ABSTRACTION FOR EFFICIENT REINFORCEMENT LEARNING

by

Alexander Telfar

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2020

Abstract

Successful reinforcement learning requires large amounts of data, compute, and some luck. We explore the ability of abstraction(s) to reduce these dependencies.

Abstractions for reinforcement learning share the goals of this abstract: to capture essential details, while leaving out the unimportant. By throwing away inessential details, there will be less to compute, less to explore, and less variance in observations. But, does this always aid reinforcement learning?

More specifically, we start by looking for abstractions that are easily solvable. This leads us to a type of linear abstraction. We show that, while it does allow efficient solutions, it also gives erroneous solutions, in the general case.

We then attempt to improve the sample efficiency of a reinforcment learner. We do so by constructing a measure of symmetry and using it as an inductive bias. We design and run experiments to test the advantage provided by this inductive bias, but must leave conclusions to future work.

Acknowledgments

Mostly, I would like to thank my advisers: Will Browne, for supporting my work and giving me the freedom to explore my interests. And Brendan McCane for timely feedback.

I would also like to thank Daniel Braithwaite, Marcus Frean and Stephen Marsland for their friendship, humour and intellectual support. And, I would like to thank my family (Ian and Evee), who, for some reason, put up with me.

And finally, I am thankful that SFTI supports theoretical work, despite its high risk.

Contents

Chapter 1

Introduction

Reinforcement learning has an efficiency problem: AlphaGo [1], the Go playing AI that beat world champion Lee Sedol, played 1.28 million games, with extra supervision from another 29.4 million positions, using 50 GPUs. Libratus [2], the poker playing AI that beat a table of professionals, constructed its poker-playing strategy over 15 million processor-core-hours. OpenAI Five [3], the Dota 2 playing AI that beat OG, the winners of The International 8 and 9, was trained over 10 months, at its peak, it collected 900 years of experience per day using 128,000 CPUs and 256 GPUs.

> *Is reinforcement learning fundamentally inefficient, or can we do better?*

1.1 Our Motivation

We think (more) efficient reinforcement learning might be achieved by the use of abstraction. Abstractions allow a learner to preserve essential structure, while discarding inessential details.

> *By throwing away inessential details, there is less to compute.*
> *By throwing away inessential details, we don't need to explore them.*

By throwing away inessential details, we reduce the variance of our observations (allowing quicker learning).

Thus, the main goal of this thesis is to:

understand how abstractions can increase the efficiency of reinforcement learning.

1.2 Overview

Given the goal above, we pick Markov Decision Problems as our setting for studying abstractions[1]and give a slightly non-standard, but general introduction to them, see section 2. Then we explore abstractions and existing theory for understanding abstraction and attempt to organise it in to a framework, see section 3.1. Next, we analyse an existing method of abstraction for efficient control and find that it does not work in general, see section 3.2. Finally, we build an abstraction using symmetries, see section 3.3.

1.3 Contributions

In this thesis, we;

- Clarify that a well cited method of linear abstraction doesn't work in general. See section 3.2.4
- Evaluate a method of combining symmetric abstractions and Thompson sampling. See section 3.3.3
- Explore the iteration complexity, effect of discounting, and the density of the Value functional polytope. See section A.3

[1]In general, when we state abstractions, we mean abstractions for reinforcement learning.

- Provide a way to visualise the dynamics of learning an Markov decision problem in higher dimensions. See section A.5
- Formulate a type of model based learner, which has some serious, but potentially solvable, issues with scaling to larger problems. See section A.4
- Explore a new task for understanding a learner's ability to generalise. See section B.4

Chapter 2

Markov Decision Problems

Reinforcement learning (RL) refers to the set of solutions to a type of problem. This general, reinforcement learning set, has two main properties; *"trial-and-error search and delayed rewards"* [4].

Unlike supervised learning, which gives the learner feedback (*Student: "I think that digit is a 5". Teacher: "No, it's a 6"*), in RL the learner only receives evaluations (*Student: "I think that digit is a 5". Teacher: "No."*). This means the learner needs to explore the possible answers via some trial-and-error search. (*Student: "Is it a 4?". Teacher: "No." Student: "How about a 0?". Teacher: "No." ... Student: "A 6?". Teacher: "Yes."*)

On top of terse teachers, many actions may be taken before any evaluation is received, thus requiring credit to be assigned to past actions, (*Student: "Is it a 4? How about a 0? A 6? Maybe a 7?". ... Teacher: "No".*) often leaving the learner wondering: *"what did I do to deserve this?"* (see pigeon superstition for an amusing example of credit assignment gone wrong [5]).

The above definition of reinforcement learning is quite general. There are many different dimensions to problems that require trial-and-error search and give delayed rewards. For example we could make a RL problem that is;

- Observable or un-observable [6]

- Deterministic or stochastic [7]
- Synchronous or asynchronous [8]
- Terminating or infinite [7]
- Discrete versus continuous [8]
- Given knowledge of the underlying model or not [9]

But, which setting should we study?[1]

A better question might be: *What is the simplest setting we can consider that still poses a challenge to the ML and / or RL communities?*

Markov decision problems (MDPs) appear to be a good candidate. Let's go through some definitions so we an more clearly understand how they can be used as a simple setting to analyse RL.

Formally, a MDP, which is a type of sequential decision problem, is defined as a tuple, $\{\mathcal{S}, \mathcal{A}, \tau, r, \gamma, d_0\}$. Where S is the set of possible states (*for example arrangements of chess pieces*), A is the set of actions (*the different possible moves, left, right, diagonal, L-shaped step, ...*), $\tau : S \times A \to \Delta(S)$[2]is the transition function which describes how the environment acts in response to the current state and to actions (*You play pawn to pawn to D4, in response your opponent moves, knight to D4, taking your pawn.*). Next is the reward function, $r : S \times A \to \mathbb{R}$, (*whether you won (+1) or lost (-1) the game*). Lastly, the policy, $\pi : S \to \Delta(A)$, is what the learner gets to choose, aka the learners strategy. It decides which action to take in different states.

The objective when solving a MDP is to find a policy that maximises the expected cumulative discounted reward V^π (aka the value). This can be written as, maximising the expected return.

[1] I will often start a chapter / section / paragraph with a question like this. These questions are not meant as research questions. Rather, they are designed to orient the reader.

[2] The notation $\Delta(S)$ represents a distribution over S.

$$V^{\pi} = \mathop{\mathbb{E}}_{\zeta \sim D(\pi,\tau)}[R(\zeta)] \qquad \text{(state value)}$$
$$\pi^* = \max_{\pi} V^{\pi}$$

Where, d_0 is the initial state distribution, γ is the discount rate, ζ collects the (s_t, a_t, r_t) triples of a game (aka trajectory or rollout) [3], $R(\zeta) = \sum_{t=0}^{H} \gamma^t \zeta_t^r$ is cumulative discounted reward (aka the 'return' of a single game), and D is the probability of a trajectory under the chosen policy and MDP.

$$\zeta = \{(s_t, a_t, r_t) : t \in [0, H]\} \qquad \text{(trajectory)}$$
$$D(\zeta, \pi, \tau, d_0) = d_0(\zeta_0^s) \prod_{t=1}^{\infty} \pi(\zeta_t^a | \zeta_t^s) \tau(\zeta_{t+1}^s | \zeta_t^s, \zeta_t^a) \qquad \text{(p}(\zeta)\text{)}$$

2.1 Sequential decision problems

A general intuition for the problem of solving a sequential decision problem is: actions (aka decisions) are made sequentially (*e.g. First we put on our socks then we put on our shoes*). These actions need to be conditioned on the current state of the world (*e.g. It is morning and time to go to work.*). The goal is to take actions that achieve higher rewards (*e.g. Lying in bed is quite rewarding...*). While instantaneous rewards are good, we really care about long term cumulative rewards (*e.g. Having a job and thus being able to afford a bed is more rewarding.*).

What does the M in MDP really mean?

When we say a decision problem is Markovian, we mean that the transition function generates a Markov chain [10]. The next transition step

[3] We allow ζ to be indexed by time and $\{s, a, r\}$. For example; $\zeta_t^s = s_t$.

depends only on the current state and action. It is invariant to any and all histories that do not change the current state. [4]

This is not to say that past actions do not effect the future. Rather, it is a special type of dependence on the past. Where the dependence is totally described by changes to the state, $s \in S$. We can return to chess for an example: in chess there are no hidden pieces, or private knowledge about the current state. I know everything there is to know.

2.2 Solving a MDP

What does it mean to solve a MDP?

A MDP is considered solved when we have found the 'optimal' policy. As above, the 'optimal' policy is the policy that gives the highest expected return (value). This notion of optimality is defined;

$$\pi^* : \quad V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall \pi \in \Pi \;\; \forall s \in S$$

But, how can we (efficiently) find the optimal policy?

If we randomly pick policies and evaluate them, we would need to test (in the worst case), all the deterministic policies, $\mathcal{O}(|A|^{|S|})$. However, we can use the Bellman equation to guide our search. The expected return can be rewritten in a recursive manner, to give the Bellman equation.

$$Q^{\pi}(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim \tau(\cdot|s,a)} [V^{\pi}(s')] \qquad \text{(Bellman equation)}$$

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [Q^{\pi}(s,a)]$$

[4] Or another way of saying the same thing, there is no hidden state that effects future transitions.

Where Q is the state-action values. The Bellman equation is sometimes written as an operator, T.

$$T(Q^\pi) = r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim \tau(\cdot|s,a)} \mathop{\mathbb{E}}_{a' \sim \pi(\cdot|s')} [Q(s', a')]$$

2.2.1 Complexity

How hard is it to find the optimal policy?

The complexity of estimating the value of a state-action under the optimal policy, ie solving the Bellman optimality equation, can be glimpsed if we unroll its recursive definition. Here we can see a series of nested maximisation problems, where the former maximisation problems are conditional on the results of the latter maximisation problems.

$$\begin{aligned} Q^\pi(s_0, a_0) = r(s_0, a_0) + \gamma \max_{a_1} \mathop{\mathbb{E}}_{s_1 \sim p(\cdot|s_0,a_0)} \Bigg[\\ r(s_1, a_1) + \gamma \max_{a_2} \mathop{\mathbb{E}}_{s_2 \sim p(\cdot|s_1,a_1)} \Bigg[\\ r(s_2, a_2) + \gamma \max_{a_3} \mathop{\mathbb{E}}_{s_3 \sim p(\cdot|s_2,a_2)} \Big[\dots \Big] \Bigg] \Bigg] \end{aligned}$$

For the final maximisation problem we need to find the best action ($|A|$) for each potential final state we might be in ($|S|$). Then we need to do this again for each maximisation problem (of which there are $|S|$). So the computational complexity is $\mathcal{O}(|S|^2|A|)$.

2.3 A tabular representation of MDPs

Back to constructing a simple RL setting.

Imagine a MDP that can be described with tables (aka arrays). A table of three dimensions can describe the transition probabilities, $P[s_{t+1}, s_t, a_t]$,

and a table of two dimensions can describe the rewards, $r[s_t, a_t]$: the states and actions act as indexes to locations in the tables. Let's formally define our tabular MDP. [5]

$$\begin{aligned}
\mathcal{M} &= \{S, A, P, r, \gamma\} && \text{(the MDP)} \\
S &= [0 : n-1] && \text{(the state space)} \\
A &= [0 : m-1] && \text{(the action space)} \\
\tau &\in [0,1]^{n \times n \times m}, \;\; \forall j, k : \sum_i \tau[i, j, k] = 1 && \text{(the transition fn.)} \\
r &\in \mathbb{R}^{n \times m} && \text{(the reward fn.)}
\end{aligned}$$

A result of this formulation is that we concisely write and solve the (Bellman equation). However, it should be noted that the ability to solve the Bellman equation analytically (via the value functional) does not allow us to solve for the optimal policy analytically. The value functional allows us to evaluate a single policy. To find the optimal policy, we many need to make many evaluations.

$$\begin{aligned}
V &= r_\pi + \gamma \tau_\pi V && \text{(tabular Bellman eqn)} \\
V &= (I - \gamma \tau_\pi)^{-1} r_\pi && \text{(Value functional)}
\end{aligned}$$

The values are written as a vector, $V \in \mathbb{R}^n$. The reward under a given policy is written $r_\pi[s, a] = \pi[s, a] r[s, a]$. And the transitions under a given policy is written $\tau_\pi[s', s] = \sum_a \tau[s', s, a] \pi[s, a]$.

An alternative derivation of the value functional, which is more verbose and more enlightening, can be found in A.1.

But why is the tabular MDP considered 'simple' enough?

[5] It should be noted that this tabular MDP setting ignores an important aspect of RL: exploration, estimation error.

Consider a MDP with deterministic actions, where $\tau(s_{t+1}|s_t, a_t) \in \{0, 1\}$. This RL problem can be efficiently solved by non-statistical methods: dynamic programming and related planning techniques [8]. This setting is too simple.

