

https://dspace.org/

Dynamics generalisation in reinforcement learning through the use of adaptive policies

Beukman, Michael

2024
https://hdl.handle.net/10539/37446

Downloaded from a DSpace repository. https://dspace.org/

Dynamics Generalisation in Reinforcement Learning through the use of Adaptive Policies

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Michael Beukman
1825748

Supervised by Mr. Devon Jarvis, Prof. Richard Klein and Prof. Benjamin Rosman

June 9, 2023



A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science.

Abstract

Reinforcement learning (RL) is a widely-used method for training agents to interact with an external environment, and is commonly used in fields such as robotics. While RL has achieved success in several domains, many methods fail to generalise well to scenarios different from those encountered during training. This is a significant limitation that hinders RL's real-world applicability. In this work, we consider the problem of generalising to new transition dynamics, corresponding to cases in which the effects of the agent's actions differ; for instance, walking on a slippery vs. rough floor. To address this problem, we introduce a neural network architecture, the *Decision Adapter*, which leverages contextual information to modulate the behaviour of an agent, depending on the setting it is in. In particular, our method uses the context – information about the current environment, such as the floor's friction – to generate the weights of an adapter module which influences the agent's actions. This, for instance, allows an agent to act differently when walking on ice compared to gravel. We theoretically show that our approach generalises a prior network architecture and empirically demonstrate that it results in superior generalisation performance compared to previous approaches in several environments. Furthermore, we show that our method can be applied to multiple RL algorithms, making it a widely-applicable approach to improve generalisation.

Declaration

I, Michael Beukman, declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.



June 9, 2023

Acknowledgements

I would like to thank my supervisors, Mr. Devon Jarvis, Prof. Richard Klein and Prof. Benjamin Rosman for their stellar supervision and support. In particular, thanks to Devon for always asking important questions, and discussing wild ideas as well as technical details in depth. Thanks to Richard for giving useful feedback on this dissertation, and looking at problems from alternative and interesting perspectives. Finally, thanks to Benji for consistently providing guidance and actionable advice throughout this project.

Next, I would like to thank Matthew Kruger for helpful discussions, proofreading all of my writing and giving helpful feedback that contributed to improving the dissertation as a whole.

Thanks to Tamlin Love, Branden Ingram and Geraud Nangue Tasse for help with setting up the robotics experiments. Beyond this, thanks to Branden for useful feedback and discussion on many of my illustrations and figures.

I would also like to thank the Mathematical Sciences Support unit at the University of the Witwatersrand for providing all of the computational resources that were used for this dissertation.

Finally, thanks to my family and friends for their constant support.

Contents

1 Introduction **1**

2 Background **4**
2.1 Reinforcement Learning 4
2.1.1 Markov Decision Processes 4
2.1.2 Common Methods 5
2.1.3 Soft Actor-Critic 6
2.2 Formalising Generalisation in RL 7
2.2.1 Contextual Markov Decision Processes 7
2.2.2 Training and Testing Procedure 8
2.2.3 Other Formalisms 9
2.3 Neural Networks 10
2.3.1 Fully Connected Neural Networks 10
2.3.2 Hypernetworks 10
2.3.3 Adapters 11
2.4 Summary 13

3 Related Work **14**
3.1 Robustness Approaches 15
3.1.1 Obtaining More Diverse Environments 15
3.1.2 Altering the Learning Objective 16
3.1.3 Summary 16
3.2 Adaptive Context-Based Approaches 17
3.2.1 How Context is Modelled 18
3.2.2 How Context is Learned 19
3.2.3 How Context is Used 19
3.2.4 Summary 21
3.3 Meta-Learning 22
3.4 Conclusion 23

4 The Decision Adapter **24**
4.1 Problem Setup 24
4.1.1 Definition of Context 24
4.1.2 Problem 25
4.2 Network Architecture 25

4.2.1 Design Decisions . . . 27
4.3 cGate as a Special Case of the Decision Adapter . . . 29

5 Experimental Setup **30**
5.1 Experimental Procedure . . . 30
5.2 Metrics . . . 30
5.2.1 Statistical Analysis . . . 32
5.3 Environments . . . 32
5.3.1 ODE . . . 33
5.3.2 CartPole . . . 35
5.3.3 Mujoco Ant . . . 36
5.4 Methods . . . 36
5.4.1 Normalisation . . . 36
5.4.2 Baselines . . . 37
5.4.3 Decision Adapter . . . 40
5.5 Summary . . . 40

6 Results **41**
6.1 Does the Decision Adapter Generalise Better than our Baselines? . . . 42
6.1.1 When Context is Necessary . . . 42
6.1.2 When Context is not Necessary, but Beneficial . . . 46
6.1.3 A More Complex Environment . . . 49
6.1.4 Real World Showcase . . . 50
6.2 How Sensitive is the Decision Adapter to its Configuration Options? . . . 52
6.2.1 Applying the Decision Adapter to another RL Algorithm . . . 53
6.2.2 Adapter Architecture . . . 54
6.2.3 Hypernetwork Chunking . . . 55
6.2.4 Skip Connection . . . 56
6.2.5 Adapter Location . . . 56
6.2.6 Activation Function Before Adapter . . . 58
6.3 What are the Limitations and Failure Cases of the Decision Adapter? . . . 59
6.3.1 Suboptimal Context Normalisations . . . 59
6.3.2 Context Noise . . . 61
6.3.3 The Effect of Training Contexts . . . 62
6.4 Summary and Discussion . . . 65

7 Conclusion and Future Work **67**

References **68**

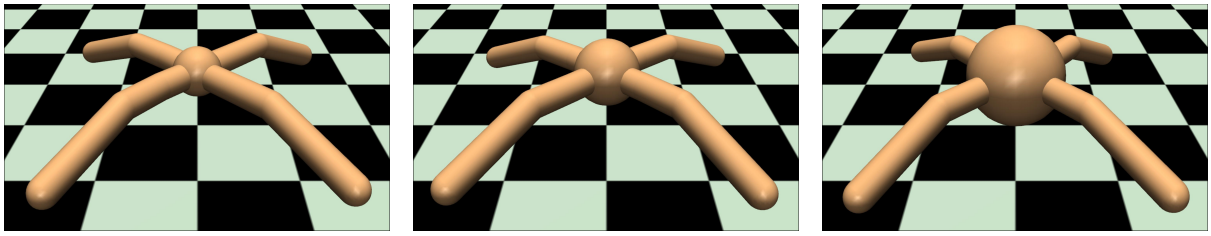
Chapter 1

Introduction

Reinforcement learning (RL) is a powerful tool, and has displayed recent success in as disparate fields as Atari games (Mnih *et al.* 2013), chess (Silver *et al.* 2017), robotics (Akkaya *et al.* 2019) and Starcraft (Vinyals *et al.* 2019). However, despite its potential, RL faces a significant challenge: generalisation. Current agents and algorithms struggle to perform well beyond the narrow settings in which they were trained (Zhang *et al.* 2018a; Cobbe *et al.* 2019; Kirk *et al.* 2023), which is a major limitation that hinders the practical application of RL. For RL algorithms to be effective in real-world scenarios, they must be capable of adapting to changes in the environment and performing well in settings that are different, but related, to those they trained on (Kirk *et al.* 2023).

To illustrate this point, consider a legged robot trained to walk on a tiled floor. While the agent would perform well in these conditions, it will likely struggle in other reasonable settings, such as walking on asphalt or when carrying cargo. When walking on another surface, the friction characteristics would differ from what the robot trained on, which may cause it to slip. When carrying additional weight, the robot is effectively heavier, with a potentially different center of mass. This may mean that the robot must exert more force to lift its legs or be more careful to prevent falling over compared to not carrying anything. This limitation is far from ideal; we want our agents to excel in tasks with minor differences in dynamics (the effect of the agent's actions) without requiring additional training.

To address this problem, several approaches have been proposed, including training a single agent in various settings in the hope that it learns a generalisable and robust behaviour (Pinto *et al.* 2017; Akkaya *et al.* 2019; Abdullah *et al.* 2019; Mankowitz *et al.* 2020; Feng *et al.* 2022). However, this approach is not without its drawbacks, as it fails when variations are too large to allow for a single behaviour to perform well in multiple different environments (Zhou *et al.* 2019; Benjamins *et al.* 2022). For instance, using the same behaviour when walking unencumbered (Figure 1.1a) and under heavy load (Figure 1.1c) would likely result in suboptimal performance in one or both situations.



(a) Light (b) Standard (c) Heavy

Figure 1.1: Showing three robots with different masses (corresponding to the *context*). This additional mass could correspond to carrying a (a) light, (b) medium or (c) heavy load.

This observation has led to other methods that instead use contextual information to choose actions, allowing the agent to adapt its behaviour to the setting it is in. For instance, if the agent knows when it is walking unencumbered vs. carrying something heavy, it can perform slightly differently in each case (Yu *et al.* 2017). This approach enables the agent to execute the appropriate behaviour in each context, without being restricted to the same behaviour across different contexts. Moreover, incorporating context information into decision making can improve the agent's ability to adapt and generalise to new and unseen environments (Benjamins *et al.* 2022). This context information can be anything that affects the dynamics, such as the friction coefficient between the robot's feet and the floor, the incline of the surface it is walking on, its total mass, etc. (Hallak *et al.* 2015; Benjamins *et al.* 2022). This differs conceptually from the *state* information, which includes everything the robot observes using its sensors, such as its joint angles, acceleration, and camera views. This is because context and state change at different timescales; for instance, the floor friction and additional mass remain the same for a long time, whereas the robot's joint angles change after every single action (Hallak *et al.* 2015).

Much progress has been made on the problem of obtaining a useful context representation, and various methods have been proposed to infer context from experience in the environment (Zhou *et al.* 2019; Rakelly *et al.* 2019; Sodhani *et al.* 2022). Despite this progress, the question of how exactly contextual information should be used to make decisions has received less attention. Currently, the predominant approach is to use context as part of the state information, ignoring the conceptual difference between these two aspects (Ball *et al.* 2021; Sodhani *et al.* 2021, 2022). However, using the context in this way may lead to the agent confounding the context and state, instead of learning a decoupled representation that generalises better (Hallak *et al.* 2015; Benjamins *et al.* 2022).

To address this problem, we introduce an improved approach to incorporating context into RL that leads to better generalisation performance. Specifically, our method separates the context and state, allowing the agent to decide how to process the state information based on the setting it is in, thus modulating its behaviour. Our experimental results demonstrate the effectiveness of our approach in improving generalisation compared to other methods. We show that our approach outperforms (1) not incorpo-

rating context information at all; (2) simply concatenating context and state; and (3) competitive baselines.

In particular, we make the following contributions:

1. We introduce the *Decision Adapter*, a neural network architecture that improves generalisation to new dynamics in reinforcement learning by effectively leveraging external information.
2. We show that our method is agnostic to the exact RL algorithm used, and can be effectively applied to on- and off-policy algorithms.
3. We demonstrate that this approach exhibits improved generalisation compared to competitive baselines, due to the effective use of *context*.
4. We theoretically show that our method is a useful generalisation of a prior network architecture, `cGate` (Benjamins *et al.* 2022), and empirically demonstrate that it outperforms this prior approach.

This dissertation is structured as follows. In Chapter 2, we cover some required background knowledge, and Chapter 3 discusses relevant research that aims to address similar questions. We detail our general approach, as well as how it generalises `cGate`, in Chapter 4, while our experimental setup and the application of our approach to several specific domains are outlined in Chapter 5. Our results are presented in Chapter 6, and we finally conclude the dissertation and outline potential avenues for future work in Chapter 7.

Chapter 2

Background

In this chapter, we provide some background necessary to understand the problem we focus on and the workings of our method. In Section 2.1.1, we describe reinforcement learning in general and discuss some common methods in Sections 2.1.2 and 2.1.3. We then define the class of problems we consider and broadly discuss generalisation in Section 2.2. Finally, in Section 2.3, we briefly introduce neural networks, and discuss two specific architectures that are particularly well suited to adapting to new settings.

2.1 Reinforcement Learning

2.1.1 Markov Decision Processes

Reinforcement learning problems are frequently modelled using a Markov Decision Process (MDP) (Bellman 1957; Puterman and Feinberg 1996). An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is the set of states.
- $\mathcal{A}$ is the set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the environment *transition function*, where $\mathcal{T}(s'|s, a)$ specifies the probability of ending up in a certain state s' after starting in another state s and performing a specific action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function*, where $R(s_t, a_t) = R_{t+1}$ specifies the reward obtained from executing action a_t in a state s_t.
- $\gamma \in [0, 1]$ is the environment discount factor, specifying how short-term and long-term rewards should be weighted.

The goal in reinforcement learning (RL) is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximises the expected *return*, where the return is defined as the sum of future discounted rewards $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ (Sutton and Barto 2018).

Policies

A policy π maps between states and actions, thereby defining how an RL agent will act in each state. Policies can be deterministic, in which case $\pi(s_t) = a_t$, or stochastic, where $\pi(a_t|s_t)$ returns a distribution over actions, which can be sampled from to obtain the action. These actions can either be discrete, with a_t being an element of a finite set $\mathcal{A}$; or continuous, with $a_t \in \mathbb{R}^n$.

Value Functions

An important concept in RL is the *value function* of a policy. This is the total expected return that will result from starting in state s and following policy π (Sutton and Barto 2018). This is denoted as

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s].$$

Relatedly, a policy also has an action-value function Q^{π}, where $Q^{\pi}(s, a)$ is the expected return obtained from performing action a in state s and following π in all states thereafter, i.e.,

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a].$$

Optimal Policies

Finally, using value functions, we can obtain a partial ordering of policies (Sutton and Barto 2018). In particular, a policy π is better than or equal to another policy π' if, for all states $s \in \mathcal{S}$, we have that $V^{\pi}(s) \geq V^{\pi'}(s)$. There always exists an *optimal policy* π_* for an MDP, which is better than or equal to all other policies. In practice, however, finding the optimal policy is often challenging or impossible, and approximate solutions are generally used instead (Sutton and Barto 2018).

2.1.2 Common Methods

Two commonly used RL methods are value-based and policy-gradient algorithms. Methods in the former class generally learn the action-value function (or an approximation thereof) and use this to make decisions, by choosing an action that maximises the action-value function in the current state (Watkins and Dayan 1992; Mnih *et al.* 2013).

Due to their prevalence and high performance in continuous control and robotics-based tasks (Schulman *et al.* 2017; Haarnoja *et al.* 2018), we focus only on policy-gradient methods in this dissertation. These methods directly learn a parametrised policy $\pi(a|s, \theta)$ and update the parameters θ to increase the policy's reward. One of the first policy-gradient methods is called REINFORCE (Williams 1992), which updates the parameter vector using the following equation, with a step size α:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}. \tag{2.1}$$

Intuitively, the update to the parameters is in the direction that would increase the probability of selecting action A_t in the future (Sutton and Barto 2018). This, however, is scaled by the return, so a high return would result in the agent performing A_t more frequently in the future, whereas a low or negative return would result in the opposite. The size of the update is also inversely proportional to the probability of taking action A_t in the first place, which serves to normalise the update so that frequently performed actions are not unduly prioritised.

However, this approach generally results in a high variance, which can be addressed by adding an arbitrary *baseline* to Equation 2.1 to reduce the noise in the gradients. This baseline must be independent of the action taken and results in the update rule

$$\theta_{t+1} = \theta_t + \alpha(G_t - b_t)\frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}. \tag{2.2}$$

This baseline can further be a learned approximation to the value function $\hat{v}(S_t|\mathbf{w}) \approx v_\pi(S_t)$, parametrised by the vector $\mathbf{w}$. This approach is very similar to that of Actor-Critic methods (Konda and Tsitsiklis 1999; Degris *et al.* 2012; Sutton and Barto 2018), which often use the one-step return, $G_{t:t+1} = R_{t+1} + \gamma\hat{v}(S_{t+1}|\mathbf{w})$ instead of the full return as their objective (Sutton and Barto 2018).

2.1.3 Soft Actor-Critic

There are many different, high-performing Actor-Critic methods (Mnih *et al.* 2016; Schulman *et al.* 2017), but we discuss only one here: Soft Actor-Critic (Haarnoja *et al.* 2018, SAC). This method was originally developed for continuous action spaces, which we discuss, but it has also been generalised to discrete action spaces (Christodoulou 2019). SAC consists of three main components: (1) an Actor-Critic architecture that learns an action-value function (the critic) and a policy (also called the actor); (2) the use of off-policy learning to enable the reuse of prior experience, thereby improving sample efficiency; and (3) an entropy maximisation objective that leads to improved exploration and stability.

Firstly, SAC learns two action-value functions $q_1(s,a|\mathbf{w_1})$ and $q_2(s,a|\mathbf{w_2})$ and uses the minimum value from either to avoid overestimation bias (van Hasselt 2010; Fujimoto *et al.* 2018). Next, SAC, as opposed to many other algorithms, is *off-policy*, meaning that its experienced transitions are stored in a *replay buffer*, and the experience from the replay buffer is used in learning (Lin 1992; Lange *et al.* 2012). This can lead to significant improvements in sample efficiency compared to on-policy methods (which use each transition only once to learn) because transitions can be used multiple times to improve the policy (Haarnoja *et al.* 2018). Finally, in standard RL, we aim to maximise the expected sum of future discounted rewards; SAC, however, uses a modified, maximum entropy objective (Ziebart 2010), where we instead maximise the following equation (with the added term shown in red):

$$\mathbb{E}[\sum_{t=0}^{\infty}(\gamma^t R_{t+1} + \beta H(\pi(\cdot|s_t,\theta)))], \tag{2.3}$$

where $H(P) = \mathbb{E}_{x\sim P}[-log(P(x))]$ denotes the Shannon entropy (Shannon 1948) and is a measure of how uncertain the outcome is when sampling from a distribution (Ziebart 2010; Achiam 2018). A uniform distribution would maximise the entropy, whereas a distribution with all of its probability mass concentrated on a single point would minimise it (Ziebart 2010). Further, $\beta > 0$ in this equation denotes the weight placed on this entropy term and how much it must be traded off with the reward. The effect of this additional entropy objective is that it (1) incentivises the agent to explore more; (2) can therefore lead to improved learning speed (Haarnoja *et al.* 2018); (3) leads to agents exhibiting more robustness to changes in the environment (Eysenbach and Levine 2022); and (4) smoothens the objective landscape, leading to easier optimisation (Ahmed *et al.* 2019).

SAC generally outperforms strong baseline algorithms in complex continuous control tasks, and exhibits stability as well as improved sample efficiency compared to baselines (Haarnoja *et al.* 2018). Due to these properties, it is a widely used base RL algorithm for numerous other methods (Rakelly *et al.* 2019; Peng *et al.* 2021; Sodhani *et al.* 2022).

2.2 Formalising Generalisation in RL

Generalisation in reinforcement learning can broadly be divided into two categories; few-shot and zero-shot generalisation. In the former setting, experience from many training tasks is leveraged to adapt quickly to a new testing task (Taylor and Stone 2009; Lazaric 2012; Vanschoren 2018; Beck *et al.* 2023). In the latter, an agent is trained in a collection of training environments, with the goal of performing well in an unseen testing environment *without* any additional training (Kirk *et al.* 2023). As mentioned previously, we focus on the zero-shot setting, as this is applicable to many cases where training in the testing domain is infeasible or dangerous, such as in real-world robotics (Peng *et al.* 2018).

2.2.1 Contextual Markov Decision Processes

When dealing with generalisation, it is essential to have a formal definition of being "similar but different". For instance, we wish to generalise to settings where the task (e.g., walking) is the same, but there are slight differences in dynamics (e.g., the floor's friction is different). One such formalism, and the one we use in this dissertation, is that of the *Contextual Markov Decision Process* (Hallak *et al.* 2015, C-MDP), which is defined by a tuple $\langle \mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M}', \gamma \rangle$, where $\mathcal{C}$ is the *context space*, $\mathcal{S}, \mathcal{A}$ are the state and action spaces respectively and $\mathcal{M}'$ is a function that maps a context $c \in \mathcal{C}$ to an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}^c, \mathcal{R}^c, \gamma \rangle$. A Contextual MDP thus defines a *family* of MDPs, that all share an action and state space, but the transition ($\mathcal{T}^c$) and reward ($\mathcal{R}^c$) functions differ depending on the context. In this work, we focus solely on generalisation over changing transition dynamics and therefore fix the reward function, i.e., $\mathcal{R}^c = \mathcal{R}, \forall c \in \mathcal{C}$.

Context Distance

Often the context is considered as a real vector (Sodhani *et al.* 2022), for example, containing the various parameters that affect the dynamics (Yu *et al.* 2017). We can use the distance between two contexts c_1 and c_2 as a proxy for the underlying difference between two environments M^{c_1}, M^{c_2}. There are a few ways to do so, the first being **Euclidean Distance**. This is the simplest approach, where we define the distance between two environments as the Euclidean distance between their corresponding contexts (Lee *et al.* 2020), i.e,

$$||M^{c_1} - M^{c_2}|| = ||c_1 - c_2||_2.$$

The second approach instead defines the distance between environments as the **Wasserstein Distance** $W(\cdot, \cdot)$ between the two transition functions $\mathcal{T}^{c_1}$ and $\mathcal{T}^{c_2}$ (Sodhani *et al.* 2022). Concretely,

$$||M^{c_1} - M^{c_2}|| = \max_{s,a} W(\mathcal{T}^{c_1}(s'|s,a), \mathcal{T}^{c_2}(s'|s,a)).$$

Intuitively, the Wasserstein distance (Vaserstein 1969) is a way to compare two probability distributions (in this case the distribution over next states) by considering each distribution as a pile of soil, and the value of the metric is the cost of transforming one pile into the other, where cost is the amount of dirt that must be moved times the distance it must be moved.

Now, while the Wasserstein distance has traditionally been used (van Breugel and Worrell 2001; Ferns *et al.* 2011), it is not the only way to compare two probability distributions, and metrics such as the Total Variation could also be used (Ferns *et al.* 2004). The Kullback-Leibler divergence (Kullback and Leibler 1951) is also used to compare probability distributions; however, it is not symmetric and thus does not satisfy the conditions of being a metric (Ferns *et al.* 2011).

2.2.2 Training and Testing Procedure

Similarly to supervised learning, when evaluating generalisation in RL, we have training and testing sets of environments. For Contextual MDPs, we have a training set of contexts $\mathcal{C}_{\text{train}}$ and a testing set $\mathcal{C}_{\text{test}}$ (Kirk *et al.* 2023). Usually, we train on the training set for some number of episodes and then evaluate the agent on some or all of the contexts in the testing set (Cobbe *et al.* 2020; Kirk *et al.* 2023). We can consider a testing context to evaluate interpolation performance if this context is inside the convex hull[1] given by the training contexts (Zhang *et al.* 2021b; Sodhani *et al.* 2022). Otherwise, this is termed extrapolation. As an example of this, consider Figure 2.1.

[1]The convex hull C_h of a set of points is the smallest set containing all these points and has the property that for any two points $a, b \in C_h$ and $0 \leq \lambda \leq 1$, we have that $(1-\lambda)a + \lambda b \in C_h$ (Boyd and Vandenberghe 2004).

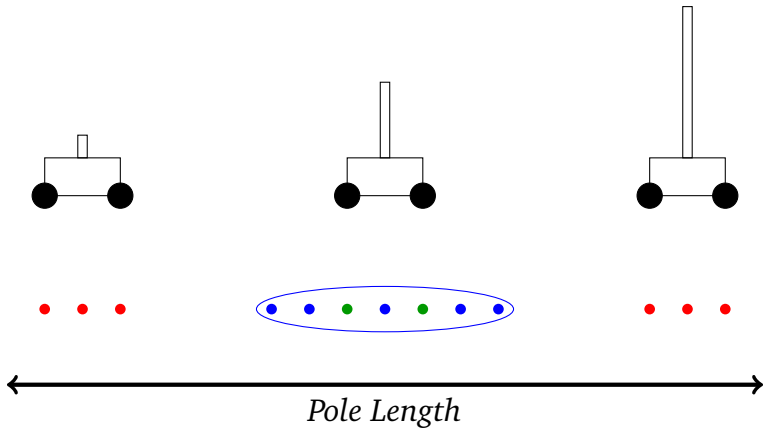


Figure 2.1: An example of generalisation in RL in the CartPole (Barto *et al.* 1983; Brockman *et al.* 2016) domain, where a cart must be moved to balance a pole. The blue points are training contexts, with green and red being evaluation settings. The blue oval represents the convex hull of the training contexts. Red points would correspond to extrapolation, with green being interpolation. Here the context is simply the pole length, and each individual length corresponds to a new environment. These environments all share the same underlying structure, which is parametrised by the unique context, leading to different, but related dynamics. This illustration is inspired by Lee *et al.* (2020).

2.2.3 Other Formalisms

While we consider the Contextual MDP in this dissertation, there have been other proposed variations of MDPs to aid research in generalisation (Kirk *et al.* 2023). One notable example is that of the Hidden Parameter MDP (Doshi-Velez and Konidaris 2016). This is similar to the Contextual MDP, but only the dynamics can vary. While we focus on the dynamics generalisation case, we use the slightly more general formalism to allow future work to investigate other types of generalisation.

Another related formalism is the *Block Contextual Markov Decision Process* (Du *et al.* 2019; Sodhani *et al.* 2021, 2022, BC-MDP). Here, a BC-MDP is identical to the Contextual MDP, the only difference being that each context could possess a unique observation space. In particular, while the state space is shared across environments, the agent's observations may depend on the context (e.g., walking outside vs. inside). We choose not to use this setting, as we focus only on state-based tasks where each context shares the exact same, low-dimensional, observation space.

Finally, *Partially Observable Markov Decision Processes* (POMDPs) can also be used to model this problem. A POMDP is similar to a normal MDP, but it contains some unobserved state information that the agent does not have access to (Kaelbling *et al.* 1998). While we can consider the context as this unobserved state information (Bai *et al.* 2013), using a more restrictive paradigm such as the Contextual MDP or the Hidden Parameter MDP can be beneficial. For instance, for each context c, the environment is an MDP, without any partial observability (Doshi-Velez and Konidaris 2016). Secondly, the context is assumed to remain fixed for each episode, while in the POMDP,

the hidden state may also change during an episode (Hallak *et al.* 2015; Doshi-Velez and Konidaris 2016).

2.3 Neural Networks

2.3.1 Fully Connected Neural Networks

Here we briefly cover fully-connected neural networks, and introduce the notation we use in the rest of this dissertation. We consider a multilayered, fully-connected neural network, also called a Multilayer Perceptron (McCulloch and Pitts 1943; Rosenblatt 1961; Minsky and Papert 1969; Werbos 1974; Haykin 1998, MLP). A neural network is a function $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, parametrised by a vector θ, which is learned from data, usually by gradient descent and backpropagation (Rumelhart *et al.* 1986).

Firstly, neural networks are generally comprised of multiple *layers*, each layer being a linear transformation followed by a nonlinear function. Let us denote a single layer as L_i, mapping an d_i-dimensional vector to a d_{i+1}-dimensional vector. L_i consists of a weight matrix $W_i \in \mathbb{R}^{d_i \times d_{i+1}}$, a bias vector $b_i \in \mathbb{R}^{d_{i+1}}$ and a nonlinear function σ_i. Given an input vector $x_i \in \mathbb{R}^{d_i}$, we can obtain the output $x_{i+1} \in \mathbb{R}^{d_{i+1}}$ as

$$x_{i+1} = L_i(x_i) = \sigma_i(W_i x_i + b_i).$$

Finally, multiple layers $L_1, L_2, \ldots, L_n$ are usually stacked together, one after the other. The overall effect of this is mapping a d_1-dimensional input vector to a d_n-dimensional output vector. We can denote an n-layer neural network as $Net = \{L_i\}_1^n$, defined as follows:

$$\begin{aligned} Net(x) &= x_{n+1} \\ x_{i+1} &= L_i(x_i) \\ x_1 &= x. \end{aligned}$$

Due to neural networks' ability to learn complex functions when given a large amount of data, they have become the de-facto standard, and state-of-the-art approach, in many fields, such as natural language processing (Vaswani *et al.* 2017), computer vision (Krizhevsky *et al.* 2012) and reinforcement learning (Mnih *et al.* 2013; Silver *et al.* 2017).

2.3.2 Hypernetworks

We next discuss *hypernetworks*, which are adaptive neural networks that generate the weights of another network. These networks have the ability to adapt to different scenarios (Ha *et al.* 2017), which can aid in generalising to new settings (Sarafian *et al.* 2021).

