4**	521.5 ± 99.1	807.6 ± 111.0	997.7 ± 5.2
	dom.	942.2 ± 177.6	999.3 ± 1.8	931.1 ± 99.9	983.4 ± 31.1	**1000.0 ± 0.0**
D-Pendulum	obs.	923.2 ± 61.5	1071.8 ± 87.1	1012.2 ± 133.3	**1098.7 ± 127.9**	979.9 ± 56.3
	act.	2110.7 ± 74.6	2161.5 ± 36.0	**2177.3 ± 74.1**	**2178.1 ± 79.1**	2135.8 ± 48.8
	env.	7494.9 ± 1189.1	8183.7 ± 144.5	**9179.4 ± 149.9**	9042.4 ± 281.7	5893.0 ± 810.0
	dom.	7103.8 ± 850.6	8174.2 ± 178.2	9133.3 ± 236.1	**9184.0 ± 131.5**	8293.3 ± 94.6

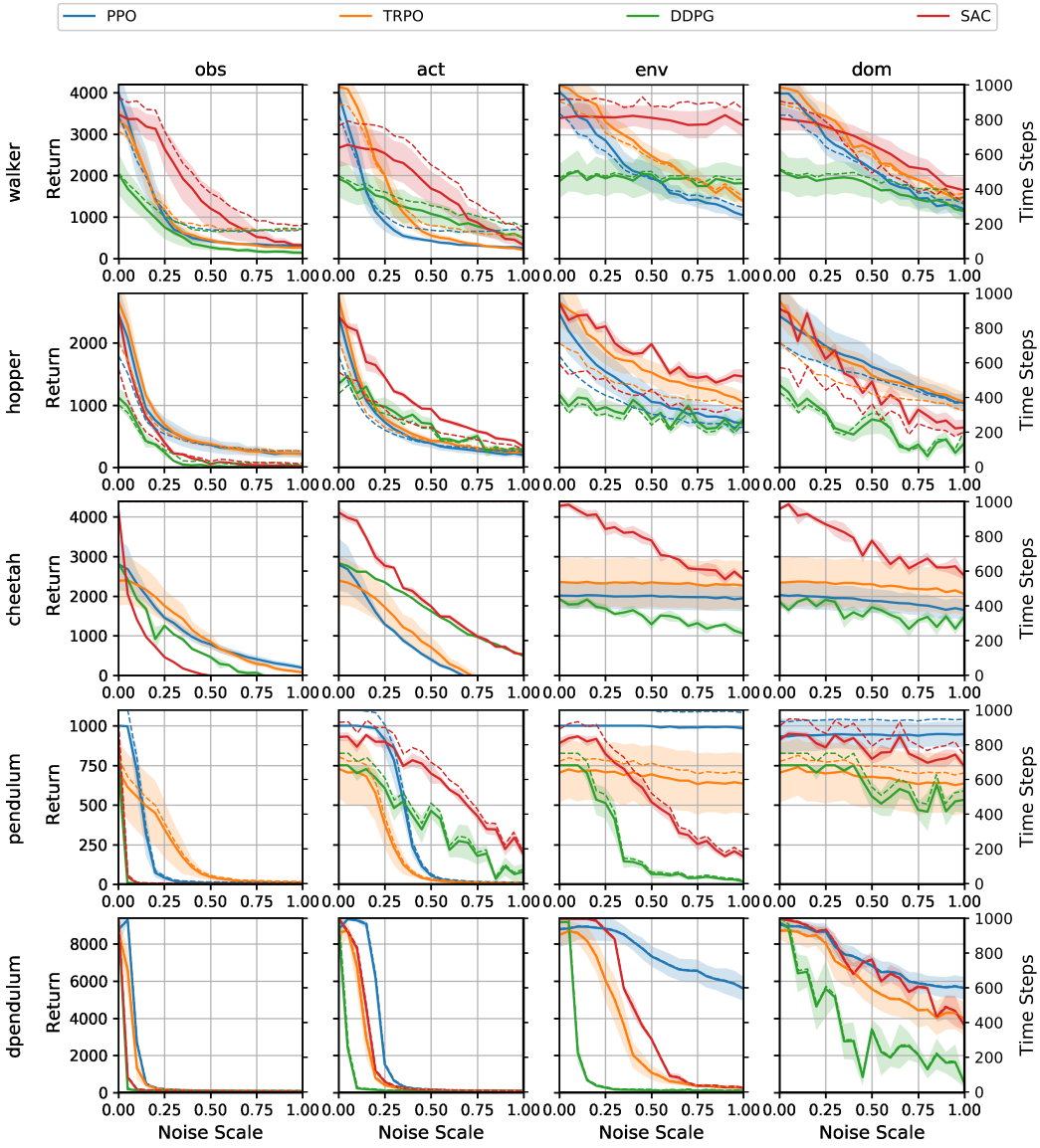


Figure 3.7: Generalisation of four basic learning algorithms across all five benchmark environments with varying testing noise $\sigma^{(o,u,s)}$ and environment variation Σ scale.

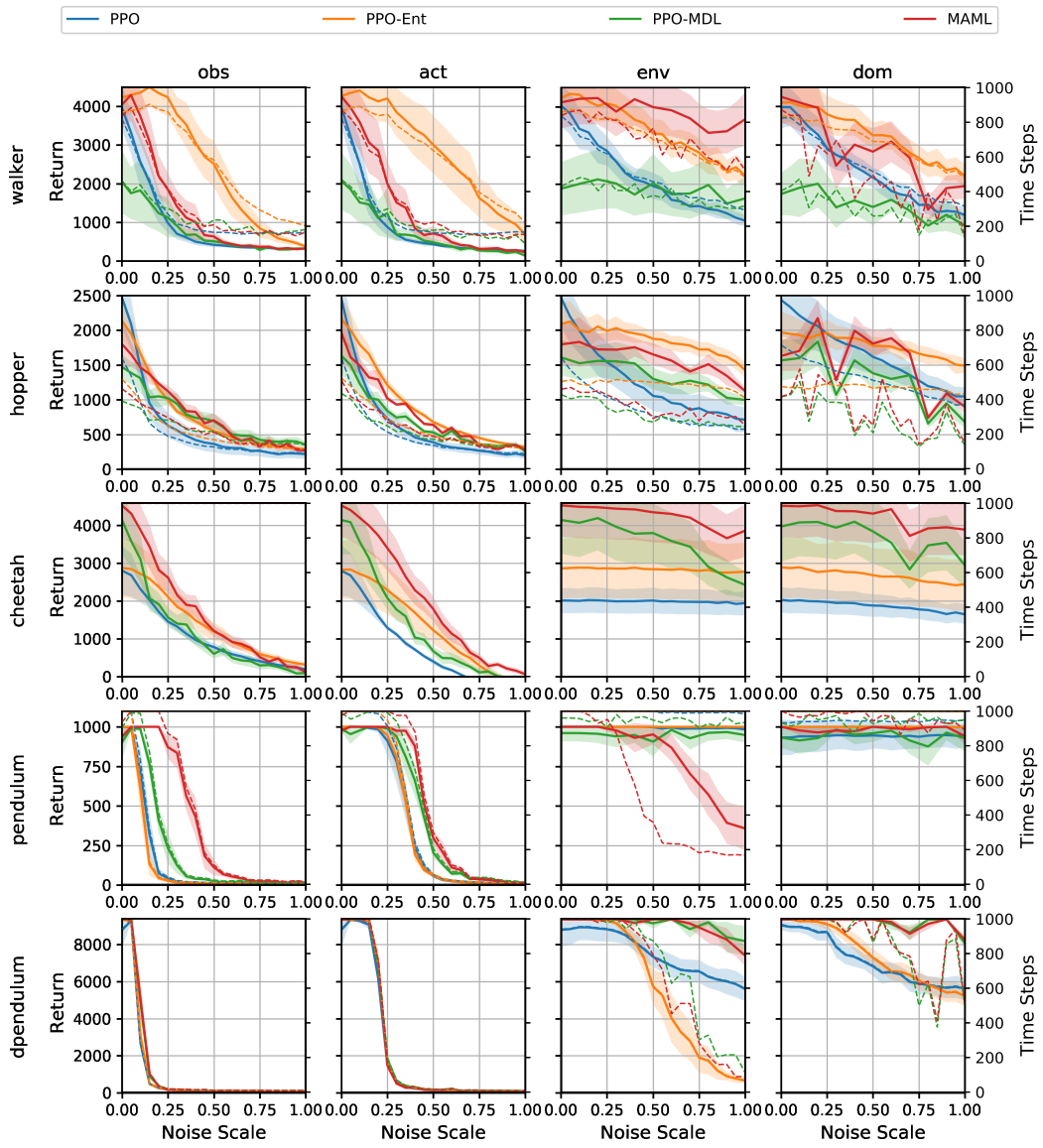


Figure 3.8: Generalisation of selected learning algorithms across all five benchmark environments with varying testing noise $\sigma^{(o,u,s)}$ and environment variation Σ scale.

Chapter 4

Regularising Multi-Domain Learning with KL Divergence

As we have shown in the previous chapter, training with multiple domains helps policies generalising against different sources of uncertainties and domain shifts. In this chapter, we take one step further towards improving generalisation with multi-domain learning (MDL).

One common strategy to aggregate information from different domains is to merge data into a single batch [Tobin et al., 2017]. This usually requires manual tuning on the specification of training domain set, because learning may become unstable with a diverse set of training domains. Another strategy is to compute the gradients individually in every domain and to optimise the policy with the average gradient. This suffers from the conflicting gradients, especially when the curvature of optimisation landscape is high and the difference in gradient magnitudes is large [Yu et al., 2020]. To avoid these negative interference, Rajeswaran et al. [2016] selected a subset of domains for updating based on the policies' performance. However, as part of data is excluded from training, this method sacrifices sample efficiency. Instead, we propose to train individual policy on each domain separately and aggregate the gradient information such that the so as to minimise the differences in policy outputs of trained individuals and aggregated policy. More specifically, considering stochastic policies which output probability distribution of actions, the main policy is optimised such that the total statistical distance between outputs of the main policy to every individuals are minimised given a batch of states. In this work, we use Kullback-Leibler (KL) divergence as an alternative of distance metrics, as it bounds the total variation distance and is easier to estimate.

Next in Section 4.1, we introduce the background and problem setting of MDL. In Section 4.2, we describe our proposed method of aggregating information from different domains regularised by KL divergence. In Section 4.3, we evaluate the proposed method on two benchmarking continuous control tasks: *Walker* and *Hopper*, and demonstrate more robust learning process and better generalisation behaviour compared to the baseline methods.

4.1 Background

4.1.1 Policy Gradient Methods

We consider Markov Decision Processes (MDPs) described by the tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where the transition function P is parameterised by the domain descriptor ξ, and stochastic policies $\pi_\theta(a|s)$, parameterised by θ. In general, the goal of an reinforcement learning (RL) agent is to find the optimal policy θ^* that maximises the expected accumulated return under a certain domain ξ_d

$$J(\theta, \xi_d) = \mathbb{E}_\tau\big[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)|\theta, \xi_d\big], \tag{4.1}$$

where the trajectory τ contains a sequence of state-action pair, $\tau = \{s_t, a_t\}_{t=0}^{T}$.

Empirically, estimating the gradient of $J(\theta, \xi_d)$ in Eq. (4.1) directly suffers from the problem of high variances. General techniques to reduce the variance includes subtracting cumulative return with baselines [Peters and Schaal, 2008; Schulman et al., 2015b], using an alternative objective function with lower variance [Schulman et al., 2015a, 2017], etc. For instance, Trust Region Policy Optimisation (TRPO) [Schulman et al., 2015a] uses the following surrogate loss as a local approximation of objective function:

$$L_{\theta^{\text{old}}}(\theta) = \mathbb{E}_{(s_t, a_t)}\big[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_{\theta_{\text{old}}}(s_t, a_t)\big], \tag{4.2}$$

where θ_{old} is the vector of policy parameters before the update, $\hat{A}$ is the estimated advantage function, and the trajectories are sampled with the $\pi_{\theta_{\text{old}}}$ and ξ_d. They provide the following lower bound for policy improvement that

$$J(\theta, \xi_d) \geq L_{\theta^{\text{old}}}(\theta) - C\alpha^2, \tag{4.3}$$

where $\alpha = \max_s D_{\text{TV}}(\pi_{\theta_{\text{old}}}(\cdot|s)|\pi_\theta(\cdot|s))$ is the maximum total variation distance between output distributions under policy θ_{old} and θ. This lower bound shows maximising the surrogate loss $L_{\theta^{\text{old}}}(\theta)$ is guaranteed to improve objective $J(\theta, \xi_d)$ given a small

update steps. TRPO uses a constraint on KL divergence between old and updated policy, i.e., trust region constraint. Proximal Policy Optimisation (PPO) uses the clipped probability ratio to discourage large policy change [Schulman et al., 2017].