Rather, a MDP with stochastic actions, $\tau(s_{t+1}|s_t, a_t) \in [0, 1]$, seems to retain much of the complexity we care about: this setting does not allow efficient solutions via dynamic programming. Further, it can be approached with algorithms that are used for state-of-the-art deep RL such as; policy gradients [11] and Q-learning [12].

Before going further with our quest for efficient RL. Let's try to understand some properties of our setting, tabular MDPs.

2.4 The value function polytope

The Value Function Polytope [13] provides intuition about the structure of a MDP and the dynamics and complexity of solvers. Let's take a look: consider a two state, two action MDP.

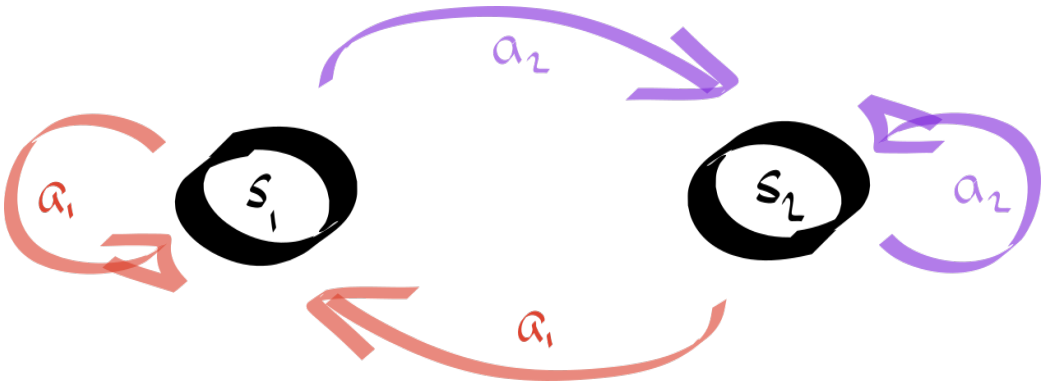

Figure 2.1: The simplest possible MDP has two states and two actions. (Any simpler setting is entirely uninteresting. A single state means actions do nothing. A single action means all policies are the same.).

The space of possible policies is a 2D surface in a 4D space. For each state, we can pick `action1` or `action2`, with some probability, p. For more intuition about this policy space see A.2. And for a method of visualising higher dimensional policies, see A.5.

$$\begin{aligned}\pi &= \begin{bmatrix} p(a=a_1|s=s_1) & p(a=a_2|s=s_1) \\ p(a=a_1|s=s_2) & p(a=a_2|s=s_2) \end{bmatrix} \\ &= \begin{bmatrix} p(a=a_1|s=s_1) & 1-p(a=a_1|s=s_1) \\ p(a=a_1|s=s_2) & 1-p(a=a_1|s=s_2) \end{bmatrix}\end{aligned}$$

Since the policies are a 2D space, we can visualise them. This square of all possible policies is not particularly interesting.

Rather, we can evaluate (calculate the expected return) each policy (using the (Value functional)). Since there are two states, the evaluation re-

turns a 2D vector of values, one value for each state. Therefore, we can visualise the value of each policy.

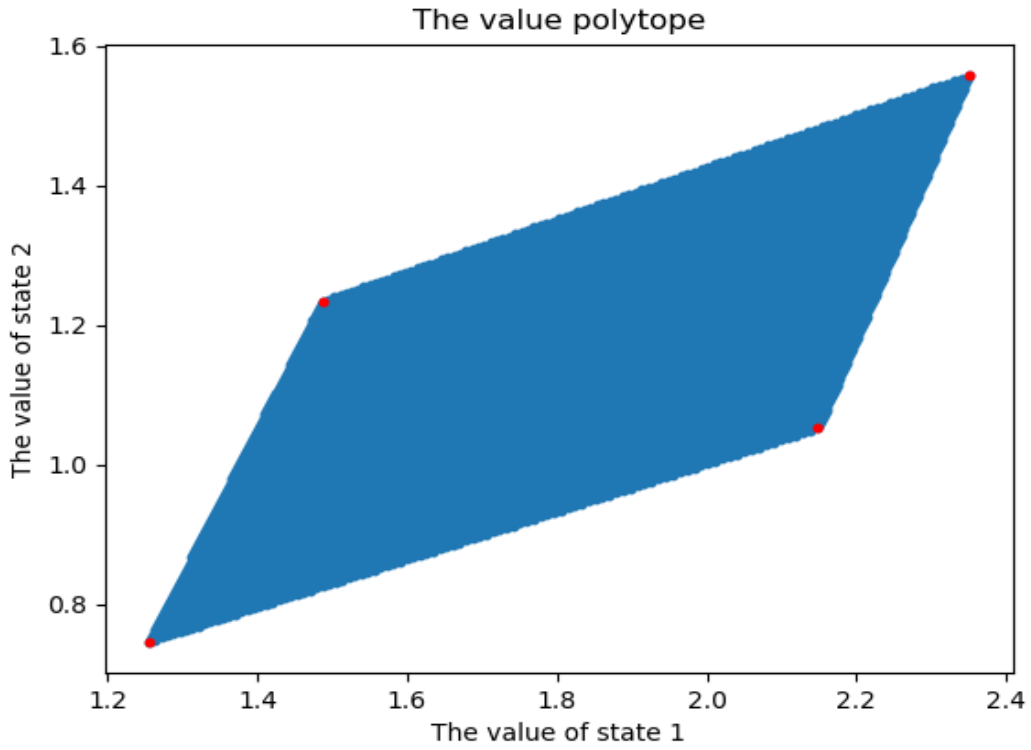

Figure 2.2: For every policy, we can plot a dot that represents the value of that policy. The red dots are deterministic policies.

Dadashi et al. [13] explored a few properties of the polytope. Specifically they focused on its geometry and dynamics.

Geometry of the polytope

Dadashi et al. remark; the polytope gives a clear illustration of the following classic results regarding MDPs [14].

1. (Dominance of V^*) The optimal value function V^* is the unique dominating vertex of V;
2. (Monotonicity) The edges of V are oriented with the positive orthant;

3. (Continuity) The space V is connected.

Let's try to understand these.

Dominance By the definition of the optimal policy, $\forall s : V^{\pi^*}(s) \geq V^{\pi}(s)$. Imagine there is some other policy, π' and state s', such that $V^{\pi^*}(s') \leq V^{\pi'}(s')$. This is a contradiction. Thus, the optimal value function must have the largest value in all states, meaning it will be the 'dominating' vertex.

Monotonicity If $V(s_2)$ increases then $V(s_1)$ must either increase or stay the same. This can be seen in the last equation below;

$$\begin{aligned} V(s_1) &= \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s_1)} r(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim\sum_a \tau(\cdot|s,a)\pi(\cdot|s)} V(s') \\ &= \sum_a \pi(a|s_1) r(s,a) + \gamma \sum_{s'} \sum_a \tau(s'|s_1,a)\pi(a|s) V(s') \\ &= \sum_a \pi(a|s_1) r(s,a) + \gamma \sum_a \tau(s_1|s_1,a)\pi(a|s) V(s_1) + \gamma \sum_a \tau(s_2|s_1,a)\pi(a|s) V(s_2) \end{aligned}$$

If $\sum_a \tau(s_2|s_1,a)\pi(a|s) = 0$ (i.e. s_1 and s_2 are not connected) then $V(s_1)$ stays the same, yielding a constant vertical line on the polytope. If $\sum_a \tau(s_2|s_1,a)\pi(a|s) > 0$ (i.e. there is some change of transitioning from s_1 to s_2) then $V(s_1)$ increases with $V(s_2)$, yielding a 'positive orthant'. $\sum_a \tau(s_2|s_1,a)\pi(a|s) < 0$ is not possible.

Continuity The policy space is connected by definition, and the value function is a continuous function. Therefore the value space, the polytope, is connected.

Dynamics on the polytope

Furthermore, Dadashi et al. [13] were interested in three aspects of different algorithms' learning dynamics;

- the path taken through the value polytope,
- the speed at which they traverse the polytope,
- any accumulation points that occur along this path.

They consider value iteration, policy iteration, policy gradients, entropy regularized policy gradients, natural policy gradients and the cross entropy method.

Their results are intriguing. They show that different RL algorithms traverse the polytope in vastly different ways. Some are not even constrained to the polytope. This raises the question;

> *How does a search algorithm interact with its search space to yield efficient search?*

In A.3 you can find further exploration of other properties of the value polytope, such as; the density of policies, and the effect of the discount rate.

2.5 Search spaces for MDPs

We want to efficiently find the optimal policy for a given MDP. But where and how should we search for this policy? We could search within;

- the set of potentially optimal policies, the $|A|^{|S|}$ discrete policies,
- the set of all possible policies $\pi \in \mathbb{R}^{|S|\times|A|} : \int_a \pi(a|s) = 1 \forall s$
- the set of possible state-action value functions, $\mathbb{R}^{|S|\times|S|}$, (which we could then use to construct the optimal policy),
- Or maybe some other space?

  Which space is best? Which space allows us to find the optimal policy in the 'cheapest' manner?

Naively, we often think smaller search spaces are better. We would rather search for our keys in a few rooms, rather than many. But added structure (for example, ordering) can be exploited to yield faster search, even when there are infinitely more states to search. For example, we might be able to order the rooms based on how recently we visited them. This should help us retrace our steps and find our keys, rather than arbitrarily picking rooms to search.

2.5.1 Policy search

We can search through policies. In my opinion, this is the most 'natural' type of search for RL. As, after all, we are searching for the optimal policy.

Searching through the space of policies supports a couple of modes of travel: policy iteration and policy gradients.

Policy iteration

In policy iteration (PI), we search for the optimal policy by evaluating our current policy and then acting greedily. In our tabular setting, policy iteration can be written as:

Algorithm 1 Policy iteration

procedure PI(τ, r, γ)
  $t = 0$
  $\pi_t \sim \mathcal{N}$ ▷ Initialise randomly
  while not converged **do**
    $V_t = (I - \gamma\tau_{\pi_t})^{-1} r_{\pi_t}$ ▷ Evaluate policy
    $Q_t = r + \gamma\tau \cdot_{s'} V_t$ ▷ Bellman operator
    $\pi_t = \text{greedy}(Q_t)$ ▷ Greedy update
    $t = t + 1$
  return π_t

The greedy operator picks the actions that give the highest state-action return, and sets their probability to be 1. $\text{greedy}(Q) = \text{onehot}(\text{argmax}_a Q[s, a], |A|)$.

This iteration converges because the state-action values capture counterfactuals, $Q^\pi(s, a)$. *What would the return be if I took an action, a, not necessarily chosen by the current policy, then I followed the current policy.* If there exists an action that achieves higher return than the current policy's choice of action, then (because of the greedy step) the PI updates to pick that action.

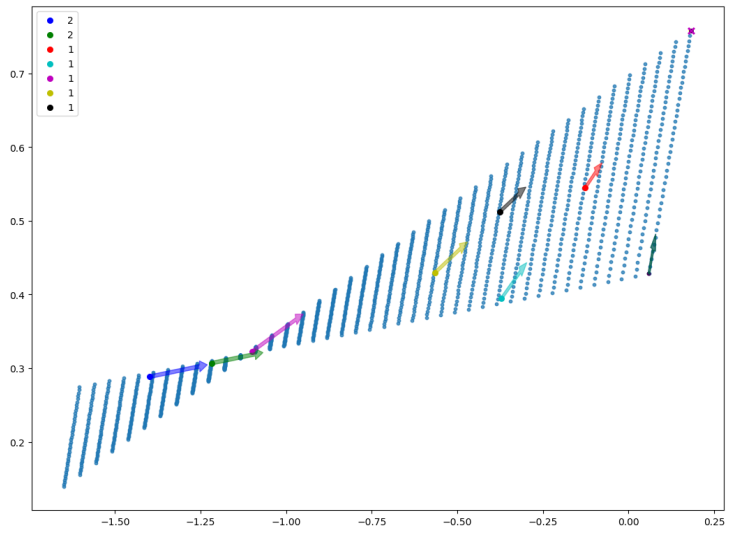

Figure 2.3: The optimal policy is shown by the cross. Each color shows PI applied to a different policy initialisation. The labels denote the number of iterations to reach convergence. The arrows point from the current to the next policy. As we can see, PI jumps between the deterministic policies (the vertices of our polytope). This is because of the greedy update step.

Policy gradients

Policy gradients (PG) are closely related to the deep learning / end-to-end paradigm. *Simply write down what you want (the loss function), estimate its derivative and apply gradient descent.* In our case, the loss function is the state value function. We can estimate the derivative by differentiating the Value functional with respect to the policy.

To ensure the optimisation problem is constrained properly (that the policy returns a distribution over actions), we pick θ as our parameters and construct the policy using $\pi = \sigma(\theta)$, where σ is the softmax function.