A hypernetwork (Ha *et al.* 2017) is a neural network that fundamentally generates the weights for another network, conditioned on some variable. In the original work (Ha

et al. 2017), there are two different networks, the *hypernetwork* and the *main network*. The hypernetwork generates the weight matrix for each layer in the neural network, and is conditioned on a learned layer embedding, which allows it to generate different weight matrices for each layer. The hypernetwork is trained using backpropagation in an end-to-end fashion. Subsequent work has expanded this to generate entire networks based on some embedding. For example, von Oswald *et al.* (2020) learned a *task embedding* that was used to generate a network catered specifically to solve a particular task. This was done to mitigate the effect of catastrophic forgetting in continual learning, by using a constraint that specified that the generated weights for previous tasks should not significantly change when learning new tasks. Huang *et al.* (2021) adapted this approach and applied it to the continual learning setting in RL. Finally, Xian *et al.* (2021) used a hypernetwork to generate the weights of a dynamics model that is specific to the context of the current objects under consideration, leading to improved predictions.

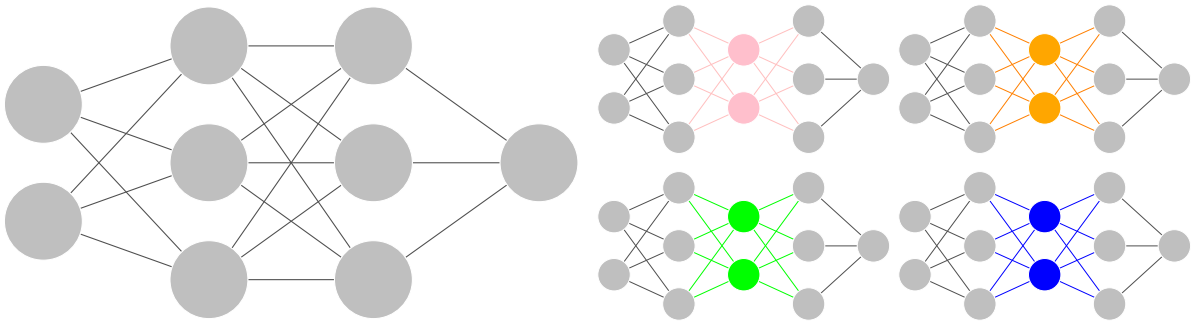
The key aspects of hypernetworks are thus the ability to (1) adapt to new scenarios without forgetting prior learned knowledge and (2) having a network tailored to the specific task (or even a single example (Volk *et al.* 2022)), while still taking advantage of other, previously seen, tasks, as the hypernetwork itself is shared across all of them.

Hypernetwork Chunking

Now, if one were to use a hypernetwork naïvely to directly predict all of the weights of a network, the output dimension would be very large. This would, in turn, lead to the number of learnable parameters in the hypernetwork also being prohibitively large. One way to circumvent this problem is to use a *chunked hypernetwork* (Ha *et al.* 2017; von Oswald *et al.* 2020). Instead of the hypernetwork predicting the entire parameter vector in one forward pass, in this paradigm it outputs a subset (also called a "chunk") of weights, of a fixed size (e.g., $1\,000$). Then, to generate the entire network's weights, this process is repeated, with different learned chunk embeddings to obtain different weights. This is similar to using layer embeddings as in the original work (Ha *et al.* 2017), but allows for a configurable chunk size, and can lead to a hypernetwork with fewer learnable parameters than the main network it generates.

2.3.3 Adapters

An adapter A is a, usually small, neural network, with the same input and output dimension. This network can then be inserted between the layers of an existing, main, network to change its behaviour in some way. For instance, instead of calculating $L_i(x_i)$, we would instead calculate $L_i(A(x_i))$, for some adapter network A. Figure 2.2b illustrates this, where different adapters (in colour) can be inserted between the layers of the main network (in grey).



(a) (b)

Figure 2.2: An example of how adapters would be used in a fully connected network. In (a), we have the main network that is shared across tasks. In (b), we have adapters for each task, with the grey parameters being the same as in (a) and the coloured ones being unique to each task.

Adapters were initially developed for multi-task visual problems (e.g., classifying images in various settings, such as natural images, medical images, etc.), to allow for a significant amount of parameter sharing per task, while still retaining some behaviour specific to each domain. In particular, Rebuffi *et al.* (2017) used a single large network that was shared across all tasks, with small, domain-specific adapters allowing for parameter-efficient adaptation to new domains. Thus, if there were ten different tasks, there would be one base model, while each domain would have its own adapter. Each adapter had the form $A(x;\alpha) = x + \alpha x$, for some learned matrix α. This technique, where the input of a function is added to its output, is commonly called a residual or skip connection, and has been shown to improve the learning speed and final performance of deep neural networks (He *et al.* 2016; Rebuffi *et al.* 2017; Orhan and Pitkow 2018). Furthermore, when a new domain is encountered, the base model can be frozen, with only a new adapter network being instantiated and trained. This technique results in high performance on new tasks without suffering from interference or catastrophic forgetting, which are common issues where neural networks forget earlier tasks after training on new ones (McCloskey and Cohen 1989; French 1999).

Adapters have also seen success in natural language processing (NLP) (Houlsby *et al.* 2019; Pfeiffer *et al.* 2020a,b; Le *et al.* 2021; Fu *et al.* 2022a; Thomas *et al.* 2022). In NLP, the current state-of-the-art performance is generally obtained by using a large language model that was pre-trained in an unsupervised way (e.g., to predict the next word in a sentence) on a large dataset (Devlin *et al.* 2019; Liu *et al.* 2019; Conneau *et al.* 2020). Then, to use this model for a downstream task, such as question answering or sentence classification, a linear layer is added to the pre-trained model, and the entire model is fine-tuned end-to-end, usually on a smaller dataset with a task-specific loss (Devlin *et al.* 2019). This, however, implies that if we are interested in n different tasks, we must store and fine-tune n different pre-trained models. This can be infeasible as these models tend to have a large number of parameters (Radford *et al.* 2019; Conneau *et al.* 2020; Brown *et al.* 2020). Houlsby *et al.* (2019) address this problem using adapters. In particular, the base pre-trained model is frozen, and its weights do not

change during the fine-tuning stage. To obtain a similar effect as fine-tuning the entire model, adapter modules are inserted between certain layers of the pre-trained model as shown conceptually in Figure 2.2. These modules have the same input and output dimensions, effectively transforming the previous layer's features before they are fed into the next layer. The number of trainable adapter parameters is a tiny fraction of the entire network, leading to much more parameter-efficient fine-tuning and adaptation.

2.4 Summary

This chapter briefly detailed several important concepts that are necessary for the rest of this dissertation. In particular, we first covered RL broadly, and then considered the specific problem of generalisation in RL. This included describing the formalism we use in the rest of this dissertation, the Contextual MDP, and the general procedure used to evaluate generalisation in RL. We closed this chapter by discussing fully-connected neural networks and two architectures – hypernetworks and adapters – that are well suited to adaptation. Next, in Chapter 3, we discuss prior work focused on improving generalisation in RL.

Chapter 3

Related Work

Generalisation in reinforcement learning is an important and widely studied problem that has attracted various approaches, perspectives (Taylor and Stone 2009; Konidaris *et al.* 2012; Witty *et al.* 2018; Zhao *et al.* 2019; Upadhyay *et al.* 2021; Wang *et al.* 2021; Lyle *et al.* 2022) and benchmarks (Fortunato *et al.* 2019; Harries *et al.* 2019; Cobbe *et al.* 2020; Ahmed *et al.* 2021; Benjamins *et al.* 2022; Chan *et al.* 2022). While there are multiple aspects of MDPs we could generalise over (Mousavi *et al.* 2014; Chen *et al.* 2019; Jain *et al.* 2020; Nasiriany *et al.* 2021; Trivedi *et al.* 2021; Roy and Konidaris 2021; Zhang *et al.* 2021a; Raileanu and Fergus 2021; Jain *et al.* 2022; Faccio *et al.* 2022; Kirk *et al.* 2023), we focus solely on the problem of generalising to different transition dynamics (Raileanu *et al.* 2020; Zhao and Hospedales 2021; Sodhani *et al.* 2022).

This setting involves performing well on unseen testing domains where the transition dynamics differ from what was observed during training. We assume that there is some shared structure in the transition function, and that all other aspects of the environment remain constant. This is a common scenario in robotics, where the task's objective (e.g., walking without falling), as well as the state and action spaces (e.g., corresponding to a robot's joint angles and forces), remain the same, but the dynamics (e.g., the floor's friction) differ (Peng *et al.* 2018). Furthermore, we consider the *zero-shot generalisation* problem, where the agent is only allowed one episode in the testing environment, so it cannot rely on any kind of fine-tuning on the unseen testing contexts (Kirk *et al.* 2023).

In this chapter, we primarily focus on prior work that aims to address the zero-shot dynamics generalisation problem, but we do discuss other works where relevant. First, we cover *robustness* approaches in Section 3.1. These methods attempt to train agents to be robust to a set of changes that are expected to occur during testing, i.e., to learn a single behaviour that performs well in many settings. We then discuss *context-adaptive* approaches in Section 3.2, which instead use context information to inform the agent's decisions; by using context, these methods can adapt to a wider range of changes than pure robustness methods. We close this chapter by briefly mentioning some methods that focus on few-shot adaptation rather than zero-shot generalisation in Section 3.3.

3.1 Robustness Approaches

Generalisation is a long-standing problem in reinforcement learning because RL agents, without any explicit incentive to the contrary, tend to overfit to the specific environment they are trained in (Zhang *et al.* 2018a; Cobbe *et al.* 2019). This means that the agents will obtain a high training reward, but fail to perform well in settings even slightly different to the training contexts (Malik *et al.* 2021). In many cases, this is due to it being easier to memorise the required steps for a single task compared to learning a general behaviour (Zhang *et al.* 2018b).

Robustness methods aim to address this limitation by learning a single, yet general, policy that can perform well in settings that are different, but close, to the training environment (Kirk *et al.* 2023). There are many ways to achieve this, the most common being (1) increasing the number and diversity of the training environments; and (2) altering the training objective to obtain more robust agents. These two approaches are covered in Sections 3.1.1 and 3.1.2, respectively.

3.1.1 Obtaining More Diverse Environments

One well-known way to train robust agents is to use a large, yet diverse, set of training environments alongside a standard RL algorithm (Zhang *et al.* 2018a; Packer *et al.* 2018). For instance, Zhang *et al.* (2018a) find that overfitting often occurs when training on a single environment, with no variation in the initial state distribution. This can be remedied by allowing more variation, which often leads to agents learning general behaviours (Cobbe *et al.* 2019).

We can obtain this collection of diverse environments by using *Domain Randomisation* (DR), which involves randomising various aspects of the environment during training (Tobin *et al.* 2017; Sadeghi and Levine 2017; Peng *et al.* 2018; Matas *et al.* 2018; Akkaya *et al.* 2019; Andrychowicz *et al.* 2020; Escontrela *et al.* 2020; Wellmer and Kwok 2021). This approach can improve generalisation in two ways. First, by training on a more diverse set of environments, the agent is more likely to learn a general behaviour (James *et al.* 2017; Tobin *et al.* 2017; Zhang *et al.* 2018a). Second, by randomising the environment's parameters, the training distribution over contexts broadens. Therefore, it is more likely that this expanded training distribution is closer to, or overlaps with, the target distribution (Zhao *et al.* 2020; Kirk *et al.* 2023).

While domain randomisation can increase robustness, it may not be readily applicable when some parameter configurations render the environment unsolvable or trivially easy (Kirk *et al.* 2023). One way to address this problem is to have a separate learning process that generates a diverse curriculum of environments for the agent to train on (Gisslén *et al.* 2021; Gur *et al.* 2021). This emerging field is promising and has achieved impressive results thus far (Wang *et al.* 2019, 2020b; Dennis *et al.* 2020; Team *et al.* 2021; Parker-Holder *et al.* 2022). However, the current focus of many of these methods is state-space generalisation, for example, an agent that learns to play on some platformer levels and can generalise to unseen ones, as opposed to the dynamics generalisation problem we care about (Kirk *et al.* 2023).

3.1.2 Altering the Learning Objective

While training on a diverse collection of environments has empirically been shown to increase the robustness of agents, many approaches still use the standard RL training procedure, which does not explicitly incentivise agents to generalise. This observation has led to an emerging research field that focuses on modifying the loss function or the training procedure, in a way to incentivise robustness.

One way to do this is to incentivise obtaining a high worst-case reward over a collection of environments (Pinto *et al.* 2017; Abdullah *et al.* 2019; Chen and Li 2020; Mankowitz *et al.* 2020), instead of simply prioritising the average reward, as is the standard (Krueger *et al.* 2021). The core idea here is that an agent that performs well on average may excel in some environments while performing terribly in others – which can lead to poor generalisation. An agent that instead has a high performance in the worst case will not catastrophically fail when faced with small variations (Pinto *et al.* 2017; Tang *et al.* 2019).

Other approaches are inspired by supervised learning, and leverage regularisation techniques (Srivastava *et al.* 2014; dos Santos and Papa 2022), such as dropout, l_2 weight decay loss and early stopping to improve generalisation performance in RL (Farebrother *et al.* 2018; Ada *et al.* 2022). Beyond this, constraining an agent's capacity by limiting the amount of information it uses from the environment, can also lead to improved robustness and less overfitting (Lu *et al.* 2020; Eysenbach *et al.* 2021).

Finally, some methods aim to encode *invariances* into agents (Zhang *et al.* 2020; Kirk *et al.* 2023), as there are often many irrelevant aspects of the agent's observations that should be ignored if a general policy is desired (Roy and Konidaris 2021; Li *et al.* 2021; Wang *et al.* 2022). For example, a self-driving car should be invariant to (i.e., it should perform the same action) whether there are clouds in the sky or not, as clouds do not affect it in any way (Zhang *et al.* 2021a). Invariance can be encouraged by optimising a loss function that incentivises visually different, but semantically equivalent, observations to map to the same encoded representation (Zhang *et al.* 2021a). One way to achieve this in practice is by applying data augmentation to RL algorithms (des Combes *et al.* 2018; Ye *et al.* 2020; Raileanu *et al.* 2021). Data augmentation, initially developed for supervised learning, is used to construct a larger and more diverse training data set by forming new examples through the augmentation or perturbation of existing training data points; for instance, flipping or rotating images (Krizhevsky *et al.* 2012; Ronneberger *et al.* 2015; Dieleman *et al.* 2015; Taylor and Nitschke 2017; Perez and Wang 2017; Shorten and Khoshgoftaar 2019). This amounts to having the same desired output for different variations of an image, resulting in the model learning to be invariant to these augmentations (Raileanu *et al.* 2021).

3.1.3 Summary

The common thread between all of the robustness methods discussed in this section is that their goal is to learn a single policy that is robust to minor variations between the training and testing environments. Many of these approaches are successful in this goal and can generalise well when faced with small perturbations. However, as Zhou

et al. (2019) note, using one policy for multiple environments (or contexts) may lead to this policy being conservative, as it cannot exploit the specifics of the current environment. Furthermore, these policies are limited to performing the same action when faced with the same state, regardless of the environment. This is potentially suboptimal, especially when the gap between the different environments widens (Kessler *et al.* 2022; Benjamins *et al.* 2022). In Section 3.2, we discuss solutions to these problems; in particular, identifying some characteristics of the current environment (the context or an approximation thereof) and using this information to allow the policy to adapt to new environments.

3.2 Adaptive Context-Based Approaches

The next class of approaches we consider are adaptive, context-based methods. These approaches often use the Contextual MDP (Hallak *et al.* 2015) formalism (or a variation of it), and use the context information to inform the agent's choice of action. This is beneficial, as it allows agents to exhibit different behaviour in different settings, which may be necessary to achieve optimality or generalisation when faced with large variations (Benjamins *et al.* 2022).

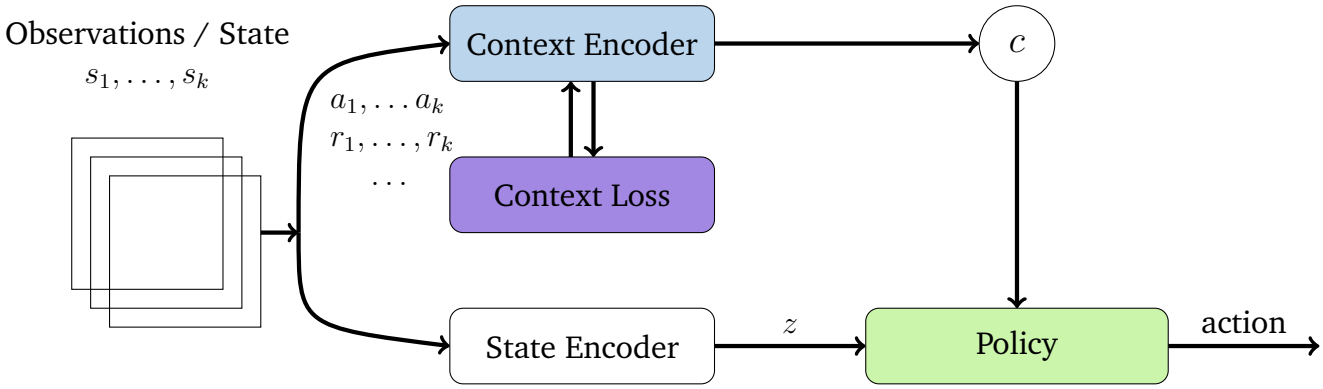


Figure 3.1: An approximation of the general architecture of the context-adaptive methods discussed in this section. Context is inferred using the context encoder, which is trained to minimise some loss function, and the policy uses the predicted context to act. This illustration is inspired by Zhang *et al.* (2021b) and Sodhani *et al.* (2022).

The three primary aspects that differ between methods in this class are (1) how context is modelled, (2) how it is learnt, and (3) how it is used by the agent. The general structure of many of these methods is illustrated in Figure 3.1. In Section 3.2.1, we first consider how context is modelled, i.e., the form of the context encoder in this figure. Next, in Section 3.2.2, we discuss how context can be learnt, i.e., the form of the context loss function. Finally, in Section 3.2.3, we detail how context can be used by the policy, corresponding to the structure of the context conditioned policy.

3.2.1 How Context is Modelled

In general, context is not always visible to the agent, which makes it necessary to build some model of context that can be used to infer an approximate context during deployment. As discussed by Sodhani *et al.* (2022), two types of settings can be considered here. The first is "known unknowns", where the context variables we care about, or we expect to vary, are known, and the task is to accurately predict their values (Åström and Torsten 1965; Swevers *et al.* 1997; Bhat *et al.* 2002; Zhu *et al.* 2018; Yu *et al.* 2017). For instance, consider the case where we have a robot that walks only on one surface, but with additional cargo that has an unknown mass. Here, we know that one aspect of the environment varies – the mass – we simply do not know its value. The second type is the "unknown unknowns", where the properties that vary are not known. This approach has recently received much attention due to its general applicability, and the context is usually modelled as an arbitrary-dimensional vector and passed to the policy (Chen *et al.* 2018; Sodhani *et al.* 2022).

A common approach is to model the context as a function $\psi : (\mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S})^n \to \mathcal{C}$ (Zintgraf *et al.* 2020, 2021; Sodhani *et al.* 2022), i.e., the context encoder receives a collection of transitions $x_i = (s_i, a_i, r_i, s_{i+1})$ as input and outputs the context. This is justified if the context affects the dynamics, observations or reward function, as it can be inferred from a sequence of transitions. This encoder is often modelled as a standard feed-forward neural network (Seo *et al.* 2020; Lee *et al.* 2020; Sodhani *et al.* 2022). Other methods follow a probabilistic approach, assuming a Gaussian distribution over contexts, fittings its parameters, and then performing inference using the posterior distribution (Doshi-Velez and Konidaris 2016; Sæmundsson *et al.* 2018). Using a probabilistic approach allows the agent to infer the context it is currently in, while taking the uncertainty in its estimation into account (Sæmundsson *et al.* 2018; Zintgraf *et al.* 2020).

Another approach for obtaining this context is using a known environment identifier (e.g., an integer from 1 to n representing the task). In particular, Zhang *et al.* (2021b) learn an encoder that maps from this identifier to a continuous vector representing the context that varies smoothly over environments; this context is then fed to the agent's policy to inform its actions. This approach, however, assumes this identifier is given, which may be an unrealistic assumption. Other works, such as Sodhani *et al.* (2021), use brief natural language descriptions of each task in *Meta-World* (Yu *et al.* 2020) as input to a pre-trained language model that outputs a context embedding.

Finally, some methods implicitly model the environment context by augmenting the agent with *memory*. In particular, by using a recurrent neural network – which has a hidden state that is used alongside the current environment state to obtain the next hidden state (Rumelhart *et al.* 1985; Werbos 1990; Hochreiter and Schmidhuber 1997) – the agent can build up a representation using information from sequential timesteps. This approach can implicitly encode the context, without any additional loss terms, allowing the model to perform differently when faced with different settings. This is because the observed transitions would be different in each setting, thereby changing the hidden state (Peng *et al.* 2018; Siekmann *et al.* 2020; Ghosh *et al.* 2021).

3.2.2 How Context is Learned

The next differentiating factor is how these context distributions are learned. One common approach is having a set of auxiliary losses that are backpropagated through to the context encoder. This is often done by training a dynamics model $M : \mathcal{S} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{S}$ conditioned on the context, and optimising the context encoder to improve the dynamics model's predictions (Sanchez-Gonzalez *et al.* 2018; Zhou *et al.* 2019; Lee *et al.* 2020; Seo *et al.* 2020; Sodhani *et al.* 2022). The intuition surrounding this idea is that the encoded context should be useful in predicting the dynamics of the environment, as the ground-truth context directly affects it (Lee *et al.* 2020).

Another approach for learning the context is to add some constraints to the latent context space. For example, Sodhani *et al.* (2022) enforce Lipschitz continuity[1] on the latent context representation. In particular, this is done by assuming that the difference between two environments, in terms of rewards and dynamics (measured as the maximum Wasserstein distance between the two transition functions, see Section 2.2.1), must be bounded by the distance between the two contexts. This is theoretically justified, as prior work (Xu and Mannor 2012) showed that the difference between training and testing error is bounded for Lipschitz-continuous functions. Standard RL, however, often does not satisfy this property, which is why this additional constraint is necessary (Sodhani *et al.* 2022). This approach resulted in improved generalisation, but did start to perform worse as the testing contexts moved further away from the training ones.

The context encoder can also be trained using supervised learning. This may be feasible because, in some – but not all (Zhou *et al.* 2019) – cases, we have access to the simulator during training (allowing us access to the ground truth context parameters) but not during evaluation (Kumar *et al.* 2021). For instance, during training, Yu *et al.* (2017) collect partial trajectories of the form $((s_i, a_i), (s_{i+1}, a_{i+1}), \ldots, (s_{i+h}, a_{i+h}))$ alongside the context c for that episode, and use a mean-squared error loss to fit a supervised model. This model can then be used during evaluation to predict the context. However, this method focuses on the "known unknowns" case discussed earlier, which may restrict it to generalising only over parameters explicitly considered during training. This, in turn, may require significant effort to manually choose all possible parameters that could feasibly change during testing.

3.2.3 How Context is Used

Finally, we come to how these contexts are used to inform the action choices of the agent. The most prevalent approach is to simply concatenate the context to the state, and use this directly as input to the policy (Ghasemipour *et al.* 2019; Rakelly *et al.* 2019; Ball *et al.* 2021; Eghbal-zadeh *et al.* 2021; Sodhani *et al.* 2021, 2022; Mu *et al.* 2022), trained using methods such as Soft Actor-Critic (Haarnoja *et al.* 2018) or

[1] A function $f : I \rightarrow \mathbb{R}$, with $I \subset \mathbb{R}$, is Lipschitz if $\exists A > 0$, such that $|f(x) - f(x')| \leq A|x - x'|$, $\forall x, x' \in I$ (Sohrab 2003).

Proximal Policy Optimisation (Schulman *et al.* 2017). The policy may then adapt its behaviour to the current environment, and act accordingly. This approach, however, may prevent the agent from being able to distinguish state and context, which could lead to worse generalisation (Benjamins *et al.* 2022).

Zintgraf *et al.* (2020) introduce VariBAD, which follows a similar approach, but instead takes a Bayesian view, explicitly modelling a probability distribution over the context c. Whenever a new transition is observed, the belief over c is updated to take this into account. The policy is then parametrised by this posterior distribution, and thus learns to take the belief over contexts, *including uncertainty*, into account. The authors state, however, that generalising to out-of-distribution environments will pose challenges to this method, as the inference procedure may become incorrect.

Sarafian *et al.* (2021), instead take a different perspective. This work uses a hypernetwork (Ha *et al.* 2017) to generate the weights of the primary policy network. Given a context c, they generate a neural network with weights $W(c)$, which processes the state and outputs an action. They applied this method to two common meta-learning approaches, MAML (Finn *et al.* 2017) and PEARL (Rakelly *et al.* 2019). When given a ground-truth context, MAML with a hypernetwork outperformed MAML equipped with the concatenation of state and context. Generating the weights conditioned on the context eased the training of the context encoder by not letting the gradients pass through too many networks. For the critic in PEARL, they found that using the state as the input to the hypernetwork and inputting the context to the policy was a better option. However, this approach still required test-time adaptation episodes, like the original PEARL algorithm, making it less suited to zero-shot generalisation. Furthermore, Sarafian *et al.* (2021) focused more on the reward-variation case, where the context changes the desired behaviour, instead of considering generalisation to new transition dynamics.

There are several additional ways to incorporate context. For instance, Yao *et al.* (2018) perform *action distillation* – learning separate policies per context and using supervised learning to learn a general, context conditional policy. In particular, this context-conditional model was trained to predict the same action in state s when given context c_i as the model that was solely trained on c_i. This method, however, may scale poorly since we must train a separate policy for each training context.

Similarly, FLAP (Peng *et al.* 2021) aims to improve generalisation by effectively learning a shared representation across tasks, which is then processed by a task-specific adapter in the form of a linear layer. Given state s, the policy network outputs features $\phi(s) \in \mathbb{R}^d$, which are shared across tasks. The final action is chosen according to $W_i\phi(s) + b_i$, with a unique weight matrix W_i and bias vector b_i for each task. During training, these weights are learned separately for each task. Concurrently with this, a supervised learning model is trained to map between transition tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ and these learned weights (instead of simply the predicted actions as Yao *et al.* (2018) do). At test time, this supervised model generates the weights, while the policy network remains fixed. Peng *et al.* (2021) demonstrated that their method generalises better to out-of-distribution tasks (e.g., where the dynamics or reward functions differ) compared to other meta-learning approaches. The linear adapter model, in particular, performs well and allows adaptation to be performed faster during testing. However, this approach

requires a separate head for each task, which may scale poorly if we have many tasks or see each task only a few times. Furthermore, if we do not have access to task IDs during training, it would be challenging to apply this method.

Benjamins *et al.* (2022) also learn separate state ($\phi(s)$) and context ($g(c)$) encoders. The action is then obtained by evaluating $f(\phi(s) \odot g(c))$, where f is the learned policy, and $\odot$ is the elementwise product. In essence, this model modulates the state-based features with contextual information before passing it on to the state. This is very similar to the idea of Feature-wise Linear Modulation (Perez *et al.* 2018; Dumoulin *et al.* 2018; Zintgraf *et al.* 2019; Eghbal-zadeh *et al.* 2021, FiLM), where the primary features (corresponding to the state information) are adapted in some way using information from a different modality (the context in this case).

Finally, we come to model-based approaches, where methods learn an approximate model of the transition function and use its predictions to inform the choice of action (Moerland *et al.* 2023). However, when there are multiple tasks (or contexts) with different dynamics, learning a single, context-unaware dynamics model could be a poor choice in cases where the context has a significant effect on the dynamics (Lee *et al.* 2020; Xian *et al.* 2021). This can be remedied by learning a context-conditioned dynamics model, and Lee *et al.* (2020) showed that this could indeed improve prediction performance across multiple contexts. Killian *et al.* (2017) use a Bayesian Neural Network to learn a context-informed model of the transition function, which includes uncertainty information, and train a policy using this approximated dynamics model. Instead of simply parametrising the dynamics model with context, recent work has demonstrated that using a hypernetwork (Ha *et al.* 2017) to generate the weights of a specialised dynamics model based on context can lead to impressive adaptation performance (Xian *et al.* 2021; Kirchmeyer *et al.* 2022). While these approaches can build accurate dynamics models, they may require time in the target domain to adapt, making them less suitable for zero-shot adaptation (Killian *et al.* 2017). Furthermore, training a policy using a learned dynamics model may be computationally expensive, with any inaccuracies possibly causing the policy to overfit to the learnt dynamics, thereby performing poorly in the real environment (Yao *et al.* 2018).

3.2.4 Summary

Generally, the task of inferring the context has received much attention, resulting in the development of multiple high-performing approaches. However, how to incorporate context into the RL setting is not as well studied. Many works simply concatenate the context to the state to form an augmented state space, passing these values directly to the agents (Rakelly *et al.* 2019; Ball *et al.* 2021; Eghbal-zadeh *et al.* 2021; Sodhani *et al.* 2021, 2022). This, however, may pose some significant problems. One potential pitfall is that this procedure artificially enlarges the state space, making it harder to generalise (Hallak *et al.* 2015). Secondly, the context and state change at different timescales, with the state changing at every time step, while the context remains constant across an entire episode. Combining both of these into the same state space may result in the agent confounding these two factors, again resulting in poorer generalisation.