4.1.2 Multi-Domain Learning

In multi-domain learning, the objective is to maximise the expectation of the expected return over a distribution of domains $\xi_d \in \Xi$:

$$J(\theta) = \mathbb{E}_{\xi_d}\big[\mathbb{E}_\tau\big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|\theta, \xi_d\big]\big]. \tag{4.4}$$

In practice, the expectation is estimated by Monte-Carlo average over the sampled domain parameters $\xi_1, \xi_2, ..., \xi_N$. And the objective function is

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} J(\theta, \xi_i). \tag{4.5}$$

While it is appealing to optimise the objective in each individual domain and take the average of all sub-policies in parameter space, empirically, it does not perform well. The gradients from different domains or batches could interfere negatively with each other. This is especially problematic under the co-occurrence of the following conditions: (1) when gradients from different domains are conflicting to each other with respect to their directions; (2) when the difference in gradient magnitudes is large, leading to one gradient dominating the others; and (3) there is high curvature in the optimisation landscape. When averaging the gradients from different domains in such case, the improvement in performance of dominating domain could be significantly overestimated, while the degradation in performance of other domains may be significantly underestimated [Yu et al., 2020]. Moreover, averaging gradients in Euclidean space could potentially break the constraints of small update step and lead to a large total variation distance. In practice, trajectory data from different domains are usually trained jointly, i.e., merged into one single batch [Rajeswaran et al., 2016; Muratore et al., 2018]. However, the variance of source domain distribution makes joint learning more unstable. It usually requires carefully tuned domain range.

4.2 Methodology

We propose a method to aggregate the information from different domains based on the distances in probability distribution space of action distributions, instead of euclidean

space of policy parameters.

Given a policy parameterised by θ_{old}, we first randomly sample a set of N domains $\{\xi_1, \xi_2, ..., \xi_N\}$. For each of the domain ξ_n, we randomly sample a batch of trajectories with T timesteps using current policy θ_{old}. We then perform the common optimising methods (e.g. PPO, TRPO, etc.) on each domain independently. Note that the sub-policies are only optimised with corresponding batch of data, i.e., no data exchange occurs during training sub-policies. The updated sub-policies are stored as $\{\theta_1, \theta_2, ..., \theta_N\}$. One conventional way to recombine these sub-policies is to take the central policy in the space of policy parameters, as in Eq. (4.6). However, this may break the constraints on gradient step sizes, especially when the variances of gradients are large.

$$\theta_{\text{new}} = \frac{1}{N} \sum_{i=1}^{N} \theta_i \tag{4.6}$$

To address this problem, when measuring distance between two policies, we consider the distance in the probability space of their induced action distributions, instead of the euclidean distance between policy parameters. Then we take a step to minimise overall probability distance between the aggregated policy and each updated policies $\{\theta_1, \theta_2, ..., \theta_N\}$. We use KL divergence as the metric. Note that although KL divergence is not a valid distance metric between distributions, it bounds the total variation distance according to Pinsker's inequality [Tsybakov, 2009] and is simple to implement in practice.

$$\theta_{\text{new}} = \underset{\theta}{\text{argmin}} \sum_{i=1}^{N} \mathbb{E}_s \left[D_{\text{KL}}(\pi_{\theta_i}(\cdot|s) || \pi_\theta(\cdot|s)) \right]. \tag{4.7}$$

We further take the quadratic approximation to KL divergences, resulting in the following problem:

$$\theta_{\text{new}} = \underset{\theta}{\text{argmin}} \sum_{i=1}^{N} (\theta_i - \theta)^T \mathcal{I}(\theta_i)(\theta_i - \theta), \tag{4.8}$$

where $\mathcal{I}(\theta_i)$ is the expected Fisher information matrix (FIM) constructed by analytically computing the Hessian of the KL divergence [Gourieroux and Monfort, 1995], i.e.,

$$\mathcal{I}(\theta_i) = \frac{\partial^2}{\partial \theta^2} \mathbb{E}_s \left[D_{\text{KL}}(\pi_{\theta_i}(\cdot|s) || \pi_\theta(\cdot|s)) \right]. \tag{4.9}$$

Note that we ignore changes in state visitation density due to changes in the policy, and trajectories generated by θ_{old} are used to estimate expected FIMs for all sub-policies.

Algorithm 3 KL Regularised Multi Domain Learning

Input: number of sampled domains N, trajectory timesteps T, update epochs K
Input: initial policy θ_{old}, initial value network ϕ_{old}
while not converged **do**
  Sample a batch of domains $\{\xi_1, \xi_2, ..., \xi_N\}$
  for all training domains ξ_i **do**
    Sample trajectory τ_i in domain ξ_i for T timesteps with policy θ_{old}
  end for
  for $k = 1$ **to** K **do**
    for all (ξ_i, τ_i) **do**
      Update the policy θ_i and value network ϕ_i with trajectory τ_i using PPO
    end for
    Compute θ_{new} following Eq. (4.11) and $\phi_{\text{new}} = \frac{1}{N}\sum_{i=1}^{N}\phi_i$
    Assign $\theta_{\text{old}} \leftarrow \theta_{\text{new}}, \phi_{\text{old}} \leftarrow \phi_{\text{new}}$
  end for
end while

Let $\{g_1, g_2, ..., g_N\}$ denote the policy change in each domain ($g_i = \theta_i - \theta_{\text{old}}$), and $\bar{g}$ denote the average change $\bar{g} = \frac{1}{N}\sum_{i=1}^{N} g_i$. This has the closed form solution that

$$\theta_{\text{new}} = \Big[\sum_{i=1}^{N} \mathcal{I}(\theta_i)\Big]^{-1} \Big[\sum_{i=1}^{N} \mathcal{I}(\theta_i)\theta_i\Big] \tag{4.10}$$

$$= \theta_{\text{old}} + \bar{g} + \Big[\sum_{i=1}^{N} \mathcal{I}(\theta_i)\Big]^{-1} \Big[\sum_{i=1}^{N} \mathcal{I}(\theta_i)(g_i - \bar{g})\Big]. \tag{4.11}$$

In large-scale problems, where it is costly to compute the inverse of FIMs, this can be approximately solved by conjugate gradient methods after multiple gradient steps without forming the full matrices $\mathcal{I}(\theta_i)$.

Note that we only apply this update procedure to policy networks, of which the outputs are probabilistic distributions. For the cases where value network is independent from the policy, we update value network with the arithmetic average. The full procedure is described in Algorithm 3.

4.3 Experiments and Results

We design our experiments to investigate the following research questions:

- How is the learning performance when training with a distribution of domains?

- Does training with the proposed method generalise better to target domains?

4.3.1 Environment and Tasks

We evaluate the proposed methods on *Walker* and *Hopper* benchmark problems using the MuJoCo physical simulator [Todorov et al., 2012] from OpenAI Gym [Brockman et al., 2016]. In both environments, the goal is to move forward as fast as possible without falling over. The reward at each timestep is given by

$$r(s_t, a_t) = \underbrace{v}_{\text{forward velocity}} + \underbrace{1}_{\text{alive bonus}} - \underbrace{c_a \times ||a_t||_2^2}_{\text{action penalty}}, \tag{4.12}$$

where the coefficient of action penalty term c_a is 0.001 by default.

We compare our methods (denoted *kl_avg*) with two baselines: (1) *joint*: merge experience from different domains into one single batch and update the policy with PPO (which corresponds to *PPO-MDL* described in section 3.2.1); and (2) *params_avg*: update the policy in each source domain and average the sub-policies in euclidean space of policy parameters (Eq. (4.6)).

We use the same hyper-parameters as the implementation in Dhariwal et al. [2017]. The policy is designed to be a two-layer Multi-Layer Perceptrons (MLP) network with 64 hidden units each layer, which outputs the expectation of distribution, and an individual variable for standard deviation variable. The value network has the same architecture design as policy MLP but with a different set of parameters. In *joint* training setting, the batch size is 4096, the size of minibatches is 64, and the number of optimiser epochs per iteration is 10. In *kl_avg* and *params_avg* settings, for each source domain, We choose 1024 as the batch size and 64 as the size of minibatches, such that the total number of timesteps and update steps both remain the same across all settings. Note that while we use stochastic policies during training, the expectation of output distribution is used when evaluating the policies.

4.3.2 Training Performance Comparison

Despite multi-domain learning is an effective practice against potential domain shift, in practice, it is difficult to choose the distribution of source domains. Ideally, we would like to train the agents with source domains as diverse as possible to maximise the chance of covering testing scenarios. However, as the set of source domains becomes more diverse, the variance in gradient estimation would increase. and therefore,

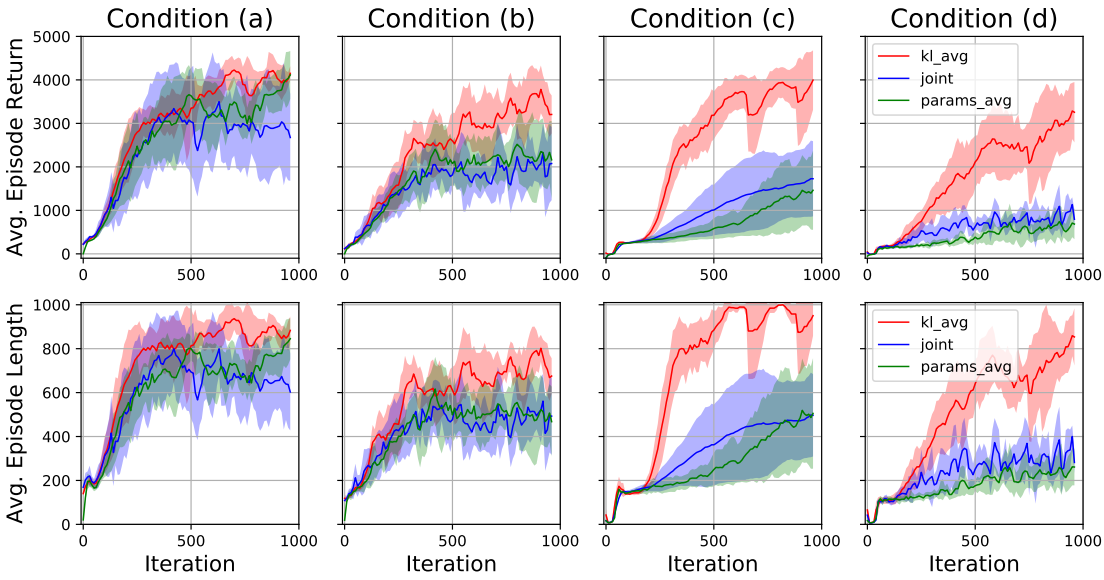
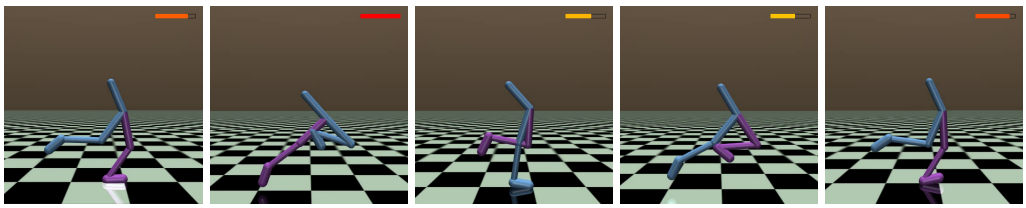


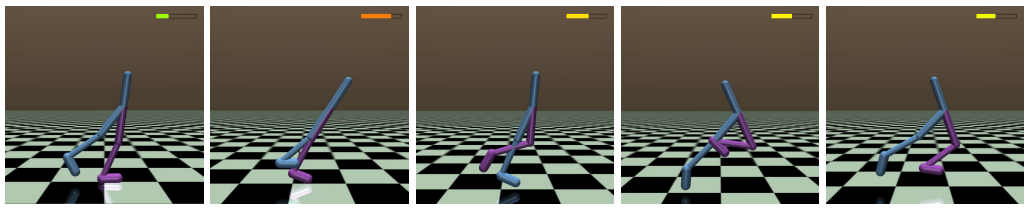
Figure 4.1: Comparisons of average episode returns and lengths during training in *Walker2d* task under four different training conditions. (a): $c_a = 0.001$, $\xi \sim \mathcal{U}(-0.5, 0.5)$, (b): $c_a = 0.001$, $\xi \sim \mathcal{U}(-2.5, 2.5)$, (c): $c_a = 1.0$, $\xi \sim \mathcal{U}(-0.5, 0.5)$, (d): $c_a = 1.0$, $\xi \sim \mathcal{U}(-2.5, 2.5)$. Each iteration consists of 4096 time steps in simulation. Results are averaged over 6 random seeds.

leads to unstable learning performance. Similar scenario happens in penalising large action: a large action penalty coefficient could make learning difficult to converge as agents need to balance different objectives while learning. We argue that, by averaging policies in the probability space of action distributions, agents are able to learn more robustly to the choices of training conditions.

For this analysis, we change the wind speeds to model domain changes during training in both *Walker* and *Hopper* tasks. Note that the wind speed is sampled once per episode. We define two different uniform distributions as training conditions: $\xi \sim \mathcal{U}(-0.5, 0.5)$ and $\xi \sim \mathcal{U}(-2.5, 2.5)$. The latter distribution covers a wider range of domains, in the meantime, it is a more difficult condition to learn jointly. We also evaluate a modified tasks with higher penalties for taking large actions. Specifically, we amplify the coefficient c_a from 0.001 to 1. Higher penalties encourage the agent to learn more natural behaviour. However, it makes learning more difficult because the agent needs to learn to balance between the objectives of moving fast and saving energy. In addition, to avoid reporting an overfitted performance, we use a different set of random seeds from the training data to evaluate the performance of policies.