Algorithm 2 Policy gradients

procedure PG(τ, r, γ, η)
  $t = 0$
  $\theta_t \sim \mathcal{N}$ ▷ Initialise randomly
  while not converged **do**
    $\theta_{t+1} = \pi_t + \eta \nabla_\theta V(\sigma(\theta_t))$ ▷ Gradient update
    t += 1
  return $\sigma(\theta_t)$

To mitigate stability / convergence issues, it is common to add weak regularisation, which maximises the entropy of the policy. This forces policies away from the edges of the polytope, where gradients are not defined.

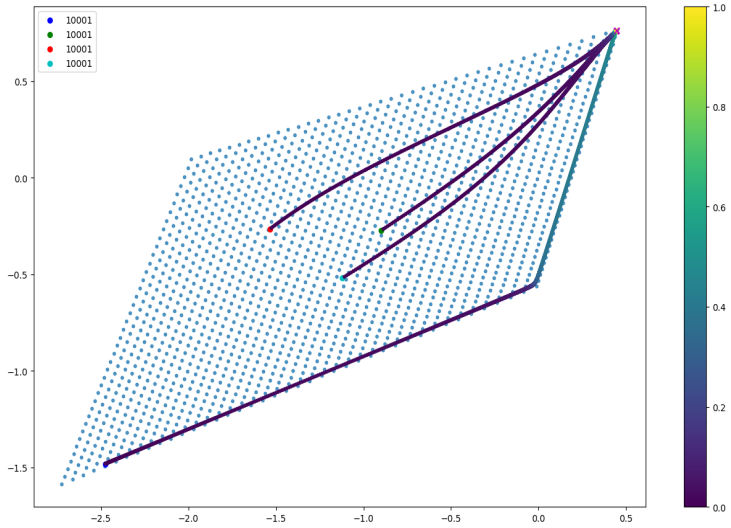

Figure 2.4: Examples of policy gradients: observe that most of the trajectories are purple, indicating that the majority of progress is made in the first 20% of training. They converge very slowly once close to the optimal policy.

2.5.2 Value search

Alternatively, we can search through possible state values, then infer the policy that achieves that value. But how can we ensure that our search will converge to a value that corresponds to a realisable policy? We can use Bellman's optimality operator to constrain the search.

(For similar reasons to why policy iteration converges) The greedy step using the state-action values will find actions with higher value.

Algorithm 3 Value iteration

procedure VI(τ, r, γ, η)
    t = 0
    $V_t = \mathcal{N}$ ▷ Initialise randomly
    while not converged **do**
        $\hat{V} = \max_a T(Q)$ ▷ Bellman optimality
        $V_{t+1} = V_t + \eta\hat{V} - V_t$ ▷ Average temporal difference
        $t+ = 1$
    $\pi = \text{argmax}_\pi \, r_\pi + \gamma\tau_\pi V_t$
    return π

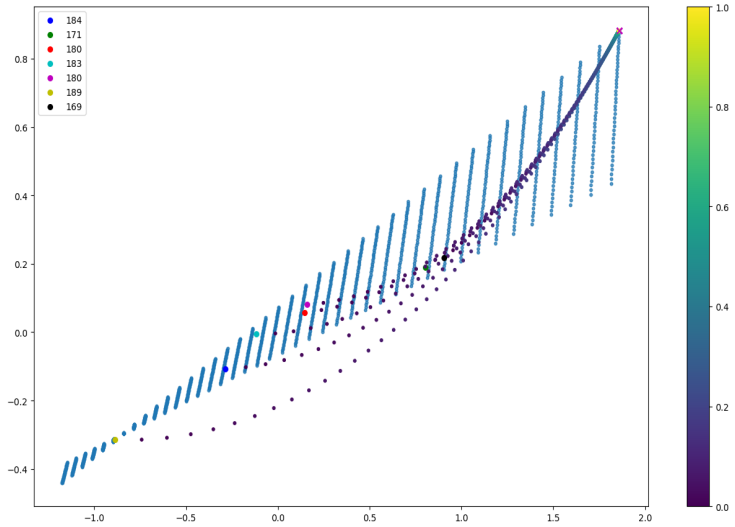

Figure 2.5: Observe that the value iterations are not constrained to map to a realisable policy (they can go out side of the polytope), but they do converge to a realisable policy, the optimal one.

Thus, there are different classes of search space: each imbued with special structure from the Bellman equation or expected return. Each with different types of search they support.

> *Which spaces support efficient search for the optimal policy? Can we characterise the properties of each space?*

For further exploration of search spaces (their iteration complexity and dynamics) see appendix A.6.

Chapter 3

Abstraction

What is an abstraction?

A recipe is an abstraction. For example, pumpkin soup:

1. Boil stock (6 cups), garlic (1 clove), thyme (0.5 tsp) and pumpkin (4 cups) in a pot for 30 mins.
2. Puree the mixture.
3. Simmer for 30 minutes and then add cream (0.5 cups).

This is an abstraction because it captures the essential details: from this recipe, you could make pumpkin soup. While it ignores inessential details: the recipe doesn't tell you; where or what to cook in, where to get the ingredients, which species of pumpkin to use, how to do the many actions needed to actually puree something, ... etc. It also doesn't mandate; the time of day to perform this recipe, what clothes to wear, the style of music.

We can make this notion more formal using, a homomorphism. Consider a ground space, X, (*the reality of what needs to be done. all the details*), and an abstract space Y (*our recipe*). We are looking for a transformation between X and Y preserves only the essential structure in X. This can be written as $f : X \to Y$ such that;

$$f(x \circ_G y) = f(x) \circ_A f(y)$$

Where the $\circ$ is the preserved structure.[1]

For example, we might want to preserve the ordering of objects $x \leq y \leq z$ (or the ordering of the steps in our recipe; boil, puree, then simmer). We can achieve this by picking $\circ$ (the preserved structure) to be the less-than operator, $\leq$. Thus, any homomorphism that preserves $\leq$ gives us an 'abstracted' description of our original set.

In general, there are three steps to using abstraction to help solve a problem:

1. Transform the problem to the 'abstract' domain. $f : X \to Y$
2. Solve the problem $\mathcal{S} : Y \to Z$
3. 'Lift' the solution back to the original domain $g : Z \to X$

3.1 Abstractions for RL

There are a few different types of abstraction that can be considered for RL: state abstractions [15, 16, 17, 18, 19, 20, 21, 22, 1], action abstractions [23, 24, 25, 26], state-action abstractions [27, 28], temporal abstractions [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44]. These abstractions are often built for two goals; efficient exploration (sample complexity) and / or efficient optimisation (computational complexity).

3.1.1 Exploiting abstractions

Given an abstraction, how might a learner use it?

[1] Something that might not be clear is that you need to be able to define $\circ$ on both X and Y. These definitions might be different from each other. For example one space might be continuous and the other discrete.

Within RL, there are broad classes of abstraction exploitation. To construct these classes, consider the central functions within RL; the state-action value function, Q, and the policy, π. How can we build an abstraction into these functions?

Let X, Y, refer to abstract spaces. We can generalise the notion of a state-action value function and policy to work on abstracted spaces. $Q_A : X \times Y \to \mathbb{R}$, $\pi_A : X \to \Delta(Y)$. We can now construct different classes of learners by giving them access to different types of abstraction.

Abstraction	**X**	**Y**	**Value fn**	**Policy**
State	$\phi : S \to X$	A	$Q(\phi(s), a)$	$\pi(a\|\phi(s))$
Action	S	$\psi : A \to Y$	$Q(s, \psi(a))$	$\pi(\psi^{-1}(y)\|s)$
State and action [2]	$\phi : S \to X$	$\psi : A \to Y$	$Q(\phi(s), \psi(a))$	$\pi(\psi^{-1}(y)\|\phi(s))$
State-action	$\varphi : S \times A \to X \times Y$		$Q(\varphi(s, a))$	$\text{argmax}_a\, Q(\varphi(s, a))$
Temporal (goal like)[3]	S	$f : S \to Y$	$Q(s, f(s))$	$\pi(f(s)\|s)$
Temporal (option like)	S	$g : A^\star \to Y$	$Q(s, g(\omega))$	$\pi(g^{-1}(y)\|s)$

Examples

State abstraction groups together 'similar' states. For example, a 20 NZD note. There are many different 20 NDZ notes that, for all intents and purposes (and actions), are the same. Someone may have written a message on the note, the note may have been folded, it might have different serial codes, it might be an older version, ... etc.

Rather than describing these inessential details, we can abstract away from them.

[3] If the Q fn is a linear function of the $\varphi(s, a)$ representation, then this is known as the successor representation [27, 28]

[3] Goal like abstraction is actually a lot more general than is. But, it is easiest to understand when goals are picked to be states.

Figure 3.1: A 20 NZD note.

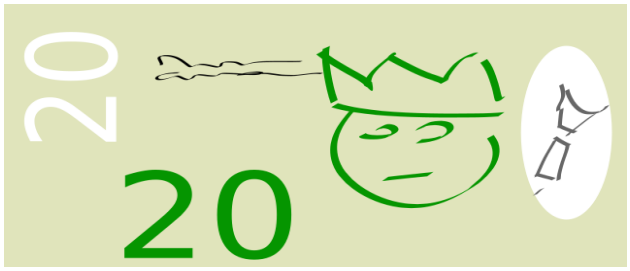

Figure 3.2: An artist's abstract rendering. This state abstraction only shows the essential details; 20, New Zealand, ...

Action abstraction groups together 'similar' actions. For example, both action i and action j might yield the same rewards and change in state, in which case, we can just relabel them as the same action.

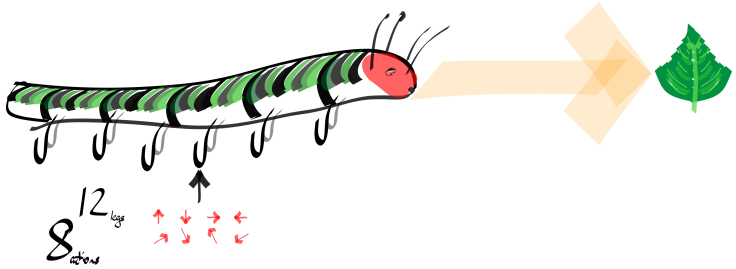

Figure 3.3: Consider a hungry caterpillar. It wants to move towards the leaf. But this is a complicated task! Move your 3rd leg up, your 7th leg down, 11th leg slightly forward, etc... To specify an action it needs to pick from 8^{12} possible actions - 8 actions per leg, 12 legs.

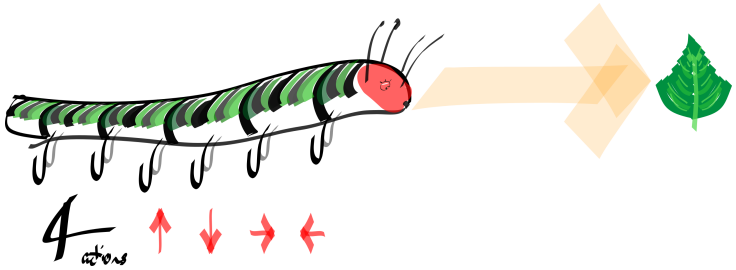

Figure 3.4: But, what if the caterpillar could specify actions in another way? Rather than specifying leg movements, it could pick a direction to move, which would specify the movement of multiple legs. Note, we are not including any temporal abstraction here.

State-action abstraction groups together 'similar' state-actions. This class of abstraction should be more powerful than state and action abstraction. Consider a mirror symmetric maze.

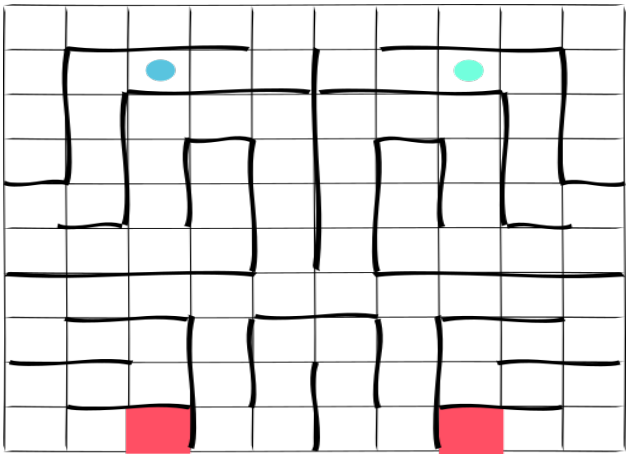

Figure 3.5: A symmetric maze, the goal(s) are shown in red. Consider two different, but 'similar' positions, shown in blue and green.

Given this setting, we can construct a representation of the state [4]where, we have $Q^{\pi}(s,a) = Q^{-\pi}(-_x s, -a), \forall \pi$. Where the negative operator $-_x$ only effects the horizontal (x) axis $s = (x,y), -_x s = (-x,y)$. And $-\pi :=$

$\pi(-a| -_x s)$. [5]

State abstraction would not be able to group together the blue and green positions. As moving left from blue is not equivalent to moving left from green.

Temporal abstraction groups together 'similar' goals / options. For example, the pumpkin recipe above: for the many ways there are to boil broth (in pot, in a jug, over a wood fire, in the oven, ...), group them together. For the many ways to puree pumpkin (with a rolling pin and some vigour, with a blender, ...), group them together.