While there are some recent techniques that do separately process the context and state, these approaches also have some limitations. Some methods require the training of separate models, one for each context (Yao *et al.* 2018; Peng *et al.* 2021). This takes advantage of context remaining constant within an episode, but these methods may not scale well as the number of training contexts increases. Other methods are limited to a linear interaction between the context- and state-based features (Benjamins *et al.* 2022), which may not be sufficient to capture complex interactions. Thus, it is still necessary to establish the utility of alternative approaches to incorporating context.

3.3 Meta-Learning

Meta-learning approaches aim to use multiple tasks during training to *learn how to learn*; this knowledge can then be used during testing to quickly adapt to a new task (Thrun and Pratt 1998; Baxter 1998; Upadhyay *et al.* 2021; Beck *et al.* 2023). There are numerous meta-learning methods, but here we only briefly mention some typical approaches, as these algorithms are often more focused on fast or few-shot adaptation instead of zero-shot generalisation (Kirk *et al.* 2023). For instance, Finn *et al.* (2017) aim to learn the weights of a neural network such that, for any new task, performing only a few gradient updates would lead to a high-performing task-specific network. This subclass of meta-learning is closely related to the idea of *fine-tuning*. In this paradigm, an initial, general policy is learned in the training domain(s), which is then updated using significantly fewer timesteps in the target domain. This is quite common when generalising from a simulator to reality, where many methods perform a small amount of fine-tuning on a real robot after training in a simulator (Julian *et al.* 2020; Peng *et al.* 2020; Smith *et al.* 2022). While this approach has potential, applying it naïvely could be problematic or dangerous in settings where a small mistake could result in catastrophe, such as those discussed by Kenton *et al.* (2019) and Srinivasan *et al.* (2020). For instance, falling while walking on grass is acceptable (Smith *et al.* 2022). When a robot is traversing a narrow bridge, however, a small mistake could destroy it (Srinivasan *et al.* 2020).

A second major subclass of meta-learning methods leverages a recurrent model that uses experience in the environment, across episodes, to adapt quickly (Nagabandi *et al.* 2019). Here, the model parameters are frozen during testing, but adaptation is still possible due to the network's hidden state. An example of this is RL^2 (Duan *et al.* 2016), where a recurrent neural network (RNN) is optimised to obtain the maximum reward across multiple episodes in the same MDP. This can be viewed as the RNN itself implementing a separate RL algorithm that is suited to few-shot adaptation (Wang *et al.* 2017). Alternatively, the RNN can be seen as a way to implicitly encode context using experience, in the form of the hidden state, which is used to choose the action (Peng *et al.* 2018). Mishra *et al.* (2018) follow a similar approach, but instead use *attention* (Bahdanau *et al.* 2015; Vaswani *et al.* 2017) alongside temporal convolutions to leverage information across time to better make decisions. However, Packer *et al.* (2018) find that methods that explicitly target adaptation, such as RL^2, can be difficult to train, and generalise worse than standard RL algorithms trained in the presence of

stochasticity. Relatedly, Fakoor *et al.* (2020) find that a standard RL algorithm augmented with context can perform comparably to common meta-learning algorithms.

Finally, meta-learning can be used to train a specialised policy that is only used for exploration and context inference, after which the context can be passed to the primary policy that aims to maximise the reward (Wang *et al.* 2020a). Zhou *et al.* (2019) follow this approach, and find that it outperforms other methods that do not have this separate exploration policy. Similarly, Zintgraf *et al.* (2021) introduce HyperX, which builds upon VariBAD (Zintgraf *et al.* 2020) by incentivising experimentation with different exploration strategies during training. This allows the model to learn in the presence of sparse rewards, a notoriously difficult setting (Nair *et al.* 2018; Sutton and Barto 2018). These methods could be used as part of a context-inference procedure, to obtain a more accurate or useful context, but may be more suited to the few-shot adaptation case.

3.4 Conclusion

Despite the large amount of work focused on improving generalisation in RL, some problems still remain. Models that do not take context into account are inherently limited in how much they can generalise when the environments differ significantly (Benjamins *et al.* 2022). Furthermore, these methods may be forced to act suboptimally, as this could be the only behaviour that is feasible in multiple environments (Zhou *et al.* 2019). While context inference has received much attention and is well understood, how best to use contextual information is less well studied. Many methods simply concatenate the state to the context, effectively constructing a larger state space that the model has to learn, which may hamper generalisation performance (Hallak *et al.* 2015; Benjamins *et al.* 2022). Other approaches are also limited in their applicability to the zero-shot dynamics generalisation problem (Rakelly *et al.* 2019; Raileanu *et al.* 2021; Kirk *et al.* 2023). This gap in current work leads us to focus on the question: "how can we best incorporate context to aid generalisation to new dynamics?"

Chapter 4

The Decision Adapter

4.1 Problem Setup

4.1.1 Definition of Context

As mentioned in Section 2.2.1, we consider the Contextual MDP formalism (Hallak *et al.* 2015). Here we list all of the assumptions that we impose on context:

1. Context is a variable that affects only the dynamics in our MDP. In particular, each MDP has the same transition function, which is parametrised by the context.
2. Context is a continuous parameter of the transition function, as opposed to it being a discrete variable, where different values result in completely different dynamics functions.
3. The context remains constant for an entire episode, and only changes between episodes.
4. Finally, we assume access to the ground truth context during training and evaluation.

Point 1 restricts our focus to generalisation to new transition dynamics where there is some shared structure, but differences in dynamics across contexts. As mentioned in Chapter 3, this is a common scenario in robotics. Relaxing this assumption would mean that other aspects differ between training and testing, such as the reward (Peng *et al.* 2021) or the state space (Cobbe *et al.* 2019). This is not our focus, but these aspects could be investigated by future work.

Point 2 formalises that the context affects the dynamics in a continuous way. This assumption is reasonable, as it corresponds well to real-world robotics use cases, where the context is some parameter (e.g., friction, gravity or mass) of a continuous dynamics function (Doshi-Velez and Konidaris 2016). However, this assumption does not preclude phase transitions that could appear discrete. For instance, consider the next state updating as $s_{t+1} = s_t + cf(s_t, a_t)$, where c is our context, and f is an arbitrary function

of the current state and action. In this case, a phase transition would occur as c moves from a positive to a negative value.

Point 3 is from the original Contextual MDP definition (Hallak *et al.* 2015) and formalises that context is separate from the state and changes on a different timescale. As a corollary to this, the agent cannot influence or change the context. This setup corresponds to having several clearly delineated settings, each associated with a particular stationary context; for instance, we have three rooms, each with its own friction coefficient. Generalisation, in this case, corresponds to training on a set of these contexts (e.g., the three rooms) and performing well in unseen settings (e.g., a fourth unseen room with a different friction coefficient). Relaxing this assumption, and allowing context to change arbitrarily often, corresponds more closely to a partially observable MDP, where the context is part of the unobserved state information that can change over time (Bai *et al.* 2013). In this setting, the agent's actions could also influence future contexts.

The final assumption allows us to isolate the effects of the agent's network architecture, i.e., the way context is incorporated into decision making. It could be relaxed by incorporating any of the existing context inference approaches instead of using the ground truth context (Lee *et al.* 2020; Seo *et al.* 2020; Kumar *et al.* 2021; Sodhani *et al.* 2022).

4.1.2 Problem

The problem we consider is thus: We have a collection of environments with the same transition function, which is parametrised by the *context*. The agent trains on a set of training contexts, where each context results in a separate MDP. After training, the agent is evaluated on a collection of unseen contexts. Our overall goal is to improve generalisation to these settings compared to prior approaches.

Having defined the problem and listed our assumptions, Section 4.2 describes our general Decision Adapter architecture in detail. Finally, in Section 4.3, we show that our approach is a generalisation of a prior network architecture, `cGate`.

4.2 Network Architecture

Our method is inspired by the idea of adapters in NLP and computer vision, with a few core differences from their traditional application. Firstly, since our aim is generalisation, having a separate adapter module for each task (corresponding to context in our case) is infeasible. This is because we would need to train a new adapter for every testing task we are interested in, which is incompatible with the goal of zero-shot generalisation. Furthermore, if we had a separate adapter per task, the number of learnable parameters would increase for every task we train on, which is not ideal. To circumvent this problem, we use a *hypernetwork* to generate the weights of the adapter module based on the context, similar to a recent technique in NLP that generates adapter weights conditioned on the current task (Ye and Ren 2021; Pilault *et al.* 2021; Mahabadi *et al.* 2021; Ivison and Peters 2022; Shi *et al.* 2022). This allows the agent to have a shared feature-extractor network, whose features are modulated

by the adapter, enabling the model to modify its behaviour for each context. Moreover, since the hypernetwork itself is shared, experience gained in one context could transfer to another (Mahabadi *et al.* 2021). Secondly, instead of having separate pre-training and adaptation phases as is common in NLP (Houlsby *et al.* 2019), we do not alter the standard RL training procedure at all. We instead change only the architecture of the agent and train it normally. This allows for the easy integration of our approach with standard RL libraries. Furthermore, our method is agnostic to the exact RL algorithm and can be applied to most algorithms without much additional effort. In particular, we make no assumptions about whether the algorithm is on- or off-policy.

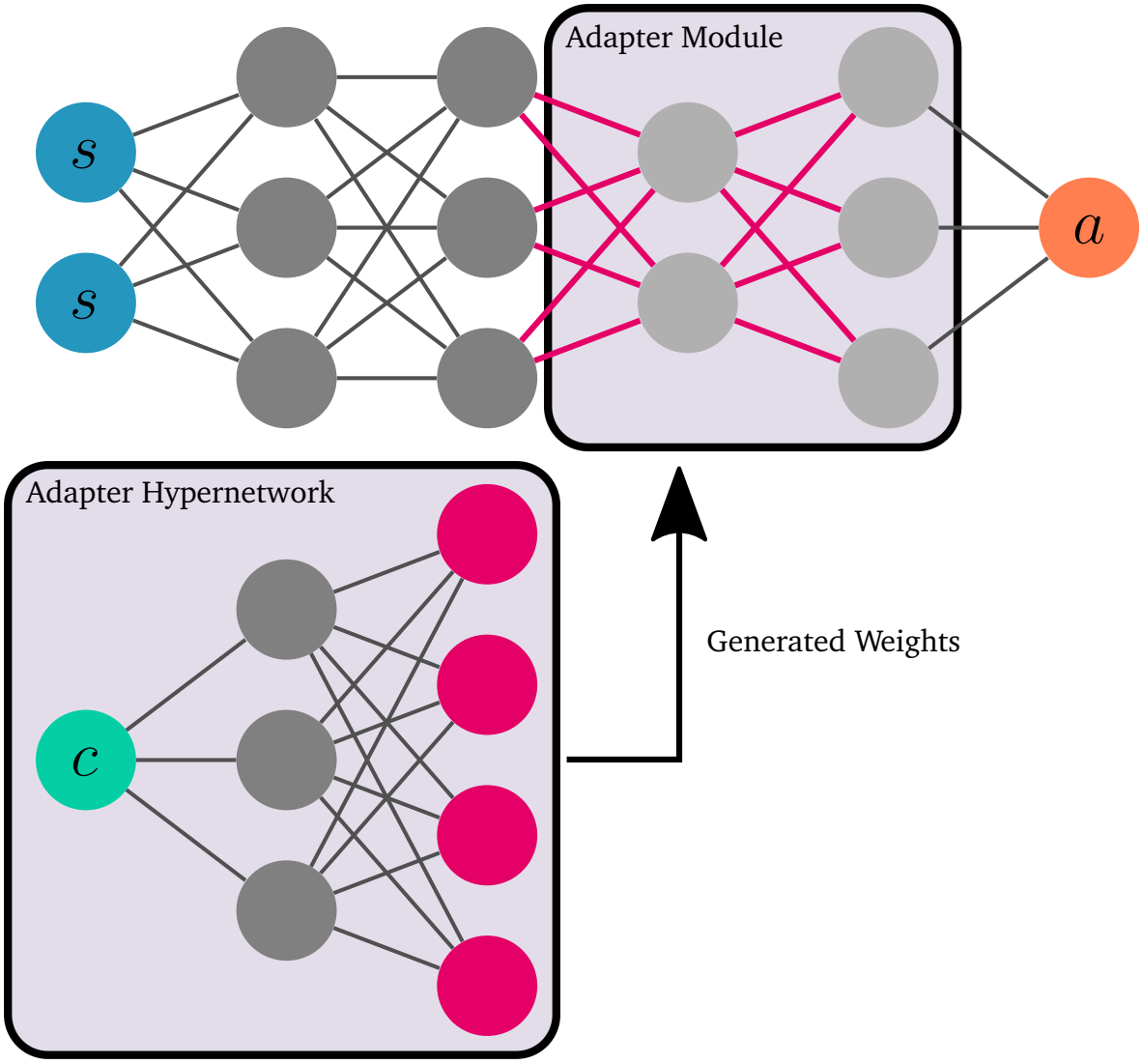
To formally describe our approach (for which pseudocode is available in Algorithm 1), suppose we have a state $s \in \mathcal{S}$, context $c \in \mathcal{C}$, and a standard n-layer fully-connected neural network architecture that predicts our action a, where

$$x_{i+1} = L_i(x_i) = \sigma_i(W_i x_i + b_i)$$
$$x_1 = s, x_{n+1} = a.$$

Our approach involves adding an adapter module A_i in-between layers L_{i-1} and L_i. This adapter module is a multilayer neural network $A_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ consisting of parameters $\theta^i_A \in \mathbb{R}^{P_i}$. These parameters are generated by the corresponding context-conditioned hypernetwork $H_i : \mathcal{C} \to \mathbb{R}^{P_i}$ (line 5 in Algorithm 1) and reshaped to form multiple weight matrices and bias vectors. Then, before we process x_i using layer L_i, we pass x_i through the adapter module A_i (line 6) defined by the generated weights and biases to obtain $x'_i = A_i(x_i|\theta^i_A)$. Furthermore, akin to the application of adapters in other fields (Rebuffi *et al.* 2017; Houlsby *et al.* 2019), we add a skip connection, by setting $x_i = x_i + x'_i$ (line 7). This updated x_i is then the input to layer L_i (line 9). Finally, we could have multiple adapter modules in the primary network, with at most one module between any two consecutive layers $i-1$ and i (lines 3 and 4). Figure 4.1a contains an illustration of our approach.

Algorithm 1 Decision Adapter – Changes to the standard forward pass in blue.

1: **procedure** CHOOSEACTION($s \in \mathcal{S}, c \in \mathcal{C}$)
2: $x_1 = s$
3: **for** $i \in \{1, 2, \ldots, n\}$ **do**
4: **if** $A_i \neq null$ **then**
5: $\theta^i_A = H_i(c)$ // Generate the weights conditioned on c
6: $x'_i = A_i(x_i|\theta^i_A)$ // Perform a forward pass using these weights
7: $x_i = x_i + x'_i$ // Skip connection
8: **end if**
9: $x_{i+1} = L_i(x_i)$
10: **end for**
11: **return** x_{n+1}
12: **end procedure**



(a) An illustration of our network architecture, the Decision Adapter. Here we have the state variables s in blue, context c in green and predicted action a in orange. The bottom network is the adapter hypernetwork, which generates the magenta weights for the top, primary, network. The light grey nodes in the top network are not directly part of the primary network, instead being added by the adapter.

4.2.1 Design Decisions

Now, our method provides a great degree of flexibility in terms of configuration, and we must make multiple design decisions to obtain a usable instantiation of the Decision Adapter. In this section, we briefly detail the default settings we use and justify why these are reasonable. We empirically investigate the merits of these choices and compare them against alternative options in Section 6.2.

Bottleneck Architecture

We use a bottleneck architecture in our generated adapter modules, as described by Houlsby *et al.* (2019). In essence, we have a down-projection layer that transforms features from d_i-dimensional to p-dimensional, with $p < d_i$. Then, the second layer up-projects this back to a d_i-dimensional vector.

The bottleneck architecture is beneficial for two reasons. Firstly, it reduces the number of required parameters in our adapter model. For instance, if we have two layers in this bottleneck architecture, the number of parameters would differ substantially compared to one layer that transforms the d_i-dimensional features to d_i dimensions. In the first case, the number of parameters would be on the order of $O(d_i \times p + p \times d_i)$, corresponding to a $d_i \times p$ matrix for the down-projection and a $p \times d_i$ matrix for the up-projection.[1] If we only have one layer, then we have a single $d_i \times d_i$ matrix. Thus, the bottleneck requires $O(pd_i)$ parameters, whereas the single layer requires $O(d_i^2)$. If we, for example, have $d_i = 256$ and $p = 32$, the bottleneck architecture has four times fewer parameters than the single layer. This, in turn, makes it easier for the adapter hypernetwork to generate useful weights.

Secondly, the bottleneck architecture imparts a low-rank inductive bias on the adapter, which may prevent it from overfitting (Huh *et al.* 2021). This idea is related to prior work that has shown that, counterintuitively, reducing the capacity of the agent's neural network can improve generalisation and reduce overfitting (Igl *et al.* 2019; Lu *et al.* 2020; Eysenbach *et al.* 2021). Although our approach does not incorporate information-bottleneck losses like these methods, the observed benefits of reduced capacity provide additional intuition for utilising the bottleneck architecture. Finally, despite any possible information loss due to the bottleneck layer, the skip connection allows the network to retain any important information from the input.

Adapter Location

Our method allows us to arbitrarily configure our adapter modules, deciding how many we desire and where to locate them in the network. By default, however, we have a single adapter module that, similarly to the use of adapters in NLP, is located near the end of the network, but not at the last layer. This means that the features used by the later layers to predict the action are adapted. Furthermore, since standard deep RL methods generally use much smaller networks than the current state of the art in NLP and computer vision (Ota *et al.* 2021), having only a single adapter module is justified. Finally, in the case of standard actor-critic RL methods, which learn both a policy and a value or action-value function, we add adapters to both networks, to allow both the behaviour and the agent's value function to be conditioned on the context.

[1] The bias vectors correspond to an additional $p + d_i$ parameters, but we omit this term as it is dominated by the number of parameters in the weight matrices.

Hypernetwork Configuration

Finally, even though our adapter module generally has few weights due to the above-mentioned bottleneck architecture, we leverage hypernetwork chunking (von Oswald *et al.* 2020, discussed in Section 2.3.2) to reduce the number of learnable parameters. In particular, we choose the chunk size to be the largest such that the hypernetwork has fewer learnable parameters than the adapter module.

4.3 cGate as a Special Case of the Decision Adapter

To conclude this chapter, we briefly show that our Decision Adapter architecture described here is, in fact, a generalisation of another network architecture introduced by prior work. For the following discussion, suppose $s \in \mathcal{S}$ is the state and $c \in \mathcal{C}$ is the context.

Benjamins *et al.* (2022) introduce `cGate`, which uses the following procedure to obtain an action:

1. Calculate $\phi(s) \in \mathbb{R}^d$ using the state encoder ϕ.
2. Calculate $g(c) \in \mathbb{R}^d$ using the context encoder g.
3. Calculate features $h = \texttt{cGate}(s, c) = \phi(s) \odot g(c)$, with $\odot$ being the elementwise, or Hadamard, product.
4. Calculate action $a = f(h)$, where $f : \mathbb{R}^d \to \mathcal{A}$ is the learnt policy.

Here, both the state encoder ϕ and the context encoder g correspond to fully-connected neural networks that each output a d-dimensional vector, where the value of d is an arbitrary design choice.

This approach is a specific instantiation of our general adapter architecture, where we have one adapter module $A = A_i$. In particular, the state encoder ϕ corresponds to the partial neural network before our adapter module, i.e., layers $L_1, L_2, \ldots, L_{i-1}$. The policy f corresponds to the layers after our adapter module, $L_i, L_{i+1}, \ldots, L_n$. Thus, we have that $\phi(s) = L_{i-1}(x_{i-1})$ and $f(h) = L_n(x_n)$.

If we define our adapter network $A(\phi(s)|c) = \phi(s) \odot g(c)$, we can recover the same result as `cGate`. This would be possible if we let our adapter consist of one layer without a nonlinearity or a bias vector, with its weight matrix being set to $W = H(c) = diag(g(c))$. This would result in

$$A(\phi(s)|c) = W\phi(s) = diag(g(c))\phi(s) = g(c) \odot \phi(s) = \texttt{cGate}(s, c),$$

where we have used the fact that, for two vectors $a, b \in \mathbb{R}^d$, the matrix multiplication $diag(a)b$ is the same as the Hadamard product $a \odot b$. This output vector would then be passed to the policy network f. Our approach, however, is strictly more powerful than a single elementwise product, as our adapter can be an entire nonlinear neural network.

Chapter 5

Experimental Setup

This chapter discusses the experimental approach we follow. In particular, we detail our training and evaluation procedures in Section 5.1 and define the metrics we use in Section 5.2. We describe the environments we use in Section 5.3. Finally, Section 5.4 contains details regarding the methods we use, including the baselines we compare against as well as our Decision Adapter configuration.

5.1 Experimental Procedure

Our general training procedure is as follows: We have a fixed set of training contexts, and a training budget corresponding to a specific number of total environment interactions (i.e., steps) we can perform. Each episode uses a new context, in a set sequence, looping to the beginning if there are more episodes than contexts. To fairly compare all methods, we use approximately the same number of parameters for each method's neural networks. Furthermore, we run each experiment independently 16 times and report the mean and standard deviation of the reward achieved.

Finally, we generally choose our training context set to be a small set of discrete contexts. This approach is opposed to a continuous range of values between two bounds or sampling from a normal distribution centered at some context (Benjamins *et al.* 2022). Choosing our contexts in this way allows us to more closely represent training in the real world. For instance, we have access to a handful of different surfaces to train a robot to walk on, instead of fine-grained control over the context variables. This setup additionally emphasises *context efficiency*, i.e., being able to generalise well after training on a small number of contexts (Kirk *et al.* 2023).

5.2 Metrics

To accurately evaluate the models' generalisation performance, our evaluation strategy is as follows: We take the trained model, and compute the average of the total episode reward over $n = 5$ episodes on each evaluation context c. We use $n > 1$ as computing the reward from just a single episode may lead to an inaccurate estimate of

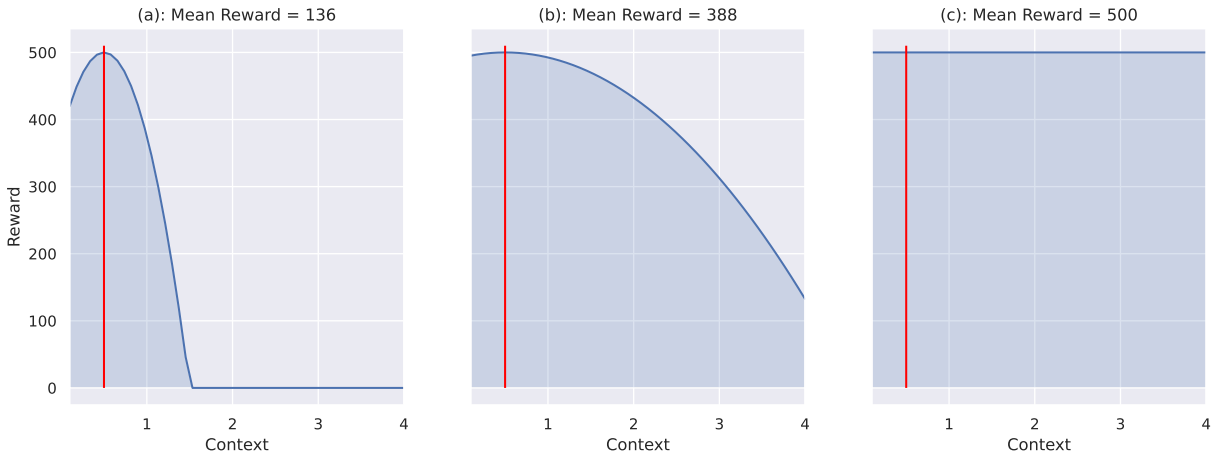


Figure 5.1: Here we illustrate three example context-reward plots. On the x-axis, we have the context c, with the red vertical line indicating the training context. The y-axis contains the rewards, with 500 being the optimal reward for this environment. Finally, the shaded areas indicate the area under the curve calculation to obtain the summary metric. (a) represents the results of an overfitting model, performing well only near the training range. (b) exhibits a more graceful decline as we move away from the training context. Finally, (c) illustrates perfect generalisation.

performance in the presence of environmental stochasticity; setting $n = 5$ provides a balance between computational load and accuracy. These evaluation contexts are generally sampled uniformly between two bounds. This procedure gives us a collection of contexts C and the corresponding rewards R.

We then use this information to calculate the primary metric we use to evaluate generalisation, the **Average Evaluation Reward**. Here, we compute the average reward obtained over the entire evaluation range, which is proportional to the area under the context-reward curve. Figure 5.1 shows an example of this. More formally, we compute the following metric:

$$\textbf{Average Evaluation Reward} = \frac{1}{c_{max} - c_{min}} \int_{c_{min}}^{c_{max}} R(c)dc,$$

where c_{min} and c_{max} are the minimum and maximum context values in C respectively, and $R(c)$ is the reward obtained on context c. Given a finite set of contexts C and their associated rewards R, we numerically calculate the integral using Simpson's rule (Press *et al.* 2007). We choose to use the integral instead of simply the average reward to more accurately measure generalisation performance when we sample unequally spaced evaluation contexts. When the context consists of multiple dimensions, we compute the integral over all dimensions; for instance, computing a double or triple integral. In this case, we obtain the normalisation factor by multiplying each context dimension's normalisation fraction (i.e., $\frac{1}{c_{max}-c_{min}}$).

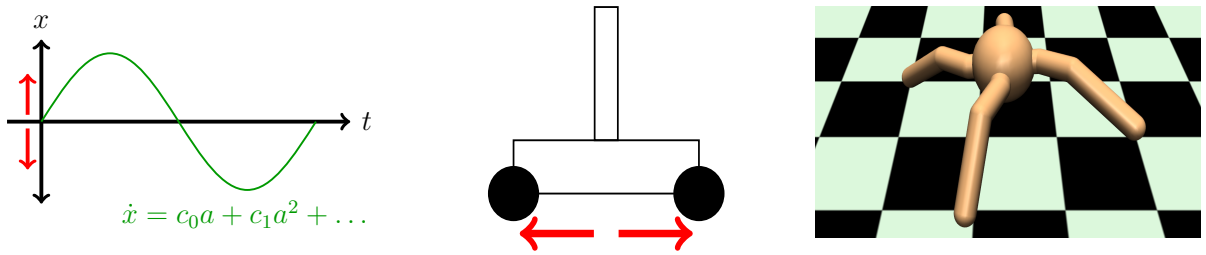
This metric is high when the agent performs well across a wide range of contexts, and thus corresponds well to generalisation performance. However, since the training con-

text set is generally a subset of the evaluation contexts, this metric does also incorporate how well the agent performs on the training contexts. Despite this, it is still a useful metric for generalisation, as our training context set typically contains only a handful of contexts, with most of the evaluation contexts being unseen. Finally, using this definition, we compute the metric value separately for each random initialisation, and then report the mean and standard deviation over the 16 seeds we consider.

5.2.1 Statistical Analysis

Our statistical analysis is as follows. Suppose we have two methods A and B that we wish to compare, with 16 seeds for each. We consider only the average evaluation reward at the end of training, as a measure of how well each method generalises. In general, we use the null hypothesis that $\mu_A = \mu_B$, i.e., that the mean performance (over seeds) of method A is the same as that of method B. We use a two-tailed Mann-Whitney U test (Mann and Whitney 1947) to determine if there exists a statistically significant difference between the two methods, or not. In particular, we set the significance threshold to $\alpha = 0.05$, i.e., if the probability of obtaining a difference in performance as large as the one we observe is less than 0.05, we reject the null hypothesis and conclude that there is a statistically significant difference between the two methods. If the probability p is greater than or equal to 0.05, we fail to reject the null hypothesis and conclude that there is no statistically significant difference. We use the Mann-Whitney U test instead of the Student's t-test (William 1908) because some of the data failed a Shapiro-Wilk normality test (Shapiro and Wilk 1965) with $p < 0.05$. The Mann-Whitney U test (hereafter abbreviated as the MWU test) does not require normally distributed data, whereas the t-test does. Finally, if we perform multiple pairwise comparisons, the chance of making a type I error (rejecting the null hypothesis when it is true) increases (Jafari and Ansari-Pour 2019; Pretorius *et al.* 2020). To remedy this, we perform Bonferroni error correction (Bonferroni 1936), which modifies the p values by multiplying them by the number of comparisons performed. We do this separately for each hypothesis we test.