(a) Walking cycle learned with *kl_avg*



(b) Walking cycle learned with *params_avg*

Figure 4.2: Illustrations of walking cycles learned with two different method. Policies are trained with $c_a = 0.1$, $\xi \sim \mathcal{U}(-2.5, 2.5)$ and evaluated under $\xi = 0$ (no wind). The best policy out of 6 random seeds is used in each method.

Figure 4.1 shows the learning performances of three algorithms in *Walker* task with four different training settings. The figures in the leftmost column show learning curves trained with the basic setting: with a small range of wind speeds and the small penalties on actions. The other columns show the performances in more complex training settings, with more diverse source domains or/and larger penalties on actions. The results demonstrate that our proposed method performs similarly to others in the basic setting (leftmost column), but outperforms other methods in more difficult settings (the other columns). This suggests our method is able to learn more robustly when the task becomes harder and is less sensitive to the choice of training conditions compared to the baselines. Moreover, to illustrate the difference among the walking gaits learned with different methods, Figure 4.2 shows five key frames from one walking cycle [1]. The walking gait is observed to be more natural with our proposed learning method.

4.3.3 Generalisation Across Domains

We next analyse how the learned policies generalise against domain shifts. Similar to the evaluation in Chapter 3, the analysis covers observation noise, action noise and environmental changes. Specifically, we test the agent with 4 different domain shifts: (1) different wind speeds, with a wider range comparing to training, (2) different masses

[1] Two videos are available on https://youtu.be/vtaBNV5bSRg and https://youtu.be/o9gjyx7-2-A.

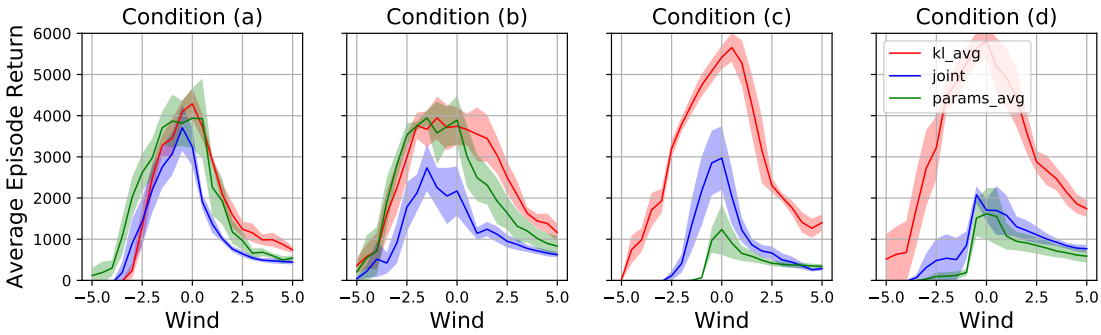
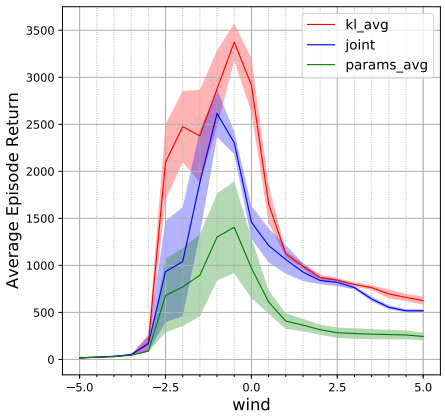
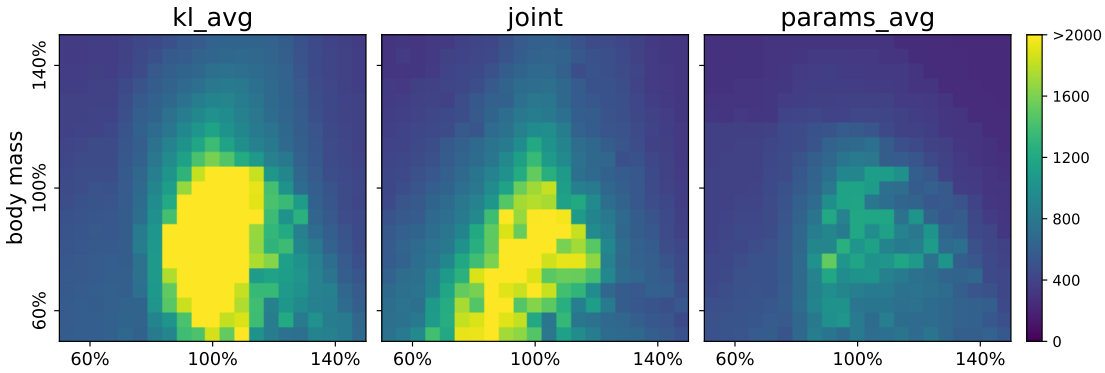


Figure 4.3: Testing performance of learned policies under different wind conditions in *Walker* task. The proposed method *kl_avg* is observed to generalise to a wider range of domain. Four training conditions are (a): $c_a = 0.001$, $\xi \sim \mathcal{U}(-0.5, 0.5)$, (b): $c_a = 0.001$, $\xi \sim \mathcal{U}(-2.5, 2.5)$, (c): $c_a = 1.0$, $\xi \sim \mathcal{U}(-0.5, 0.5)$, (d): $c_a = 1.0$, $\xi \sim \mathcal{U}(-2.5, 2.5)$. Results are averaged over 6 random seeds.



(a) Testing performance under different wind speeds



(b) Testing performance under other environmental changes

Figure 4.4: Testing performance of learned policies in *Hopper* task. Policies are trained with condition $c_a = 0.1$, $\xi \sim \mathcal{U}(-0.5, 0.5)$. Curves in (a) show the degradation of testing returns with varying wind speeds. Heat maps in (b) show testing returns under different pairs of robot masses and friction coefficients. The centre of each heat map corresponds to the training condition.

and friction coefficients, (3) different observation noise scales, and (4) different action noise scales.

Performance under different wind speeds For this analysis, we evaluate the learned policies with wind speeds varying within $[-5, 5]$, to demonstrate their generalisation performance in unseen domains. Figure 4.3 shows the results of comparison in *Walker* task with different training conditions. As we can summarise from the figures, our proposed method achieves a competitive performance in the easiest task setting, but is able to perform more robust when the task becomes harder, i.e., when the training set of domains is more diverse and the greater penalty on large actions is applied. This demonstrates that the proposed method is able to generalise better against extrapolated domains, i.e., domains outside the range of perturbation from training domain set. A similar analysis in *Hopper* task is shown in Figure 4.4(a).

Performance under other environmental changes Next, we evaluate policies with other sources of environmental changes that the agents have not experienced during training. As suggested in [Rajeswaran et al., 2016], we change the body masses of robot and the friction coefficients during testing, which have pronounced impact on performance. Figure 4.5 shows the policies' performance under different dynamic model instances. The centre of each heat map corresponds to the training setting of the dynamics parameters. We can see that both *joint* and *params_avg* tends to suffer more in the grids which are further from the centre, and *kl_avg* is robust to a wide range of changing dynamics. A similar analysis in *Hopper* task is shown in Figure 4.4(b).

Performance under observation and action noise Finally, we follow the evaluation protocol in Chapter 3 and see whether the policies also generalise against observation noise and action noise. Gaussian noise is used to model both observation and action noise. Figure 4.6(a) and 4.6(b) show the degradation of testing performance as increasing the scales of observation and action noise, respectively. We can see that the performances of learned policies with *kl_avg* degrade more slowly compared to two other baselines.

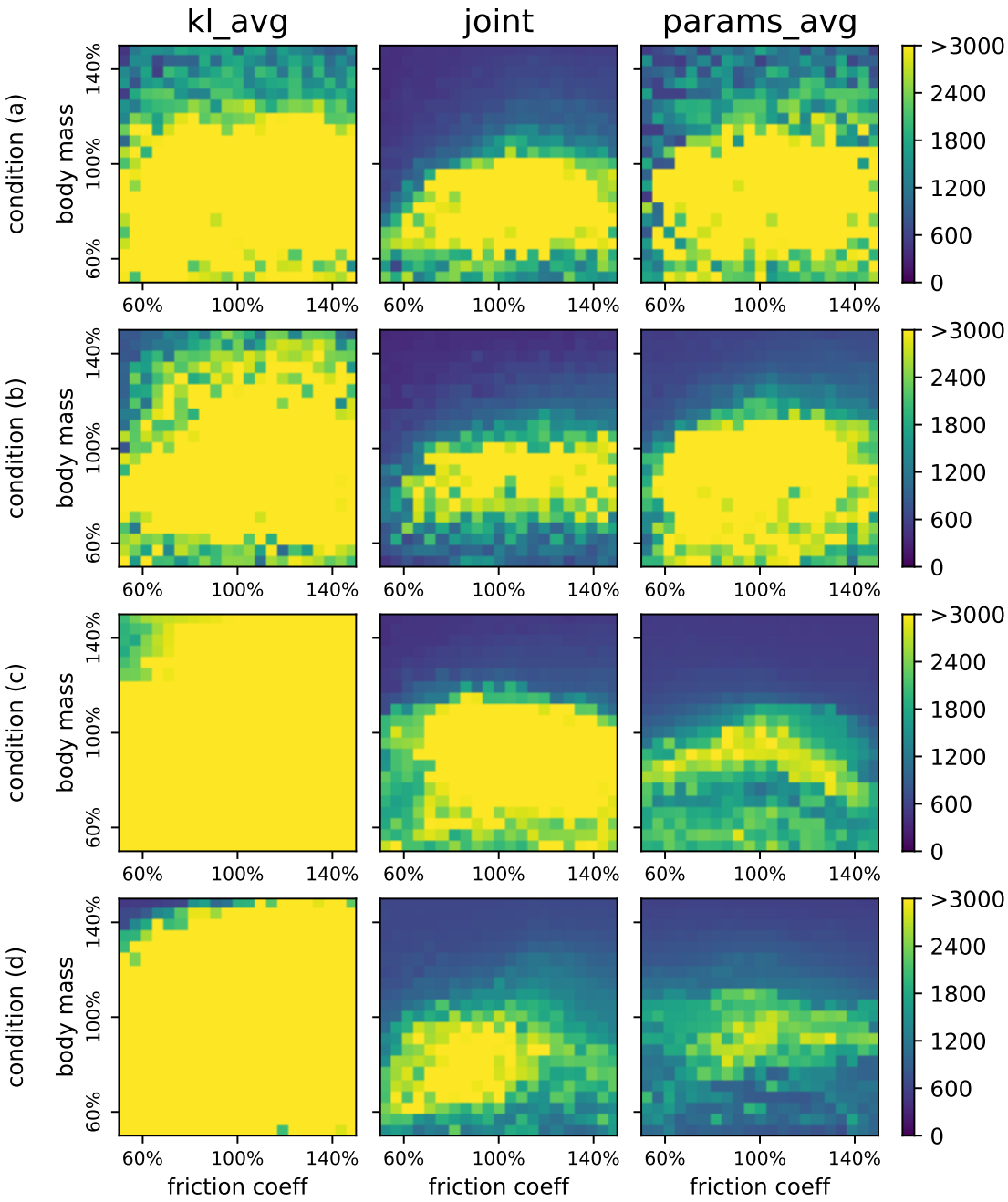
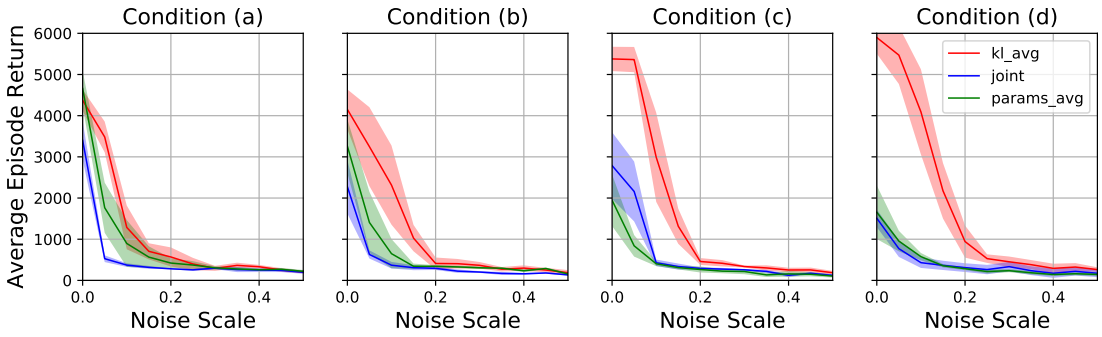
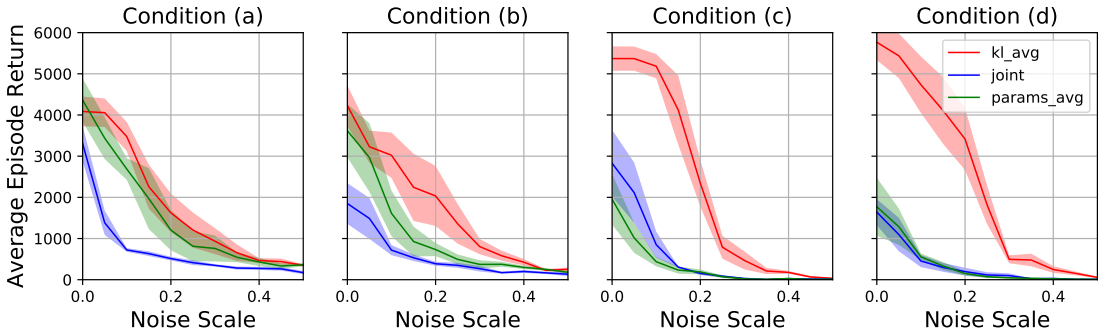


Figure 4.5: Performance of policies under different combinations of body masses and friction coefficients in *Walker* task. The testing return are depicted as heat maps, of which the axes indicate the ratio of testing dynamic parameters to training ones. The centre of each heat map corresponds to the training dynamic parameters and the brightness of each cell indicates the testing return under each dynamic model instance. Four training conditions are (a): $c_a = 0.001, \xi \sim \mathcal{U}(-0.5, 0.5)$, (b): $c_a = 0.001$, $\xi \sim \mathcal{U}(-2.5, 2.5)$, (c): $c_a = 1.0, \xi \sim \mathcal{U}(-0.5, 0.5)$, (d): $c_a = 1.0, \xi \sim \mathcal{U}(-2.5, 2.5)$.