The intuition behind both goal-like and option-like temporal abstractions is that we can decompose a sequential decision problem into a set of shorter, easier to solve, parts.

3.1.2 Evaluating abstractions

> *Which class of abstraction should we use? Which class of similarity measure should we use? We need to be able to evaluate choices.*

Ideally, we would pick the most 'coarse' abstraction, as a coarse abstraction means we have grouped together many objects. Thus it has potential for the greatest increases in computational and sample efficiency. However, coarseness is not the only property of an abstraction we care about. An abstraction must also allow us to represent near optimal policies and it must allow us to efficiently find those near optimal policies.

[5] The trade off is that with state-actions, now we need to keep track of $(|S| \times |A|) \cdot (|S| \times |A| - 1)$ similarities, rather than $|S| \cdot (|S| - 1) + |A| \cdot (|A| - 1)$ similarities.

[5] To construct this representation, we set the state to be centered around the mirror plane. So the blue player is at location $(-3, 9)$ and the green player is at location $(3, 9)$. We equip the players with the actions $(+1, 0), (-1, 0), (0, -1), (, +1)$ (aka; up, down, left, right).

Ideally, we would be able to summarise the 'coarseness', 'sub-optimality' and 'efficiency' all in a single number. Then we could try to optimise it. But for now, let's define these constraints on 'good' abstractions.

Coarse abstractions

Coarse (as opposed to fine) is a notion from topology. A (more) coarse abstraction describes an abstraction that has a looser notion of similarity (more things are considered similar). [6]

We say [state abstraction] ϕ_2 is coarser than [state abstraction] ϕ_1, denoted $\phi_1 \geq \phi_2$, iff for any states $s_1, s_2 \in S, \phi_1(s_1) = \phi_1(s_2)$ implies $\phi_2(s_1) = \phi_2(s_2)$ [16]

Any time s_1 and s_2 are considered similar in an abstraction, then a coarser abstraction will also consider them to be similar, and there might also be other similar states[7].

Why are coarse abstractions desirable?

By grouping together more states, there fewer states over all, and thus;

- there is less to compute,
- there is less to explore,
- there is less variance in our observations and thus allow quicker learning.

These arguments have been formalised in the case of symmetries and are described in B.3.1.

[6] Coarseness captures concepts such as a 'high level' abstraction or a 'general' abstraction.

[7] So the most coarse abstraction would be where everything is mapped to the same abstract state. $\forall s_i, s_j \in S : \phi(s_i) = \phi(s_j)$.

Near optimal abstractions

Imagine we have a state abstraction, a road is a road, there is no real difference between them; gravel, winding, motorway, cliffs-on-either-side... One of the first things we want to know about the abstraction is: is it possible for me to act optimally using this abstraction? If not, what's the damage? In this example, does driving 100kph on every road – because all roads are pretty-much-the-same – lead to suboptimal results? Probably. More precisely, we want know to whether the optimal policy can be approximately represented within an abstracted MDP.

This notion of near-optimality, ϵ, (aka sub-optimality) can be formalised as the representation error of the optimal policy. [22] Given an abstraction, we want to know how well the abstraction can represent the optimal policy.

$$\forall_{s \in S} \mid V^{\pi^*}(s) - V^{\pi^*_A}(s) \mid \leq \epsilon \tag{3.1}$$

Where π^* is the optimal policy, and π^*_A is the lifted optimal policy found using the abstraction.

This notion can be generalised to other types of abstraction, for example Nachum et al. 2018 [44] extend this concept of near optimal representation of the optimal policy to goal-like temporal abstraction[8].

[8]To do this, Nachum et al. 2018 [44] quantify how many of the possible low level behaviours can be produced by different choices of goal. This is roughly a measure of the invertability of the 'low level' policy from state-goals to temporally extended actions. This allows them to reason about which behaviours are representable using goals, and thus whether a 'high level' policy choosing these goals can approximate the optimal policy.

Efficiently solvable abstractions

Just because a near optimal policy exists (under our abstraction), that does not mean it will be easy to find. We might require large amounts of data or computation.

Efficient control: *What is needed for efficient control? And, when is 'it' preserved?*

We want to preserve our ability to use the Bellman equation to efficiently guide the search for optimal controls. But, what is necessary or sufficient to preserve the Bellman equation's ability to guide search? It appears that: Preserving the value of the optimal action is sufficient to preserve the Bellman equation's ability to guide search[45, 16, 22].

Consider an abstraction that has preserved the value of the optimal action. When constructing the abstraction, we may have grouped together states that don't have similar transitions, or similar value under all policies, ... etc. For example we might have: $s_i \sim s_j$ despite $Q(s_i, a_k) - Q(s_j, a_k) > \epsilon, a_k \neq a^*$. But this doesn't matter as $Q(s, a^*)$ must still be larger than $Q(s, a_k)$. Thus, when applying a greedy update to the Bellman equation, we will be pull towards the 'true' value of the optimal policy[9].

Efficient exploration: *What is needed for efficient exploration? And, when is 'it' preserved?*

For Q-learning, Jin et al. [46] show that efficient exploration ($\mathcal{O}(\sqrt{H^3SAT})$)[10]can be achieved with a UCB-Bernstein type exploration bonus, compared to the inefficient exploration ($\mathcal{O}(\sqrt{H^4SAT})$), achieved by the UCB-Hoeffding type exploration bonus. The details of these algorithms is not too important in this discussion. The important part is that: the UCB-Bernstein type exploration bonus needed to incorporate an estimate the variance of the

[9]However, this argument does not take into account how the topology of the Q values may have changed.

values, while the Hoeffding exploration bonus relies only on visitation counts.

This tells us that, if we want to preserve our ability to efficiently explore (in the sense above) we need to preserve the ability to accurately estimate the variance of the value function.

The complexity of abstraction

We hope to build learners capable of discovering and exploiting an abstraction to efficiently solve the problems they are given. But, the efficiency gains of using an abstraction (as measured by its evaluation, see 3.1.2) must offset the cost of discovering that abstraction! Else, why did we bother...?

> *"The challenge in the single-task case is overcoming the additional cost of discovering [an abstraction]; this results in a narrow opportunity for performance improvements, but a well-defined objective. In the ... transfer case, the key challenge is predicting the usefulness of a particular [abstraction] to future tasks, given limited data."*[47]

For arbitrary problems, as often as we guess right about "the best abstraction for this task" or "the usefulness of a particular abstraction to future tasks", there will be another set of future tasks where we are wrong. This is a result of the no free lunch theorem [48]. However, we can hope for good performance on restricted classes of problems.

To evaluate an abstraction for the single task case we need to compare the complexity of finding a solution via abstraction (abstract, solve, lift) against the performance against solvers that do not abstract.

[10] $\mathcal{O}(\sqrt{H^3SAT})$ tells us that the expected regret scales, in the worst case, as a function of $\sqrt{H^3SAT}$. Where H is the planning horizon, S is the size of the state space, A is the size of the action space and $T = KH$, where K is the total number of episodes played.

In the transfer case, we need to construct a set of tasks. Then we accumulate; the complexity of discovering the abstraction, and the complexity of solving and lifting the abstraction for each new task.

3.1.3 Discovering abstractions

Where do abstractions come from?

Earlier, we considered how an agent might use a given abstraction. But, where did that abstraction come from? Did someone construct it? Or can abstractions be discovered automatically?

To build an abstraction, you need a notion of how two objects might be similar (or a notion of what it is you want to preserve). This allows you to group the 'similar' objects together, building an abstraction. In RL there are many possible notions of similarity. Which ones are sensible? Which ones allow us to efficiently find the optimal policy?

Classes of similarity for RL

Li et al. [16] give five classes of state abstraction in which they group states based on various similarities [11];

intuitively, ϕ_{model} preserves the one step model, ϕ_{Q_π} preserves the state-action value function for all policies, ϕ_{Q^*} preserves the optimal state-action value, ϕ^*_a preserves the optimal action and its value, ϕ_{π^*} preserves the optimal action.

We can construct a family of similarity functions for building abstractions for RL. To do so, we can measure the similarity between two 'RL

[11] Similarity and preservation are dual notions. Similarity tells us which objects we can group together, 'preservation' tells us which objects we group together to preserve (for lack of a better word...) a notion of similarity. For example, the *"preservation of the one step model"* means: two states can be considered similar if they have the same one step model.

objects'[12]as: for different objects, x and x', how similar are their likely futures starting from those objects. Where we evaluate the similarity under (potentially) many different policies. Let's make this formal.

Let $c(s, a, r)$ be a cumulant, and let C be the discounted sum of that cumulant under a given trajectory[13]. Then $\mathcal{C}$ is the expected value of C under a given policy.

$$
\begin{aligned}
C(\zeta) &= \sum_{t=0}^{\infty} \gamma^t c(\zeta_{t+1}) \\
\mathcal{C}(x, \pi) &= \mathop{\mathbb{E}}_{\zeta \sim D(\cdot|\pi,x)} [C(\zeta)] \\
\chi(x, x') &= \int_{\pi \in B} e^{-\|\mathcal{C}(x,\pi) - \mathcal{C}(x',\pi)\|_2} d\pi
\end{aligned}
$$

Where we construct $D(\cdot|\pi, x)$ as the probability distribution over trajectories, given that we started at x and followed π.

Examples Consider: an abstraction that preserves the n-step model, and another that preserves the value of the optimal state-actions. We show that these abstractions are captured by our family 1.

Preserving the n-step model: We start with $x = (s, a), x' = (s', a')$. Set the cumulant be the next states and the rewards, $c(\zeta_t) = [s_{t+1}, r_t]$. Then the expected discounted cumulant, $\mathcal{C} = [N, V]$, represents the discounted state visitation count, N, and the state value V. if x and x' have the same state visitation and value for all policies must have the same transitions[14].

Preserving the optimal state-action value: Again, we start with $x = (s, a), x' = (s', a')$. Next, we pick $c(\zeta_t) = r_t$ which means $\mathcal{C}(x, \pi) = Q^\pi(s, a)$. Finally,

[12]Where RL objects could be one of; states, actions, state-actions, action-rewards, or state-action-rewards, ... etc.

[13]This is is closely related to the notion of a general value function [49]

[14]This statement assumes we have encoded the states as onehots. Also, it does not seem so obvious that it doesn't need proof.

we only integrate over the optimal policy. Thus, x and x' are similar if $Q^\pi(s,a) \approx Q^\pi(s',a')$.

Inference of the similarity

How might we infer that two objects are similar?

In the RL setting, we are often not given the transition or the reward function. Thus, we must infer similarities from (noisy) observations. There are two ways we can do this;

We can infer that two objects are similar because we have observed it, we have sufficient data (/ experience) to allow us to conclude that, with high certainty, x_i and x_j are similar (with probability δ, $f(x_i) - f(x_j) \leq \epsilon$).

Alternatively, we can infer that two objects are similar because we have found a pattern. For example, we might have seen that rotations of $45, 90, 135, 180, 225$, are all similar, therefore we guess that rotations of $270, 315$ are also similar. This motivates our work on symmetric abstractions (in section3.3). Because of the structure in a symmetry, some similarities can imply others. Thus we can generalise.

Note: if we are relying on inferring similarity from data, and (say) $f(x_i) = Q^{\pi^*}(s_i, a_i)$. Then learning that x_i and x_j are similar doesn't help. Because, to estimate that x_i and x_j are similar, we needed to know, $Q^{\pi^*}(s_i, a_i)$ and $Q^{\pi^*}(s_j, a_j)$. Rather, we care about (correctly) guessing that x_i and x_j are similar, so we can exploit that knowledge that to improve efficiency, for example, we want to use the knowledge of $Q_i = Q^{\pi^*}(s_i, a_i)$ to help us learn $Q^{\pi^*}(s_j, a_j)$.

3.2 Efficiently solvable abstractions

Abstractions for efficient control.

In optimisation for supervised learning we know that linear models can be solved analytically and convex problems converge quickly. What makes an RL problem easy to solve? Is there linearity or convexity within RL problems that we can use to find solutions more efficiently?

While some interesting work has been done exploring convex RL [50, 51], we choose to explore linear RL.

3.2.1 Linear Markov decision problems (LMDPs)

Why linear MDPs?

Imagine if your life were linear, in the sense of effort and achieving goals. More work equals proportionally more rewards. This makes decision making a lot more simple. Pick the most rewarding activity, and work hard.

What do we mean by a linear MDP?

Do we mean that the value function is a linear function of (say) a policy? Of the reward function? Of the transition function? Previous research has tried to incorporate linearity into MDPs in a few different ways.

- Todorov constructs a linearised MDP that is linear in the 'control' (a proxy for the policy) [52].
- Jin et al. construct a MDP where the value is a linear function of a state-action representation and of a policy embedding [53].
- Levine et al. construct a learner that assumes a linear transition function, allowing the use of LQR solvers [54].
- Pires et al. construct a factored linear MDP that allows the TD operator to be applied in a lower dimensional space [55].