5.3 Environments



(a) The ODE environment, with state x and action a.

(b) CartPole, actions move the cart to the left or right.

(c) Mujoco Ant, with an 8-dimensional action space.

Figure 5.2: The environments we use. Red arrows indicate the allowed actions. In (c), the actions are individual joint forces.

We evaluate our method on three different environments, described below and illustrated in Figure 5.2. For each environment, we also list which aspects we vary to obtain different dynamics. We first introduce *ODE*, a conceptually simple, yet highly configurable, environment in Section 5.3.1. Next, Section 5.3.2 discusses a more grounded environment that is widely used when studying dynamics generalisation; namely, Cart-Pole. Finally, in Section 5.3.3, we discuss a high-dimensional, complex and challenging environment, that of the Mujoco Ant.

5.3.1 ODE

This environment is a dynamical system described by an ordinary differential equation (ODE) that is parametrised by n variables making up the context. We use the following dynamics equation, truncated at some c_{n-1}.

$$\dot{x} = c_0 a + c_1 a^2 + c_2 a^3 + \dots \tag{5.1}$$

This results in our state x updating according to $x_{t+1} = x_t + \dot{x}_t dt$, where the goal is to provide control a to keep the state close to $x = 0$. The episode terminates after 200 timesteps.

The reward function is defined as follows:

$$R_t = \begin{cases} 1 & \text{if } |x| < 0.05 \\ \frac{1}{2} & \text{if } |x| < 0.1 \\ \frac{1}{3} & \text{if } |x| < 0.2 \\ \frac{1}{4} & \text{if } |x| < 0.5 \\ \frac{1}{20} & \text{if } |x| < 2 \\ 0 & \text{otherwise.} \end{cases}$$

This incentivises the agent to keep the state close to $x = 0$ by reducing the reward as the state moves away from 0. Figure 5.2a shows an illustration of this environment.

The context in this environment is $c = [c_0, c_1, c_2, \dots, c_{n-1}] \in \mathbb{R}^n$. The action space is two-dimensional, continuous and bounded, i.e., $a_0, a_1 \in [-1, 1]$. The action is interpreted as a complex number, $a = a_0 + a_1 i$, to allow Equation 5.1 to be solvable, even in normally unsolvable cases such as $\dot{x} = a^2$. Here, restricting a to $\mathbb{R}$ would result in $\dot{x} \geq 0$ and an unsolvable system for initial conditions $x_0 > 0$. We keep only the real part of the updated state. Having continuous actions results in agents needing to consider the magnitude of the action to perform instead of just the direction or a coarse discretisation. Having unbounded actions has the potential consequence of making the environment solvable in a single step, which is not desired, as this is unlike many RL problems which have a temporal aspect (Sutton and Barto 2018). Moreover, we opt to not have $\dot{x}$ defined in terms of x, as this (combined with bounded actions) may result in some environments being unsolvable. We also clip the state x_t to always fall between -20 and 20. Finally, we have a separate parameter, the *force* F, which is used to scale the agent's actions before they are performed in the environment. Thus, the

agent predicts an action between -1 and 1, which is then multiplied by the force. The force impacts the magnitude of the agent's actions, and can be increased to make the environment more challenging. By default, we use a value of $F = 100$, kept constant across contexts. When we have multidimensional contexts, we rather set $F = 10$, to ensure that the environment is still reasonably solvable. If we set $F = 100$, then the effects of the agent's actions are very large, due to having a $(Fa)^2$ term in the equation for $\dot{x}$.

The ODE acts as a conceptually simple environment where we can arbitrarily scale both the number (i.e., n) and magnitude (i.e., $|c_i|$) of the contexts and precisely measure the effects. Furthermore, since many dynamical systems can be modelled using differential equations (Moore 1990; Sutton 1995; Brockman *et al.* 2016), this environment can be considered a distilled version of this class of environments.

Experimental Procedure

During training, for each context, we cycle between starting states, with each episode taking a new starting state from the set $\{1.0, 0.5, -0.5, -1.0\}$. Thus, the agent trains on the first context for 4 episodes, each with a different initial state. Then it goes on to the second context, and so on. When we reach the end of the context list, we simply start at the first context again. We do this to ensure training is comparable for each method, and that there is sufficient diversity in the initial state distribution. During evaluation, we fix the starting state at $x_0 = 1.0$ to isolate the effect of context.

On the Necessity of Context

Furthermore, in the ODE environment, under certain conditions, we can show that using context is strictly necessary to perform well in multiple settings. Consider the case where we have a one-dimensional context c_0, with the corresponding dynamics equations

$$\begin{aligned} \dot{x} &= c_0 a \\ x_{t+1} &= x_t + \dot{x}_t dt \\ \implies x_{t+1} &= x_t + c_0 \cdot a \cdot dt. \end{aligned}$$

Then, if we were to choose an optimal action $a \in \mathbb{C}$ such that $x_{t+1} = 0$, ignoring the action bounds and assuming $c_0 \neq 0$, we would obtain

$$\begin{aligned} x_{t+1} &= x_t + c_0 \cdot a \cdot dt \\ \implies 0 &= x_t + c_0 \cdot a \cdot dt \\ \implies -x_t &= c_0 \cdot a \cdot dt \\ \implies a &= -\frac{x_t}{c_0 \cdot dt}. \end{aligned}$$

This equation, crucially, shows that the optimal action depends on the context. Furthermore, for a fixed state x_t and context magnitude $|c_0|$, the optimal action depends solely

on the sign of the context. In fact, choosing the incorrect sign is close to being the worst possible action for that state and context. To better illustrate this, suppose that $x_t = 1$ and $c_0 = 1$, with $d_t = 0.1$. Then, $a = -\frac{1}{1 \cdot 0.1} = -10$. If the context is -1 instead, then performing the action $a = -10$ would result in $x_{t+1} = 1 + (-1)(-10)(0.1) = 2$, i.e., the state moving further away from the desired state of 0. This means that a model that does not take contextual information into account *cannot* simultaneously perform well in environments defined by differently signed contexts. This property makes this environment a good candidate to compare context-unaware models to context-conditioned ones.

5.3.2 CartPole

CartPole (Barto *et al.* 1983, see Figure 5.2b) is a task where an agent must control the movement of a cart to balance a pole vertically placed upon it. The observation space is a 4-dimensional real vector containing the cart's position x and velocity $\dot{x}$, as well as the pole's angle θ and angular velocity $\dot{\theta}$. The value for each state variable at the start of the episode is randomly sampled from the interval $[-0.05, 0.05]$. We use a continuous action space where $a \in [-1, 1]$ corresponds to the force applied to the cart (where negative values push the cart to the left and positive values to the right). The reward function is $+1$ for each step that the pole is upright. An episode terminates when the pole is tilted too far off-center, the cart's position is outside the allowable bounds or the number of timesteps is greater than 500.

CartPole can be described by the following system of second-order differential equations (Barto *et al.* 1983; Florian 2007; Brockman *et al.* 2016):

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta\left(\frac{-F - m_p l \dot{\theta}^2 \sin\theta}{m_c + m_p}\right)}{l\left(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p}\right)}$$

$$\ddot{x} = \frac{F + m_p l(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)}{m_c + m_p}.$$

Here, θ is the pole's angle and F is the force, consisting of the agent's action times a constant force magnitude. The other parameters and their default values are listed in Table 5.1. We follow prior work and consider the variables in this table collectively as the context (Benjamins *et al.* 2022). In particular, this means that the context is represented as a 5-dimensional vector, even when we vary only a subset of the dimensions during training.

Table 5.1: The five context variables and their default values.

Name	Symbol	Default Value
Gravity	g	9.80
Cart Mass	m_c	1.00
Pole Mass	m_p	0.10
Pole Length	l	0.50
Force Magnitude	$\tilde{F}$	10.00

This environment is useful as it (1) is a simple setting with models that can be trained quickly, and (2) still has desirable aspects, namely a way to perturb some underlying variables to change the dynamics of the environment. Furthermore, prior work has shown that even standard, unaware RL algorithms can generalise well in this environment (Packer *et al.* 2018), which allows us to investigate the benefits of context in this case.

5.3.3 Mujoco Ant

As a more complex and high dimensional problem, we consider *Ant* from the Mujoco suite of environments (Todorov *et al.* 2012, Figure 5.2c), originally introduced by Schulman *et al.* (2016). Here, the task is to control a 4-legged robot such that it walks. The observation space consists of the robot's joint angles and velocities, as well as contact forces and torques applied to each link. This totals 111 dimensions. The action space $\mathcal{A} = [-1, 1]^8$ is 8-dimensional, representing the torque applied to each of the 8 joints. The initial joint positions are such that the ant is standing and facing forward, with some uniform noise in the range of $[-0.1, 0.1]$ added to it. The initial joint velocities are sampled from $\mathcal{N}(0, 0.1)$. For this environment, we consider two cases. In the first, the context is the robot's mass. In the second, the context is the friction between the robot's legs and the floor. In both cases, the context is represented as a 1-dimensional vector. Furthermore, each episode is terminated after 1000 timesteps.

We choose the Ant environment due to its high-dimensional state and action spaces, and the fact that it is more challenging than either CartPole or the ODE. Furthermore, it is a well-known environment that has been used in prior work (Nagabandi *et al.* 2018; Haarnoja *et al.* 2018; Fu *et al.* 2022b).

5.4 Methods

This section details the methods we use, including our baselines, and the relevant implementation details.

5.4.1 Normalisation

As mentioned in Section 4.1, we give each method access to the ground-truth context during training and evaluation. However, as shown in Table 5.1, the magnitudes of the different context variables may be significantly different, which may cause problems for the neural network-based policies (Hastie *et al.* 2009). To remedy this, we normalise the contexts to a range between 0 and 1. In particular, for each context dimension i, we calculate the context to be given to the agents $c_{norm,i}$ as

$$c_{norm,i} = \frac{c_i}{max_i},$$

where max_i is the largest value for dimension i in the training context set. This results in the largest training contexts having values of 1.0. During evaluation, we perform

the same normalisation, so contexts outside the convex hull of the training set will potentially result in values larger than 1.0.

5.4.2 Baselines

As our approach revolves around introducing a new network architecture with the goal of improving generalisation, we consider several baselines to compare our model against. For the following discussion of the baselines, we denote $s \in \mathcal{S}$ to be the state and $c \in \mathcal{C}$ to be the context. We use `ReLU` for all activation functions. We also aim to have an equal number of learnable parameters for each method and adjust the number of hidden nodes to achieve this.

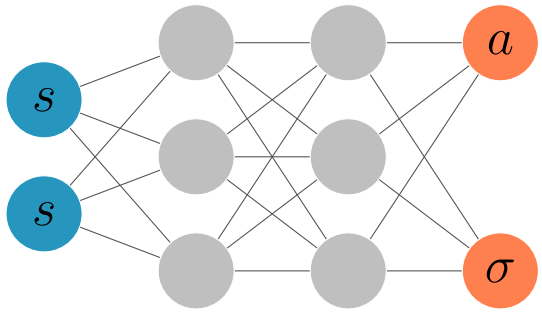
We use Soft-Actor-Critic (Haarnoja *et al.* 2018, SAC) for all methods, to isolate and fairly compare the network architectures. We use the high-performing and standard implementation of SAC from the *CleanRL* library (Huang *et al.* 2022) with neural networks being written in PyTorch (Paszke *et al.* 2019). We use the default hyperparameters, which are listed in Table 5.2. SAC learns both an actor and a critic. The actor takes as input the state and outputs μ and $\log(\sigma)$, which are used to construct the probability distribution over actions $\mathcal{N}(\mu, \sigma)$ which is sampled from to obtain an action. The critic receives the performed action a as well as the state s, and outputs a single number, representing the approximate action-value $\hat{q}(s, a)$. The following sections describe each of our baselines.

Name	Value
Buffer Size	$1\,000\,000$
γ	0.99
τ	0.005
Batch Size	256
Exploration Noise	0.1
First Learning Timestep	5000
Policy Learning Rate	0.0003
Critic Learning Rate	0.001
Policy Update Frequency	2
Target Network Update Frequency	1
Noise Clip	0.5
Automatically Tune Entropy	Yes

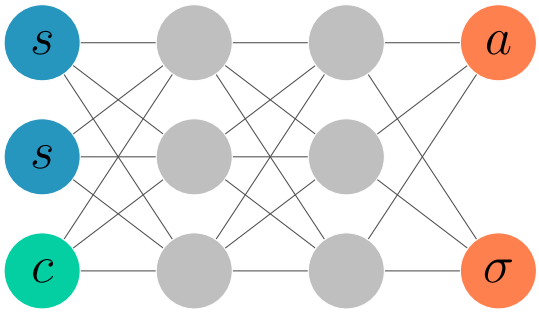
Table 5.2: The default hyperparameters we use in the CleanRL implementation.

Unaware

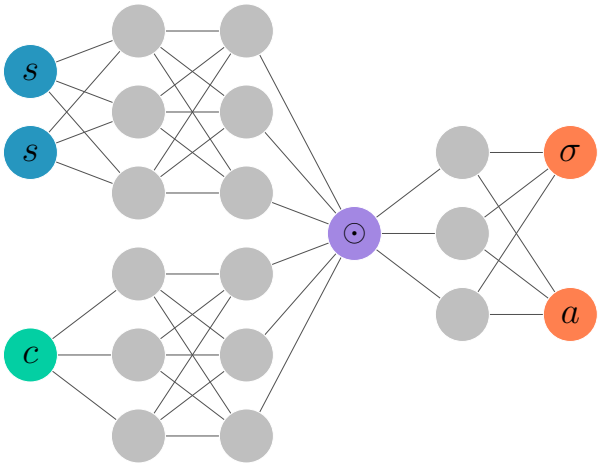
The first baseline we consider is the context-unaware model, shown in Figure 5.3a, which simply ignores the contextual information and just uses the state. This approach allows us to evaluate how well a method that does not incorporate context performs. We do not use any of the additional robustness techniques mentioned in Section 3.1 as many of them can also be applied to context-aware models. Thus, we keep all factors



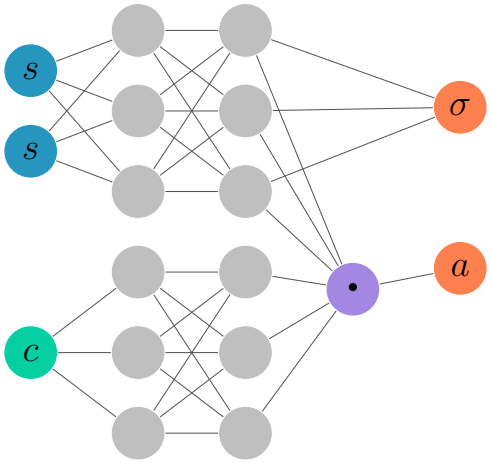
(a) The Unaware Model



(b) The Concatenate Model



(c) The `cGate` Model, with $\odot$ indicating an elementwise product



(d) The FLAP Model, with $\cdot$ indicating a dot product

Figure 5.3: Illustrating the architectures of each of our baselines. Here we consider the actor's architecture, with two separate output heads, one predicting the mean action a and the other predicting the log of the standard deviation (denoted as σ in this figure). In each plot, the blue nodes represent the state inputs whereas the green one represents the context. In general, for the bottom row, the network architectures for the context and state encoders are not necessarily the same.

other than the model architecture consistent to evaluate the merits of context. In particular, we use a fully-connected architecture with two hidden layers with 288 neurons each for both the actor and the critic. We chose this architecture as it outperformed a deeper network with three hidden layers and the same number of overall parameters.

Concatenation

This approach is very prevalent in the current literature, and it involves forming a new, augmented state space $\mathcal{S}' = \mathcal{S} \times \mathcal{C}$, and then feeding the model the concatenation $[s; c]$ of the state and context (Ball *et al.* 2021; Eghbal-zadeh *et al.* 2021; Sodhani *et al.* 2021, 2022). This baseline, shown in Figure 5.3b, allows us to compare to the current standard approach of using contextual information, to determine whether our method is indeed superior. The network architecture we use consists of three hidden layers

with 205 nodes each, as this outperformed a shallower network with a similar number of parameters. Hereafter we refer to this method as the Concat model.

cGate

The next baseline is another recent way of incorporating context into RL, introduced by Benjamins *et al.* (2022), and described more in Section 4.3. The core idea here is to have separate state ($\phi(s)$) and context ($g(c)$) encoders. Then, we obtain the action by evaluating $f(\phi(s) \odot g(c))$, where f is the learned policy, and $\odot$ is the elementwise product. Figure 5.3c shows an example of this. The exact architecture we use for our experiments consists of a state encoder with two hidden layers of 205 nodes each, a context encoder consisting of one hidden layer with 205 nodes and a final policy network with one 205-node hidden layer. The network head that outputs the standard deviation is a single linear layer, mapping $\phi(s) \odot g(c)$ to $\log(\sigma)$. We choose this configuration, as it is similar to the one used by Benjamins *et al.* (2022), and has a comparable number of parameters to all of the other baselines. This work originally used other algorithms, such as TD3 (Fujimoto *et al.* 2018), but to ensure comparability, we apply the network architecture to SAC. As such, we add the $\log(\sigma)$ head to the actor, which is absent in TD3 (Huang *et al.* 2022). The critic's network architecture is the same as the actor's, just without the $\log(\sigma)$ head. Since our approach is a theoretically more powerful generalisation of `cGate`, this baseline allows us to determine whether this generalisation results in improved performance.

FLAP

Our final baseline, called Fast Linearised Adaptative Policy (Peng *et al.* 2021, FLAP) – described in more detail in Section 3.2.3 – works by effectively learning a shared state representation across tasks. This representation is then processed by a task-specific linear layer that is generated by conditioning on the context. We implement the training procedure as described by Peng *et al.* (2021), i.e., where a separate policy head is trained for each task and a weight-predicting network is trained to predict the weights of the linear layer. We use the following network architectures: The actor and critic networks have three hidden layers of 120 nodes each, and the weight-predicting network has three hidden layers of 240 nodes each. This configuration was chosen to be comparable to the other baselines in terms of the number of learnable parameters, while having the same number of layers as in the original work. Furthermore, we give this method access to the ground truth context to be comparable to the other baselines. Preliminary experiments demonstrated that using the context to generate the weights outperformed using the (s, a, r, s') transition tuples and that the (much larger) network architecture used by Peng *et al.* (2021) performed comparably to the one we use. Finally, while most of the default parameters above were the same as the ones used by Peng *et al.* (2021), they used a critic learning rate of 0.0003, which we also use only for this baseline. Figure 5.3d illustrates this model.

5.4.3 Decision Adapter

Our Decision Adapter model is configured as follows for the experiments in Chapter 6. The base network architecture is the same as the Unaware model (e.g., Figure 5.3a). The main actor network consists of a *trunk* that outputs a hidden state of size 256, with one hidden layer of size 256. We choose 256 as this is the default size of the networks' hidden layers in CleanRL (Huang *et al.* 2022). The $\log(\sigma)$ head is a single linear layer, mapping this hidden state to $\log(\sigma)$. We do not add an adapter module to the $\log(\sigma)$ head because we found that having the context affect the policy's stochasticity did not improve performance. The action head, however, contains our adapter module, followed by a single linear layer mapping a 256-dimensional state to the mean action μ. This adapter consists of one hidden layer of size 32, and is generated by a chunked hypernetwork which has two hidden layers of 37 nodes each. Each chunk has a size of 370 and the hypernetwork outputs 46 chunks to obtain the entire set of parameters for the adapter. As mentioned in Section 4.2, these values were chosen such that the hypernetwork has fewer parameters than the adapter network. The critic has a very similar architecture to the actor, just without the $\log(\sigma)$ head. Finally, for the actor, we do not have an activation function after the trunk, as this improved performance slightly. For the hypernetwork component of our method, we use the `hypnettorch` package (von Oswald *et al.* 2020; Ehret *et al.* 2021; Henning *et al.* 2021).[1]

While we did experiment with several hyperparameter settings for the Decision Adapter, most options performed comparably to our chosen configuration, and we found that our method is quite robust when changing hyperparameters (see Section 6.2 for more details). While we generally used the default hyperparameters for our baselines, we did perform some tuning on the ODE domain. For instance, we compared two versions of the Unaware and Concat models that had the same number of parameters – one with three hidden layers and one with two – and chose the best. Similarly, for FLAP, we ensured that the network size we use performed comparably to the larger network used in the original work, and found that using the context to generate the weights outperformed using a single transition tuple. For all methods, the hyperparameters that performed the best in the ODE domain were used in the other environments.

5.5 Summary

In this chapter, we detailed our experimental procedure and defined the primary metric we use to evaluate the generalisation performance of each model. We further discussed the three environments we use – ODE, CartPole and Ant – and our reasons for using them. Finally, we described the baseline algorithms we compare against and discussed their implementation details. Now, in Chapter 6, we present the results of our experiments, and discuss the implications of our findings.

[1] https://github.com/chrhenning/hypnettorch

Chapter 6

Results

This chapter describes our experiments, details our results and discusses the conclusions we can draw from these results. We structure this chapter around answering three primary questions. The first is **Does the Decision Adapter generalise better than our baselines?** By answering this question, we can validate the contribution of our Decision Adapter. We address this question in Section 6.1 by experimenting on the three environments discussed in Section 5.3. First, in Section 6.1.1, we consider the ODE environment, where context is necessary to perform well. Next, in Section 6.1.2, we use the CartPole domain, where context is not required, and an unaware policy can generalise well. Lastly, in Section 6.1.3, we consider Ant, which is a more complex, challenging and real-world environment.

The second question we ask is **How sensitive is the Decision Adapter to its configuration options?** These configuration options correspond to the design decisions we made in Sections 4.2.1 and 5.4.3. Answering this question will allow us to (1) justify our default options; (2) make better recommendations about how to configure the Decision Adapter; and (3) determine how useful our model will be in practice, with minimal tuning of these options. We answer this question in Section 6.2, by comparing the performance of the default design decisions against alternative options. In this section, we also apply our approach to another popular RL algorithm, Proximal Policy Optimisation (Schulman *et al.* 2017), to illustrate the general applicability of the Decision Adapter.

Our third question, which we answer in Section 6.3, is **What are the limitations and failure cases of the Decision Adapter?** This question is important as it discusses potential pitfalls of our method, i.e., where care must be taken when using it in practice. In particular, we investigate the effect of incorrectly normalising the context in Section 6.3.1. In Section 6.3.2, we also consider our method's sensitivity to receiving an inaccurate context instead of the ground truth. We then examine the effect of changing the set of training contexts and consider the problem of overfitting in Section 6.3.3. Finally, Section 6.4 concludes and summarises this chapter with a discussion about our results and the conclusions we can draw from our experiments.

6.1 Does the Decision Adapter Generalise Better than our Baselines?

6.1.1 When Context is Necessary

In this section, we consider the ODE environment. Our aim is to show that (1) context is necessary in this environment, i.e., that the Unaware model is not able to generalise; and (2) that our adapter-based approach generalises better than our baselines. In this experiment, we consider a training context set of $\{-5, -1, 1, 5\}$ and train for 300k timesteps. To measure generalisation, we use an evaluation context range consisting of 201 equally spaced points between -10 and 10 (inclusive). This allows us to measure how well each method generalises across a wide range of contexts.

In Figure 6.1, we plot the average performance across the entire evaluation range (as discussed in Section 5.2) at different points during training. Firstly, we can see that the Unaware model fails to generalise well, since the optimal actions in the same state for two different contexts may be completely different. This result substantiates our theoretical analysis in Section 5.3, highlighting the importance of using context in this domain. Concat, `cGate` and FLAP perform reasonably well, but our Decision Adapter outperforms all methods and converges rapidly.

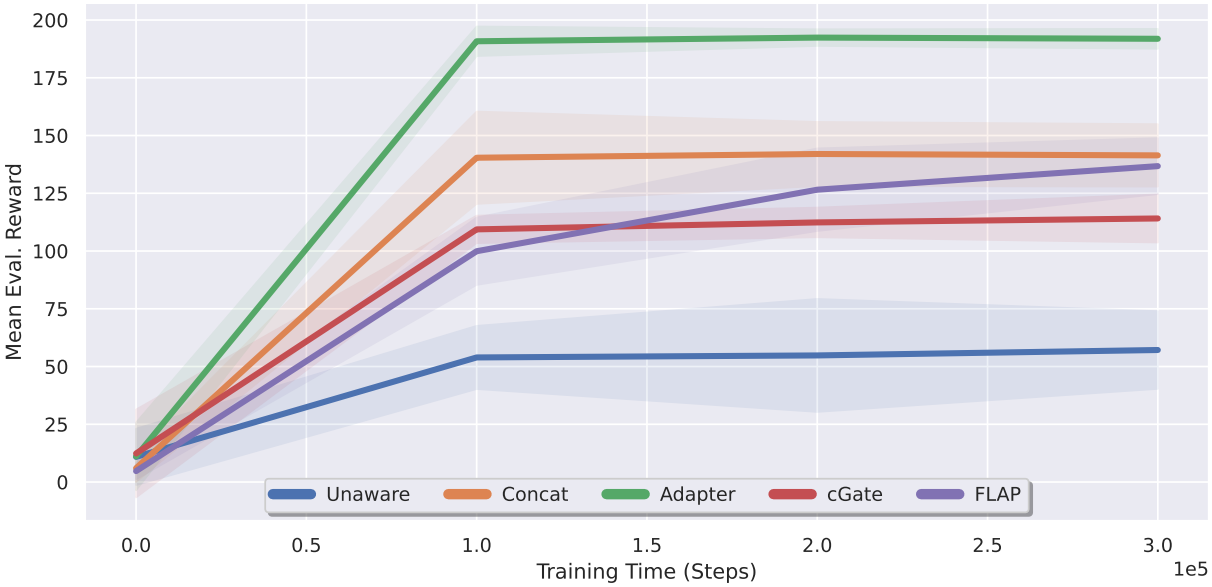


Figure 6.1: Showing the average reward over the evaluation range $[-10, 10]$ at various points during training for the ODE domain. Mean performance is shown, with standard deviation (across 16 seeds) shaded.

Interpolation vs. Extrapolation

We now examine these results in more detail by separately plotting the following three subsets of evaluation contexts in Figure 6.2:

Train This is the reward obtained on the training contexts, i.e., $\{-5, -1, 1, 5\}$, shown in the left pane of Figure 6.2. In particular, this considers all contexts within a Euclidean distance of $\epsilon = 0.05$ of these training contexts.

Interpolation The reward when interpolating, i.e., the evaluation contexts between two of the training contexts, consisting of the region $[-5, 5]$ (center).

Extrapolation Extrapolation refers to the evaluation contexts outside the context hull of the training contexts, containing $c \in [-10, -5) \cup (5, 10]$ (right).

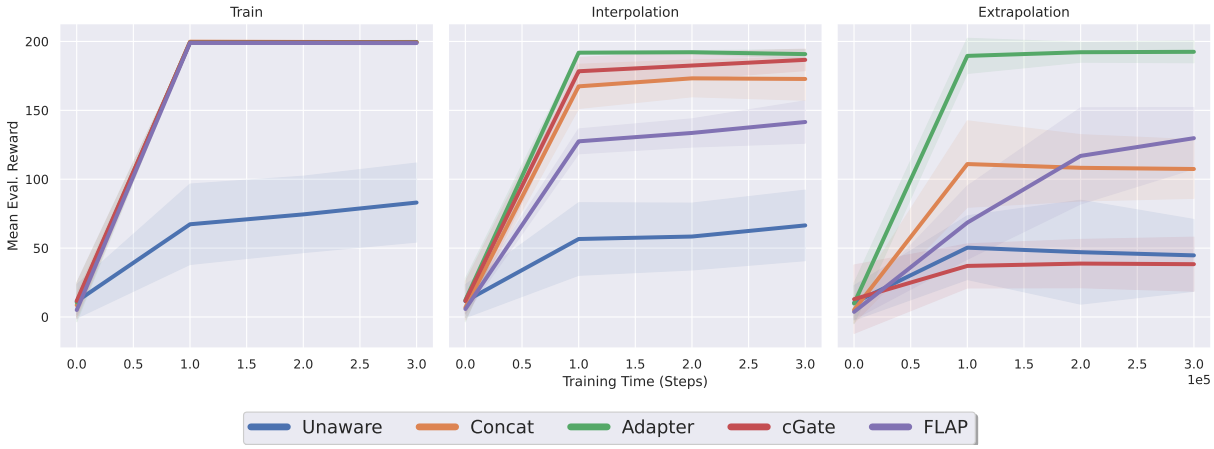


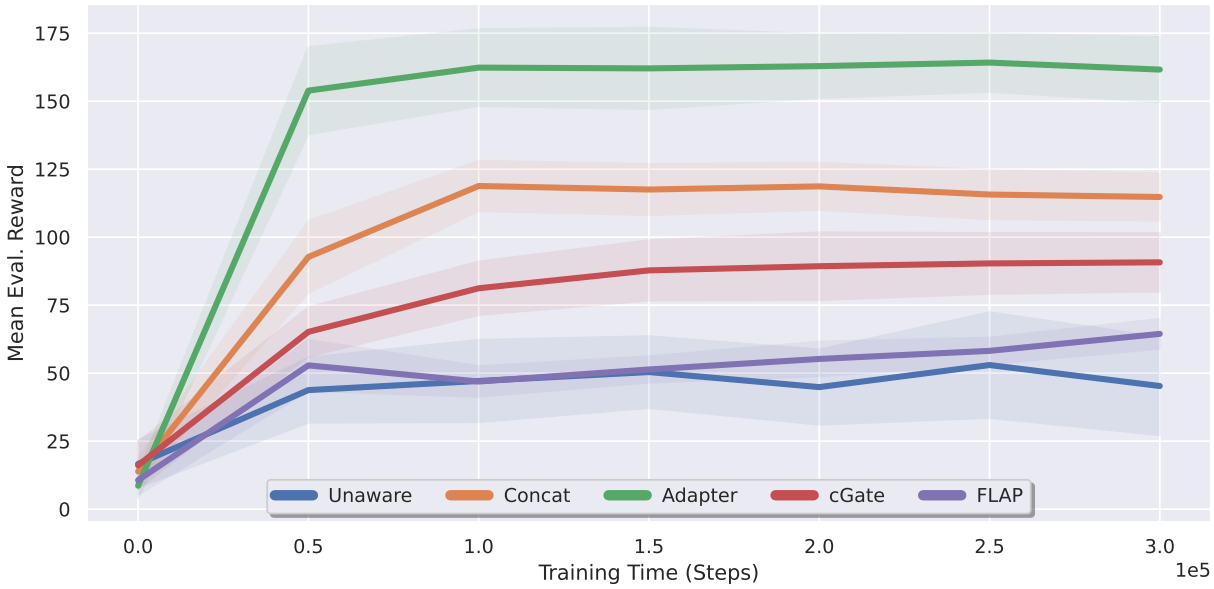
Figure 6.2: Showing the performance of each method in three different evaluation regimes, (left) the training contexts, (center) interpolation and (right) extrapolation. Mean performance is shown, with standard deviation shaded.