(a) Robustness against observation noise



(b) Robustness against action noise

Figure 4.6: Testing performance of learned policies under different observation and action noise. Noise is modelled as Gaussian noise with varying scales. Four training conditions are (a): $c_a = 0.001$, $\xi \sim \mathcal{U}(-0.5, 0.5)$, (b): $c_a = 0.001$, $\xi \sim \mathcal{U}(-2.5, 2.5)$, (c): $c_a = 1.0$, $\xi \sim \mathcal{U}(-0.5, 0.5)$, (d): $c_a = 1.0$, $\xi \sim \mathcal{U}(-2.5, 2.5)$.

4.3.4 Discussion

A similar work to ours is [Teh et al., 2017], where the idea to regularise policy updates with KL divergence is used to improve multi-task learning in the context of discrete Deep RL. Teh et al. [2017] introduced a *central* policy θ_0, which is used to guide task-specific policy learning via a regulariser $\ell^{\mathrm{KL}} = \mathbb{E}\left[D_{\mathrm{KL}}\left(\pi_{\theta_i}(\cdot|s)||\pi_{\theta_0}(\cdot|s)\right)\right]$. Similar to ours, the agent alternates between (1) a training step, where each individual θ_i is trained to optimise the regularised task-specific objectives, and (2) a distillation step, where the central policy θ_0 is trained to minimise the summation of its KL divergence to each individual θ_i. Differently, we (1) instead of using regulariser, use FIMs and conjugate gradient methods, and (2) focus on the problem setting of cross domain generalisation, where the goal is to learn an individual policy that performs robust against potential domain shifts, therefore, no domain-specific tuning is allowed.

Moreover, as shown in Figure 4.7, our method helps to learn policies with higher

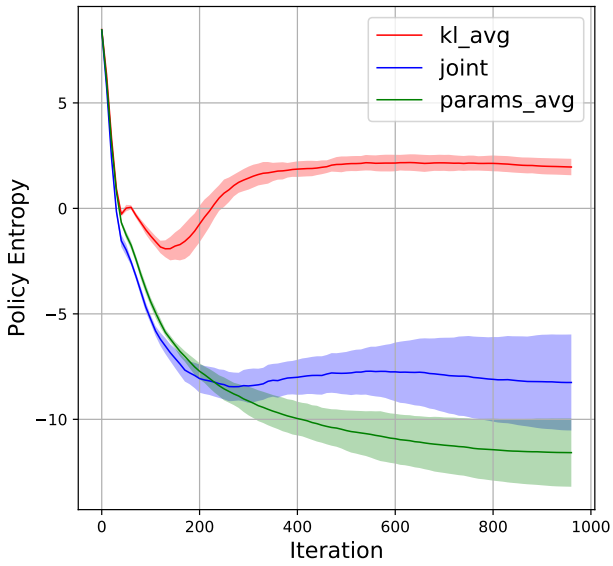


Figure 4.7: Policy entropy changes during learning process. Policies are trained in *Walker* task with $c_a = 0.1$, $\xi \sim \mathcal{U}(-0.5, 0.5)$.

entropy. Entropy regularisation is often used in single task learning, to counter premature convergence to greedy policies, which is particularly severe when learning with policy gradients [Mnih et al., 2016]. With averaging policy updates in the probability space of action distributions, the entropy will decrease only when the policy updates align in source domains. Thus, a soft entropy regularisation is naturally applied.

4.4 Summary

In this chapter, we explored multi-domain learning in dynamic tasks, which enables better generalisation across domains. By averaging the updated policies from domains in the space of action probabilities, our method is able to learn a more generalised policy. The experimental results showed that the learned policy does not only perform more robustly within the source domain distribution, but also generalises better in unforeseen target domains. For future work, we would like to extend this idea of averaging policies in probability distribution space to multi-task learning where part of the policy network parameters are shared among different tasks.

Chapter 5

Knowledge Transfer Across Tasks for Robot Control

In this chapter, we develop a new tensor based transfer learning technique for complex robot control problems. Inspired by lifelong learning [Thrun, 1996b; Ruvolo and Eaton, 2013b], we aim to develop learning agent that is able to extract task-agnostic generalisations knowledge from experience and learn new tasks progressively more easily as a wider and deeper set of prior tasks are mastered. To achieve this, we define a class of neural network controllers loosely inspired by the *Dynamic Movement Primitives* (DMPs) [Schaal et al., 2005] commonly used to solve these dynamic tasks. We first learn a set of multiple source tasks individually from a given family (e.g., throwing objects of various weights to various locations), and the controllers for these correspond to a stack of such neural networks. Then we factorise this set of tasks to obtain transferrable latent skills. By using these latent skills as a basis to construct a policy network, we are able to learn a set of target tasks autonomously, without demonstration. Uniquely our tensor-based transfer framework enables simultaneous discovery and sharing of latent skills across both task categories and actuators [Luck et al., 2014].

Our contributions are two-fold: (i) We introduce a DMP-inspired neural network that can represent a variety of dynamic skills, and show how multiple networks can be factored to obtain transferrable latent skills, (ii) We evaluate this idea with four challenging dynamic tasks, including reaching to a target position, throwing at a target [Deisenroth et al., 2014; Kober et al., 2010], casting at a target [Kober and Peters, 2010], and ball-in-cup [Stulp et al., 2014]. We then show how transferring latent skills can dramatically speed up learning target tasks, ultimately allowing challenging new

skills to be mastered autonomously by policy-search reinforcement learning (RL).

5.1 Background

Learning challenging robot control skills is usually achieved in practice by supervised learning from demonstration. Nevertheless, the vision of learning-based robot control is for robots to learn new skills autonomously. Direct application of RL however requires prohibitive amounts of training, risking physical damage to the robot. This has motivated a fruitful line of research into contextual (parameterised) policies and options where a robot learns to a task under different domains, such as throwing objects of different weights [Stulp et al., 2014]. By contrast, we aim to achieve autonomous learning of a new task through transfer learning.

5.1.1 Dynamic Movement Primitives

Dynamic movement primitives are a representation of motor primitives proposed by [Schaal et al., 2005]. The core idea behind DMPs is to perturb into a simple yet stable linear control system with a nonlinear term to acquire the desired trajectory. To help the system follow arbitrary trajectory and finally stabilise around target state, an additional canonical system is introduced. Consider a discrete movement, the dynamic systems can be expressed as the following set of equations:

$$\ddot{y} = \alpha(\beta(g-y)-\dot{y}) + f(x), \tag{5.1}$$

$$f(x) = \frac{\sum_{i=1}^{N} \theta_i \phi_i}{\sum_{i=1}^{N} \phi_i} x(g-y_0), \tag{5.2}$$

$$\phi_i = \exp\left(-h_i(x-c_i)^2\right), \tag{5.3}$$

$$\dot{x} = -\alpha_x x. \tag{5.4}$$

where y is the system state, y_0 is the initial state and g is the goal state. The first half in Eq. (5.1) is a point attractive system with target state at g. The second half $f(x)$ is a nonlinear function that forces the system to follow an arbitrary trajectories. The nonlinear function is represented as a radial basis function (RBF) network as in Eq. (5.2), with weight parameters denoted as θ and Gaussian kernels Eq. (5.3). To make sure the influence of this function vanish at the end of the system, a canonical dynamical system is introduced, denoted as x. x is initialised as 1 and goes to 0 as time goes infinity. With such dynamics, DMPs converge from its initial state y_0 towards

the target g as time goes on. The general shape of the trajectory is determined by the weight parameters θ and the Gaussian kernals in Eq. (5.3), where the temperature parameter α_x controls the

5.1.2 Transfer and Multi-task Learning

Transfer (TL) and multi-task (MTL) learning aim to improve and accelerate learning by sharing knowledge across different tasks, in a uni-and multi-directional way respectively. These are well studied topics in supervised machine learning. One of the earliest applications to RL showed that for classic pole balance problems, it was faster to learn a policy for a novel system with knowledge transferred from other similar systems [Selfridge et al., 1985]. TL has since been widely applied to accelerate RL [Taylor and Stone, 2009]. MTL has been used to jointly optimise multiple RL tasks, such as stacking various numbers of blocks [Deisenroth et al., 2014]. Recently, multi-task Deep RL has been applied in learning to play multiple video games [Czarnecki et al., 2019; Parisotto et al., 2016; Rusu et al., 2015].

A related work to ours is PG-ELLA [Ammar et al., 2014], which adapts the successful GO-MTL algorithm [Kumar and Daume III, 2012] from supervised learning to the policy gradient based RL setting. Their idea is to apply low-rank matrix factorisation to a stack of linear models and share information through the resulting subspace, and PG-ELLA applies this to control tasks such as pole-balancing. We go beyond this: PG-ELLA/GO-MTL deal with linear control problems only. To solve more complicated control tasks requires dynamic trajectory planning function that is highly non-linear, such as radius basis functions. Secondly, PG-ELLA shares latent knowledge across tasks only. Our tensor-based TL framework represents policy for each actuator as a slice of tensor and therefore, discovers and shares latent skill across both tasks [Ammar et al., 2014] and actuators [Luck et al., 2014] simultaneously.

5.1.3 Lifelong and Curriculum Learning

Lifelong learning takes TL/MTL ideas further with the vision that as more tasks are learned, better task-agnostic abstract knowledge can be extracted. The resulting more humanlike "*learning to learn*" should make each new task progressively easier to master [Thrun, 1996a]. These ideas have also been studied in robot control, for example by treating different environments [Ring, 1998], or robot hardware platforms [Isele et al., 2016] as multiple tasks to be mastered in a lifelong learning manner. The choice of

task sequence is important to the outcome in lifelong learning [Bengio et al., 2009], but despite some work [Ruvolo and Eaton, 2013a], how to predict a good curriculum in advance is still an open question. In this paper we demonstrate that with an intuitive choice of curriculum, our framework can ultimately learn a challenging dynamic control skill autonomously through RL.

5.1.4 Low-Rank Tensors and Factorisation

Low-rank tensor models are often studied for missing data imputation, and compressing/speeding up large neural networks [Lebedev et al., 2015]. A few MTL studies used low-rank tensors for knowledge sharing across tasks with a structured description, rather than a simple index [Wimalawarne et al., 2014]. However, in these studies the underlying per-task representation in each case is still a linear model (single vector) that predicts a single output. We address tensor factorisation based transfer to share knowledge across both tasks and multiple outputs (actuators) with non-linear neural network-based models.

5.2 Methodology

5.2.1 Policy Representation

The challenging dynamic control tasks are commonly solved by Dynamic Movement Primitive (DMP) based approaches [Schaal et al., 2005]. DMPs combines a non-linear open-loop term (which depends on time) with a linear closed-loop term. (which depends on state). Inspired by this idea, we design a policy network $a_{t,i} = \pi(t, \mathbf{x}^o_{t,i})$ which produces the ith actuator's force $a_{t,i}$ given the current time t and observed state $\mathbf{x}^o_{t,i}$. The network has inputs for t and $\mathbf{x}^o_{t,i}$, and includes two learnable layers. The first is a radial-basis function (RBF) layer that functions as a trajectory planner, allowing temporally extended and highly non-linear movements to be generated. The second is a linear fully-connected (FC) layer that encodes learned controller parameters that will ensure the trajectory is followed. The roles of these roughly correspond to the two

components in a DMP. Specifically, for actuator i the policy is parameterised as:

$$\phi_n(t;\mu,\Sigma) = \frac{1}{\sqrt{2\pi}\sigma_n}\exp\left(\frac{(t-\mu_n)^2}{2\sigma_n^2}\right), \quad (5.5)$$

$$x^d_{t,i} = \Theta_i\Phi(t;\mu,\Sigma), \quad (5.6)$$

$$\dot{x}^d_{t,i} = \frac{1}{2\Delta t}(x^d_{t+\Delta t,i} - x^d_{t-\Delta t,i}), \quad (5.7)$$

$$a_{t,i} = \Psi_i(\mathbf{x}^d_{t,i} - \mathbf{x}^o_{t,i}). \quad (5.8)$$

Here the learnable parameters are the matrices Ψ_i that implement the controller, and Θ_i containing the weights of basis functions describing the non-linear trajectory. Total number of basis functions used is denoted as N. Figure 5.1 illustrates the network for the ith actuator. The dash lines indicate that desired velocity is not generated from the feed forward network directly, but via the Euler method as in Eq. (5.7). In contrast to the conventional DMP-based pipeline, our approach jointly learns the inter-related problems of trajectory planning and PD controller.