Todorov's formulation appears to be the most powerful. By having a linear relationship between the value and the control, we can easily search for controls that are optimal. For the rest of this work we will refer to these as LMDPs.

Constructing a LMDP

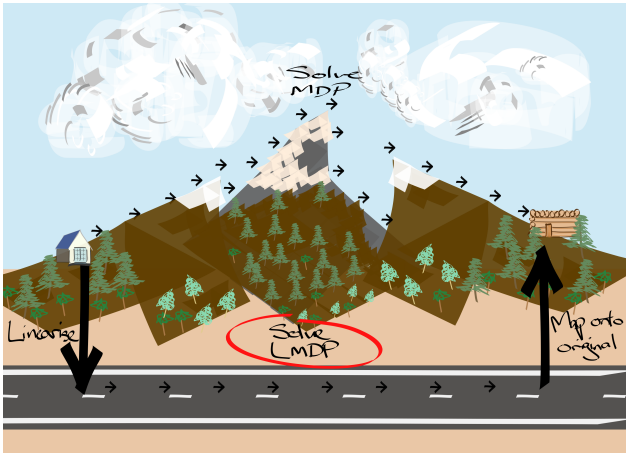

Figure 3.6: Solving the LMDP

To build a LMDP that acts similarly to a MDP, Todorov [52] incorporate a few changes to the typical MDP setting;

- allow the agent to pick actions in the space of possible transitions, which they name 'controls', $u : S \to S$.
- ensure that chosen controls are possible under the given transition function, a new reward is added. Controls are rewarded if they are close to the 'unconstrained dynamics', $p(s'|s)$.
- maximise the exponentiated cumulative rewards. $\max_u \mathbb{E}_{\zeta \sim D(u)}\, e^{R(\zeta)}$ [56] (rather than maximizing the cumulative reward).

The intuition behind these changes seems to be; we have allowed the learner to pick an arbitrary control. This simplifies the optimisation problem. But, now the learner might pick a control that is not possible under

the given transition function. So we incentivise controls that are close to the underlying dynamics.

> *"It is reminiscent of a linear programming relaxation in integer programming"* [57].

Putting these changes together, a linear Markov decision process can be formulated as; LMDP $= \{S, U, p, q, \gamma\}$. Where S is the state space, U is the space of possible controls (i.e. any transitions), $p : S \to S$ is the unconditioned dynamics, and $q \in \mathbb{R}^{|S|}$ is state rewards. Our goal is to find the control, $u : S \to S$, that gives the highest exponentiated cumulative reward $z \in \mathbb{R}_+^{|S|}$.

$$\log z_{u^*}(s) = \max_u q(s) - \mathrm{KL}(u(\cdot|s) \parallel p(\cdot|s)) + \gamma \mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \log z_u(s') \tag{1}$$

$$u^*(\cdot|s) = \frac{p(\cdot|s) \cdot z_{u^*}(\cdot)^\gamma}{\sum_{s'} p(s'|s) z_{u^*}(s')^\gamma} \tag{2}$$

$$z_{u^*} = e^{q(s)} \cdot p z_{u^*}^\gamma \tag{3}$$

By definition, a linearised Bellman equation is constructed as (1). After some algebra, it can be shown that the optimal policy has the form in (2). Most importantly! We can solve for the value of the optimal control by solving the linear equation in (3). (see B.1.1 for further explanation and a derivation of a LMDP).

Let's try and understand this LMDP that has been constructed.

The unconstrained dynamics and state rewards

What do p and q do?

Rather than state-action rewards $r : S \times A \to \mathbb{R}$, we have state rewards $q : S \to \mathbb{R}$. How can state rewards direct behaviour? They can't, and they can. State rewards are not capable of directly giving rewards for actions

taken. Rather, the action dependent part of the reward is captured by the KL divergence between the control and the unconstrained dynamics (this relationship is shown in rewards). But, they do implicitly direct behaviour through their presence in the exponentiated value function z_{u^*}.

While it may seem intuitive to think of the unconstrained dynamics as the expected result of using random policy $p(s'|s) = \int_a \tau(s'|s,a)U(a|s)$ (a random walk using the transition function). This turns out to be wrong. The unconstrained dynamics are responsible for rewarding actions.

The optimal policy

What can we interpret about the form of the optimal control?

$$u^*(\cdot|s) = \frac{p(\cdot|s) \cdot z_{u^*}(\cdot)^\gamma}{\sum_{s'} p(s'|s) z_{u^*}(s')^\gamma}$$

Interpreting the equation above, the optimal control is the control that transitions to new state, s', with a probability proportional to the discounted exponentiated value of that state, $z(s)$. That seems reasonable, and is closely related to the RL as inference framework [58], which picks actions with probability proportional to their exponentiated Q values.

$$\pi(\cdot|s) \propto e^{Q(s,\cdot)}$$

Solving for the optimal value

What can we interpret about the form of the value estimate?

$$z_{u^*} = e^{q(s)} \cdot p z_{u^*}^\gamma$$

In the undiscounted case (aka first exit case), where $\gamma = 1$, this turns into an eigen value problem $z = QPz = Az$, where $Q = e^{q(s)}$. As these are linear problems, they are efficiently solvable. Similarly, in the discounted case, the exponentiated value, z, is guaranteed to converge quickly [57].

3.2.2 Solving a MDP

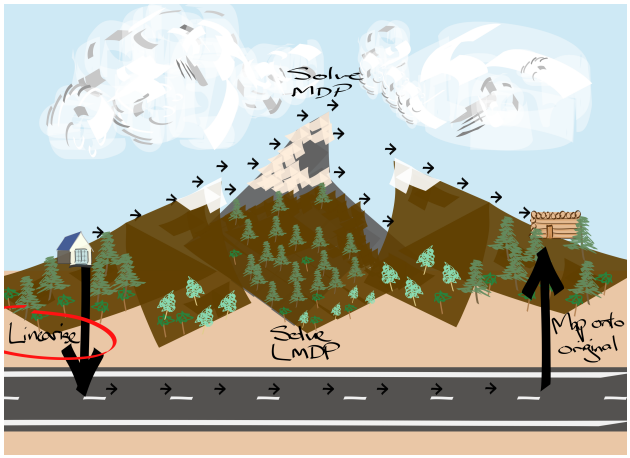

Figure 3.7: Solving a MDP

So, LMDPs can be easily solved. But how does solving a LMDP help us solve a MDP?

We need a way to transform a MDP into a LMDP, while preserving the 'structure' of the original MDP. But what do we mean by a MDP's structure? According [57], the LMDP should be able to represent the same transition dynamics as the original MDP, and give the the same rewards was the original MDP.

$$\forall s, s' \in S, \forall a \in A, \exists u_a \text{ such that;}$$

$$\tau(s'|s,a) = u_a(s'|s)p(s'|s) \qquad \text{(transition dynamics)}$$

$$r(s,a) = q(s) - \mathrm{KL}(\tau(\cdot|s,a) \parallel u_a(\cdot|s)) \qquad \text{(rewards)}$$

So, given a reward function, r, and a transition function, P, (from an MDP), we can use the transition dynamics and rewards to solve for the unconditioned dynamics p and a state reward q. This transformation, from $P, r \rightarrow p, q$ requires $|S| \times |A|$ computations, as for each state in the MDP we need to satisfy constraints for each action. For a derivation and explanation see appendix B.1.2.

However, an important condition is missing! We should preserve the value of optimal actions (as discussed in 3.1.2). Alternatively, we might settle for preserving the optimal policy.

$$\begin{aligned} &\forall s, s' \in S, \forall a \in A, \forall \pi \in \Pi \\ &z_{u_\pi}(s) = e^{V_\pi(s)} && \text{(value)} \\ &\tau(s'|s,a) \cdot \pi^*(a|s) = u^*(s'|s) && \text{(optimal policy)} \end{aligned}$$

Where z_{u_π} is the value (as evaluated by the LMDP) of the control $u_\pi(s'|s) = \tau(s'|s,a) \cdot \pi(a|s)$, and V_π is the expected return of policy π as evaluated by the original MDP.

It seems that Todorov hopes that the constraints transition dynamics, rewards, will be sufficient to give value, optimal policy. But he does not prove it. This is a problem that we will return to in 3.2.3.

3.2.3 Lifting the optimal LMDP policy

> *How can we use the optimal control, u^*, to construct a policy that is optimal in the original MDP, π_{u^*}?*

The LMDP has disentangled the search for the optimal controls (solve the LMDP) (go to this or that state) from the search for the optimal policy (how to actually realise that control).

This decomposition is reminiscent of goal conditioned approaches to hierarchical RL. Where a higher level agent gives goals (go to this or that

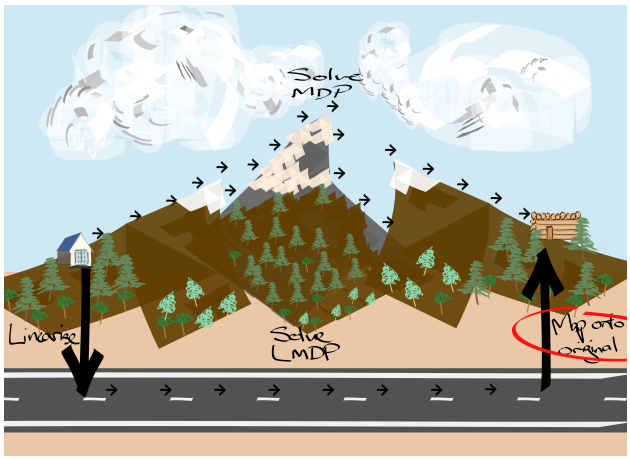

Figure 3.8: Lifting the LMDP's solution to the MDP

state) to a lower level agent, who must figure out how to achieve that goal [59].

This decomposition within the LMDP can be seen as the optimal control decoding step (currently being considered). We know which states we want to be in, the z^*/u^*, from solving the LMDP, but, we do not know how to implement those controls with the actions we have available in the original MDP. We can formulate this problem as;

$$\tau_\pi(\cdot|s) = \sum_a \tau(\cdot|s,a)\pi(a|s)$$
$$\pi = \underset{\pi}{\text{argmin}}\, \text{KL}\Big(u(\cdot|s)) \parallel \tau_\pi(\cdot|s)\Big)$$

We would hope that the knowledge of u^* would make it easy to find the optimal actions. But, this optimisation problem has very little structure for us to exploit. In the worst case it has complexity $\mathcal{O}(|S|^2|A|)$ (which is the same as solving a MDP).

We have now built a way to solve a MDP with a LMDP. We have a method to transform a MDP to a LMDP 3.2.2. We can solve the LMDP

3.2.1. Finally, we can lift the solution to the original MDP 3.2.3. Let's try it out.

Optimality of solutions via LMDPs

Do these two paths lead to the same place?

One of the main questions we have not addressed yet is; if we solve the MDP directly, or we linearise then solve then lift, do we end up in the same place? This is a question about the sub-optimality of our abstraction 3.1.2. Can our abstraction approximately represent (and find) the same solutions that the original can? We can formalise this question as the difference between the optimal values or optimal policies.

$$\epsilon = \| \ Q_{\pi^*} - Q_{\pi_{u^*}} \ \|_\infty$$

We answer this question using experiments, described in figures 3.9 and 3.10.

We also investigate the sensitivity of the LMDP's optimal control given different MDPs. The optimal control is very sensitive to the sign and magnitude of the rewards, as can be seen in 3.11 and 3.12.

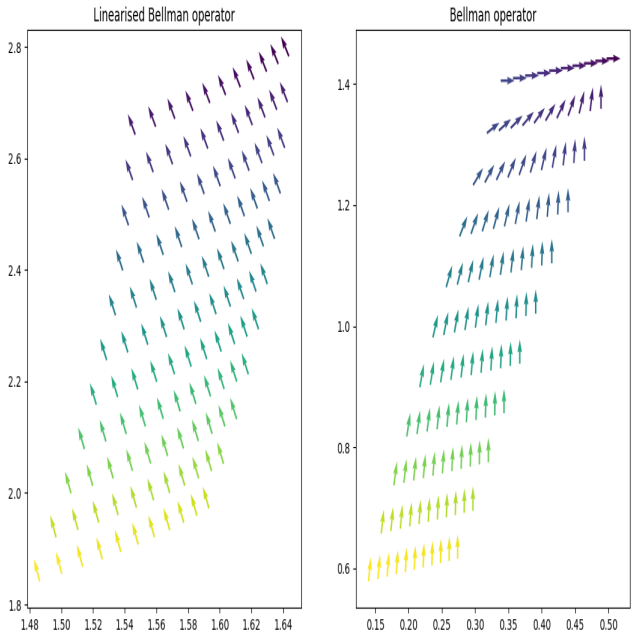

Figure 3.9: For the same MDP, shown is a comparison of the linear temporal difference operator (left), versus the true, Bellman, temporal difference operator (right). As expected, the temporal difference operator points towards the optimal value. But, the linear temporal difference operator points elsewhere.