All context-aware methods perform close to optimal on the training contexts. The Unaware model, however, does not, explained by the fact that it cannot simultaneously perform well on negative and positive contexts. For interpolation, our Adapter performs comparably to `cGate`, which outperforms the Concat model. The reason for the Concat model losing some performance after training for a long time may be due to it overfitting to the training contexts. Finally, when considering only extrapolation, the Adapter outperforms all methods, and converges rapidly. FLAP performs better than the Concat model, and `cGate` exhibits poor extrapolation.

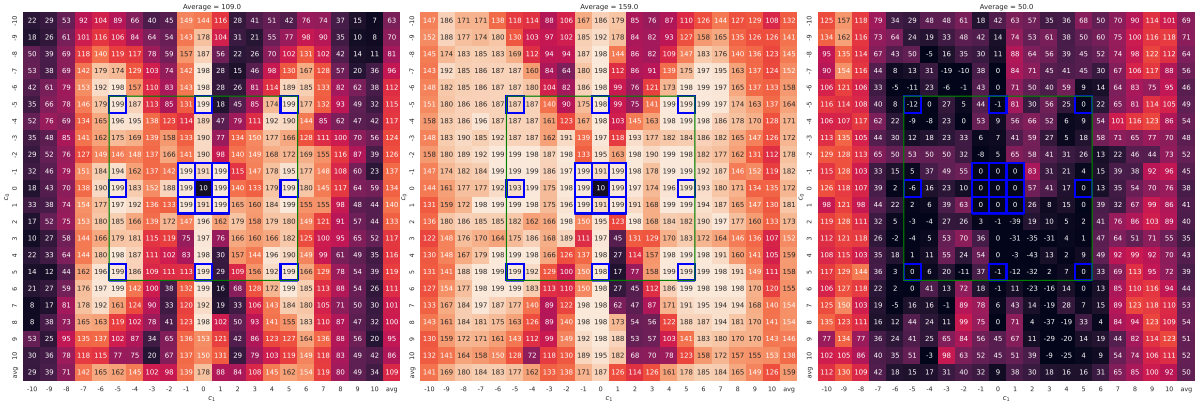
Multidimensional Contexts

We next show that these results hold even when the context is multidimensional in the ODE environment. We train on the context set given by $\{1, 0, -1\}^2 \cup \{5, 0, -5\}^2$, with the context $(0, 0)$ being omitted due to it being unsolvable.[1] We again train for 300k timesteps, as in Figure 6.1. The evaluation range consists of the cartesian product A^2, where A contains 21 equally-spaced points between -10 and 10. These results are shown in Figure 6.3.

[1]Here, for a set S, $S^2 = S \times S = \{(a, b) | a, b \in S\}$ denotes the cartesian product of S with itself.



(a)



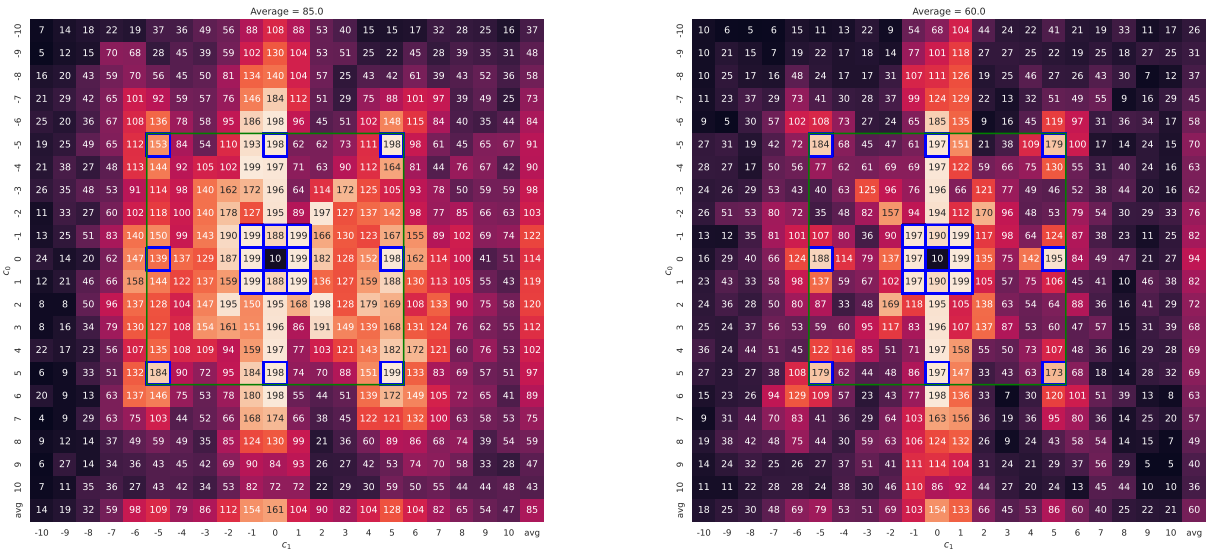
(b) Concat (c) Adapter (d) Subtracting (b) from (c)

Figure 6.3: Multidimensional contexts in the ODE domain. In (a) we plot the aggregate performance across the entire evaluation range. In (b) and (c) we plot the performance on each context in the evaluation set for the Concat and Adapter models, respectively, at the end of training. Each cell represents the reward obtained when evaluated in the corresponding context, averaged over 16 seeds, with lighter colours indicating higher reward and darker colours indicating lower reward. The blue squares indicate the specific training contexts, whereas the green square represents the convex hull of training contexts. For the heatmaps, the last rows and columns, labelled *avg*, represent the average reward over the particular column or row, respectively. In (d), we plot (c) − (d), allowing us to see the relative improvement of the Adapter over the Concat model.

Our Decision Adapter outperforms all baselines and, as can be seen in Figures 6.3b, 6.3c and 6.3d, generalises further away from the training contexts than the Concat model. In particular, these heatmaps show that both models perform well on the training contexts (the blue squares). However, the Adapter generalises much better, especially to contexts far outside the training range. For example, when $|c_1| \in [8, 10]$ (corresponding to the far left and far right of the heatmaps), the Adapter's rewards are higher than the

Concat model's, as indicated by the positive differences in Figure 6.3d. The Adapter obtains between 50 and 150 more reward than the Concat model – which is a substantial difference, as the maximum reward in this environment is 200. This region of context-space is far outside of the convex hull of the training range, which contains values of $|c_1|$ only up to 5.0. Closer to the training range; for instance, around the boundary of the green square, the Adapter still outperforms the Concat model, but the difference is less pronounced.

FLAP and `cGate` perform poorly, even though they obtain near-perfect rewards on the training contexts (as can be seen in Figure 6.4). There may be several reasons for this. First, here we train on 16 relatively sparsely distributed contexts, with 300k steps in total, resulting in fewer episodes in each context compared to Figure 6.1. Second, with two dimensions, the interpolation range makes up a smaller proportion of the evaluation range compared to the one-dimensional case. This is because, in one dimension, the range $[-5, 5]$ makes up half of the $[-10, 10]$ evaluation range. In two dimensions, however, the interpolation range makes up only a quarter of the evaluation set. Thus, the overall average performance in Figure 6.3a is more skewed towards extrapolation. This, coupled with the fact that both `cGate` and FLAP perform poorly on extrapolation (Figure 6.2, right), may explain why they underperform in this case. Lastly, it is generally easier to generalise in the one-dimensional ODE, as the optimal action depends mainly on the signs of the context and state variables. By contrast, the multidimensional case is more difficult, as the dynamics equation includes a nonlinear action term. This is also why, for instance, our Decision Adapter performs worse in this case (around 160 overall evaluation reward) compared to the one-dimensional ODE (around 190).



(a) `cGate` (b) FLAP

Figure 6.4: Heatmap Performance for (a) `cGate` and (b) FLAP. These plots have the same meaning as Figures 6.3b and 6.3c. In particular, the blue squares indicate the training contexts and the green squares indicate the convex hull of training contexts.

6.1.2 When Context is not Necessary, but Beneficial

In our next experiment, we consider the widely-used CartPole domain. First, we demonstrate that, in contrast to the ODE, context is not explicitly required, and that some unaware policies can generalise well. Next, we show that incorporating contextual information is not detrimental to generalisation performance in these cases. After this, we show that using context can, in fact, be beneficial.

Demonstrating that Context is not Necessary

Here we demonstrate that, in CartPole, there exists some context-unaware behaviour that performs well on many different contexts. To do so, we consider each of the five context dimensions (e.g., gravity, pole length, etc.) separately. For each dimension, we wish to measure how well various Unaware models generalise when varying only this dimension. We do this by training Unaware models on specific values of this dimension and evaluating how well these models generalise. In particular, for each factor in Table 6.1, we have six separate training settings. Five of these are single training contexts, ranging from "`Small`" to "`X-Large`". The sixth training setting is a set consisting of $\{\texttt{Small}, \texttt{Medium}, \texttt{Default}, \texttt{Large}, \texttt{X-Large}\}$. For instance, when considering gravity, we train six unaware models, one with weak gravity, one with medium gravity, one with high gravity, etc. Then, we evaluate how well each of these models generalises when varying gravity. This allows us to determine whether there is some context-unaware behaviour that can generalise across a wide range of values for this context dimension.

The specific context values were obtained by taking the default setting of this parameter v_d, and setting `Small` to be $\frac{v_d}{10}$ and `X-Large` to be $10v_d$, with `Medium` and `Large` in-between. The evaluation range consists of 30 equally-spaced points (on a logarithmic scale) between $\frac{v_d}{20}$ and $20v_d$. Thus, for each context variable, we fix all of the other variables at their default values, and train separate unaware models on each of the aforementioned training sets. In total, we have 30 Unaware models, corresponding to six models for each of the five context variables. These models are then evaluated on how well they generalise when varying the corresponding context variable.

Table 6.1: The different training contexts for each variable that we consider.

	Gravity	Cart Mass	Pole Mass	Pole Length	Force Magnitude
`Small`	0.98	0.10	0.01	0.05	1.00
`Medium`	3.10	0.32	0.03	0.16	3.16
`Default`	9.80	1.00	0.10	0.50	10.00
`Large`	30.99	3.16	0.32	1.58	31.62
`X-Large`	98.00	10.00	1.00	5.00	100.00

The performance of Unaware models on these contexts is shown in Figure 6.5. In most cases, the model trained on the `X-Large` setting (which generally corresponds to more difficult cases) generalises nearly perfectly when varying the corresponding context dimension. However, the models trained on the other training settings perform

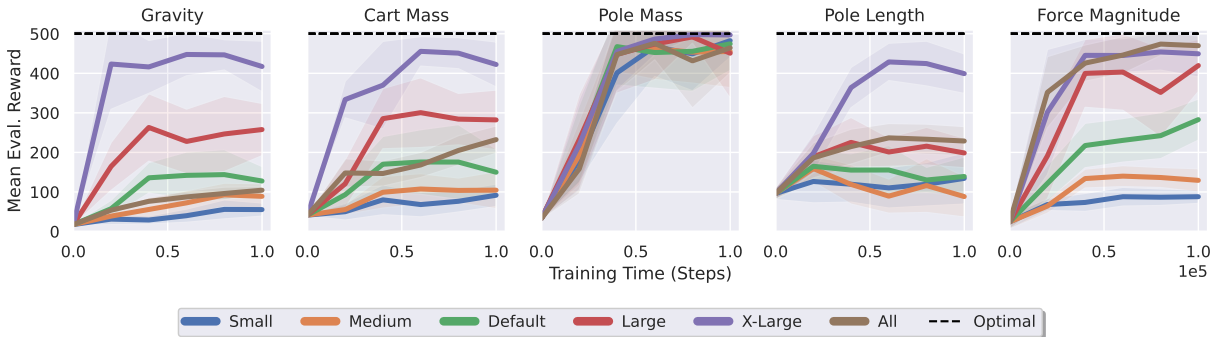


Figure 6.5: Comparing the performance of Unaware models on each context variable in CartPole. The dotted line indicates optimal performance, corresponding to perfect generalisation across the entire evaluation range. In each plot, only the variable listed in the title differs for each coloured line, with all others being left at their default value during training. Furthermore, each plot only measures generalisation with respect to a single context dimension. For instance, the *Gravity* plot measures generalisation when varying gravity. Mean performance is shown, with standard deviation shaded.

much worse for most of the context variables. Interestingly, the models trained on the entire range of contexts perform poorly when varying gravity, the cart's mass and the pole length. This may be due to interference, where the learning signal differs for each context, preventing the model from learning a useful policy (Kessler *et al.* 2022). Finally, the pole's mass is a variable that does not greatly affect the dynamics, as all models generalise equally well. From these results, we can see that, for each context variable in CartPole, there is an unaware behaviour that generalises well when varying this variable. However, we note that it is non-trivial to choose the appropriate training context that leads to strong generalisation, with most options resulting in poor performance.

Demonstrating that Context is not Detrimental

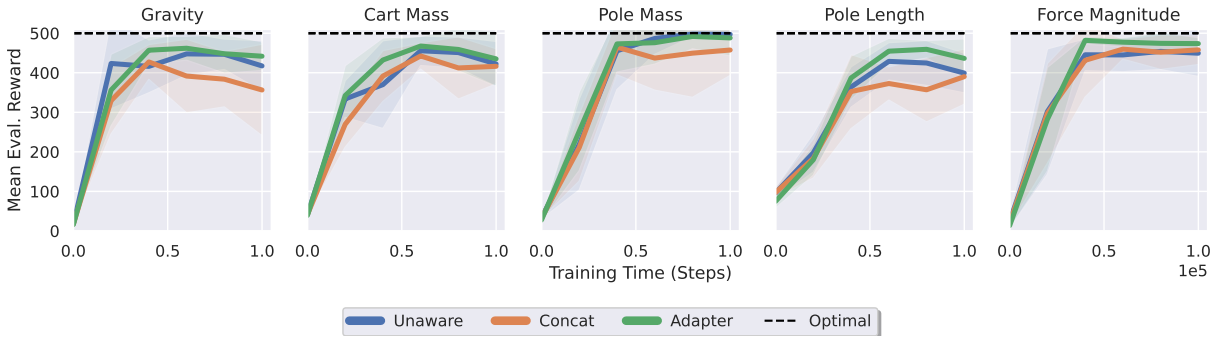


Figure 6.6: Training the Unaware, Concat and Adapter models on the `X-Large` context, as listed in Table 6.1. Mean performance is shown, with standard deviation shaded.

Before we present our primary experimental results for CartPole, we briefly note that, when training on the `X-Large` setting for each context variable, the Adapter and Concat

models perform comparably to the Unaware model. This is shown in Figure 6.6. Thus, in this case, being conditioned on context does not harm the agent's generalisation.

Context Improves Sample Efficiency

Having shown that there exist context-unaware policies that can generalise well in CartPole (even though they require correctly choosing an appropriate training context), we now investigate the performance of context-adaptive approaches in this domain. In particular, we vary only the pole length, and consider the training contexts as $\{1, 4, 6\}$, with the evaluation contexts being 301 equally-spaced points between 0.1 and 10. We choose pole length as it was the variable where the `X-Large` training setting vastly outperformed all other training contexts, indicating that this variable greatly affects the dynamics.

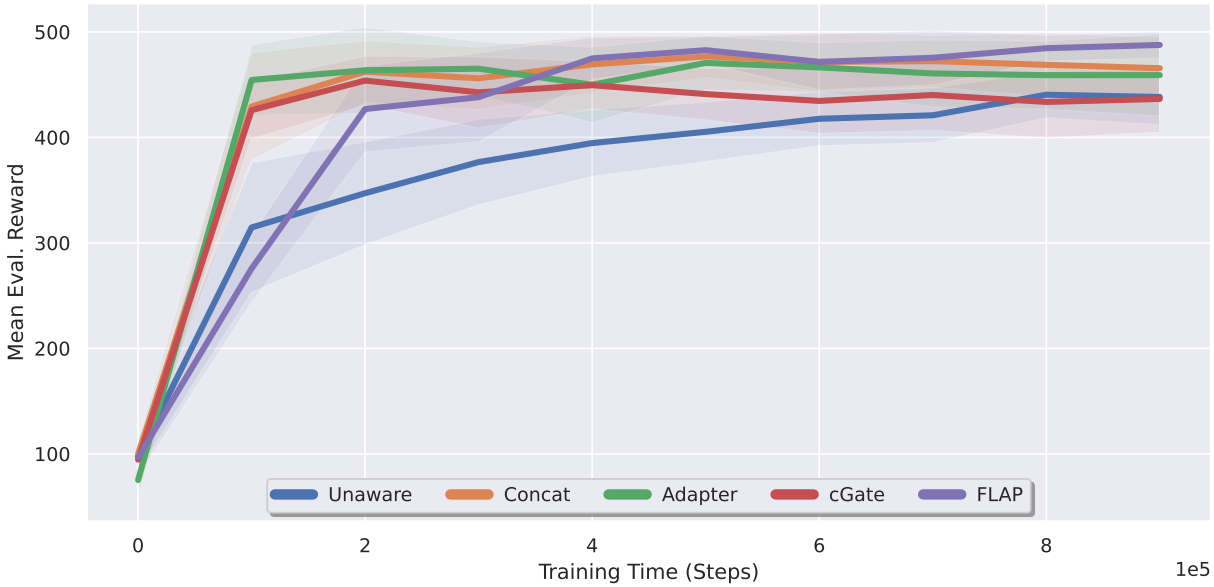


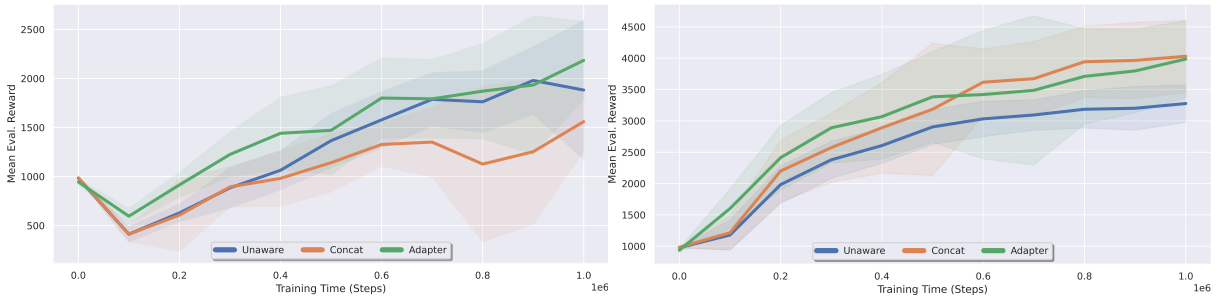
Figure 6.7: Showing the average evaluation reward as we increase training time for CartPole. Mean performance over seeds is shown and standard deviation is shaded.

These results are shown in Figure 6.7. While the Unaware model can achieve comparable generalisation performance to the context-informed models, it takes significantly longer to do so. In particular, to obtain an average reward above 400, the Unaware model must train for more than 600k steps, whereas `cGate`, Concat and our Adapter reach this threshold before 100k steps. Thus, while context is unnecessary, incorporating it may be beneficial and can lead to much more sample-efficient generalisation compared to the Unaware model. This is particularly relevant when training on multiple training contexts, which seems to cause interference for the Unaware model (as seen above in Figure 6.5). The context-aware approaches, however, are not as susceptible to this problem, as they are able to distinguish experience from different contexts. Furthermore, most of the context-aware models perform similarly in this case. In particular, our Adapter, Concat and `cGate` converge rapidly, while FLAP converges slightly slower, but achieves comparable generalisation performance at the end of training.

There is no statistically significant difference in final performance between the Adapter and each of the other models, using a MWU test with a significance level of 0.05.

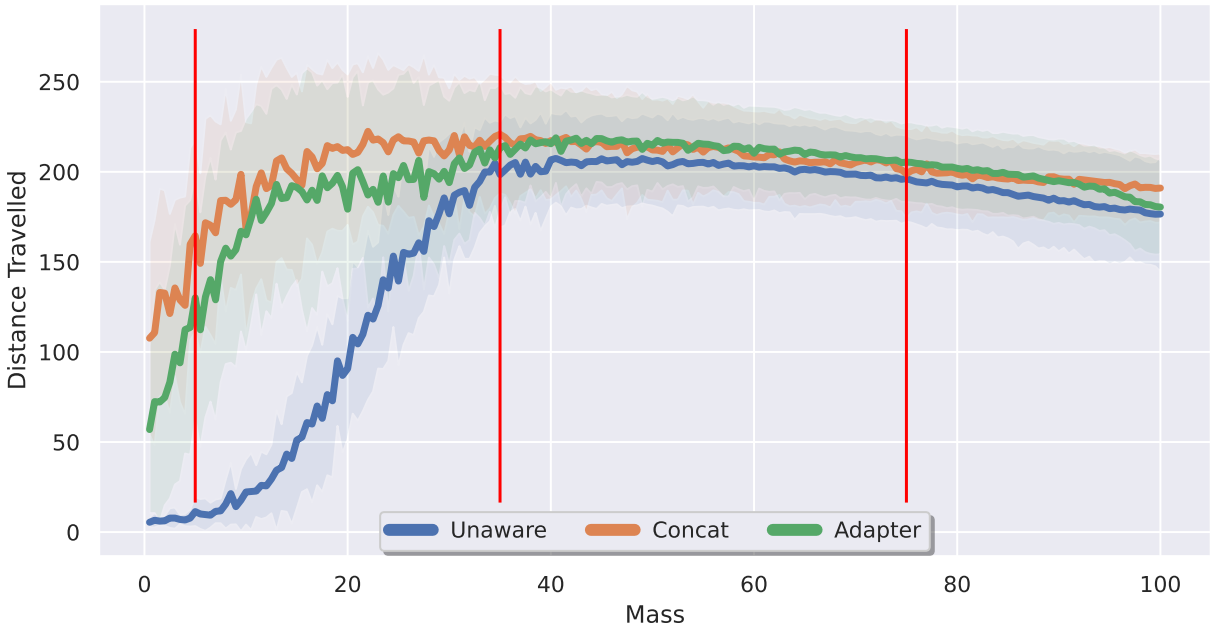
6.1.3 A More Complex Environment

In this section, we consider a more complex and challenging continuous-control environment, that of the Mujoco Ant. We consider two settings here; first, we vary the friction between the ant's legs and the floor. Here we train on friction coefficients of 1.0 and 3.0 (where 1.0 is the default), and evaluate on 24 evenly spaced contexts between 0.2 and 4.8. In the second setting, we change the mass of the ant's body (which corresponds to 5 by default), training on the set $\{5, 35, 75\}$ and evaluating on 200 evenly spaced points between 0.5 and 100.



(a) Varying Friction

(b) Varying Mass



(c) Plotting Mass vs. Distance Travelled, at the end of training, for those models in Figure 6.8b. The red lines indicate the training contexts.

Figure 6.8: Illustrating the performance on the Ant domain, where we vary (a) friction or (b, c) mass. Mean performance is shown, with standard deviation shaded. The rewards in (a) and (b) are different, as increasing friction generally results in a more challenging task and thus a lower reward. In (c), we plot the mass vs. the reward obtained when evaluating on each mass (context).

For baselines, we consider only the Concat and Unaware models. The reasons for this are that (1) the Concat model performed comparably to or better than the other baselines in the previous sections, making it a representative example of the context-aware approaches; (2) the Concat model is currently a very prevalent way of incorporating context (Ghasemipour *et al.* 2019; Rakelly *et al.* 2019; Ball *et al.* 2021; Eghbal-zadeh *et al.* 2021; Sodhani *et al.* 2021, 2022; Mu *et al.* 2022); and (3) the Unaware model demonstrates whether context is necessary or not.

Figure 6.8 shows these results. Firstly, when varying friction (in Figure 6.8a), the Adapter outperforms the Unaware model initially, and they both obtain roughly the same performance at the end of training, with no statistically significant difference. The Concat model, however, performs worse, with a statistically significant difference compared to the Adapter, $p < 0.05$. In Figure 6.8b, the Adapter and Concat models perform similarly, but the Unaware agent does not generalise as well as the context-aware approaches. Notably, there is a statistically significant difference between the performance of the Adapter and Unaware models. This performance gap is due to the Unaware model being unable to simultaneously perform well on light and heavy masses, as seen in Figure 6.8c. In contrast, by utilising context, the agent can perform well in both settings despite their significant differences.

Thus, in this domain, some context variables do not affect the dynamics significantly, resulting in the Unaware model performing well. Other variables, such as the mass, have a significant effect, and the Concat model vastly outperforms Unaware. In both cases, the Adapter performs comparably to or better than the best approach.

6.1.4 Real World Showcase

As a final qualitative showcase, we apply our Decision Adapter to a physical robot locomotion task. In particular, we consider the *RealAnt* quadruped robot (Boney *et al.* 2020), shown in Figure 6.9. This robot is similar, but slightly different, to the ant used in Section 6.1.3. We consider the context as a three-dimensional vector consisting of *Mass*, *Friction* and *Slope Angle*. *Slope Angle* refers to the incline or decline of the surface that the agent walked on. We train the policies in simulation[2] for 300k steps on the following collection of contexts, using a normalisation value of $(2.5, 3.5, 2)$:

$$\{(1, 1.5, 0), (2, 1.5, 0), (1, 3.5, 0), (2, 3.5, 0), (1, 1.5, 2), (1, 1.5, -2)\}$$

In the simulation, we add noise as recommended by Boney *et al.* (2020). In particular, we add Gaussian noise sampled from $\mathcal{N}(0, 0.02)$ to the body's position and orientation. Further, we use a latency of 2 steps and give the agent the 6 latest observations, stacked into one vector. We then transfer these policies to reality as follows:

- Set up the simulation using a context approximately similar to the real-world case.

[2]We use the Mujoco-based simulation from here: https://github.com/AaltoVision/realant-rl/



(a) The unencumbered robot



(b) The robot with an additional mass

Figure 6.9: Showing the RealAnt robot we use for this section.

- Apply the policy's action to the simulation at every step.
- Set the real robot's joint positions to be the same as the simulator.

We follow this approach instead of using state information from the physical robot for two reasons. First, the camera used for pose estimation suffered from significant motion blur and other inaccuracies, frequently leading to significant errors in the inferred state. Second, the semantic meaning of the actions differs slightly between the simulator and reality, which we circumvent by simply setting the real robot's joint positions to be the same as the simulator at each step. While there may be some drift between the simulator and reality, this was not a significant problem as our rollouts were generally quite short (around 15 seconds). Due to the time-consuming nature of real-world experiments, we consider only the Unaware model as a baseline.