The network can be trained in a supervised way via learning from demonstration, or by RL via policy-search. For RL, we use Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) [Kern et al., 2004], a direct-policy search method for optimisation of non-linear non-convex functions in continuous space. CMA-ES has connections to policy-gradient and has been shown to outperform other RL algorithms for our type of policy [Stulp and Sigaud, 2013]. For an actuator i, both layers' parameters can be summarised as a vector $\mathbf{w}_i$ containing both vectorised RBF weights $\Theta_i \in \mathbb{R}^N$ and controller parameters $\Psi_i \in \mathbb{R}^{\dim(\mathbf{x})}$. Thus, for a given task, the parameters are represented as a matrix $\mathbf{W} \in \mathbb{R}^{D\times A}$ where $D = (N + \dim(\mathbf{x}))$ and $A = \dim(\mathbf{a})$, where $\dim(\mathbf{x}) = 2$ in the case of a PD controller.

5.2.2 Low-Rank Tensor Factorisation: Latent Skills

Given the above policy representation, we aim to discover latent skills that can be shared across tasks and also actuators. As summarised above, the policy for each task is represented as a matrix, so multiple tasks stack into a 3-way tensor. We will achieve knowledge sharing through low-rank modelling of this tensor. Unlike for matrices, there are many definitions of low-rank tensor factorisation, and we use one of the most general, Tucker decomposition [Tucker, 1966]. Tucker decomposition factors an N-way tensor into a lower-rank N-way core tensor and N matrices along each mode. A

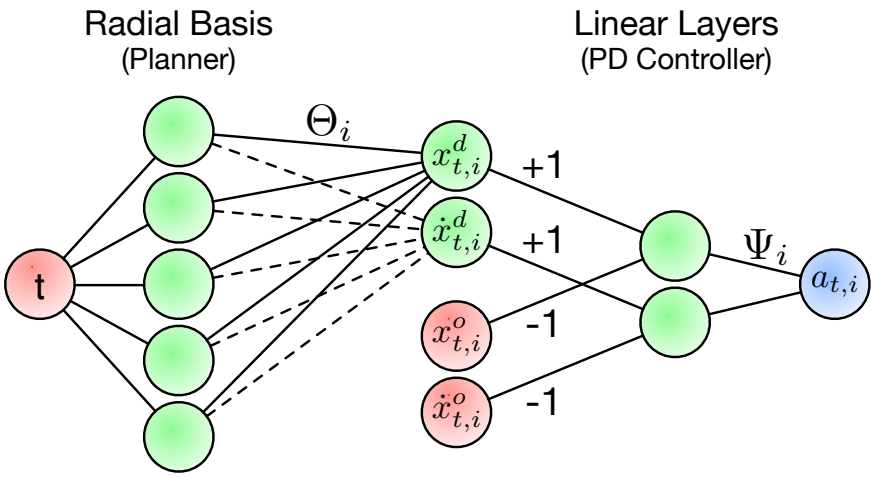


Figure 5.1: Network defining our policy $\mathbf{a}_i = \pi(\mathbf{x}^o_{t,i}, t)$ from time-step t and observed state $\mathbf{x}^o_{t,1}$ to action $a_{t,i}$.

3-way tensor $\mathcal{X} \in \mathbb{R}^{D_1 \times D_2 \times D_3}$ is assumed to be composed as:

$$\mathcal{X}_{m,n,p} = \sum_{k_m=1}^{K_1} \sum_{k_n=1}^{K_2} \sum_{k_p=1}^{K_3} \mathcal{G}_{k_m,k_n,k_p} \mathbf{U}1_{k_m,m} \mathbf{U}2_{k_n,n} \mathbf{U}3_{k_p,p}, \tag{5.9}$$

or in a matrix form:

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3, \tag{5.10}$$

where $\mathcal{G} \in \mathbb{R}^{K_1 \times K_2 \times K_3}$ is the lower-rank core tensor, $\mathbf{U}_n \in \mathbb{R}^{K_n \times D_n}$ are the factor matrices and can be regarded as the principal components in each mode, $\times_i$ is a tensor operator that multiplys a N-way tensor with a matrix along the i-th dimension, and $K_i \leq D_i$. This can be efficiently solved as a higher-order singular value decomposition (HOSVD) problem [Lathauwer et al., 2000] and obtaining $\mathbf{U}_n$ as the $\mathbf{U}$ matrix from SVD of mode-n flattening of $\mathcal{X}$, after which the core tensor is obtained by

$$\mathcal{G} = \mathcal{X} \times_1 \mathbf{U}_1^T \times_2 \mathbf{U}_2^T \times_3 \mathbf{U}_3^T. \tag{5.11}$$

A detailed explanation is described in Algorithm 4.

5.2.3 Multi-Task and Transfer Learning Strategy

Given a set of tasks/skills to learn, we aim to use the above tensor factorisation strategy to extract task-agnostic information to share between them (multi-task learning), and to transfer to benefit the learning of new tasks (transfer learning). A schematic overview of the procedure is given in Figure 5.2.

Algorithm 4 Tucker decomposition of a 3-way tensor with HOSVD

INPUT: A 3-way tensor $\mathcal{X} \in \mathbb{R}^{D_1,D_2,D_3}$, core tensor dimensions K_1, K_2, K_3

for $i = 1,2,3$ **do**

Flatten tensor $\mathcal{X}$ along axis i as $\mathbf{X}_{(i)}$

$\mathbf{U}_i \leftarrow K_i$ leading left singular vectors of $\mathbf{X}_{(i)}$

end for

$\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{U}_1^T \times_2 \mathbf{U}_2^T \times_3 \mathbf{U}_3^T$

RETURN: $\mathcal{G}, \mathbf{U}_{1,2,3}$

Formalisation We assume that we have P source tasks where $\mathbf{W}_p$ represents the policy parameter matrix for the p-th task's network (as described in Sec. 5.2.1). The policy parameters for all source tasks can be stacked as slices of a 3-way tensor $\mathcal{W}$ of size $D \times A \times P$. The latent task assumption is to factorise the weight tensor as per Eq. (5.10). In this case $\mathcal{G}$ is the core tensor of size $K_1 \times K_2 \times K_3$ that contains knowledge abstracting both skills and actuators. $\mathbf{U}_2 \in \mathbb{R}^{K_2 \times A}$ contains actuator specific knowledge, and $\mathbf{U}_3 \in \mathbb{R}^{K_3 \times P}$ encodes task specific knowledge. Based on this decomposition, we consider both multi-task and transfer learning.

Multi-task Learning To jointly learn several tasks, we exploit 'constructive' multi-task learning [Yang and Hospedales, 2015, 2017a]. That is, the parameters we actually train with CMA-ES are the factors $\mathcal{G}$ and $\mathbf{U}_{1,\ldots,3}$, which are then multiplied out (Eq. (5.10)) to obtain the tensor parameterising the policies for all tasks (neural network RBF and FC parameters) in order to perform a rollout of a given task. For regularisation we place a L_1-norm on $\mathbf{U}_3$ to encourage sparsity in combining latent tasks and L_2-norm on the others to prevent overfitting and stabilise learning. Simultaneously training the parameters in this way shares knowledge across actuators and across tasks.

Learning a Target Task We next consider transferring knowledge to a target task, given a set of source tasks (modelled by $\mathcal{G}$ and $\mathbf{U}_{1,\ldots,3}$ above). To achieve this, we gather the task-agnostic knowledge (latent skills) by contracting the task independent factors into a tensor $\mathcal{L} \in \mathbb{R}^{D \times A \times K}$ as $\mathcal{L} = \mathcal{G} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2$. The tensor $\mathcal{L}$ is then transferred to the target task. This provides a good initial subspace, so that a challenging target task in a different category can in practice now be learned autonomously with RL. The policy parameters for each target task are initialised as a random point in the transferred subspace $\mathbf{W}_{target} = \mathcal{L} \times_3 \mathbf{s}_{target}$ by initialising the weight vector $\mathbf{s}_{target}$ randomly. The learner then searches for the policy $\mathbf{W}_{target}$ by direct policy search in the space of $\mathbf{s}_{target}$ and $\mathcal{L}$ with a L_2-norm regulariser on output weights. Algorithm 5

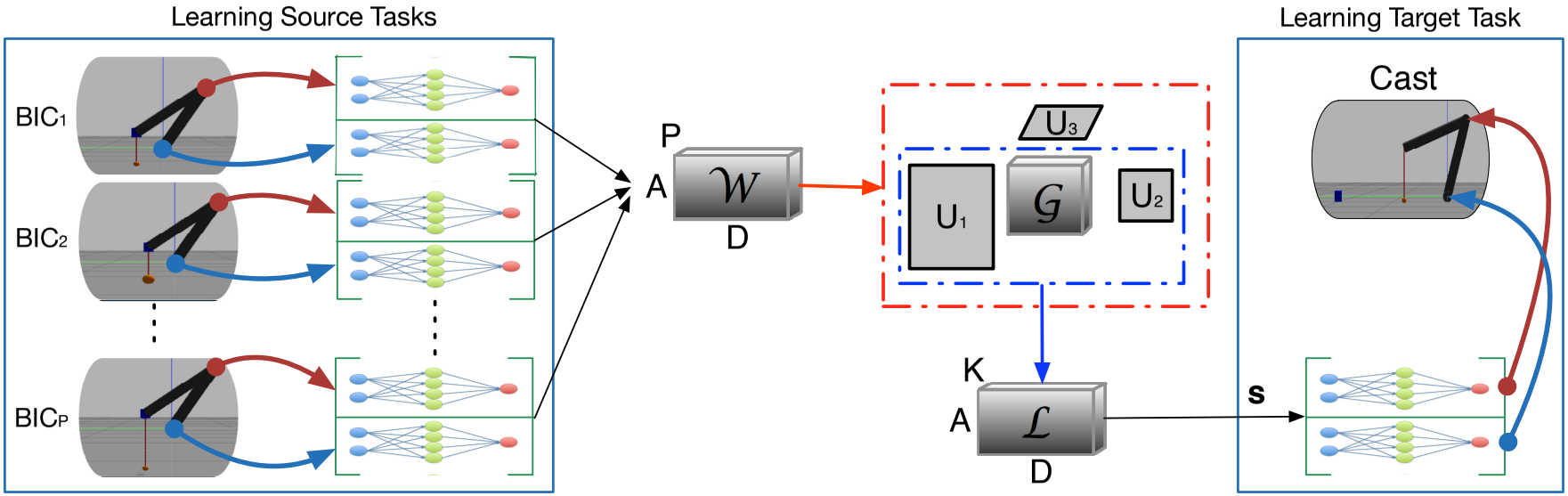


Figure 5.2: Transfer Learning Schematic. A set of P source tasks are learned and the matrices representing each source policy are stacked in to a tensor $\mathcal{W}$. Factorising $\mathcal{W}$ separates task-specific, actuator-specific and shared knowledge. Task-independent latent skills are gathered in tensor $\mathcal{L}$. For the target task, the agent learns the weights $\mathbf{s}$ that reconstructs $\mathcal{L}$ into policy parameters.

Algorithm 5 Tensor-based Transfer Learning

{Source Task Learning}

for $p = 1$ **to** P source tasks **do**

Learn source task policy Θ_p and Ψ_p.

end for

Initialise tensor $\mathcal{W}$ containing weights as slices.

MTL initial condition: $\mathcal{W} \rightarrow \mathcal{G} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3$.

Multi-task learn source tasks.

{Target Task Policy Search}

Task-agnostic knowledge tensor: $\mathcal{W}' = \mathcal{G} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2$.

Initialise $\mathcal{L}$ with slices of tensor $\mathcal{W}'$.

Initialise $\mathbf{s}$ as one random source task.

while not converged **do**

Fix $\mathcal{L}$ and update $\mathbf{s}$ with CMA-ES

Fix $\mathbf{s}$ and update $\mathcal{L}$ with CMA-ES

end while

summarises the overall procedure.

Discussion In contrast to some other one-to-one transfer learning approaches [Taylor and Stone, 2009], an important difference is that our algorithm exploits transfer from *multiple* source tasks in order to extract task-agnostic information that is likely to be transferable. The subspace/latent task set $\mathcal{L}$ is updated when learning the target, so we can see that transfer learning here is about providing a good initial condition for the subspace before performing RL.