3.2.4 Discussion

Unfortunately, Todorov [52, 57] never actually produced any experiments that use the linear bellman equation to solve a MDP. His experiments were with Z-learning, the linearised counterpart to Q-learning.

$$\tilde{z}(s) = e^{q(s)} z(s')^{\gamma} \qquad \text{(linearised bellman eqn)}$$

$$z_{t+1}(s_i) = (1-\eta) z_t(s_i) + \eta \tilde{z}(s_i) \qquad (Z \text{ iteration})$$

He makes this work by hand picking q so that the optimisation will converge to the optimal value. Then he evaluates the accuracy of the value estimates. They do not give a general way of constructing q from a MDP that actually works (in the sense of yielding the same optima). Nor do they give an efficient way to map an optimal control to an optimal policy.

Because there are no benchmarks that we can validate our implementation on, there is some doubt about whether I implemented the LMDP correctly. While the LDMP solution strategy does occasionally pick the correct optimal policy, more often than not, its outputs are seriously wrong.

Linearity in MDPs

Todorov's formulation of LMDPs does not seem to allow a MDP to be to reliably embed a LMDP. Despite this, it might still be possible to construct a linear MDP given different assumptions.

Let's reconsider Jin et al.'s approach [53]: construct a MDP where the value is a linear function of a state-action representation and of a policy embedding. It is important to note that, not every (policy) embedding can be realised as a policy, so to constrain the search dynamics properly we must use the Bellman equation.

And this captures the essence of the problem. We need the Bellman equation for efficient search, but it is not linear in the policy. So, we can linearise a MDP, but we will lose the ability to use the Bellman equation to guide the search for the optimal policy.

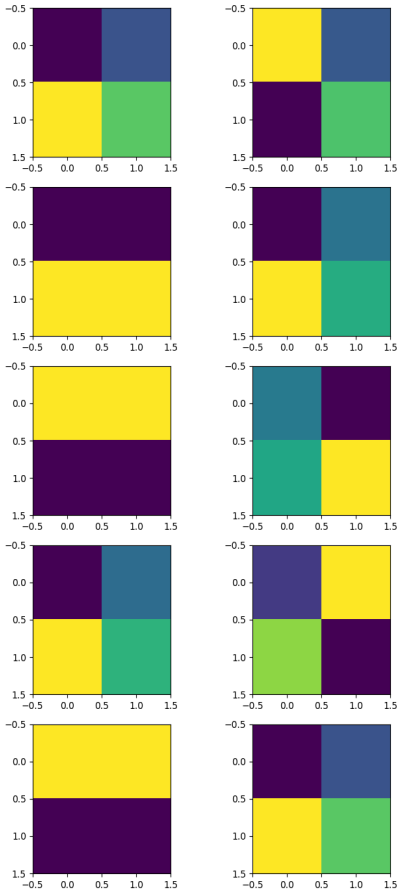

Figure 3.10: A comparison between the optimal control $u^*(s'|s)$ (shown in the left column) and the dynamics of the optimal policy (MDP) $\sum_a \tau(s'|s,a)\pi^*(a|s)$ (shown in the right column). Because the optimal control, and dynamics of the optimal policy can be written as 2×2 matrices, we can plot them. Yellow indicates high probability, purple indicates low probability. But the main thing to observe is that the optimal control is often not the same as the dynamics of the optimal policy.

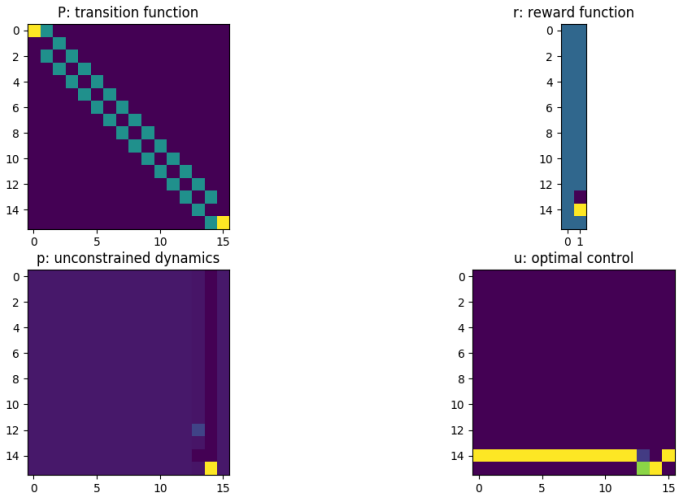

Figure 3.11: A chain problem [60] with zero reward on all states except the last two. A small negative reward, then a large positive one. The optimal control to this problem is not realisable: in every state, jump to the state with positive reward.

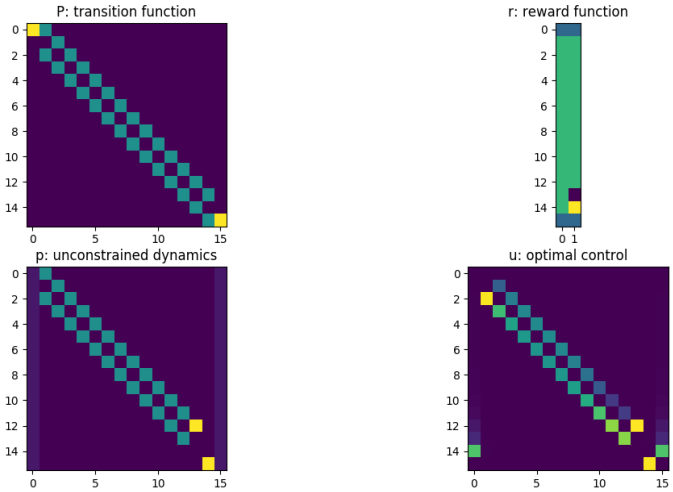

Figure 3.12: The same chain problem as above, but with an positive reward added to all states. We can see that the structure of the transition function has now made it into the optimal control! The LMDP's control can be decoded into a realisable policy.

3.3 Symmetric abstractions

Symmetry is a concept from pure mathematics, which has found major success in physics (for example, see Noether's theorem). Symmetry has also had success in machine learning. For example, the notion has led to; data augmentation strategies [61], understanding of convolutional neural networks [62], and a definition of disentanglement [63]. We are interested in using symmetry to build abstractions for RL. But first, what do we mean by symmetry?

Definition

We say that an object is symmetric if it has 'group' structure.

A group is a set, G (*say the set of rotations,* $\{0, 90, 180, 270\}$), and an operation $\circ$ that combines any two elements a and b to form another element (*rot* 90 *composed with rot* 180 *is rot* 270*, or* $a \circ b = a + b \bmod 360$). To qualify as a group, the set and operation, $(G, \circ)$, must satisfy four requirements;

- **Closure:** For all $a, b \in G$, the result of the operation $a \circ b$ is also in G. (*every composition of two rotations, must also be a rotation*)
- **Associativity:** For all $a, b, c \in G$, $(a \circ b) \circ c = a \circ (b \circ c)$.
- **Identity element:** There exists and element $e \in G$ such that, for every element $a \in G$, the equation $e \circ a = a \circ e = a$ holds. (*there must exist a rotation that doesn't rotate*)
- **Inverse element:** For each $a \in G$, there exists an element $b \in G$, commonly denoted a^{-1}, such that $a \circ b = b \circ a = e$. (*we must be able to undo any rotation*)

How can symmetries be used to build abstractions?

We can use a group structure to define a similarity (aka an equivalence relation), $a \sim_G b$ iff $\exists g \in G : a = g \circ b$. For some examples of symmetries within RL see B.2.3. Therefore, we can use this notion of similarity, as we did in 3, to construct abstractions.

3.3.1 A symmetric inductive bias

In 3.1.3 we considered using similarities to build an abstraction. But, learning pairwise similarities does not give you the ability to generalise[15]. To generalise you need (accurate) priors [48].

Existing approaches to estimating similarity can generalise if the (measure of) similarity encodes priors. For example: If our measure of similarity is parameterised as a CNN, then the CNN implicitly encodes a preference for continuous and local functions[64]. If our measure of similarity is trained using SGD, SGD implicitly encodes a preference for low rank solutions [65].

In this section we explore how to construct a prior that says:

> *We believe that the problems we are given will have group structure.*

This symmetric prior allows us to impose abstract structure on potential similarities, without specifying the 'details' of these similarities.

3.3.2 A measure of symmetry

> *To build this prior, we need to be able to measure symmetries.*

We need to construct a measure of symmetry, $S : X \to [0, 1]$, that returns higher values for 'more' symmetric objects. We can then use this measure to bias a learner towards more symmetric guesses (see 3.3.3).

Given an object, $x \in X$, we want to know, how symmetric is that object? But, what makes something more or less symmetric? We define the amount of symmetry to be: the order [16]of the largest group that x is invariant to.

[15]Knowing that a and b are similar tells you nothing about c and / or d.

[16]The order of a group is defined as the cardinality of the set of group elements, which is written $|G|$.

For example, $x_{ab} = [a, a, b, b]$ should be considered more symmetric than $x_{abc} = [a, a, b, c]$ because, x_{ab} is invariant to actions[17]of $S_2 \times S_2$, while x_{abc} is (only) invariant to actions of S_2.

Note, the larger the group order, the coarser the abstraction we can build. We can group together $x_{ab}[0] \sim x_{ab}[2], x_{ab}[1] \sim x_{ab}[3]$, thus building an abstraction of size 2.

We write this measure of symmetry as;

$$
\begin{aligned}
G^* &= \max_G |G| && \text{(pick the largest group)} \\
\text{s.t. } & \forall g \in G, x = \phi(g, x) && \text{(that } x \text{ is invariant to)} \\
S(x) &= e^{-\|n - |G^*|\|_1} && \text{(transform so } S(x) \in [0, 1])
\end{aligned}
$$

While this measure of symmetry captures some of what we want, it does not work on inputs of interest. It only considers exact symmetries.

Approximate symmetries

Our intuition tells us that $x = [1, 1.001, 2, 2]$ is close to being as symmetric as $x = [1, 1, 2, 2]$. So rather than asking: *is x invariant to G?*, we can ask *How close is x to being invariant to G?* We can write this as;

$$
\begin{aligned}
C(x) &= \min_G \sum_{g_i \in G} \| \, x - \phi(g_i, x) \, \|_2 - \beta \, \| \, n - |G^*| \, \|_1 \\
S(x) &= e^{-C(x)}
\end{aligned}
$$

Implementation issues

In its current state, this measure of approximate symmetry has two main implementation issues; the search over groups, the representation of the action(s).

[17] An action is defined as $\phi : G \times X \to X$ such that $\phi(e, x) = x$ and $\phi(g \circ h, x) = \phi(g, \phi(h, x))$.

Searching through groups: For a given order, the number of possible groups is roughly equal to the number of integer factorisations of the order itself. We could try to explicitly construct these groups, however, there are many special cases (known as the sporadic groups [66]). This complicates their construction[18].

Representations of actions: The function ϕ is not unique. There can be many ways a group acts on X. As the dimensionality of X grows, the number of possible group actions also grows quickly.

Consider the representation of group actions as permutation matrices, $\phi(g_i, x) = P_i \cdot x$ where $P_i \in 0, 1^{d \times d}$ and $P \cdot \mathbf{1} = \mathbf{1}$. Then there are $d!$ possible permutations. And checking that a collection of actions is closed requires $\mathcal{O}(d! \times d!)$ compositions.[19]

As an example, we construct the possible actions of S_2 and $S_2 \times S_2$ in B.3.2.

For these reason, we start with a restricted family of groups: the groups which can be constructed by direct products of S_2. By using this family of groups as a prior, we are saying: *we prefer objects that are invariant to flips / reflections*.

The actions of these groups can be easily constructed (the permutations that swap n-grams[20]), and (more importantly) their order can be identified by the existence of idempotent permutations generated by an n-gram swap.

Still, the number of actions of this restricted family grows quickly. We are constrained to consider small problems.

[18]Complicated from a program design perspective, not a computational perspective.

[19]Why is this so expensive? The reason is that we treat each dimension as independent from the others. They are discrete, unordered and unrelated.

[20]For example, a 1-gram swap could be $0 \leftrightarrow 1$, a 2-gram swaps could be $(0, 1) \leftrightarrow (2, 3)$. Where these numbers represent the indicies of a vector.

	2	3	4	5	6	7	8	9	10
S_2	1	3	6	10	15	21	28	36	45
$S_2 \times S_2$	0	0	3	15	45	105	210	378	630
$S_2 \times S_2 \times S_2$	0	0	0	0	15	105	420	1260	2150

In the table above, we count the total number of possible actions (minus the identity) for different groups (rows) and Xs of different dimension (columns).

Topology

Let's try to understand this measure we have constructed. How does it behave? Does it give us the properties we wanted?

What properties did we want?

1. Approximate symmetries should be measured as 'less symmetric' than exact symmetries.
2. Larger symmetries should be preferred.

These two properties have been met. When using 3.3.2 as our symmetry measure, we get; $S([1, 1.001, 2, 2]) = 0.956 < 1 = S([1, 1, 2, 2])$ and $S([1, 1, 2, 2]) = 1 > 0.026 = S([1, 1, 2, 3])$.