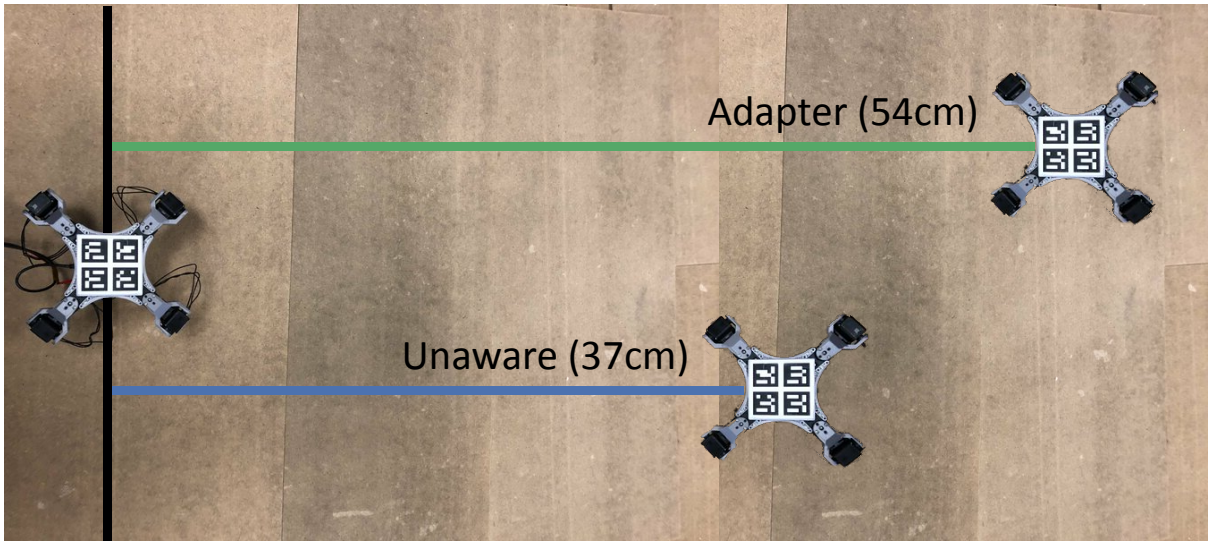


Figure 6.10: An illustration of the results of the Adapter compared to the Unaware model, walking on a rough surface without any additional mass. The relative lengths of the lines are accurate, but the absolute lengths are not to scale.

In each setting, we run the policy five times and measure the overall distance travelled in each rollout. In each case, the simulator is parametrised by an approximate context that corresponds to the real-world case. For instance, the additional mass weighs about twice as much as the ant. Thus, the default mass is 1.0 and the overall mass with the additional weight is set to 3.0. We do note that the real robot's friction characteristics differ slightly from the simulator, possibly due to the simulated feet being larger than the real feet. Thus, for the surface we used, we set the simulation friction as 1.5, but give the Adapter module a friction input of 0.5. This resulted in the robot slipping less, and demonstrates that the Adapter's ability to learn separate behaviours can be useful. Some of our results are shown in Table 6.2, with an illustration in Figure 6.10. Overall, the Adapter was able to perform well in several cases, whereas the Unaware model struggled. In particular, we noticed that the Unaware model often veered from its path, causing it to bump into the walls surrounding our experimental area. By contrast, the Adapter module walked in a straight line more often. We emphasise that these results are not meant to be a rigorous evaluation of the Decision Adapter, but rather a qualitative showcase of its potential in a real-world robotics setting.[3]

Table 6.2: Results for the different settings we considered in the RealAnt experiments. We present the $mean \pm standard\ deviation$ of the distance travelled in centimetres and mark the best performance in **bold**. A negative slope indicates an uphill surface. All of these experiments used the same surface, so the friction did not change.

Name	Mass	Friction	Slope	Adapter distance	Unaware distance
Default	1	1.5	0	$\mathbf{54.4} \pm 6.7$	37.4 ± 1.4
Heavy	3	1.5	0	$\mathbf{59.2} \pm 1.6$	50.1 ± 5.7
Incline	1	1.5	-4	40.6 ± 2.9	$\mathbf{44.8} \pm 1.4$
Heavy Incline	3	1.5	-4	$\mathbf{53.6} \pm 3.2$	47.8 ± 0.7

6.2 How Sensitive is the Decision Adapter to its Configuration Options?

Here we experimentally justify some of the design decisions we made in Section 4.2.1. The experiments in this section also give us insight into how robust the Decision Adapter is to changing its configuration options. First, in Section 6.2.1, we investigate if our Adapter can successfully be applied to a different RL algorithm. We find that **The Decision Adapter can generalise well when using PPO.**

Thereafter, we investigate the effects of the following factors, with our conclusions listed in **bold**.

[3]Video demonstrations are available at https://sites.google.com/view/decisionadapter.

- The network architecture of the Adapter module. We find that **Most adapter architectures perform well.** This result is expanded upon in Section 6.2.2.
- Whether the hypernetwork uses chunking. **Hypernetwork chunking outperforms non-chunking methods while using significantly fewer parameters** (Section 6.2.3).
- Using a skip connection. **The Decision Adapter can perform well with or without a skip connection** (Section 6.2.4).
- The location of the adapter module in the main network. **Most locations lead to high performance, except if the adapter is at the very start or very end of the network** (Section 6.2.5).
- Having an activation function before the adapter. **Not having an activation function before the adapter in the actor model outperforms the alternatives** (Section 6.2.6).

For this entire section, we consider the ODE environment and follow the same procedure as in Section 6.1.1. We use the ODE due to its fast training, and the fact that context is necessary, which enables us to determine if a particular design choice of the Adapter results in poor use of the context. Furthermore, we consider the null hypothesis to be that the Adapter (Section 6.2.1) or its base variation (Sections 6.2.2 – 6.2.6) performs comparably to the alternatives at the end of training. As mentioned in Section 5.2.1, we use a two-sided Mann-Whitney U test with $\alpha = 0.05$.

6.2.1 Applying the Decision Adapter to another RL Algorithm

The first experiment in this section demonstrates that our Adapter architecture is indeed agnostic to the RL algorithm used. In particular, we apply our model to Proximal Policy Optimisation (Schulman *et al.* 2017, PPO), a widely-used on-policy actor-critic method (Packer *et al.* 2018; Spitznagel *et al.* 2021). We again use the CleanRL implementation (Huang *et al.* 2022) and consider the Unaware and Concat models as baselines. Furthermore, we train for more steps than SAC, since PPO is generally less sample efficient, partially because it is an on-policy algorithm (Haarnoja *et al.* 2018).

The results are shown in Figure 6.11. Our Adapter still outperforms the Unaware model, and there is no statistically significant difference between the Adapter model and the Concat baseline at the end of training. We note that here, both the Unaware and Concat models used a three-layer network architecture of 204 nodes, as this outperformed a shallower but wider network with an equal number of parameters.

Overall, PPO performed slightly worse than SAC. This is not unexpected, as prior work has shown that SAC can outperform PPO in certain continuous-control tasks (Haarnoja *et al.* 2018). This may be due to SAC's maximum entropy objective that incentivises robustness (Eysenbach and Levine 2022). Furthermore, PPO generally results in more noise across seeds, indicating that its training is potentially less stable. This instability could also contribute to worse generalisation performance.

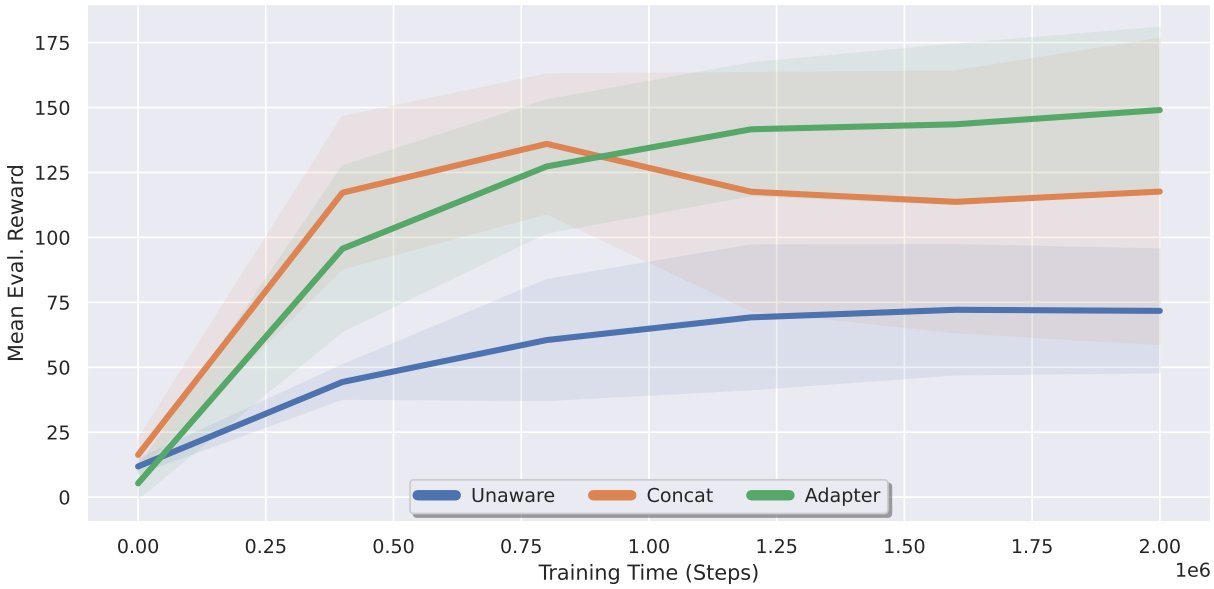


Figure 6.11: Illustrating the performance of our Decision Adapter when using PPO. Mean performance is shown, with standard deviation shaded.

6.2.2 Adapter Architecture

Here we examine the effects of changing the Adapter module's network architecture. We are interested in how the architecture impacts performance for two reasons. First, we wish to empirically justify our bottleneck architecture. Second, we wish to show that the Adapter performs well with a wide variety of network architectures, and is not particularly sensitive to this choice.

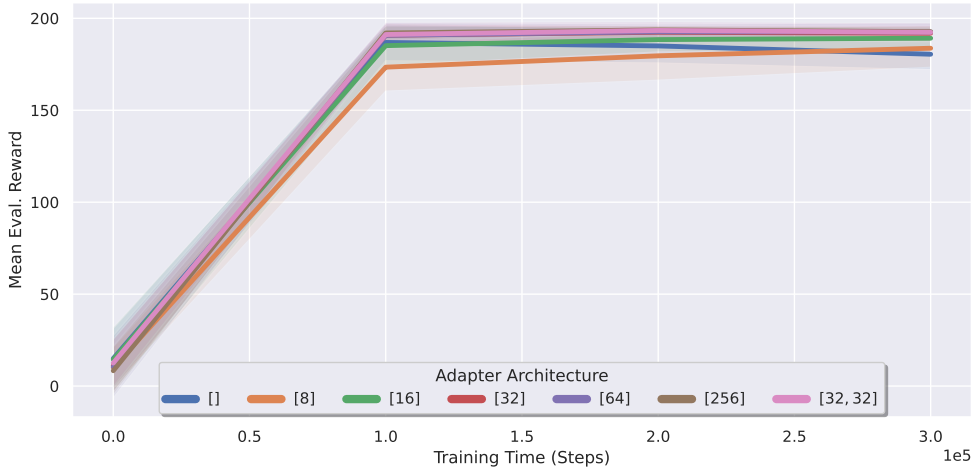


Figure 6.12: Evaluating the effect of different adapter architectures. Mean performance is shown, with standard deviation shaded.

As discussed in Section 4.2 and Section 5.4, we use a bottleneck architecture of size 32. Thus, the 256-dimensional features are transformed into 32-dimensional features and then projected back to 256 dimensions. We denote this architecture as $[32]$, since the adapter has a single hidden layer of 32 neurons. Figure 6.12 illustrates the performance

for various other adapter architectures. Most methods perform comparably, but using 8 hidden nodes or not having any hidden layers performs slightly worse than the base architecture. A large hidden layer of 256 nodes also does not outperform our bottleneck architecture. Finally, there is no particular benefit to having multiple layers in the adapter module. Our statistical analysis indicates that there is a statistically significant difference in performance between $[32]$ and both the $[8]$ and $[\,]$ configurations, with $p < 0.05$. Conversely, the differences between the $[32]$ bottleneck architecture and the other alternatives are not statistically significant.

6.2.3 Hypernetwork Chunking

Next, we compare our default architecture choice, which uses hypernetwork chunking, against non-chunking settings. We do this because hypernetwork chunking allows us to have significantly fewer learnable parameters (resulting in faster training) compared to the alternative of predicting the entire weight vector in one forward pass. Thus, if our approach performs comparably or better than non-chunking settings, then the chunking architecture is justified.

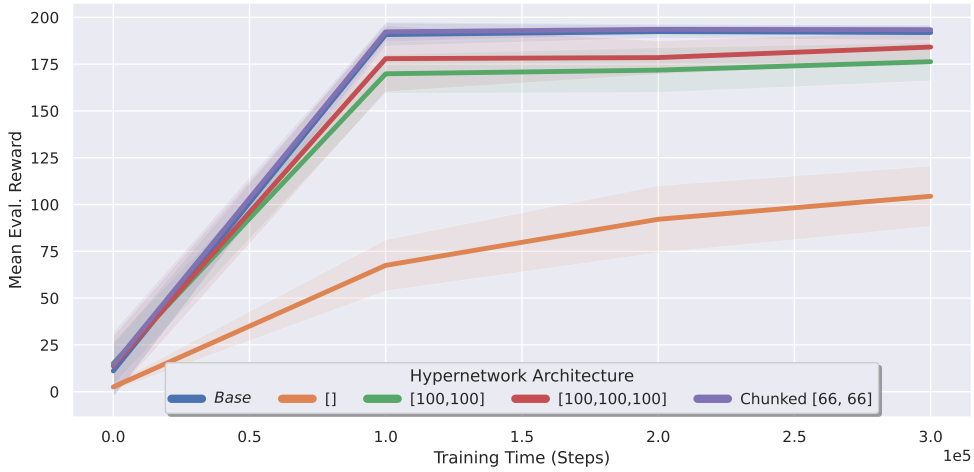


Figure 6.13: Comparing hypernetwork chunking ($Base$) compared to various non-chunking configurations as well as using larger chunks (Chunked$[66, 66]$). Mean performance is shown, with standard deviation shaded. Here, the values in the brackets represent the number of hidden nodes per layer, with $[\,]$ corresponding to a hypernetwork that has no hidden layers.

We consider a non-chunking hypernetwork architecture of no hidden layers, as well as 2 and 3 hidden layers of size 100, respectively. Our final option is a chunked hypernetwork, with chunks of size 660, and two hidden layers of size 66 each (this is roughly twice as large as our default configuration option discussed in Section 5.4.3). These results are shown in Figure 6.13. We find that both chunking configurations perform the best, with no statistically significant difference between the base and the larger chunking variation. The non-chunking approaches perform slightly worse, particularly if the hypernetwork has no hidden layers. The difference between the base model and each of the non-chunking options is statistically significant, with $p < 0.05$.

These non-chunking configurations also have significantly more learnable parameters than the default. For instance, the base setting results in the actor's hypernetwork having around 16k parameters, whereas the $[100, 100]$, non-chunking setting corresponds to roughly 1.7M parameters. In summary, by using hypernetwork chunking, we can obtain high performance while using a very small number of learnable parameters.

6.2.4 Skip Connection

Here we investigate whether a skip connection is beneficial or not. As mentioned in Section 4.2, the skip connection sets the updated x as $x = A(x) + x$, whereas if we remove it, the function becomes just $x = A(x)$. In Figure 6.14, we can see that using a skip connection performs similarly to the alternative, and we find no statistically significant difference in performance. This shows that the Decision Adapter is robust to this parameter, and can perform well with or without a skip connection.

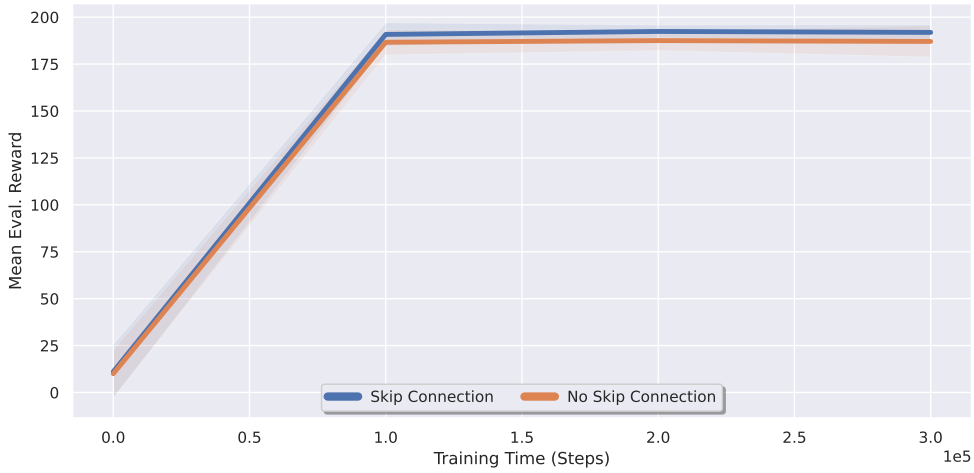


Figure 6.14: Evaluating the effect of using a skip connection. Mean performance is shown, with standard deviation shaded.

6.2.5 Adapter Location

We next examine the effects of changing the location of the adapter module in our network, and what happens if we have multiple modules. This information would be useful when using our method in practice, and deciding where and how many modules to place. There are generally many options regarding the location, as an adapter module can be placed between any two layers of the primary network.

As described in Section 5.4, our actor network consists of a "trunk", containing one hidden layer, which maps the state to a 256-dimensional feature vector, and a one-layer action head, directly mapping the trunk features to an action. Here we consider the following locations for the adapter in both the actor and critic networks:

Base The standard adapter used in the rest of this chapter, which is placed before the action head's layer. In the critic, the adapter is placed before the last layer. This corresponds to location (C) in Figure 6.15.

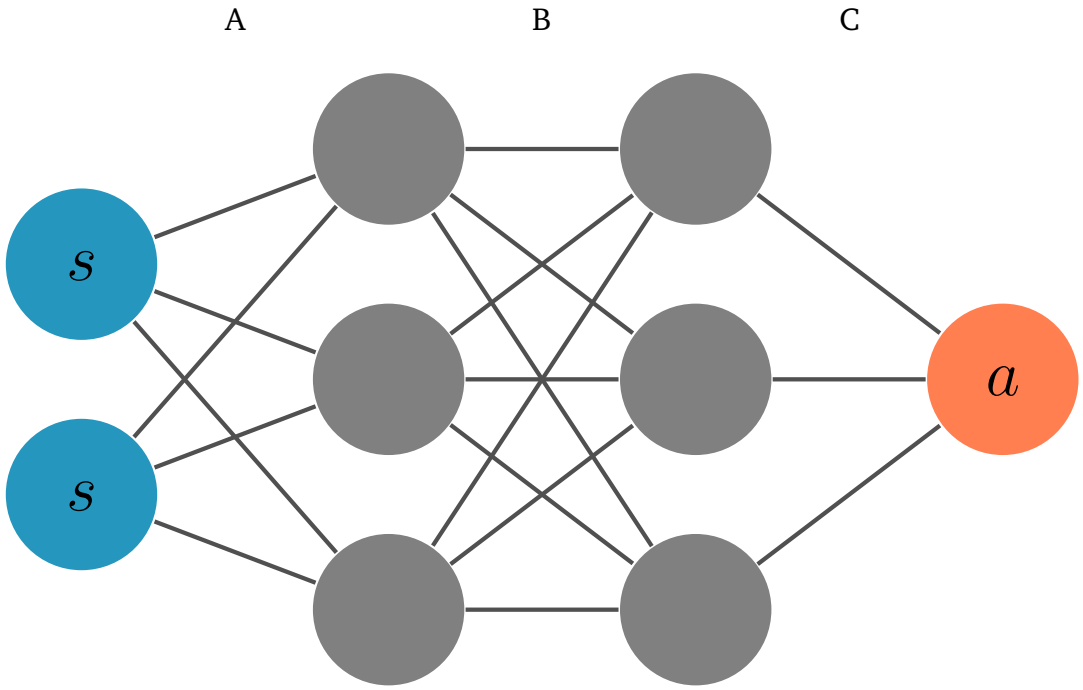


Figure 6.15: Showing the base network architecture, with letters indicating adapter positions. For instance, an adapter placed at location (D) receives the action as input and returns a modified one.

Base$_F$ Placing the adapter in the actor network before the final layer of the trunk, leaving the critic unchanged (location B).

Start The adapter here is at the start of both networks, i.e., the adapter module receives the state as input (location A).

End Here, we place the adapter at the very end. Thus, the adapter takes in the predicted action and outputs a modified one. The critic's adapter is also placed at the end. This corresponds to location (D) in Figure 6.15.

All$_F$ Three adapter modules, one at the start of the trunk, one before and one after its final layer. The critic also has three adapters, one at the start, one before the last layer and one after the last layer. The actor has adapters in locations (A), (B) and (C).

In all cases, to ensure comparability against the base model, there is no activation function after the trunk in the actor model. These results are illustrated in Figure 6.16. Overall, the base model performs the best, with statistically significant differences between the base and all other options. The adapter in the trunk performs slightly worse and the performance is quite poor if we put the adapter modules at the start of the networks. If we have three adapter modules, the performance is similar to only having one before the final layer. Finally, when adapting only the final action, the performance is also poor. Overall, the performance is low if we have the adapter only at the very

start or end of the network. Within a range of locations in the "middle" of the network, however, the performance is high and only slightly worse than the default setting.

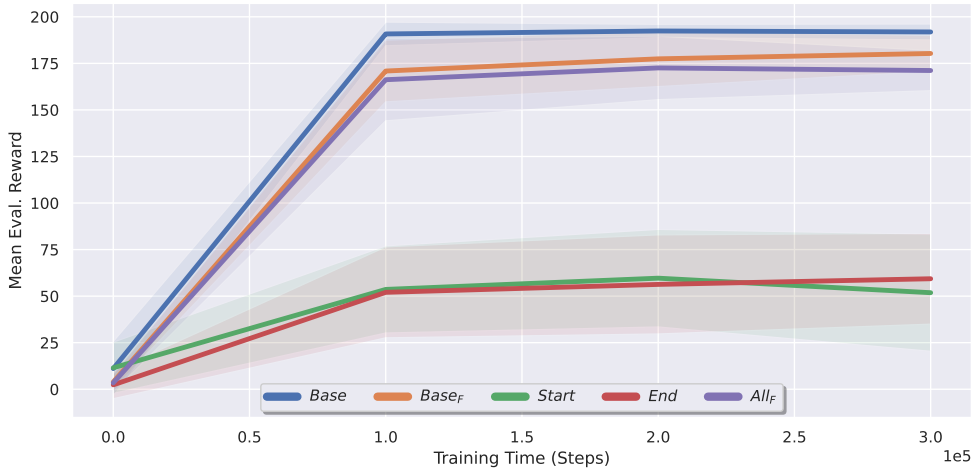


Figure 6.16: Evaluating the effect of the adapter module's location. Mean performance is shown, with standard deviation shaded.

6.2.6 Activation Function Before Adapter

Our final ablation considers the benefits of not having an activation function before the adapter module inside the actor network. We perform this experiment to determine if our choice of not having an activation function before the adapter in the actor is justified, and if it is necessary to do the same for the critic.

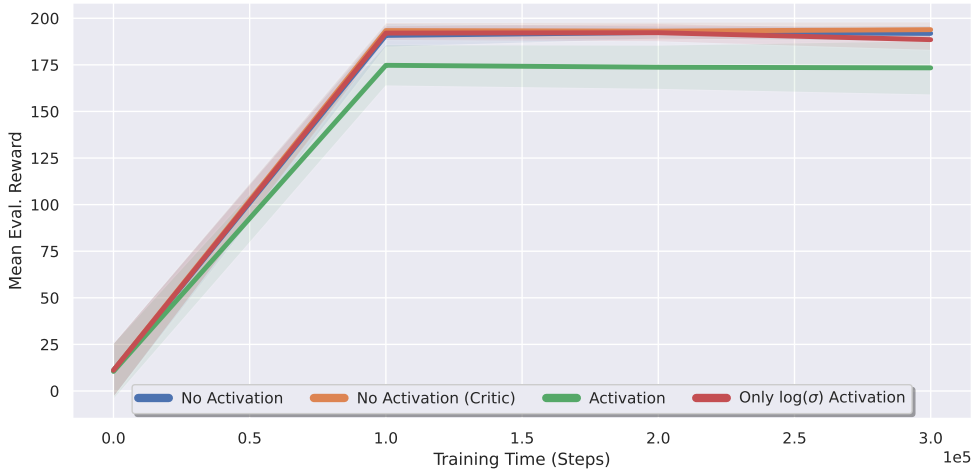


Figure 6.17: Comparing the effect of having an activation function before the adapter vs. not. The blue, green and red lines alter the actor's architecture, whereas orange changes the critic. Blue corresponds to the base choice outlined in Section 5.4.3. Mean performance is shown, with standard deviation shaded.

Here we consider four options: (1) not having an activation function after the trunk; (2) having one; (3) having an activation function after the trunk, but only for the standard

deviation head; and (4) not having an activation before the adapter in the critic. Option (1) corresponds to the default choice. These results are shown in Figure 6.17. Not having an activation function outperforms having one, and having an activation for only the standard deviation head is similar to not having one. While there is a statistically significant difference between the base option and having an activation function, there is no significant difference between the base option and the other alternatives. Thus, not having an activation function before the adapter inside the *critic* network performs as well as the base option, which has this activation. Overall, this result justifies our design choice by showing that it performs comparably to or better than the alternatives.

6.3 What are the Limitations and Failure Cases of the Decision Adapter?

Having shown that, in general, incorporating context is beneficial, and that the Decision Adapter outperforms the baselines, we now consider some counterpoints and failure cases. Identifying these failure cases is important, and would allow us to develop approaches that address these issues. In particular, we investigate how incorrectly normalising contexts affects the Adapter in Section 6.3.1. Next, we examine the effects of not receiving the exact ground truth context during evaluation in Section 6.3.2. We finally determine how much the exact set of training contexts matters, and how susceptible our method is to overfitting in Section 6.3.3. For each section, we are interested in how the change affects the Adapter and provide results for the Concat model for reference.

6.3.1 Suboptimal Context Normalisations

ODE

As discussed in Section 5.4.1, we first normalise the context before passing it to the agents. This is generally done by using the largest absolute value encountered during training. For instance, for the experiment in Section 6.1.1, we used $c_{norm} = \frac{c-0}{5}$ as the normalisation function. Here we determine how sensitive the Decision Adapter is to changes in this procedure. This is useful as we may often need to choose the normalisation factor without knowing exactly which variations we will encounter during evaluation. To do this, we run four additional experiments similar to the one in Section 6.1.1, with normalisation values of 0.1, 1.0, 2.0 and 15.0, respectively. The results are shown in Figure 6.18, and we can see that the Concat and Adapter models are both robust to the normalisation value. Our Decision Adapter loses less performance than the Concat model as we change the normalisation value. Despite losing some performance as the context normalisation value increases, both context-aware models still outperform the Unaware model.

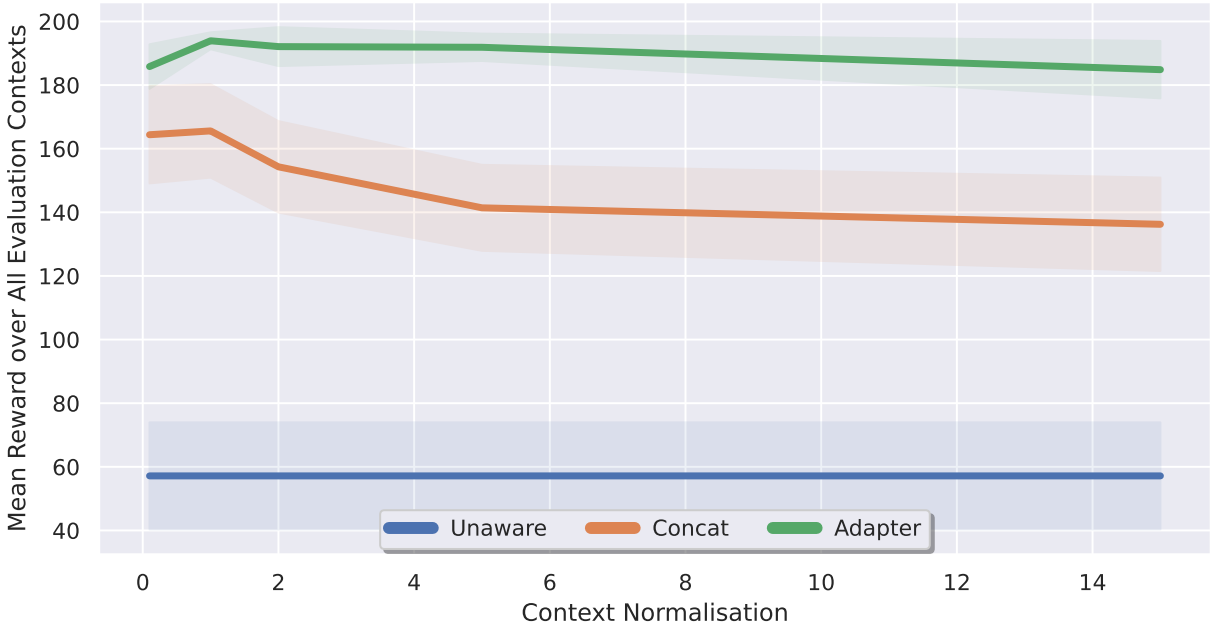


Figure 6.18: Showing the overall performance (y-axis) of each context-aware model after 300k steps as we change the context normalisation value (x-axis). Mean performance is shown, with standard deviation shaded.