5.3 Experiments

5.3.1 Environments and Tasks

We exploit a simulated robot arm whose end-effector operates in a 2-dimensional space. In the dynamical simulator, we assume an ideal inverse kinematic system which is able to convert the movement of end-effector into angular movement of each joint. Our policy network outputs the acceleration of end-effector in both horizontal and vertical directions at each time step t. In detail, the agent observes the positions and velocities, and outputs the accelerations for two actuators ($A = 2$). We consider four

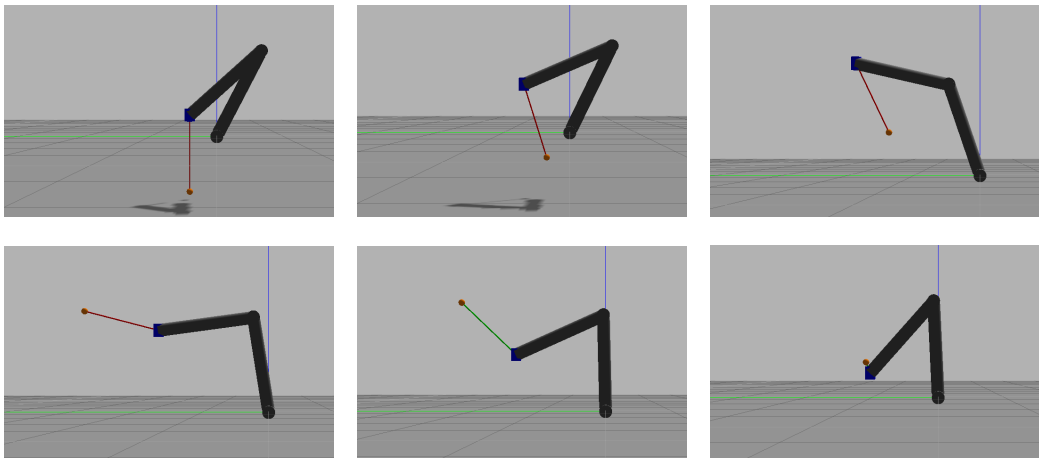


Figure 5.3: Successful movement trajectory of Ball In Cup. The arm starts from static state and moves rightward firstly, generating the ball a momentum toward right, then moves leftward fast, generating the ball a large momentum toward upper left. After the ball is pulled back through the string, the arm moves the cup towards the ball's dropping point and catches the ball.

different dynamic tasks as follows:

Target Reaching (Reach) The agent learns to move the end-effector, reaching specific target position with time discounted reward for each time step and penalties on large velocities and accelerations. Tasks within the category are to reach different positions, so task parameters are the 2-dimensional positions of goal points.

Object Throwing (Throw) The learner aims to throw a ball into a basket by holding it with the end-effector and then releasing it during dynamic arm motion [Kober et al., 2010]. We fix the ball release time. The task parameters only includes the position of target basket in 2-dimensional space.

Ball-In-Cup (BIC) In this task [Stulp et al., 2014; Kober and Peters, 2010], the robot holds a cup in its end-effector and the cup has a string attached, from which a ball hangs. Initially, the ball is hanging at rest vertically below the cup. The task is to induce motion in the ball through the string: swinging the ball up and catching the ball with the cup. This is illustrated in Figure 5.3, the red line shows the position of the ball when the string is taunt and the green lines show the ball position when the string is loose. The task parameters include the length of the string and the mass of the ball.

Casting (Cast) In casting [Kober and Peters, 2010], a ball is attached to the end-effector by a string. The task is to get the ball into a small cup, placed in front of the robot. Because the end-effector controls the ball only indirectly via the string (which

may change between loose and taunt depending on the dynamics), the dynamics are very different to throwing. We fixed the position of target cup and the task parameters include the length of the string and the mass of the ball only.

Cost Functions The cost for reaching tasks is defined with action penalties and a time discounted linear summation of quadratic functions with respective to the distance to target state $J = \sum_{t=0}^{T}[\gamma^t((\mathbf{x}-\mathbf{x}_{target})^2 - b) + \alpha\|\mathbf{a}_t\|]$, where constant b is a pre-defined baseline. The other three tasks use a cost function of the same parametric form. The cost (cf. reward) J of each episode is based on the difference between horizontal position of the ball x_b and the cup x_c at t_m when they are at the same vertical height. The cost is assigned as 0 if the ball fails to reach the same height as the cup. L_2-norms on actions are added to avoid extreme accelerations. For our experiments, we choose $\gamma = 0.99, \alpha = 0.001$.

$$J = \begin{cases} \sum_{t=0}^{T} \alpha\|\mathbf{a}_t\| & \text{if vertical heights never match} \\ \gamma^{t_m}\min(b_1(x_b - x_c)^2 - b, 0) + \sum_{t=0}^{T} \alpha\|\mathbf{a}_t\| & \text{otherwise} \end{cases} \tag{5.12}$$

5.3.2 Transfer Learning

In the first experiment we investigate autonomously learning a target task with RL, given a set of known different source tasks. We use CMA-ES as base learner with the initial parameter $\sigma = 0.01$. Our focus is on comparing the impact of different types of knowledge transfer, assuming the source tasks are well learned. For simplicity, we therefore learn the source with supervised demonstration followed by MTL RL refinement. There are multiple ways to evaluate transfer learning performance [Taylor and Stone, 2009]. We report (i) the total reward/cost during learning and (ii) target task success rate – the percentage of experiments that the robot successfully completes (e.g., gets the ball in the cup) when learning terminates. For parsimony, we adopt an experimental design where each task category is considered in turn as both a source and a target. Therefore 4 task categories entail 16 transfer experiments. Each experiment considers families of 16 source and 50 target tasks.

We compare our tensor-based transfer approach with three baselines. *Scratch*: learning the target from scratch. *Direct:* A simple direct transfer learning baseline of initialising the target task to that of a (randomly chosen) learned source task. This is a common strategy of transfer by 'warm start' followed by fine-tuning [Taylor and Stone, 2009]. *Matrix:* Matrix-based transfer approach, where the policy parameters for all tasks are structured as a matrix of size $DA \times P$, knowledge sharing is achieved

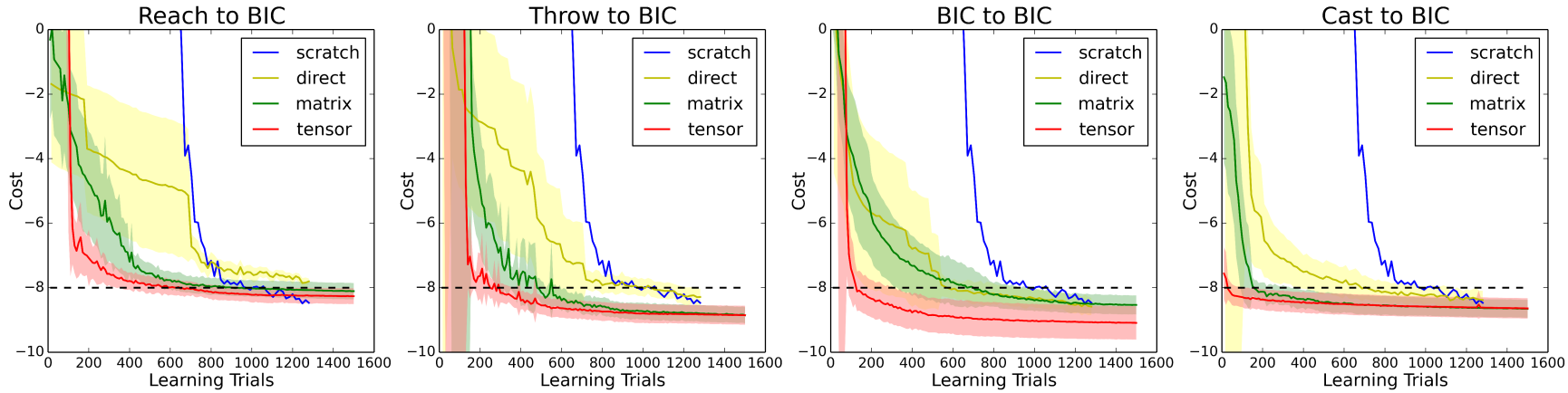
by SVD – thus only transferring knowledge across tasks and not also across actuators. These transfer approaches are thus about providing a good initial condition for the policy-search based RL of the target task, but ours transfers the latent tasks in a tensor structure.

Figure 5.4(a) shows illustrative learning curves of 4 of the 16 experiments. Note that only successful learned experiments are counted for the learning curves. Figure 5.4(b) summarises learning success rate and total cost for all 16 experiments. We observe that: (i) Learning from scratch is feasible (it achieves success) although slow/costly for Reaching and Throwing. However it mostly fails to succeed at the harder BIC and Casting tasks. (ii) Among the transfer learning approaches, we see that our tensor-based approach is best in terms of total cost/learning speed, followed by matrix-based transfer, and direct's warm-start approach. (iii) Our tensor-based transfer is the only one to solve (high success rate) most tasks given most sources. Overall the results show that with a suitably designed policy and TL strategy, it is possible to learn challenging dynamic control tasks autonomously, even when transferring from very different and much easier source tasks (e.g., Reach→BIC). Previous solutions to BIC have required demonstration [Stulp et al., 2014].

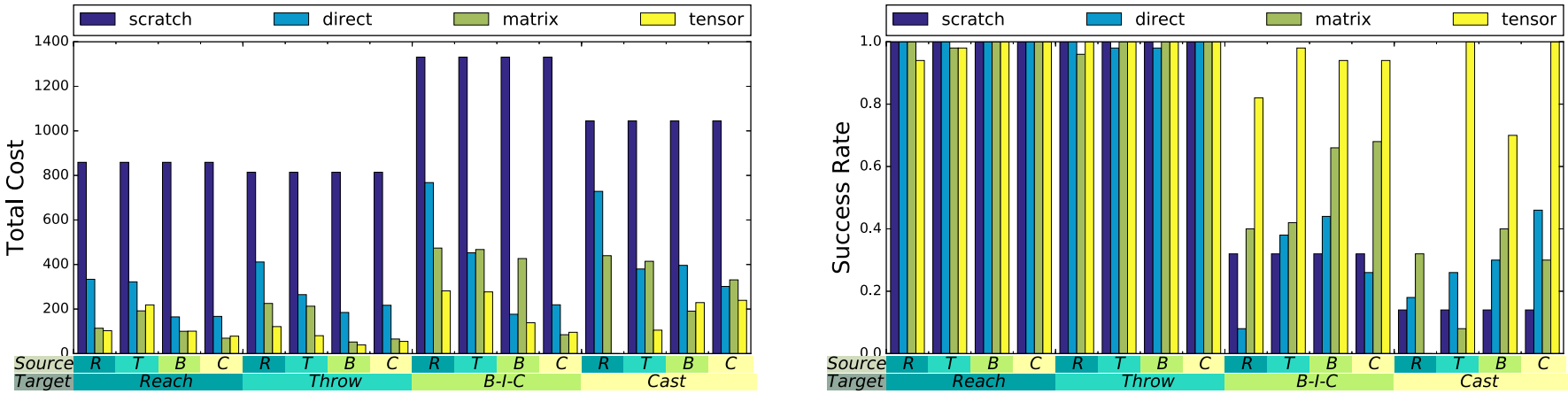
5.3.3 Comparative Analysis

We next compare our framework against three competitors with alternative policy representations, and transfer strategies. **ELLA:** Our implementation of [Ruvolo and Eaton, 2013b], a state of the art framework for matrix-based transfer of linear models. We adapt to our purpose by providing it the inputs $[\mathbf{x}_t, t]$ our model uses. **RBF-ELLA:** As ELLA is designed for linear models, we generalise it for non-linear tasks through RBF-tiling the input space. **FCNN:** A fully connected multi-layer NN policy mapping $[\mathbf{x}_t, t]$ to actions via one RELU hidden layer of 8 units is a conventional alternative to our DMP-inspired policy.

We perform the same experiment as the previous section, considering 16 task pairs for transfer. Figure 5.5 shows an example of the learning curve of each method along with summary statistics (i) Average learning cost, and (ii) % of total cost 'wins' (which method had the smallest cost in a given experiment). Figure 5.5(a) illustrates our tensor-transfer approach starting off better and converging quicker and to better asymptote than alternatives. Figure 5.5(b) shows that we achieve much better total cost on average, and considering each experiment achieve the best total cost in most cases.