A final decision, is whether to normalise the distance measure (see 3.13). As, we believe the measure of symmetry should be invariant to changes in norm. For example, if we scale an object by a constant, it should have the same measure of symmetry, $S(10 \cdot x) = S(x)$.

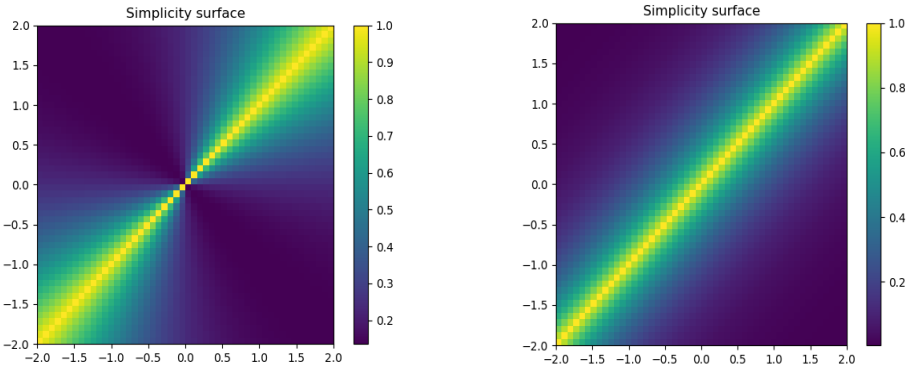

Figure 3.13: A final decision is about whether we want to normalise the distances, $\| x - P \cdot x \|$ (shown on the left), or leave the unnormalised (shown on the right).

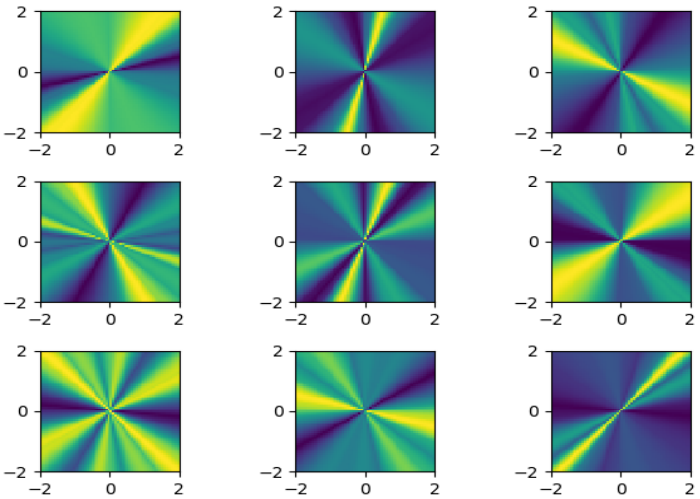

Figure 3.14: A visualisation of our measure of symmetry applied to eight dimensional vectors. We have randomly picked linear projections from the eight dimensions, to two dimensions. Each pixel represents a vector in the 8D space. Lighter color is more symmetric.

Symmetric Rejection Sampling

How can we use this measure of symmetry as a prior?

We can use rejection sampling to bias produce distributions biased towards symmetric samples, or in other words, we can use rejection sampling to incorporate our symmetric prior. Let's see how.

Rejection sampling allows you to generate samples from a target distribution, $p(\cdot)$ (which we cannot efficiently sample from) and a 'related' distribution that we can easily sample from, $q(\cdot)$. The notion of relatedness of $p(\cdot)$ and $q(\cdot)$ is captured by $k = \max_x \frac{p(x)}{q(x)}$. This value, k tells us, the average number of rejections before accepting a sample. Thus, to closely related distributions will have $k \sim 1$. [67, 68]

Algorithm 4 Rejection sampling

```
procedure RS(p, q, k)
    t = 0
    while not accepted do
        x ~ U([0, 1])
        if x < p(x) / (k q(x)) then
            Break
        t += 1
    return x
```

Incorporating a prior: We might have some belief over parameters values θ, given some data, D_t, which we write as $p(\theta|D_t)$ (the posterior). We might also have a prior about likely parameter values, in this case, our belief that they should be symmetric $P_{\text{sym}}(\theta)$. Therefore, we can construct the 'target' distribution using Bayes rule.

$$q(\theta|D_t) = \frac{p(D_t|\theta)P_{\text{sym}}(\theta)}{p(D_t)}$$

Thus we have a distribution over parameters, $q(\theta|D_t)$, which we can optimise using maximum a posteriori.

Fortunately, we don't need to construct $P_{\text{sym}}(\theta)$, we only need an unnormalised function that it is proportional to [68]. Meaning, we can use our measure of symmetry $P_{\text{sym}}(\theta) \to S(x)$.

Consider an example: we might have some estimate of a parameter's value, μ, σ, which follows a Gaussian distribution $p(\cdot)$. But, we believe the parameters should be symmetric. Therefore, we can construct a new distribution, $q(x|D_t) = p(\mu, \sigma|x)S(x)$. We can use rejection sampling to generate samples from $q(x|D_t)$. See the results below.

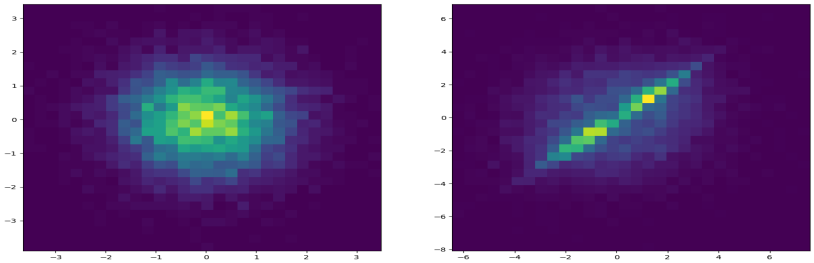

Figure 3.15: On the left we can see a Gaussian distribution of mean zero and variance one, $q(x)$. On the right, we have used rejection sampling generate samples from our symmetry biased distribution, $p(x)$.

These results only confirm that we have implemented rejection sampling correctly, and it works as described.

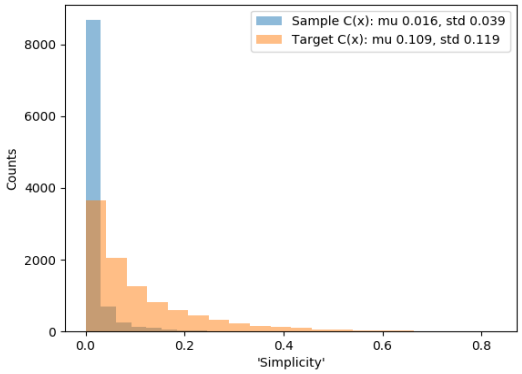

Figure 3.16: Here we have samples from the 'sample distribution', $q(x)$, and the 'target distribution', $p(x)$. We can see that the rejection sampling with the symmetry augmented target distribution was successful in producing, on average, more symmetric samples.

3.3.3 Biased Thompson Sampling

> *We have a way to apply a symmetric prior to a distribution. How can we use that to make RL more efficient?*

Here we consider how to use a symmetric prior to make Thompson sampling more sample efficient[21].

Thompson sampling

> *What is Thompson Sampling?*
>
> *Thompson sampling is an algorithm for online decision problems where actions are taken sequentially in a manner that must balance between exploiting what is known to maximize immediate performance and investing to accumulate new information that may improve future performance.*[69]

[21] For a more general picture of the combination of symmetry and RL see B.2.1.

Less cryptically, Thompson sampling estimates the model, Pr. We then act optimally with respect to the estimated model. However, the estimate uncertainty, so ...? We can write this as follows.

Algorithm 5 Thompson Sampling

procedure TS(γ)
  $t = 0$
  while not converged **do**
    (s, a, r, s') ▷ Observe
    $H_t \leftarrow (s, a, r, s', a')$ ▷ Update history
    $\tau, r \sim P(\cdot|H_t)$ ▷ Sample a model
    $Q_{t+1}(s, a) = r(s, a) + \gamma\tau(s'|s, a)Q_t(s', a')$ ▷ Bellman operator
    $\pi_t = \text{greedy}(Q_{t+1})$ ▷ Act greedily
    $t+ = 1$
  return π_t

Where H_t is the history of observations, $\{(s_i, a_i, r_i, s'_i) : \forall i \in [0, t]\}$. Further, we construct the estimate of the model as independent distributions of the transition and reward functions $P(\tau, r|H_t) = P(\tau|H_t) \cdot P(r|H_t)$. Where $P(\tau|H_t)$ is modelled as the normalised state transition counts. And $P(r|H_t)$ is modelled as a isotropic Gaussian. These can be estimated by storing state-action-state transition counts and the incremental mean and variance of the rewards.

Note that we act greedily with respect to the Q function. Normally, this would lead to sub optimal behaviour, because no exploration is being done. But, Thompson sampling directs exploration through its uncertainty in the model, τ, r. Thus, exploration occurs by acting greedily with respect to a sampled model.

Biased Thompson Sampling

Given uncertainty, prefer symmetry

The difference between *Thompson Sampling* and *Biased Thompson Sampling* is how we construct the distribution over models, $P(\tau, r|D)$.

We believe that the states will have group structure, a symmetry. We use the symmetric rejection sampling method, outlined above (3.3.2) to bias the distribution over models towards more symmetric models.

To simplify, we estimate $P_{\text{sym}}(\tau, R)$ as $P_{\text{sym}}(R)$. That is, we believe there will be symmetries in the reward function, not necessarily in the transition function.

Experiments

We want to demonstrate that Biased Thompson Sampling 3.3.3 has greater sample efficiency (when applied to problems with a symmetry present) than Thompson Sampling 3.3.3. And therefore that this learner can exploit symmetries present in a RL problem[22].

We ran experiments with symmetric n-armed bandit problems, and symmetric grid worlds B.3.2, the results are shown in figures 3.17 and 3.18.

These figures show that Biased Thompson Sampling provides no advantage in sample efficiency (while using a lot more compute). However, we cannot conclude that Biased Thompson Sampling provides no advantage. This is because we also tested a Biased Thompson Sampler given explicit knowledge of the symmetry within a MDP, and it, also, did not show any advantage (see 3.18). This result tell us that our method of biasing a distribution over models (via rejection sampling) does not seem to help RL. It says nothing about the utility of similarity measure.

Also, we expect the advantage should become clearer as we scale to larger state spaces (as there is more benefit to symmetry, there are more similar states). However, we were unable to scale to larger problems due to the computational complexity of building the family of symmetries (see 3.3.2).

[22]Note, we explore another, more complicated, setting to test a learners ability to exploit symmetries in B.4.

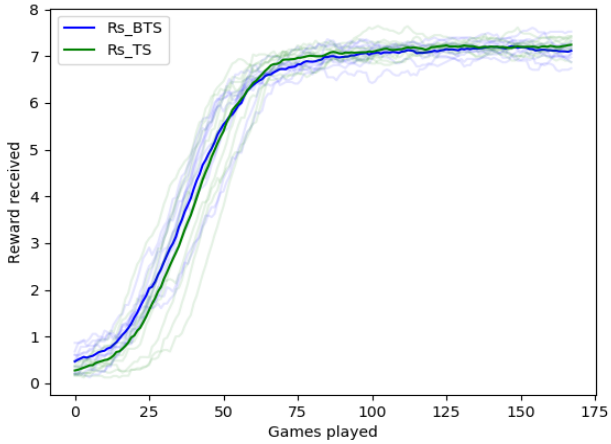

Figure 3.17: A 9 state symmetric grid world problem. We can see that Biased Thompson sampling, using our symmetry measure, does not perform better than Thompson sampling in any significant sense.

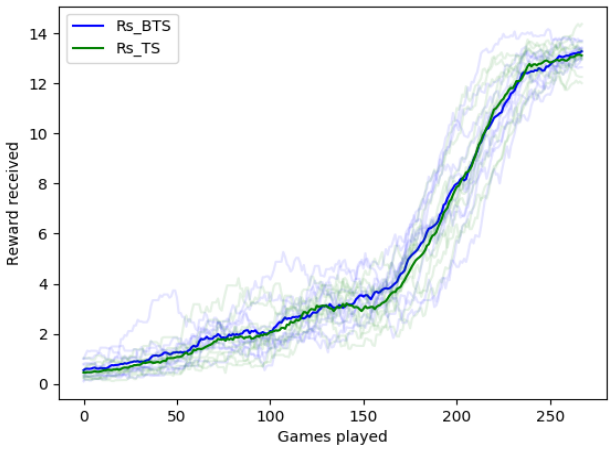

Figure 3.18: A 25 state symmetric grid world problem. This Biased Thompson Sampler was given explicit knowledge of the underlying symmetry. Despite this, we can see that Biased Thompson sampling does not perform better in any significant sense.

Chapter 4

Final remarks

4.1 Summary

To build abstractions for reinforcement learning, we need notions of how to objects can be similar. But, where does these notions of similarity come from? And how do we pick one over the other? While there are many unanswered questions (below), most importantly, there is no well defined way to evaluate an abstractions (in the general case), and thus there is no way to compare them.