CartPole

We now consider a similar setup in CartPole. In particular, we train on the `Small` setting of the Pole Mass context feature (see Table 6.1). We choose this variable as the Unaware model generalises equally well regardless of the training contexts (see Figure 6.5). We consider two scenarios: In the first, we normalise with respect to the `Small` setting (i.e. $c_{norm} = \frac{c}{0.01}$). Thus, during training, the normalised context had a value of 1.0; during evaluation, it was nearly always larger than 1.0. In the second case, we normalise with respect to the `X-Large` setting (corresponding to $c_{norm} = \frac{c}{1}$). We trained on the `Small` pole length for both settings. Figure 6.19 shows the results for this experiment. When normalising with respect to the `Small` setting, the Concat model fails to generalise at all, and the Adapter performs worse than Unaware. When changing the normalisation, however, the context-aware models perform significantly better.

Summary

Thus, the Decision Adapter is relatively robust to small changes in context normalisations. However, when encountering evaluation contexts that are 100 times larger than during training, appropriate normalisation is crucial and can make a drastic difference – even when training on the exact same contexts. As seen in the ODE, having a normalisation value that is slightly too large does not cause significant performance penalties. Thus, normalising the contexts with a larger value than expected during training is a good heuristic.

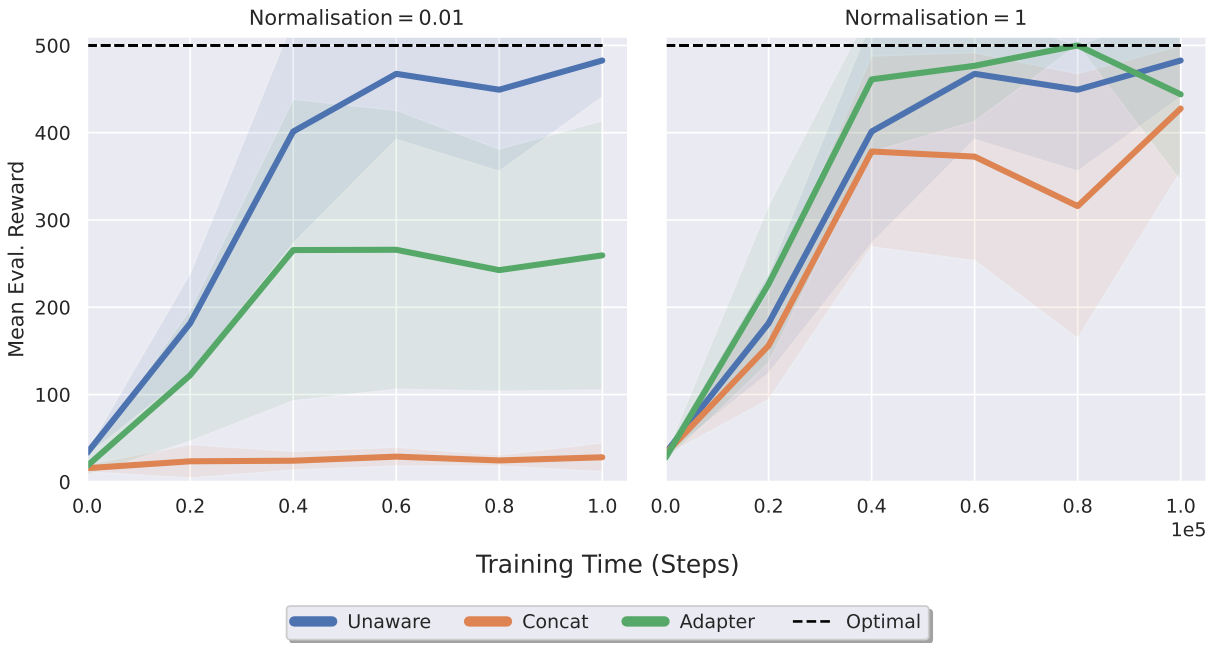


Figure 6.19: Evaluating on changing Pole Mass (similar to Figure 6.5) when normalising with respect to (left) Mass = 0.01 and (right) Mass = 1.0. In both plots, the models trained on Mass $= 0.01$. Mean performance is shown, with standard deviation shaded.

6.3.2 Context Noise

Now, while we use the ground truth context to isolate the effect of the network architecture on generalisation performance, this is not always available (Zhou *et al.* 2019; Sodhani *et al.* 2022). As discussed in Section 3.2, a common approach is to learn a context encoder that uses experience in the environment to infer an approximation to the context, which is then used to act. This encoded context, however, may not always be entirely accurate. Thus, in this experiment, we examine how sensitive each context-aware model is to receiving noisy contexts during evaluation. We consider two cases; the first is where the models received the uncorrupted context during training, and the second is where the models also encountered noise during training. The first case could reasonably occur in reality, since we may have access to the ground truth context during training, but not evaluation (Yu *et al.* 2017). The second case, where we train with noise, could correspond to training the RL policy alongside the context encoder, leading to some noise during training. This case additionally allows us to examine whether training with noise increases the models' robustness to context noise during evaluation.

We take the models trained during the primary CartPole experiments, in Section 6.1.2, where we varied the pole length. We use the model checkpoints that were trained for 600k timesteps, as both the Concat and Adapter models did not improve much by training for longer than this (see Figure 6.7). The noise we add here, corresponding to normally distributed noise with varying standard deviations, was added to each context dimension instead of just the one representing pole length. We experiment with a standard deviation $\sigma \in \{0, 0.05, 0.1, 0.2, 0.5, 1\}$. The noise was added after normalising the

contexts such that the largest training context had a value of 1.0. Thus, a standard deviation of 1.0 represents a substantial amount of noise. In addition to the models trained without noise, we additionally train models that encountered noise *during* training. In particular, we consider standard deviations $\sigma \in \{0.1, 0.5\}$, and change no other aspect of training.

The results are illustrated in Figure 6.20. The left pane shows the performance of the models trained without noise. Overall, these results show that the Concat model is more susceptible to noisy contexts than our Adapter, bolstering the argument that treating context as a state dimension is not ideal (Benjamins *et al.* 2022). Both models, however, generalise worse as the context becomes more inaccurate. This is in contrast to the Unaware model, which performs equally well regardless of the level of context noise. The Adapter still outperforms the Unaware model until the noise level reaches around $\sigma = 0.5$. This noise level, however, is very large, and $\sigma = 0.5$ corresponds to a pole length of 3 meters. The default pole length is 0.5 meters, and the largest training length is 6 meters.

When training with a small amount of noise (Figure 6.20, middle) both models perform similarly to training without noise. However, when training with a large amount of noise, corresponding to the right pane of Figure 6.20, the Concat model becomes more robust to noisy contexts, but the Adapter performs worse. In particular, the Adapter model exhibits a large amount of variation across seeds, indicating that some seeds performed badly. On average, the Adapter performs similarly to the Unaware model until the level of context noise increases beyond the training noise of $\sigma = 0.5$.

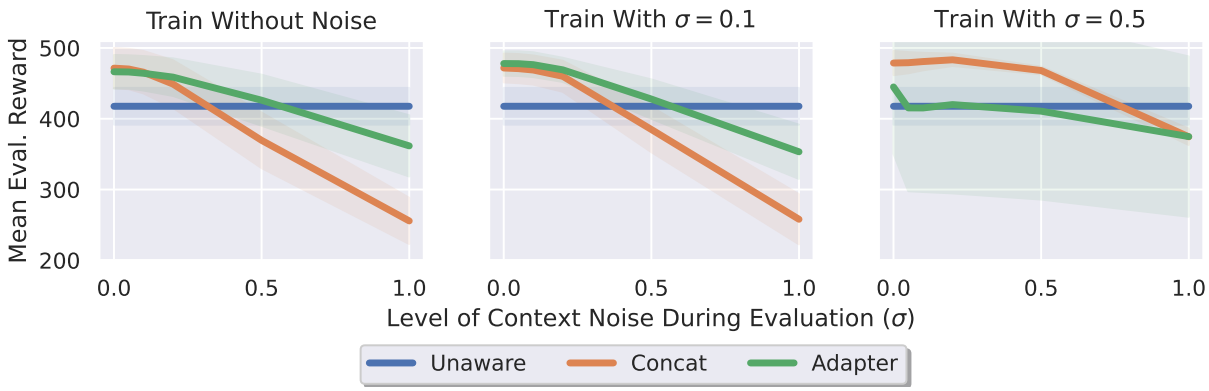


Figure 6.20: Comparing the average evaluation reward (after 600k steps) as a function of the standard deviation σ of the Gaussian noise added to the context during evaluation. In the left pane, the models trained without noise are shown. The middle pane corresponds to training with $\sigma = 0.1$ and the right pane shows the results when training with $\sigma = 0.5$. Mean performance is shown, with standard deviation over 16 seeds shaded.

6.3.3 The Effect of Training Contexts

We next consider the effect of changing the set of training contexts. Here we aim to briefly illustrate that the training context range can have a large effect on the perfor-

mance of the agents. In particular, we consider the ODE domain and 8 separate sets of training contexts. In all of these cases, we keep the normalisation consistent at $c_{norm} = \frac{c}{5}$ to ensure comparability. The results when training on each context set are shown in Figure 6.21. Overall, when we have a single positive and negative training context, performance is poor when this context is very small (a), and increases as it becomes larger (d and f). When we have multiple contexts, but insufficient variation (b, c, e), then performance is also suboptimal. For instance, in (b), the training contexts cover only the small region $[-1, 1]$. In (c), the training set of contexts does not contain any negative contexts, leading to poor generalisation. Finally, if contexts are varied, but spread out too far (g), the performance also suffers.

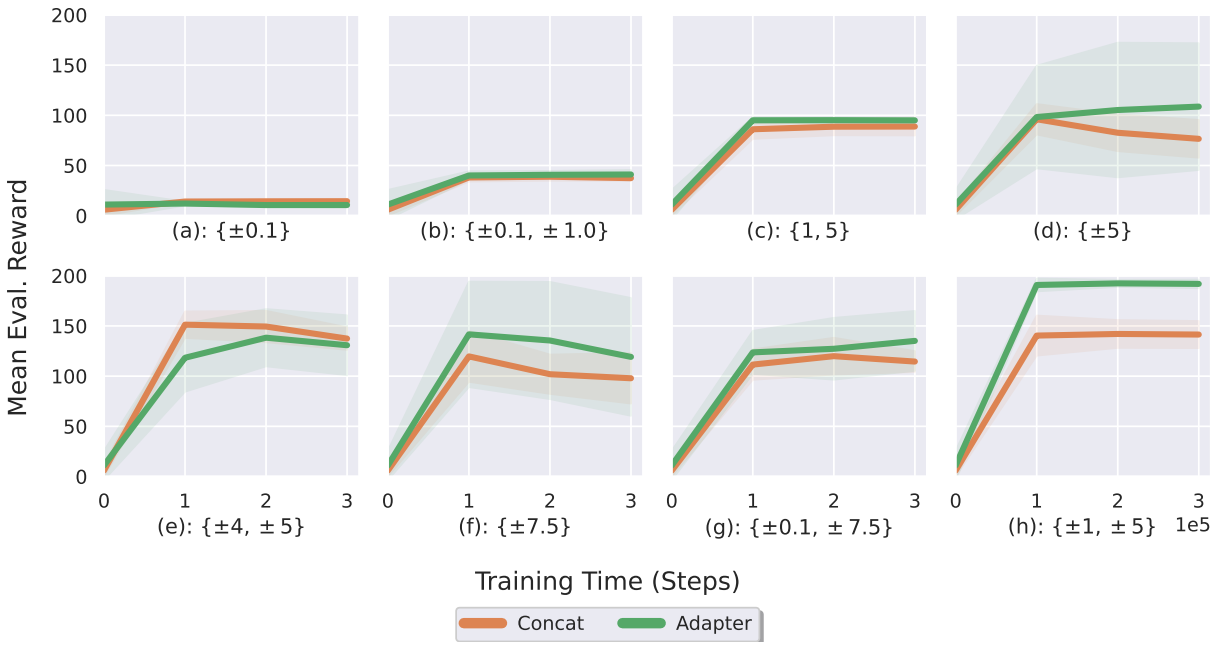


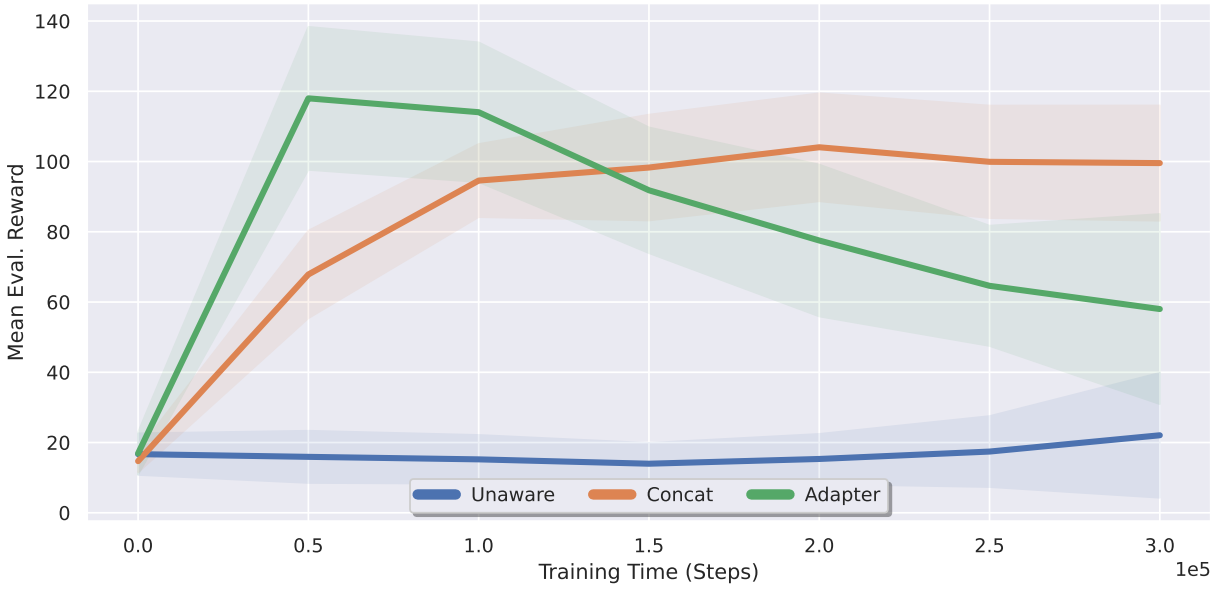
Figure 6.21: Here we examine the performance when training on different sets of contexts. The text beneath each plot indicates the training context set, with $\pm c$ indicating that both c and $-c$ are in the context set. For instance, in (a), the models were trained on a context set of $\mathcal{C}_{train} = \{-0.1, 0.1\}$. Mean performance is shown, with standard deviation shaded.

Overfitting

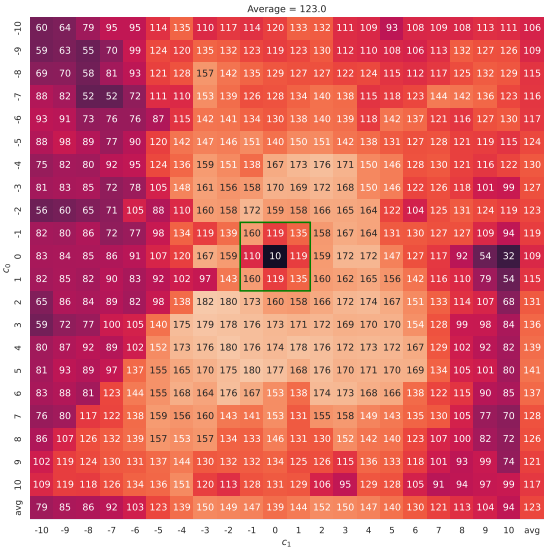
Another potential issue we may encounter is overfitting, which can occur when the training range is not diverse enough. To illustrate this problem, we conduct an experiment in the multidimensional ODE setting using the following set of training contexts:

$$\{(1, 1), (1, -1), (1, 0), (-1, 1), (-1, -1), (-1, 0), (0, 1), (0, -1)\}$$

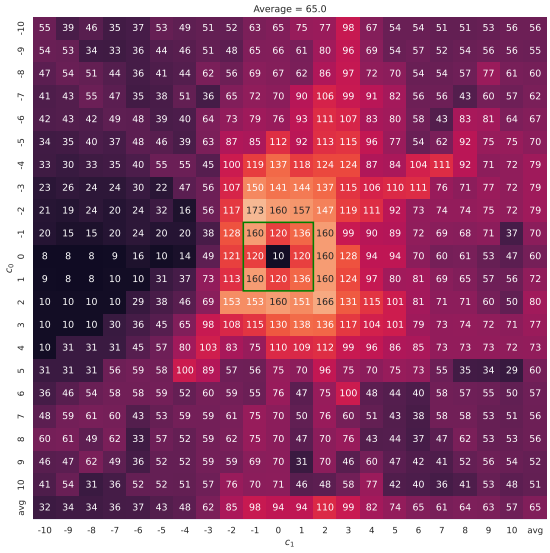
The results, presented in Figure 6.22, demonstrate that the Decision Adapter performs well initially but its generalisation performance suffers as training progresses. Specifically, as shown in Figure 6.22b and Figure 6.22c, our model exhibits worse extrapolation performance due to overfitting on the narrow training contexts.



(a)



(b) 50k steps



(c) 300k steps

Figure 6.22: Showing (a) the overall performance over time on the entire evaluation range. In (b) and (c) we show the performance of the Adapter at two points in training. In (a), we plot mean performance and shade the standard deviation; for (b) and (c), we show only the mean over the seeds.

6.4 Summary and Discussion

In this chapter, we demonstrated the effectiveness of our Adapter in particular, and context in general, in several settings. We originally set out to answer three primary questions, which we discuss now.

Does the Decision Adapter Generalise Better than our Baselines?

The answer to this question is yes. Overall, we demonstrated in Section 6.1.1 that our Adapter generalises better than our baselines when context is necessary. Next, in Section 6.1.2, we showed that incorporating context generally can lead to faster generalisation, and that our Adapter performs comparably to our baselines. Finally, in the more challenging Ant environment (Section 6.1.3), our Adapter performed the best across the two different variation settings. In particular, when varying friction, the Adapter learned faster than the Unaware model, both of which outperformed Concat. When varying mass, the Adapter and Concat models both outperformed the Unaware agent, as context was necessary. Thus, in more complex settings, the Adapter provides a good balance between these two models, and can mitigate the limitations of each approach.

`cGate` exhibited poor extrapolation performance, showing that our generalisation of this approach is indeed beneficial, and contributes to improved performance. While FLAP performed well in some cases, it generally took longer to converge, and struggled with multiple dimensions in the ODE environment. Additionally, FLAP needs to train a separate policy for each training context, whereas our Adapter has a consistent number of learnable parameters regardless of the number of training environments. Finally, FLAP requires a modified training procedure, while our approach can directly be applied to the standard RL training loop, as it is a drop-in replacement for the agent's network.

How Sensitive is the Decision Adapter to its Configuration Options?

The Decision Adapter is not particularly sensitive to the exact configuration options. In particular, in Section 6.2.1, we demonstrated that our method could be applied to another popular RL algorithm, PPO, making it widely applicable – which is one of the benefits of the concatenation-based model. Furthermore, the default configuration options we chose performed well, and, in most cases, we found several other settings that performed comparably. However, placing the adapter only at the start or end of the agent's network resulted in poor generalisation.

Due to this robustness to hyperparameters, we expect our Adapter model to readily generalise to other environments with the default hyperparameters we have chosen – even if these default hyperparameters are not completely optimal in the new environment. Provided that the context normalisation is performed adequately (using, e.g., the approach outlined in Section 5.4.1), we foresee no major obstacles to applying our method to other domains or algorithms.

What are the Limitations and Failure Cases of the Decision Adapter?

We found that using an incorrect and noisy context generally resulted in the Adapter and Concat models performing worse, particularly as we increased the level of noise. While our Adapter was more robust to this effect than the Concat model, its generalisation performance still suffered. Although training in the presence of a small amount of context noise did not affect the Adapter much, training on a large amount of noise resulted in worse performance. Secondly, while our Adapter performs well when the context normalisation is incorrect by a factor of 2 or 3, its performance does suffer when the normalisation value is orders of magnitudes too small. The Adapter still outperformed the Concat model in this case, but was itself outperformed by the Unaware model. In cases such as these, it may be that a context-unaware model performs the best, as it is not affected by poor normalisation or a low-quality context approximation. This, however, would only work if there exists a single policy that performs well in multiple settings; for instance, in CartPole. In environments where context is imperative, such as the ODE or Ant, incorporating context is our only option, even if it has some potential pitfalls. Furthermore, when training on a context range that was too narrow or did not exhibit sufficient variation, our model was susceptible to overfitting, which led to worse generalisation. Even if our model does not overfit, an overly narrow context range would also lead to poor generalisation. Thus, ensuring that agents are trained in sufficiently diverse environments is still important.

A final limitation of our Adapter is its longer training time compared to other methods. In particular, while we trained for the same number of timesteps as the other methods, and used a similar number of parameters, the Adapter generally takes longer (in terms of wall clock time) to train than other methods. This is likely caused by the hypernetwork that we use, which we did not optimise. Caching the generated weights for a particular context (which would be possible because context remains constant for an entire episode), or more effectively batching the forward passes through the Adapter could ameliorate this issue.

Chapter 7

Conclusion and Future Work

Reinforcement learning has great potential in various domains, but its real-world application is hindered by poor generalisation (Kirk *et al.* 2023). In this dissertation, we considered the problem of generalising to new transition dynamics, an important scenario with applications to control and robotics tasks (Doshi-Velez and Konidaris 2016). We introduced a new neural network architecture, the *Decision Adapter*, inspired by the success of adapters in other fields (Rebuffi *et al.* 2017; Houlsby *et al.* 2019; Mahabadi *et al.* 2021). This model uses context to generate a neural network that processes the state information, allowing the agent to modulate its behaviour based on context. Our approach allows much of the agent's policy to be shared across contexts, with the adapter module providing context-specific modifications. We empirically demonstrated that our model could improve generalisation compared to alternative approaches in several domains. In particular, our model outperformed the current default paradigm of simply concatenating the state and the context. We further showed that – when not encountering noise during training – our model is more robust to receiving noisy contexts than this concatenation-based model. Moreover, our approach can be easily integrated with multiple RL algorithms, making it a versatile and effective tool for addressing the generalisation problem. Additionally, our Adapter significantly outperformed a context-unaware model in cases requiring context to generalise well. In other cases, where context is not essential, we still performed comparably to, or better than, the Unaware model. Lastly, we demonstrated that our Adapter can effectively be applied to a real-world robotics task, where it performed well in various settings.

There are numerous avenues for future work. The most immediate aspect that merits further investigation would be the application of the Decision Adapter to other types of variation, such as generalising over changing observations or rewards. While we focus on the dynamics generalisation case, there is no fundamental reason why our architecture could not be applied to other settings. Next, while we assumed access to the ground truth context to isolate the method of context incorporation, it may not always be available, especially in real-world settings (Chen *et al.* 2018). Therefore, incorporating our approach into a context inference pipeline – as an alternative to simply concatenating the context and state – would be a promising next step. Using context encoders such as those introduced by Lee *et al.* (2020) or Sodhani *et al.* (2022)

alongside our Adapter would ideally improve the generalisation of these methods while not assuming any external information. Additionally, while our approach is agnostic to the RL algorithm used, it would be promising to investigate whether some algorithms or learning methods are particularly well-suited to our hypernetwork-based architecture.

Moreover, as discussed in Section 6.4, our Adapter does have some limitations. For instance, if the normalised evaluation contexts have large magnitudes ($|c_{norm}| \gg 1$), the Adapter's performance suffers. This would not be a problem when an inferred context is used, because it could be constrained to be within a specific range by using an appropriate activation function. When using the ground truth context, clipping the normalised context or applying an activation function to it to limit its range could be feasible options that merit investigation. Additionally, the Decision Adapter performs worse as the context becomes more inaccurate. The Adapter's robustness could potentially be improved by, as is commonly done, training the RL policy while the context encoder is still busy learning (Sodhani *et al.* 2022). If a large amount of additional noise is added during training, the Adapter's performance suffers. Investigating ways to mitigate this would be important; for instance, by starting with a small amount of noise and increasing it as training progresses. Furthermore, the agent's architecture is but one aspect required for generalisation; another important consideration is the distribution over contexts (Kirk *et al.* 2023). In particular, if the training contexts are not diverse enough, the Decision Adapter is susceptible to overfitting, leading to worse generalisation. Thus, it would be promising to incorporate our approach into the field of unsupervised environment design (Wang *et al.* 2019; Dennis *et al.* 2020; Parker-Holder *et al.* 2022) as an alternative to the current approach of solely training unaware models. Additionally, approaches such as early stopping or regularisation could be used to avert the problem of overfitting (Ada *et al.* 2022). Finally, we focused on empirically demonstrating the benefits of our approach. However, incorporating theoretical guarantees about how far an agent can generalise outside of its training context set would be helpful to safely deploy agents in the real world. Kenton *et al.* (2019) and Fu *et al.* (2023) are promising preliminary works in this direction, but more research is needed.

In summary, we have presented the Decision Adapter as an effective way to incorporate context into reinforcement learning. Our experiments demonstrated that the Decision Adapter outperforms several baseline methods, including the standard approach of concatenating the state and context. Importantly, the Decision Adapter can serve as a drop-in replacement for the concatenation-based method, as it does not require any modifications to the training procedure and is agnostic to the RL algorithm used. Thus, the Decision Adapter offers a practical and high-performing solution for incorporating context into a wide range of reinforcement learning problems.

It is important to note that our work is just one step towards addressing the generalisation problem in reinforcement learning. There are still some settings where the unaware model outperforms the context-aware ones, and more work must be done to address this issue. Nevertheless, we hope that our research will be useful for future RL practitioners, and will contribute to the ongoing efforts to improve generalisation in the field.

Bibliography

[Abdullah *et al.* 2019] Mohammed Amin Abdullah, Hang Ren, Haitham Bou-Ammar, Vladimir Milenkovic, Rui Luo, Mingtian Zhang, and Jun Wang. Wasserstein robust reinforcement learning. *CoRR*, abs/1907.13196, 2019.

[Achiam 2018] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[Ada *et al.* 2022] Suzan Ece Ada, Emre Ugur, and H. Levent Akin. Generalization in transfer learning: robust control of robot locomotion. *Robotica*, 40(11):3811–3836, 2022.

[Ahmed *et al.* 2019] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 151–160. PMLR, 2019.

[Ahmed *et al.* 2021] Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Manuel Wuthrich, Yoshua Bengio, Bernhard Schölkopf, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. In *9th International Conference on Learning Representations*. OpenReview.net, 2021.

[Akkaya *et al.* 2019] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[Andrychowicz *et al.* 2020] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *Int. J. Robotics Res.*, 39(1), 2020.

[Åström and Torsten 1965] Karl-Johan Åström and Bohlin Torsten. Numerical identification of linear dynamic systems from normal operating records. *IFAC Proceedings Volumes*, 2(2):96–111, 1965.

[Bahdanau *et al.* 2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, 2015.

[Bai *et al.* 2013] Haoyu Bai, David Hsu, and Wee Sun Lee. Planning how to learn. In *2013 IEEE International Conference on Robotics and Automation*, pages 2853–2859. IEEE, 2013.

[Ball *et al.* 2021] Philip J. Ball, Cong Lu, Jack Parker-Holder, and Stephen J. Roberts. Augmented world models facilitate zero-shot dynamics generalization from a single offline environment. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 619–629. PMLR, 2021.

[Barto *et al.* 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.*, 13(5):834–846, 1983.

[Baxter 1998] Jonathan Baxter. Theoretical models of learning to learn. In *Learning to Learn*, pages 71–94. Springer, 1998.

[Beck *et al.* 2023] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa M. Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *CoRR*, abs/2301.08028, 2023.