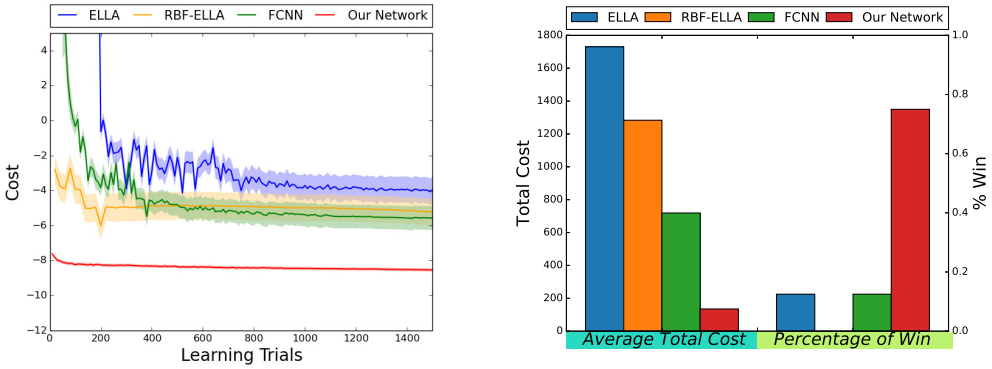


(a) Learning curves of transfer learning from 4 source tasks to BIC tasks



(b) Total Cost of Learning and Success Rate at Convergence

Figure 5.4: Learning curves and summary statistics of different transfer strategies. Figure (a) compares the learning curves of different transfer strategies given different transfer task pairs. Figure (b) shows the cost and the percentages of trials that the robot successfully catches the ball using different learned policies. Our method is shown to both converge faster and achieve a better final performance compared to baselines.



(a) Casting to BIC transfer

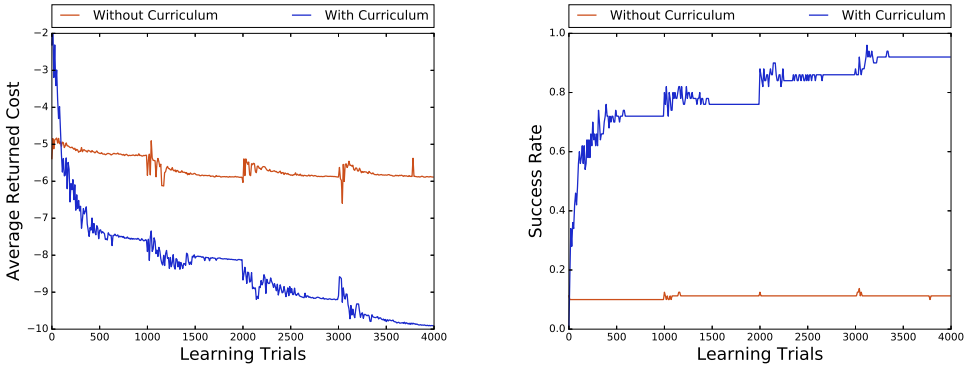
(b) Summary of all transfer pairs

Figure 5.5: Comparison of our method vs. three baselines. (a) illustratively shows our method outperforms others in Casting to BIC transfer. (b) shows an overall comparison in terms of the total cost and percentage of transfer pairs that each algorithm wins in over all pairs.

Discussion Why does our method perform so much better than ELLA which has previously shown convincing results on benchmarks like cart-pole, quadrotor, etc? The linear policy of ELLA can solve these classic benchmarks as they have a specific target state (e.g., balance/hover position) and are linearly controllable closed loop systems [Mokhtari and Benallegue, 2004] that do not require extended dynamic trajectory planning. However in our dynamic manipulation tasks, the objective is not a simple target state: it is an extended movement trajectory. Our policy class including a non-linear open loop layer as trajectory planner, and closed-loop controller can address this. ELLA can be seen as implementing only the linear feedback controller in the final layer of our network. RBF-ELLA is much more flexible and could potentially solve our tasks. But it suffers from the needing to tile RBFs in 3/5D (P/PD control), which means either an under-fitting policy, or very many parameters to train (we used $5 \times 5 \times 5$ tiling to give it a similar number of parameters to ours).

5.3.4 Curriculum Learning

The previous experiments showed that with pre-learned source tasks, we can autonomously learn a new task category with transfer and RL. In this section we explore whether it is possible to learn dynamic manipulation tasks autonomously with no supervision anywhere in the pipeline, by constructing an appropriate training curriculum. The procedure starts with learning the easiest task-category from scratch with multi-task RL,



(a) Learning Curve of BIC by CL (b) Success Rate of BIC by CL

Figure 5.6: Comparison of learning curves of autonomously learning ball-in-cup from scratch via curriculum and without curriculum. Figure (a) shows the average returned cost and (b) shows the success rate.

and the task-agnostic tensor is extracted and transferred to bootstrap learning the next task-category with transfer multi-task RL. We consider the task category curriculum Reach-Throw-BIC. We compare following this curriculum (CL) using tensor transfer against no curriculum (NC). We control for the total cost in rollouts giving both conditions 6000 trials of pre-training. In NC these iterations are used on MTL learning of BIC family tasks. In CL, they are spent on MTL learning of reaching then throwing task categories.

Based on these initial conditions, Figure 5.6(a), 5.6(b) shows the average cost and success rate of the following 4000 trials of MTL training on BIC for both approaches. Note that at learning trial 0, the policies are initialised as the pre-trained policies learned in previous tasks of the curriculum. And the periodic bumps are caused by the alternating optimisation of $\mathcal{L}$ and $\mathbf{s}$. The results show that the challenging BIC task can be autonomously solved by knowledge transfer from a curriculum of autonomously learned source tasks, thus achieving BIC without any demonstrating supervision.

5.4 Summary

In this chapter, we explored transfer learning to enable autonomous RL of non-linear dynamic control tasks. Through our effective policy network representation and tensor based transfer of the latent task subspace, the speed and asymptotic success rate of autonomous RL of target tasks is significantly improved. With curriculum transfer

learning, we were ultimately able to learn hard tasks such as BIC from scratch autonomously without any demonstration. In future work we will explore lifelong MTL within and across tasks, and extend our framework to model similar tasks, such as throwing to multiple targets, as contextual policies rather than discrete tasks.

Chapter 6

Conclusion and Future Work

In this thesis, we have presented a line of research on generalisation in reinforcement learning (RL) focusing on two aspects: cross-domain generalisation and cross-task generalisation. The problem of learning continuous control tasks from a tabular rasa has become one of the most intensively studied topics in the community of RL. Although the breakthroughs in Deep RL show great success in solving simulated complex dynamic tasks, such as *Walker*, *Hopper*, or even *Humanoids* [Schulman et al., 2015a, 2017; Lillicrap et al., 2015; Haarnoja et al., 2017b], deploying the learned policies to control robots end-to-end in a real life scenario remains to be challenging [Levine et al., 2016; Irpan, 2018]. Two of the major concerns are (1) learned policy cannot generalise against the reality gap, i.e., domain shifts between training and testing scenarios; and (2) the amount of data required for learning an individual task is not scalable on real robots.

The first concern leads us to revise the current common practice of evaluating Deep RL algorithms. With the emergence of vast Deep RL algorithms, a standardised task set and systematic evaluation and comparison are importance for us to understanding the strengths and limitations of algorithms, and therefore suggest directions for future research. Existing evaluation work includes Duan et al. [2016a] evaluating the policies with the set of domains same as training, and Henderson et al. [2018] addressing the reproducibility problems in Deep RL. Yet, there is no thorough evaluation on robustness against domain shifts, even though the existence of reality gap between training and testing environment is inevitable. Targeting dynamic control problems, in Chapter 3 [Zhao et al., 2019], we provide a thorough characterisation of potential generalisation challenges and a contribute the Deep RL community with a comprehensive benchmark for Deep RL generalisation performances. We expand the environment in RL with a

sensory module, an actuation module and a dynamic module and characterise different domain shift problems based on the sources of uncertainties. Based on this characterisation, we conduct a comprehensive evaluation of existing common RL algorithms in terms of their ability to generalise against potential domain shifts. Our analysis shows that the common practice of picking algorithms and architectures based on training performance leads to the wrong choice in terms of generalisation performance. We show that many popular RL methods are vulnerable to overfitting, showing poor generalisation to testing. This is particularly so in cases of domain shift, for example transferring from a deterministic to stochastic simulation; or where system parameters such as robot mass vary between training and deployment. Therefore, we recommend that future methods are evaluated using benchmarks such as ours that test generalisation, in preference to training performance. Based on our benchmark, we thoroughly evaluate a suite of existing and new techniques that potentially improve generalisation and define the best practice in terms of algorithms and training strategies. Our benchmark and identified best practice provide a good starting point for future research and practical applications.

As our evaluation results suggest that learning with multiple domains does improve policies' robustness against domain shifts, in Chapter 4, we take one step further to investigate how to improve cross-domain generalisation with multi-domain learning. Existing techniques usually require intensive tuning on source domain distributions so as to balance between generalisation and learning performance [Vuong et al., 2019]. The learning performance may degrade when source distribution becomes wider, because averaging the gradients from different domains may be detrimental to the learning process. To tackle this problem, instead of euclidean distance in the space of policy parameters, we propose to update the policy to minimise its Kullback-Leibler (KL) divergence to sub-policies updated in each source domain. We evaluate the method on *Walker* and *Hopper* tasks and show that our method improve training performance as well as cross-domain generalisation performance in both tasks. This provides an alternate perspective of aggregating update information for multi-domain learning, and potentially for multi-task learning: aggregating towards minimal distances in statistical space of predicted distributions, instead of Euclidean space of network parameters.

Finally in Chapter 5, we move on to generalisation across tasks. As RL algorithms usually require large amount of data to get the agents to converge, it will be helpful to bootstrap the learning process with, for example, general skills acquired from previous tasks. In Chapter 5 [Zhao et al., 2017], we take the idea of tensor based transfer on

the latent task subspace [Yang and Hospedales, 2017b] and extract generalised latent skills across tasks with our effective policy network representation. The results show that given experience on source tasks, the learning speed and asymptotic success rate of autonomous RL of target tasks are significantly improved with transfer across tasks. Moreover, with a designed transfer curriculum, we are ultimately able to learn harder tasks such as Ball-In-Cup from scratch autonomously without any demonstration. This work explores the use of tensor and tensor decomposition to represent policies of multiple tasks, abstract common knowledge among tasks, and transfer the knowledge to bootstrap learning.

6.1 Summary of Contributions

The main contributions of this thesis is summarised as follow:

1. We present a thorough characterisation of the generalisation challenges in overfitting and domain shift in continuous control problems (Chapter 3).

2. To quantify these generalisation challenges empirically, we contribute a comprehensive benchmark for measuring RL generalisation performance, both within- and across-domain scenarios, and evaluate several current algorithms and modifications (Chapter 3).

3. We introduce a practical method to learn with multiple domains: alternately learning sub-policies in each domain individually and updating the generalised policy to minimise overall distance to sub-policies in the space of probability distributions (Chapter 4).

4. We evaluate the idea in two dynamic tasks and show how the learned policies can achieve both better training performance in source domain distribution and better generalisation performance in unforeseen target domains (Chapter 4).

5. We show how multiple policies can be factorised to obtain generalised and transferrable latent skills across tasks with a DMP-inspired network architecture that can represent a variety of dynamic skills (Chapter 5).

6. We evaluate the multi-task/transfer learning idea with four challenging dynamic tasks, and show how transferring latent skills can dramatically speed up learn-

ing target tasks, ultimately allow challenging new skills to be mastered autonomously by policy-search RL (Chapter 5).

The work in this thesis has led to the following publications:

- Investigating Generalisation in Continuous Deep Reinforcement Learning
  Chenyang Zhao, Olivier Sigaud, Freek Stulp and Timothy M. Hospedales
  Preprint, *Journal of Machine Learning Research*, submitted.
  Bibliographic Reference: [Zhao et al., 2019]

- Tensor Based Knowledge Transfer Across Skill Categories for Robot Control
  Chenyang Zhao, Timothy M. Hospedales, Freek Stulp and Olivier Sigaud
  Proc. International Joint Conference on Artificial Intelligence (IJCAI '17)
  Bibliographic Reference: [Zhao et al., 2017]

6.2 Future Work

6.2.1 Towards Understanding RL Generalisation

Chapter 3 presents an empirical evaluation for better understanding the cross-domain generalisation properties of different RL algorithms. For future work, there are two directions of future work that stand out: one is towards a broader coverage of different RL algorithms in evaluation, and the other is towards the understanding of cross-task generalisation.

The evaluation in Chapter 3 covers various model-free RL algorithms and policies represented by feedforward neural networks. However, some important categories are missing from the current evaluation, including model-based RL and recurrent neural networks (RNNs). Model-based RL is a family of RL methods where the agent has access to not only the trajectory information, but also the model (or an estimation of the model) of the environment. Model-based RL generally achieves a better sample efficiency compared with model-free RL [François-Lavet et al., 2018]. How about comparing their generalisation performance? Despite some claims that model-based RL improves robustness of results, Zhang et al. [2018a] suggested the opposite that learning with dynamic models leads to larger generalisation gaps (i.e., more overfitted to training seeds). On the other hand, different from the feedforward neural networks, RNNs use an internal "memory" state which captures information about the past experience. RNNs are shown able to identify different objectives given dense reward sig-

nals, for example, walking in different speeds [Wang et al., 2016; Duan et al., 2016b]. Intuitively, RNNs with memory could potentially learn to identify and adapt to a target domain, without any explicit reward signal, nor changing their policy parameters. Moreover, various components in RL methods may have impact on the outcome of generalisation performance, such as the choice of optimiser, the discount constant, etc. We think it is important to evaluate these aspects in order to build a more thorough benchmark, especially since no constant winner to all facet of generalisation has been found so far in Chapter 3.