[Bellman 1957] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[Benjamins *et al.* 2022] Carolin Benjamins, Theresa Eimer, Frederik Schubert, Aditya Mohan, André Biedenkapp, Bodo Rosenhahn, Frank Hutter, and Marius Lindauer. Contextualize me - the case for context in reinforcement learning. *CoRR*, abs/2202.04500, 2022.

[Bhat *et al.* 2002] Kiran S. Bhat, Steven M. Seitz, and Jovan Popovic. Computing the physical parameters of rigid-body motion from video. In *7th European Conference on Computer Vision*, volume 2350 of *Lecture Notes in Computer Science*, pages 551–565. Springer, 2002.

[Boney *et al.* 2020] Rinu Boney, Jussi Sainio, Mikko Kaivola, Arno Solin, and Juho Kannala. Realant: An open-source low-cost quadruped for education and research in real-world reinforcement learning. *CoRR*, 2020.

[Bonferroni 1936] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62, 1936.

[Boyd and Vandenberghe 2004] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Brockman *et al.* 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[Brown *et al.* 2020] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom

Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[Chan *et al.* 2022] Stephanie C. Y. Chan, Andrew Kyle Lampinen, Pierre Harvey Richemond, and Felix Hill. Zipfian environments for reinforcement learning. In *Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pages 406–429. PMLR, 2022.

[Chen and Li 2020] Shiyu Chen and Yanjie Li. An overview of robust reinforcement learning. In *IEEE International Conference on Networking, Sensing and Control*, pages 1–6. IEEE, 2020.

[Chen *et al.* 2018] Tao Chen, Adithyavairavan Murali, and Abhinav Gupta. Hardware conditioned policies for multi-robot transfer learning. In *Advances in Neural Information Processing Systems*, pages 9355–9366, 2018.

[Chen *et al.* 2019] Yu Chen, Yingfeng Chen, Yu Yang, Ying Li, Jianwei Yin, and Changjie Fan. Learning action-transferable policy with action embedding. *CoRR*, abs/1909.02291, 2019.

[Christodoulou 2019] Petros Christodoulou. Soft actor-critic for discrete action settings. *CoRR*, abs/1910.07207, 2019.

[Cobbe *et al.* 2019] Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 2019.

[Cobbe *et al.* 2020] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2048–2056. PMLR, 2020.

[Conneau *et al.* 2020] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451. Association for Computational Linguistics, 2020.

[Degris *et al.* 2012] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. In *Proceedings of the 29th International Conference on Machine Learning*. icml.cc / Omnipress, 2012.

[Dennis *et al.* 2020] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, 2020.

[des Combes *et al.* 2018] Remi Tachet des Combes, Philip Bachman, and Harm van Seijen. Learning invariances for policy generalization. In *6th International Conference on Learning Representations*. OpenReview.net, 2018.

[Devlin *et al.* 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics, 2019.

[Dieleman *et al.* 2015] Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *CoRR*, abs/1503.07077, 2015.

[dos Santos and Papa 2022] Claudio Filipi Goncalves dos Santos and João Paulo Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Comput. Surv.*, 54(10s):213:1–213:25, 2022.

[Doshi-Velez and Konidaris 2016] Finale Doshi-Velez and George Dimitri Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1432–1440. IJCAI/AAAI Press, 2016.

[Du *et al.* 2019] Simon S. Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudík, and John Langford. Provably efficient RL with rich observations via latent state decoding. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1665–1674. PMLR, 2019.

[Duan *et al.* 2016] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

[Dumoulin *et al.* 2018] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations.

[Eghbal-zadeh *et al.* 2021] Hamid Eghbal-zadeh, Florian Henkel, and Gerhard Widmer. Context-adaptive reinforcement learning using unsupervised learning of context variables. In *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*, volume 148 of *Proceedings of Machine Learning Research*, pages 236–254. PMLR, 11 Dec 2021.

[Ehret *et al.* 2021] Benjamin Ehret, Christian Henning, Maria R. Cervera, Alexander Meulemans, Johannes von Oswald, and Benjamin F. Grewe. Continual learning in recurrent neural networks. In *International Conference on Learning Representations*, 2021.

[Escontrela *et al.* 2020] Alejandro Escontrela, George Yu, Peng Xu, Atil Iscen, and Jie Tan. Zero-shot terrain generalization for visual locomotion policies. *CoRR*, abs/2011.05513, 2020.

[Eysenbach and Levine 2022] Benjamin Eysenbach and Sergey Levine. Maximum entropy RL (provably) solves some robust RL problems. In *International Conference on Learning Representations*, 2022.

[Eysenbach *et al.* 2021] Ben Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Robust predictable control. In *Advances in Neural Information Processing Systems*, pages 27813–27825, 2021.

[Faccio *et al.* 2022] Francesco Faccio, Vincent Herrmann, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. Goal-conditioned generators of deep policies. *CoRR*, abs/2207.01570, 2022.

[Fakoor *et al.* 2020] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. Meta-q-learning. In *8th International Conference on Learning Representations*. OpenReview.net, 2020.

[Farebrother *et al.* 2018] Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in DQN. *CoRR*, abs/1810.00123, 2018.

[Feng *et al.* 2022] Gilbert Feng, Hongbo Zhang, Zhongyu Li, Xue Bin Peng, Bhuvan Basireddy, Linzhu Yue, Zhitao Song, Lizhi Yang, Yunhui Liu, Koushil Sreenath, and Sergey Levine. Genloco: Generalized locomotion controllers for quadrupedal robots. In *CoRL*, volume 205 of *Proceedings of Machine Learning Research*, pages 1893–1903. PMLR, 2022.

[Ferns *et al.* 2004] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Proceedings of the 19th National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence,*, pages 950–951. AAAI Press / The MIT Press, 2004.

[Ferns *et al.* 2011] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM J. Comput.*, 40(6):1662–1714, 2011.

[Finn *et al.* 2017] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

[Florian 2007] Razvan V Florian. Correct equations for the dynamics of the cart-pole system. *Center for Cognitive and Neural Studies (Coneural), Romania*, 2007.

[Fortunato *et al.* 2019] Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charles Deck, Joel Z Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[French 1999] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[Fu *et al.* 2022a] Chin-Lun Fu, Zih-Ching Chen, Yun-Ru Lee, and Hung-yi Lee. Adapterbias: Parameter-efficient token-dependent representation shift for adapters in NLP tasks. In *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 2608–2621. Association for Computational Linguistics, 2022.

[Fu *et al.* 2022b] Haotian Fu, Shangqun Yu, Saket Tiwari, George Dimitri Konidaris, and Michael L. Littman. Meta-learning transferable parameterized skills. *CoRR*, abs/2206.03597, 2022.

[Fu *et al.* 2023] Haotian Fu, Jiayu Yao, Omer Gottesman, Finale Doshi-Velez, and George Konidaris. Performance bounds for model and policy transfer in hidden-parameter MDPs. In *The Eleventh International Conference on Learning Representations*, 2023.

[Fujimoto *et al.* 2018] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR, 2018.

[Ghasemipour *et al.* 2019] Seyed Kamyar Seyed Ghasemipour, Shixiang Gu, and Richard S. Zemel. Smile: Scalable meta inverse reinforcement learning through context-conditional policies. In *Advances in Neural Information Processing Systems*, pages 7879–7889, 2019.

[Ghosh *et al.* 2021] Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34, 2021.

[Gisslén *et al.* 2021] Linus Gisslén, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar. Adversarial reinforcement learning for procedural content generation. In *Conference on Games*, pages 1–8. IEEE, 2021.

[Gur *et al.* 2021] Izzeddin Gur, Natasha Jaques, Yingjie Miao, Jongwook Choi, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Environment generation for zero-shot compositional reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4157–4169, 2021.

[Ha *et al.* 2017] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations*. OpenReview.net, 2017.

[Haarnoja *et al.* 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018.

[Hallak *et al.* 2015] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *CoRR*, abs/1502.02259, 2015.

[Harries *et al.* 2019] Luke Harries, Sebastian Lee, Jaroslaw Rzepecki, Katja Hofmann, and Sam Devlin. Mazeexplorer: A customisable 3D benchmark for assessing generalisation in reinforcement learning. In *Conference on Games*, pages 1–4. IEEE, 2019.

[Hastie *et al.* 2009] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*, chapter 11. Springer Series in Statistics. Springer, 2009.

[Haykin 1998] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE Computer Society, 2016.

[Henning *et al.* 2021] Christian Henning, Maria R. Cervera, Francesco D'Angelo, Johannes von Oswald, Regina Traber, Benjamin Ehret, Seijin Kobayashi, Benjamin F. Grewe, and João Sacramento. Posterior meta-replay for continual learning. In *Advances in on Neural Information Processing Systems*, 2021.

[Hochreiter and Schmidhuber 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[Houlsby *et al.* 2019] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.

[Huang *et al.* 2021] Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *International Conference on Robotics and Automation*, pages 799–805. IEEE, 2021.

[Huang *et al.* 2022] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[Huh *et al.* 2021] Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *CoRR*, abs/2103.10427, 2021.

[Igl *et al.* 2019] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pages 13956–13968, 2019.

[Ivison and Peters 2022] Hamish Ivison and Matthew E. Peters. Hyperdecoders: Instance-specific decoders for multi-task NLP. In *Findings of the Association for Computational Linguistics*, pages 1715–1730. Association for Computational Linguistics, 2022.

[Jafari and Ansari-Pour 2019] Mohieddin Jafari and Naser Ansari-Pour. Why, when and how to adjust your p values? *Cell Journal (Yakhteh)*, 20(4):604, 2019.

[Jain *et al.* 2020] Ayush Jain, Andrew Szot, and Joseph Lim. Generalization to new actions in reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4661–4672. PMLR, 13–18 Jul 2020.

[Jain *et al.* 2022] Ayush Jain, Norio Kosaka, Kyung-Min Kim, and Joseph J Lim. Know your action set: Learning action relations for reinforcement learning. In *International Conference on Learning Representations*, 2022.

[James *et al.* 2017] Stephen James, Andrew J. Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 334–343. PMLR, 2017.

[Julian *et al.* 2020] Ryan Julian, Benjamin Swanson, Gaurav S. Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. In *4th Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 2120–2136. PMLR, 2020.

[Kaelbling *et al.* 1998] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[Kenton *et al.* 2019] Zachary Kenton, Angelos Filos, Owain Evans, and Yarin Gal. Generalizing from a few environments in safety-critical reinforcement learning. *ICLR SafeML Workshop*, 2019.

[Kessler *et al.* 2022] Samuel Kessler, Jack Parker-Holder, Philip J. Ball, Stefan Zohren, and Stephen J. Roberts. Same state, different task: Continual reinforcement learning without interference. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 7143–7151. AAAI Press, 2022.

[Killian *et al.* 2017] Taylor W. Killian, George Dimitri Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Proceedings of the Thirty-First Conference on Artificial Intelligence*, pages 4949–4950. AAAI Press, 2017.

[Kirchmeyer *et al.* 2022] Matthieu Kirchmeyer, Yuan Yin, Jérémie Donà, Nicolas Baskiotis, Alain Rakotomamonjy, and Patrick Gallinari. Generalizing to new physical systems via context-informed dynamics model. 162:11283–11301, 2022.

[Kirk *et al.* 2023] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *J. Artif. Intell. Res.*, 76:201–264, 2023.

[Konda and Tsitsiklis 1999] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014. The MIT Press, 1999.

[Konidaris *et al.* 2012] George Dimitri Konidaris, Ilya Scheidwasser, and Andrew G. Barto. Transfer in reinforcement learning via shared features. *J. Mach. Learn. Res.*, 13:1333–1371, 2012.

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

[Krueger *et al.* 2021] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5815–5826. PMLR, 18–24 Jul 2021.

[Kullback and Leibler 1951] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[Kumar *et al.* 2021] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: rapid motor adaptation for legged robots. In *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*, 2021.

[Lange *et al.* 2012] Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 45–73. Springer, 2012.

[Lazaric 2012] Alessandro Lazaric. Transfer in reinforcement learning: A framework and a survey. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 143–173. Springer, 2012.

[Le *et al.* 2021] Hang Le, Juan Miguel Pino, Changhan Wang, Jiatao Gu, Didier Schwab, and Laurent Besacier. Lightweight adapter tuning for multilingual speech translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 817–824. Association for Computational Linguistics, 2021.

[Lee *et al.* 2020] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In *Proceedings of the 37th International Conference on Ma-*

chine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 5757–5766. PMLR, 2020.

[Li *et al.* 2021] Bonnie Li, Vincent François-Lavet, Thang Doan, and Joelle Pineau. Domain adversarial reinforcement learning. *CoRR*, abs/2102.07097, 2021.

[Lin 1992] Long Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8:293–321, 1992.

[Liu *et al.* 2019] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[Lu *et al.* 2020] Xingyu Lu, Kimin Lee, Pieter Abbeel, and Stas Tiomkin. Dynamics generalization via information bottleneck in deep reinforcement learning. *CoRR*, abs/2008.00614, 2020.

[Lyle *et al.* 2022] Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 14560–14581. PMLR, 2022.

[Mahabadi *et al.* 2021] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 565–576. Association for Computational Linguistics, 2021.

[Malik *et al.* 2021] Dhruv Malik, Yuanzhi Li, and Pradeep Ravikumar. When is generalizable reinforcement learning tractable? *Advances in Neural Information Processing Systems*, 34, 2021.

[Mankowitz *et al.* 2020] Daniel J. Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Yuanyuan Shi, Jackie Kay, Todd Hester, Timothy A. Mann, and Martin A. Riedmiller. Robust reinforcement learning for continuous control with model misspecification. In *8th International Conference on Learning Representations*. OpenReview.net, 2020.

[Mann and Whitney 1947] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60, 1947.

[Matas *et al.* 2018] Jan Matas, Stephen James, and Andrew J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. 87:734–743, 2018.

[McCloskey and Cohen 1989] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[McCulloch and Pitts 1943] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[Minsky and Papert 1969] Marvin Minsky and Seymour Papert. *Perceptron: an introduction to computational geometry*. Cambridge, MA: MIT Press, 1969.

[Mishra *et al.* 2018] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *6th International Conference on Learning Representations*. OpenReview.net, 2018.

[Mnih *et al.* 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[Mnih *et al.* 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

[Moerland *et al.* 2023] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Found. Trends Mach. Learn.*, 16(1):1–118, 2023.

[Moore 1990] Andrew William Moore. *Efficient memory-based learning for robot control*. Technical report, University of Cambridge, Computer Laboratory, 1990.

[Mousavi *et al.* 2014] Amin Mousavi, Babak Nadjar Araabi, and Majid Nili Ahmadabadi. Context transfer in reinforcement learning using action-value functions. *Comput. Intell. Neurosci.*, 2014:428567:1–428567:10, 2014.

[Mu *et al.* 2022] Yao Mu, Yuzheng Zhuang, Fei Ni, Bin Wang, Jianyu Chen, Jianye Hao, and Ping Luo. Decomposed mutual information optimization for generalized context in meta-reinforcement learning. *CoRR*, abs/2210.04209, 2022.

[Nagabandi *et al.* 2018] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation*, pages 7559–7566. IEEE, 2018.

[Nagabandi *et al.* 2019] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *7th International Conference on Learning Representations*. OpenReview.net, 2019.

[Nair *et al.* 2018] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *International Conference on Robotics and Automation*, pages 6292–6299. IEEE, 2018.

[Nasiriany *et al.* 2021] Soroush Nasiriany, Vitchyr H. Pong, Ashvin Nair, Alexander Khazatsky, Glen Berseth, and Sergey Levine. Disco RL: distribution-conditioned reinforcement learning for general-purpose policies. In *International Conference on Robotics and Automation*, pages 6635–6641. IEEE, 2021.

[Orhan and Pitkow 2018] A. Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. In *6th International Conference on Learning Representations*. OpenReview.net, 2018.

[Ota *et al.* 2021] Kei Ota, Devesh K. Jha, and Asako Kanezaki. Training larger networks for deep reinforcement learning. *CoRR*, abs/2102.07920, 2021.

[Packer *et al.* 2018] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282, 2018.

[Parker-Holder *et al.* 2022] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17473–17498. PMLR, 2022.

[Paszke *et al.* 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[Peng *et al.* 2018] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *International Conference on Robotics and Automation*, pages 1–8. IEEE, 2018.

[Peng *et al.* 2020] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems XVI*, 2020.

[Peng *et al.* 2021] Matt Peng, Banghua Zhu, and Jiantao Jiao. Linear representation meta-reinforcement learning for instant adaptation. *CoRR*, abs/2101.04750, 2021.

[Perez and Wang 2017] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[Perez *et al.* 2018] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th*

AAAI Symposium on Educational Advances in Artificial Intelligence, pages 3942–3951. AAAI Press, 2018.

[Pfeiffer *et al.* 2020a] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54. Association for Computational Linguistics, 2020.

[Pfeiffer *et al.* 2020b] Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online, November 2020. Association for Computational Linguistics.

[Pilault *et al.* 2021] Jonathan Pilault, Amine El hattami, and Christopher Pal. Conditionally adaptive multi-task learning: Improving transfer learning in NLP using fewer parameters & less data. In *International Conference on Learning Representations*, 2021.

[Pinto *et al.* 2017] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR, 06–11 Aug 2017.

[Press *et al.* 2007] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[Pretorius *et al.* 2020] Arnu Pretorius, Elan Van Biljon, Benjamin van Niekerk, Ryan Eloff, Matthew Reynard, Steven D. James, Benjamin Rosman, Herman Kamper, and Steve Kroon. If dropout limits trainable depth, does critical initialisation still matter? A large-scale statistical analysis on relu networks. *Pattern Recognit. Lett.*, 138:95–105, 2020.

[Puterman and Feinberg 1996] Martin L Puterman and Eugene A Feinberg. Markov decision processes: discrete stochastic dynamic programming. *SIAM review*, 38(4):689, 1996.

[Radford *et al.* 2019] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Raileanu and Fergus 2021] Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8787–8798. PMLR, 18–24 Jul 2021.

[Raileanu *et al.* 2020] Roberta Raileanu, Max Goldstein, Arthur Szlam, and Rob Fergus. Fast adaptation to new environments via policy-dynamics value functions. In

Proceedings of the 37th International Conference on Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 7920–7931. PMLR, 2020.

[Raileanu *et al.* 2021] Roberta Raileanu, Maxwell Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

[Rakelly *et al.* 2019] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5331–5340. PMLR, 2019.

[Rebuffi *et al.* 2017] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, 2017.

[Ronneberger *et al.* 2015] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[Rosenblatt 1961] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[Roy and Konidaris 2021] Josh Roy and George Dimitri Konidaris. Visual transfer for reinforcement learning via wasserstein domain confusion. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence*, pages 9454–9462. AAAI Press, 2021.

[Rumelhart *et al.* 1985] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[Rumelhart *et al.* 1986] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[Sadeghi and Levine 2017] Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

[Sæmundsson *et al.* 2018] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta reinforcement learning with latent variable gaussian processes.

In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 642–652. AUAI Press, 2018.

[Sanchez-Gonzalez *et al.* 2018] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter W. Battaglia. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4467–4476. PMLR, 2018.

[Sarafian *et al.* 2021] Elad Sarafian, Shai Keynan, and Sarit Kraus. Recomposing the reinforcement learning building blocks with hypernetworks. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9301–9312. PMLR, 2021.

[Schulman *et al.* 2016] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations*, 2016.

[Schulman *et al.* 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Seo *et al.* 2020] Younggyo Seo, Kimin Lee, Ignasi Clavera Gilaberte, Thanard Kurutach, Jinwoo Shin, and Pieter Abbeel. Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[Shannon 1948] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[Shapiro and Wilk 1965] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[Shi *et al.* 2022] Haoxiang Shi, Rongsheng Zhang, Jiaan Wang, Cen Wang, Yinhe Zheng, and Tetsuya Sakai. Layerconnect: Hypernetwork-assisted inter-layer connector to enhance parameter efficiency. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3120–3126. International Committee on Computational Linguistics, 2022.

[Shorten and Khoshgoftaar 2019] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019.

[Siekmann *et al.* 2020] Jonah Siekmann, Srikar Valluri, Jeremy Dao, Lorenzo Bermillo, Helei Duan, Alan Fern, and Jonathan W. Hurst. Learning memory-based control for human-scale bipedal locomotion. In *Robotics: Science and Systems XVI*, 2020.

[Silver *et al.* 2017] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan

Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[Smith *et al.* 2022] Laura M. Smith, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In *International Conference on Robotics and Automation*, pages 1593–1599. IEEE, 2022.

[Sodhani *et al.* 2021] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9767–9779. PMLR, 2021.

[Sodhani *et al.* 2022] Shagun Sodhani, Franziska Meier, Joelle Pineau, and Amy Zhang. Block contextual MDPs for continual learning. In *Learning for Dynamics and Control Conference, L4DC 2022, 23-24 June 2022, Stanford University, Stanford, CA, USA*, volume 168 of *Proceedings of Machine Learning Research*, pages 608–623. PMLR, 2022.

[Sohrab 2003] Houshang H Sohrab. *Basic real analysis*, volume 231, chapter 4, page 163. Springer, 2003.

[Spitznagel *et al.* 2021] Martin Spitznagel, David Weiler, and Klaus Dorer. Deep reinforcement multi-directional kick-learning of a simulated robot with toes. In *International Conference on Autonomous Robot Systems and Competitions*, pages 104–110. IEEE, 2021.

[Srinivasan *et al.* 2020] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep RL with a safety critic. *CoRR*, abs/2010.14603, 2020.

[Srivastava *et al.* 2014] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[Sutton and Barto 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sutton 1995] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pages 1038–1044. MIT Press, 1995.

[Swevers *et al.* 1997] Jan Swevers, Chris Ganseman, Dilek Bilgin Tükel, Joris De Schutter, and Hendrik Van Brussel. Optimal robot excitation and identification. *IEEE Trans. Robotics Autom.*, 13(5):730–740, 1997.

[Tang *et al.* 2019] Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. Worst cases policy gradients. In *3rd Annual Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1078–1093. PMLR, 2019.

[Taylor and Nitschke 2017] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. *CoRR*, abs/1708.06020, 2017.

[Taylor and Stone 2009] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, 2009.

[Team *et al.* 2021] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michaël Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents. *CoRR*, abs/2107.12808, 2021.

[Thomas *et al.* 2022] Bethan Thomas, Samuel Kessler, and Salah Karout. Efficient adapter transfer of self-supervised speech models for automatic speech recognition. In *International Conference on Acoustics, Speech and Signal Processing*, pages 7102–7106. IEEE, 2022.

[Thrun and Pratt 1998] *Learning to Learn*. Springer, 1998.

[Tobin *et al.* 2017] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems*, pages 23–30. IEEE, 2017.

[Todorov *et al.* 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[Trivedi *et al.* 2021] Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J. Lim. Learning to synthesize programs as interpretable and generalizable policies. In *Advances in Neural Information Processing Systems*, pages 25146–25163, 2021.

[Upadhyay *et al.* 2021] Richa Upadhyay, Ronald Phlypo, Rajkumar Saini, and Marcus Liwicki. Sharing to learn and learning to share - fitting together meta-learning, multi-task learning, and transfer learning: A meta review. *CoRR*, abs/2111.12146, 2021.

[van Breugel and Worrell 2001] Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In *Automata, Languages and Programming, 28th International Colloquium*, volume 2076 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2001.

[van Hasselt 2010] Hado van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[Vanschoren 2018] Joaquin Vanschoren. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018.

[Vaserstein 1969] Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[Vaswani *et al.* 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[Vinyals *et al.* 2019] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[Volk *et al.* 2022] Tomer Volk, Eyal Ben-David, Ohad Amosy, Gal Chechik, and Roi Reichart. Example-based hypernetworks for out-of-distribution generalization. *CoRR*, abs/2203.14276, 2022.

[von Oswald *et al.* 2020] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *8th International Conference on Learning Representations*. OpenReview.net, 2020.

[Wang *et al.* 2017] Jane Wang, Zeb Kurth-Nelson, Hubert Soyer, Joel Z. Leibo, Dhruva Tirumala, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matt M. Botvinick. Learning to reinforcement learn. In *Proceedings of the 39th Annual Meeting of the Cognitive Science Society*. cognitivesciencesociety.org, 2017.

[Wang *et al.* 2019] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired Open-Ended Trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *CoRR*, abs/1901.01753, 2019.

[Wang *et al.* 2020a] Haozhe Wang, Jiale Zhou, and Xuming He. Learning context-aware task reasoning for efficient meta reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, pages 1440–1448. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[Wang *et al.* 2020b] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth O. Stanley. Enhanced POET: open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9940–9951. PMLR, 2020.

[Wang *et al.* 2021] Haoxiang Wang, Han Zhao, and Bo Li. Bridging multi-task learning and meta-learning: Towards efficient training and effective adaptation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10991–11002. PMLR, 2021.

[Wang *et al.* 2022] Tongzhou Wang, Simon Du, Antonio Torralba, Phillip Isola, Amy Zhang, and Yuandong Tian. Denoised mdps: Learning world models better than the world itself. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22591–22612. PMLR, 2022.

[Watkins and Dayan 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Wellmer and Kwok 2021] Zac Wellmer and James T. Kwok. Dropout's dream land: Generalization from learned simulators to reality. In *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part I*, volume 12975 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2021.

[Werbos 1974] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[Werbos 1990] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[William 1908] S William. The probable error of a mean. *Biometrika*, 6(1):1–25, 1908.

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

[Witty *et al.* 2018] Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael L. Littman, and David D. Jensen. Measuring and characterizing generalization in deep reinforcement learning. *CoRR*, abs/1812.02868, 2018.

[Xian *et al.* 2021] Zhou Xian, Shamit Lal, Hsiao-Yu Tung, Emmanouil Antonios Platanios, and Katerina Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. In *9th International Conference on Learning Representations*. OpenReview.net, 2021.

[Xu and Mannor 2012] Huan Xu and Shie Mannor. Robustness and generalization. *Mach. Learn.*, 86(3):391–423, 2012.

[Yao *et al.* 2018] Jiayu Yao, Taylor Killian, George Konidaris, and Finale Doshi-Velez. Direct policy transfer via hidden parameter markov decision processes. In *LLARLA Workshop, FAIM*, volume 2018, 2018.

[Ye and Ren 2021] Qinyuan Ye and Xiang Ren. Learning to generate task-specific adapters from task description. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference*

on Natural Language Processing, pages 646–653. Association for Computational Linguistics, 2021.

[Ye *et al.* 2020] Chang Ye, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. Rotation, translation, and cropping for zero-shot generalization. In *Conference on Games*, pages 57–64. IEEE, 2020.

[Yu *et al.* 2017] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

[Yu *et al.* 2020] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *3rd Annual Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR, 2020.

[Zhang *et al.* 2018a] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *CoRR*, abs/1806.07937, 2018.

[Zhang *et al.* 2018b] Chiyuan Zhang, Oriol Vinyals, Rémi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *CoRR*, abs/1804.06893, 2018.

[Zhang *et al.* 2020] Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block MDPs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11214–11224. PMLR, 2020.

[Zhang *et al.* 2021a] Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *9th International Conference on Learning Representations*. OpenReview.net, 2021.

[Zhang *et al.* 2021b] Amy Zhang, Shagun Sodhani, Khimya Khetarpal, and Joelle Pineau. Learning robust state abstractions for hidden-parameter block mdps. In *9th International Conference on Learning Representations*. OpenReview.net, 2021.

[Zhao and Hospedales 2021] Chenyang Zhao and Timothy M. Hospedales. Robust domain randomised reinforcement learning through peer-to-peer distillation. In *Asian Conference on Machine Learning, ACML 2021, 17-19 November 2021, Virtual Event*, volume 157 of *Proceedings of Machine Learning Research*, pages 1237–1252. PMLR, 2021.

[Zhao *et al.* 2019] Chenyang Zhao, Olivier Sigaud, Freek Stulp, and Timothy M. Hospedales. Investigating generalisation in continuous deep reinforcement learning. *CoRR*, abs/1902.07015, 2019.

[Zhao *et al.* 2020] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *Symposium Series on Computational Intelligence*, pages 737–744. IEEE, 2020.

[Zhou *et al.* 2019] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. In *7th International Conference on Learning Representations*. OpenReview.net, 2019.

[Zhu *et al.* 2018] Shaojun Zhu, Andrew Kimmel, Kostas E. Bekris, and Abdeslam Boularias. Fast model identification via physics engines for data-efficient policy search. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 3249–3256. ijcai.org, 2018.

[Ziebart 2010] Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, USA, 2010.

[Zintgraf *et al.* 2019] Luisa M. Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7693–7702. PMLR, 2019.

[Zintgraf *et al.* 2020] Luisa M. Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep RL via meta-learning. In *8th International Conference on Learning Representations*. OpenReview.net, 2020.

[Zintgraf *et al.* 2021] Luisa M. Zintgraf, Leo Feng, Cong Lu, Maximilian Igl, Kristian Hartikainen, Katja Hofmann, and Shimon Whiteson. Exploration in approximate hyper-state space for meta reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12991–13001. PMLR, 2021.