Another important aspect of generalisation is cross-task transfer. In the context of computer vision, it is a common procedure to transfer pre-trained feature extractor and learn the decision making layer (e.g. a linear classifier) in target task. [Yosinski et al., 2014] carried out an experimental analysis on what to transfer between tasks with ImageNet data [Deng et al., 2009]. Such evaluation is still missing in RL. In continuous RL domain, an agent usually consists of a policy network and a value network (and a model in the case of model-based RL). An interesting question is what component should be transferred and which algorithm could benefit the most from such transfer to a novel target task. This could lead to a better understanding of transfer learning in RL and would be a starting point for future work in transfer learning and multi-task learning.

6.2.2 Towards Improving RL Generalisation

Cross-Domain Generalisation Chapter 4 shows that, when learning with multiple domains, it is better to find the centre of all sub-policies in the probability space of action distributions, instead of the euclidean space of policy parameters. The distance is measured by KL divergence, because an analytical estimation of it is available through Fisher Information Matrix. In a future experiment, comparisons should be made between different distance metrics (e.g., Wasserstein distance), policy representation (e.g., deterministic policy), and to add more complex dynamic tasks, such as *Humanoid* [Brockman et al., 2016].

Cross-Task Generalisation Finally, another future direction is to apply this idea to a cross-task generalisation setting. Chapter 5 has presented a framework of tensor based multi-task learning and transfer learning with a multiplicative sharing model. The experimental results show that learning novel tasks can be bootstrapped by transferring shared knowledge base. However, the specific network design does not scale well to

learn more complex tasks, such as the *Humanoid* task which has a 376-dimensional observation space and a 17-dimensional action space. Deep neural networks are generally used to represent policies for such tasks. In supervised learning problems, [Yang and Hospedales, 2017a] extended the idea of learning a multiplicative sharing model to deep neural networks. However, it's not trivial to extend the similar idea to RL problems, because of the interference among gradient information from different tasks. As demonstrated in Chapter 4, as well as in [Teh et al., 2017], KL regularisation could help to stabilise learning. An interesting aspect of future work is to apply KL regularisation to learn a deep multiplicative sharing model for multiple tasks.

Bibliography

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*.

Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. (2014). Online multi-task learning for policy gradient methods. In *International Conference on Machine Learning*, pages 1206–1214.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20.

Argyriou, A., Evgeniou, T., and Pontil, M. (2008). Convex multi-task feature learning. *Machine Learning*, 73(3):243–272.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *ICML*.

Berseth, G., Xie, C., Cernek, P., and Van de Panne, M. (2018). Progressive reinforcement learning with distillation for multi-skilled motion control. *arXiv preprint arXiv:1802.04765*.

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

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

Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., and Zaremba, W. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2018). Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*.

Csurka, G. (2017a). Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*.

Csurka, G. (2017b). *Domain Adaptation in Computer Vision Applications*. Springer.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*.

Czarnecki, W. M., Pascanu, R., Osindero, S., Jayakumar, S. M., Swirszcz, G., and Jaderberg, M. (2019). Distilling Policy Distillation. *arXiv e-prints*, page arXiv:1902.02186.

Deisenroth, M. P., Englert, P., Peters, J., and Fox, D. (2014). Multi-task policy search for robotics. In *ICRA*, pages 3876–3881.

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. `https://github.com/openai/baselines`.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016a). Benchmarking deep reinforcement learning for continuous control. In *ICML*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016b). Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

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

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., et al. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Gourieroux, C. and Monfort, A. (1995). *Statistics and Econometric Models*.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2016). Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*.

Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. (2017). Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*.

Ha, D. (2017). A visual guide to evolution strategies. *blog.otoro.net*.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017a). Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2017b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. *AAAI*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

Irpan, A. (2018). Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`.

Isele, D., Luna, J. M., Eaton, E., de la Cruz, G. V., Irwin, J., Kallaher, B., and Taylor, M. E. (2016). Lifelong learning for disturbance rejection on mobile robots. In *IROS*.

Isidori, A. (2013). *Nonlinear control systems*. Springer Science & Business Media.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274.

Kakade, S. M. (2002). A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538.

Kern, S., Müller, S. D., Hansen, N., Büche, D., Ocenasek, J., and Koumoutsakos, P. (2004). Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing*, 3(1):77–112.

Kober, J., Oztop, E., and Peters, J. (2010). Reinforcement learning to adjust robot movements to new situations. In *Robotics: Science and Systems (R:SS)*.

Kober, J. and Peters, J. (2010). Policy search for motor primitives in robotics. *Machine Learning*, 84(1):171–203.

Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856.

Kolb, B. and Whishaw, I. Q. (2001). *An introduction to brain and behavior*. Worth Publishers.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Koos, S., Mouret, J.-B., and Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *Trans. Evol. Comp*, 17(1):122–145.

Kumar, A. and Daume III, H. (2012). Learning task grouping and overlap in multi-task learning. In *ICML*.

Kupcsik, A., Deisenroth, M. P., Peters, J., Loh, A. P., Vadakkepat, P., and Neumann, G. (2017). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439.

Kupcsik, A. G., Deisenroth, M. P., Peters, J., and Neumann, G. (2013). Data-efficient generalization of robot skills with contextual policy search. In *AAAI*.

Lathauwer, L. D., Moor, B. D., and Vandewalle, J. (2000). A multilinear singular value decomposition. In *SIAM Journal on Matrix Analysis and Applications*.

Lazaric, A. and Ghavamzadeh, M. (2010). Bayesian multi-task reinforcement learning. In *ICML-27th International Conference on Machine Learning*, pages 599–606. Omnipress.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., and Lempitsky, V. (2015). Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*.

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.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. (2018a). Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. (2018b). Learning to generalize: Meta-learning for domain generalization. *AAAI*.

Li, D., Zhang, J., Yang, Y., Liu, C., Song, Y.-Z., and Hospedales, T. M. (2019). Episodic training for domain generalization. In *ICCV*.

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*.

Luck, K. S., Neumann, G., Berger, E., Peters, J., and Amor, H. B. (2014). Latent space policy search for robotics. In *IROS*.

Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. (2017). Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *IROS*.

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

Mnih, V., Kavukcuoglu, K., Silver, D., 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.

Mokhtari, A. and Benallegue, A. (2004). Dynamic feedback controller of euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle. In *ICRA*.

Mordatch, I., Lowrey, K., and Todorov, E. (2015). Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5307–5314. IEEE.

Muandet, K., Balduzzi, D., and Schölkopf, B. (2013). Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, pages 10–18.

Muratore, F., Treede, F., Gienger, M., and Peters, J. (2018). Domain randomization for simulation-based policy optimization with transferability assessment. In *Conference on Robot Learning*, pages 700–713.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., and Song, D. (2018). Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2016). Actor-mimic: Deep multitask and transfer reinforcement learning. In *ICLR*.

Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Chowdhary, G. (2018). Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*.

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

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

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702*.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. (2016). Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. (2017). Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6550–6561.

Ring, M. B. (1998). Child: A first step towards continual learning. *Machine Learning*, pages 261–292.

Ruder, S. (2017). Transfer learning: Machine learning's next frontier.

Ruder, S., Bingel, J., Augenstein, I., and Søgaard, A. (2017). Learning what to share between loosely related tasks. *arXiv preprint arXiv:1705.08142*.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016a). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2016b). Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*.

Ruvolo, P. and Eaton, E. (2013a). Active task selection for lifelong machine learning. In *AAAI*.

Ruvolo, P. and Eaton, E. (2013b). Ella: An efficient lifelong learning algorithm. In *ICML*.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2005). Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

Selfridge, O. G., Sutton, R. S., and Barto, A. G. (1985). Training and tracking in robotics. In *IJCAI*, pages 670–672.

Sigaud, O. and Stulp, F. (2019). Policy search in continuous action domains: an overview. *Neural Networks*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Srouji, M., Zhang, J., and Salakhutdinov, R. (2018). Structured control nets for deep reinforcement learning. *ICML*.

Stulp, F., Herlant, L., Hoarau, A., and Raiola, G. (2014). Simultaneous on-line discovery and improvement of robotic skill options. In *IROS*.

Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. (2013). Learning compact parameterized skills with a single regression. In *Humanoids*.

Stulp, F. and Sigaud, O. (2013). Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61.

Sugiyama, M. and Storkey, A. J. (2007). Mixture regression for covariate shift. In *Advances in Neural Information Processing Systems*, pages 1337–1344.

Sunderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., and Corke, P. (2018). The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Tamar, A., Glassner, Y., and Mannor, S. (2015). Optimizing the cvar via sampling. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

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*.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506.

Thrun, S. (1996a). Is learning the n-th thing any easier than learning the first? In *NIPS*.

Thrun, S. (1996b). Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646.

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. In *IROS*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *IROS*.

Tsybakov, A. B. (2009). *Introduction to Nonparametric Estimation*.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311.

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. *arXiv preprint arXiv:1511.07111*, 2(3).

Vuong, Q., Vikram, S., Su, H., Gao, S., and Christensen, H. I. (2019). How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies? *arXiv preprint arXiv:1903.11774*.

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):73.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wimalawarne, K., Sugiyama, M., and Tomioka, R. (2014). Multitask learning meets tensor factorization: task imputation via convex optimization. In *NIPS*.

Yang, Y. (2017). *Knowledge sharing: From atomic to parametrised context and shallow to deep models*. PhD thesis, Queen Mary University of London.

Yang, Y. and Hospedales, T. M. (2015). A unified perspective on multi-domain and multi-task learning. In *ICLR*.

Yang, Y. and Hospedales, T. M. (2017a). Deep multi-task representation learning: A tensor factorisation approach. *ICLR*.

Yang, Y. and Hospedales, T. M. (2017b). Unifying multi-domain multitask learning: Tensor and neural network perspectives. In *Domain Adaptation in Computer Vision Applications*, pages 291–309. Springer.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*.

Zhang, A., Ballas, N., and Pineau, J. (2018a). A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. (2018b). A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.

Zhao, C., Hospedales, T. M., Stulp, F., and Sigaud, O. (2017). Tensor based knowledge transfer across skill categories for robot control. In *IJCAI*, pages 3462–3468.

Zhao, C., Siguad, O., Stulp, F., and Hospedales, T. M. (2019). Investigating generalisation in continuous deep reinforcement learning. *arXiv preprint arXiv:1902.07015*.

Ziebart, B. D. (2010). Modeling purposeful adaptive behavior with the principle of maximum causal entropy.

Appendix A

Hyperparameters

We use the implementation of PPO, TRPO and DDPG from OpenAI baselines codebase [Dhariwal et al., 2017] and the implementation of SAC from OpenAI Spinning Up [1]. The hyperparameters we use for some training algorithm used in chapter 3 and 4 are listed below:

• PPO

- Policy Network: (64, tanh, 64, tanh, Linear) + policy standard deviation variable
- Value Network: (64, tanh, 64, tanh, Linear)
- Normalised observations with running mean filter
- Number of time steps per batch: 2048
- Number of optimiser epochs per iteration: 10
- Size of optimiser minibatches: 64
- Optimiser learning rate: $3e-4$
- Generalised Advantage Estimator (GAE) factor $\lambda = 0.98$
- Discount factor $\gamma = 0.99$
- Cliprange parameter: 0.2
- Number of total training timesteps: $4e6$

• TRPO

- Policy Network: (64, tanh, 64, tanh, Linear) + policy standard deviation variable
- Value Network: (64, tanh, 64, tanh, Linear)
- Normalised observations with running mean filter
- Number of time steps per batch: 1024
- Maximum KL divergence: 0.01

[1] https://github.com/openai/spinningup

- Conjugate gradient iterations: 10
- CG damping factor: 0.1
- Generalised Advantage Estimator (GAE) factor $\lambda = 0.98$
- Discount factor $\gamma = 0.99$
- Value network update epochs per iteration: 5
- Value network learning rate: $1e-3$
- Number of total training timesteps: $4e6$

• DDPG
- Actor Network: (64, relu, 64, relu, tanh)
- Critic Network: (64, relu, 64, relu, Linear)
- Normalised observations with running mean filter
- Noise during training: OU-Noise 0.2
- Learning rates: actor LR: $1e-4$, critic LR: $1e-3$
- L2 normalisation coeff: 0.01
- Batch size: 64
- Discount factor $\gamma = 0.99$
- Soft target update $\tau = 0.01$
- Reward Scale: 1.0
- Number of total training timesteps: $4e6$

• SAC
- Actor Network: (64, relu, 64, relu, tanh)
- Critic Network: (64, relu, 64, relu, Linear)
- Number of time steps per batch: 4096
- Size of optimiser minibatches: 128
- Entropy regulariser coefficient: 0.2
- Learning rates: actor LR: $1e-3$, critic LR: $1e-3$
- Batch size: 64
- Discount factor $\gamma = 0.99$
- Soft target update $\tau = 0.01$
- Number of total training steps: $4e6$

• MAML for multi-domain RL
- Number of domains sampled per epoch: 4

- Number of timesteps per domain per epoch: 1024
- Learning rates: $1e-3$
- Number of total training steps: $4e6$

• Reptile for multi-domain RL
- Number of domains sampled per epoch: 4
- Number of timesteps per domain per epoch: 1024
- Learning rates: $1e-3$
- Number of total training timesteps: $4e6$