Linear Markov Decision Problems attempt to preserve the space of transition dynamics and the rewards. But, they failed to preserve the value of the optimal actions, and thus cannot guarantee performance in general. Ultimately, this was because the Bellman equation is non-linear.

We develop a measure of symmetry, and use it to give reinforcement learners a preference towards symmetry, a prior. Our experiment did not show any advantage by using this prior, but due to computational and temporal[1]constraints, we were unable to conclude that there is no advantage.

[1]Temporal constraints meaning: A masters is a finite amount of time.

4.2 Future work

There is a large amount of future work to be done if we want to:

> *understand how abstractions can increase the efficiency of reinforcement learning.*

Our exploration of **Abstractions**, in 3.1, raises a few fundamental questions;

- What is the advantage, if any, of *state and action abstraction* versus *state-action abstraction* 3.1.1?
- Of the two approaches to temporal abstraction, *goal-like* and *option-like* temporal abstraction, is one strictly better that the other? If not, then in which cases does *goal-like* temporal abstraction perform better?
- Are the facets of evaluation, presented in 3.1.2, necessary and / or sufficient for 'efficient' performance of an abstraction in practice?
- Do many, or even all, abstractions of interest to RL live in the family defined in 3.1.3?
- Is there a difference between trajectory based similarity measures (that set the similarity $\chi(x, x')$ to be built from distances between the cumulants $D(c(x, \pi), c(x', \pi))$, rather than expected discounted cumulants $D(\mathcal{C}(x, \pi), \mathcal{C}(x, \pi))$)?
- What is necessary (rather than sufficient - which is proved in existing work) for the preservation of Bellman equation's ability to guide search? Can we weaken the requirements to: preserving the ordering of the value of optimal actions (rather than their absolute values as in existing work)?

Similarly, our results on **LMDPs** (3.2.4) leave a open few questions about whether LMDPs whether can be saved from irrelvance;

- In which cases does the LDMP give the right solution?
- What are the properties that make a MDP easily solvable (via LDMPs)?
- Can easily solvable MDPs (via LDMPs) be easily identified?

Next, our results on **symmetric abstractions** (3.3) leave a few questions unanswered;

- What is the (computational) efficiency of rejection sampling for symmetry biased distributions, and how does it scale with dimension? And how much data (sample efficiency) does that computation buy?
- What happens if we use our measure of symmetry 3.3.2 as a regulariser (as it is differentiable)?
- How can we use invariants B.2.3 of the transition and / or reward and / or the value functions to identify symmetries?
- How can representations of states (or state-actions or ...) be ordered or structured? And how does this structure reduce the combinatorial space of possible symmetries?

Finally, the **appendices** (4.2) ask more questions than they answer;

- What is the significance of the projection of the value polytope onto a single dimension when we increase the discount rate? A.4
- When attempting to evaluate the integral over all policies, how can we exploit structure in the given MDP to reduce the computational complexity. 1 Is there a way to intelligently (or randomly) pick which mixtures of deterministic policies to evaluate (like compressed sensing [70]).
- Convexity in optimisation problems allows efficient solutions to be found, are there other properties that are mutual exclusive to convexity that also allow efficient solutions. A.6.2

- What is the cost of discovering temporal symmetries? How does this cost scale with the length of time? Can we amortize this cost by building symmetries from smaller symmetries in shorter sequences? B.2.2
- How can we use invariant of the value function to identify symmetries? B.2.3
- Are there more methods of exploiting symmetries? Or would any new method reduce to one listed in B.3.1. And which method of exploiting symmetries is best?
- Explain difference between the *Discrete* and *MultiBinary* action spaces in B.4.3 (as the hardness of learning a binary decoder is not sufficient to explain it).

Bibliography

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] N. Brown and T. Sandholm, "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. Pondé, D. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," 2019.

[4] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2018.

[5] S. Box, "Skinner, B. F. (1948). Superstition in the pigeon.," no. 1948, pp. 168–172, 1997.

[6] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains Leslie," 1998.

[7] M. Putterman, *Markov Decision Processes*. 1994.

[8] D. Bertsekas, *Dynamic programming and optimal control*. 1995.

[9] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[10] A. A. Markov, "Classical text in translation: An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains," *Science in Context*, vol. 19, no. 4, pp. 591–600, 2006.

[11] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1889–1897, 2015.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[13] R. Dadashi, A. A. Taïga, N. L. Roux, D. Schuurmans, and M. G. Bellemare, "The Value Function Polytope in Reinforcement Learning," 2019.

[14] D. P. Bertsekas and J. Tsitsiklis, "Neuro-Dynamic Programming," 1996.

[15] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised State Representation Learning in Atari," no. NeurIPS, 2019.

[16] L. Li, T. J. Walsh, and M. Littman, "Towards a Unified Theory of State Abstraction for MDPs," 2006.

[17] G. Cuccu, J. Togelius, and P. Cudre-Mauroux, "Playing Atari with Six Neurons," jun 2018.

[18] M. Zhong and E. Todorov, "Aggregation Methods for Lineary-solvable Markov Decision Process ⋆,"

[19] G. Vezzani, A. Gupta, L. Natale, and P. Abbeel, "Learning latent state representation for speeding up exploration," 2019.

[20] D. Abel, D. Arumugam, L. Lehnert, and M. Littman, "State Abstractions for Lifelong Reinforcement Learning," *35th International Conference on Machine Learning*, vol. 80, pp. 10–19, 2018.

[21] Y. Duan, Z. T. Ke, and M. Wang, "State Aggregation Learning from Markov Transition Data," no. NeurIPS, 2018.

[22] D. Abel, D. E. Hershkowitz, and M. L. Littman, "Near Optimal Behavior via Approximate State Abstraction," vol. 48, pp. 1–18, 2017.

[23] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. S. Thomas, "Learning Action Representations for Reinforcement Learning," 2019.

[24] C. J. Bester, S. D. James, and G. D. Konidaris, "Multi-Pass Q-Networks for Deep Reinforcement Learning with Parameterised Action Spaces," 2019.

[25] G. Tennenholtz and S. Mannor, "The Natural Language of Actions," 2019.

[26] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–17, 2019.

[27] P. Dayan, "Improving Generalization for Temporal Difference Learning: The Successor Representation," *Neural Computation*, vol. 5, no. 4, pp. 613–624, 1993.

[28] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. Van Hasselt, and D. Silver, "Successor features for transfer in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 4056–4066, 2017.

[29] P. Christodoulou, R. T. Lange, A. Shafti, and A. A. Faisal, "Reinforcement Learning with Structured Hierarchical Grammar Representations of Actions," 2019.

[30] J. Rafati and D. C. Noelle, "Learning Representations in Model-Free Hierarchical Reinforcement Learning," pp. 1–35.

[31] D. J. Mankowitz, T. A. Mann, P.-L. Bacon, D. Precup, and S. Mannor, "Learning Robust Options," no. 1, pp. 6409–6416, 2018.

[32] A. Harutyunyan, P. Vrancx, P.-L. Bacon, D. Precup, and A. Nowe, "Learning with Options that Terminate Off-Policy," 2017.

[33] R. Fruit and A. Lazaric, "Exploration–Exploitation in MDPs with Options," 2017.

[34] M. Riemer, M. Liu, and G. Tesauro, "Learning Abstract Options," no. NeurIPS, pp. 1–11, 2018.

[35] P.-L. Bacon, "Temporal Representation Learning," no. June, 2018.

[36] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel, "Variational Option Discovery Algorithms," pp. 1–29, 2018.

[37] V. Pham, S. Tunyasuvunakool, S. Liu, D. Tirumala, N. Heess, and G. Wayne, "HIERARCHICAL VISUOMOTOR CONTROL OF HUMANOIDS," no. 2002, pp. 1–19, 2019.

[38] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[39] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine, "Latent Space Policies for Hierarchical Reinforcement Learning," tech. rep.

[40] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[41] R. Fruit, A. Lazaric, M. Pirotta, and E. Brunskill, "Empirical Evaluation of Optimism with Options," *NIPS Hierarchical Reinforcement Learning Workshop*, 2017.

[42] P.-L. Bacon, J. Harb, and D. Precup, "The Option-Critic Architecture," 2016.

[43] Y. Jinnai, D. Abel, M. Littman, and G. Konidaris, "Finding Options that Minimize Planning Time," 2018.

[44] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-Optimal Representation Learning for Hierarchical Reinforcement Learning," pp. 1–18, 2018.

[45] N. K. Jong and P. Stone, "State abstraction discovery from irrelevant state variables," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 752–757, 2005.

[46] S. Bubeck and M. I. Jordan, "Is Q-learning Provably Efficient ?," no. NeurIPS, 2018.

[47] G. Konidaris, "On the necessity of abstraction," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 1–7, 2019.

[48] D. H. Wolpert and W. Macready, "No Free Lunch Theorems for Optimization," 1996.

[49] A. White, "Developing a predictive approach to knowledge," 2015.

[50] B. O'Donoghue, "Suboptimal control policies via convex optimisation," no. December, 2012.

[51] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning Convex Optimization Control Policies," pp. 1–26, 2019.

[52] E. Todorov, "Linearly-solvable Markov decision problems," no. 1, 2006.

[53] Chi Jin, Zhuoran Yang, Z. Wang, and M. I. Jordan, "Provably Efficient Reinforcement Learning with Linear Function Approximation," pp. 1–28.

[54] N. Levine, Y. Chow, R. Shu, A. Li, M. Ghavamzadeh, and H. Bui, "Prediction, Consistency, Curvature: Representation Learning for Locally-Linear Control," pp. 1–27, 2019.

[55] B. Á. Pires and C. Szepesvári, "Policy Error Bounds for Model-Based Reinforcement Learning with Factored Linear Models," vol. 49, pp. 1–31, 2016.

[56] F. P. S. Eric Warren Fox, J. Seth, and Gordon, "a Note on Nonparametric Estimates of Space-Time Hawkes Point Process Models for Earthquake Occurrences," *Annals of Statistics*, vol. 44, no. 2, pp. 629–659, 2016.

[57] E. Todorov, "Efficient computation of optimal actions," *Proceedings of the National Academy of Sciences*, vol. 106, no. 28, pp. 11478–11483, 2009.

[58] S. Levine, "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review," 2018.

[59] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "FeUdal networks for hierarchical reinforcement learning," *34th International Conference on Machine Learning, ICML 2017*, vol. 7, pp. 5409–5418, 2017.

[60] R. S. Sutton and A. G. Barto, *Reinforcement learning*. 2018.

[61] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," no. Icdar, pp. 1–6, 2003.

[62] T. S. Cohen and M. Welling, "Steerable CNNs," no. 1990, pp. 1–12, 2017.

[63] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner, "Towards a Definition of Disentangled Representations," pp. 1–29, 2018.

[64] L. Yann and Y. Bengio, "Convolutional Networks for Images, Speech, and Time Series," 1995.

[65] S. Gunasekar, B. Woodworth, S. Bhojanapalli, B. Neyshabur, and N. Srebro, "Implicit regularization in matrix factorization," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 6152–6160, 2017.

[66] J. Conway, "Atlas of finite groups: maximal subgroups and ordinary characters for simple groups," 1985.

[67] G. W.R. and W. P., "Adaptive Rejection Sampling for Gibbs Sampling," *Applied Statistics*, vol. 41, pp. 337–348, 1992.

[68] A. B. Owen, "Monte Carlo theory, methods and examples," 2013.

[69] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, "A Tutorial on Thompson Sampling," pp. 1–96, 2017.

[70] E. Candes, J. Romberg, and T. Tao, "Stable Signal Recovery from Incomplete and Inaccurate Measurements," 2005.

[71] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking Model-Based Reinforcement Learning," pp. 1–25, 2019.

[72] M. G. Bellemare, W. Dabney, R. Dadashi, and A. A. Taiga, "A Geometric Perspective on Optimal Representations for Reinforcement Learning," pp. 1–27, 2019.

[73] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[74] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," pp. 1–12, 2017.

[75] S. Arora, N. Cohen, and E. Hazan, "On the Optimization of Deep Networks : Implicit Acceleration by Overparameterization," 2018.

[76] O. E. Ganea, G. Bécigneul, and T. Hofmann, "Hyperbolic neural networks," *Advances in Neural Information Processing Systems*, vol. 2018-Decem, no. NeurIPS, pp. 5345–5355, 2018.

[77] B. Ravindran and A. G. Barto, "Model minimization in hierarchical reinforcement learning," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2371, pp. 196–211, 2002.

[78] B. Ravindran and A. G. Barto, "Approximate Homomorphisms: A Framework for Non-exact Minimization in Markov Decision Processes," *Proceedings of the Fifth International Conference on Knowledge Based Computer Systems (KBCS 04)*, pp. 19–22, 2004.

[79] B. Ravindran and A. G. Barto, "SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1011–1016, 2003.

[80] Peter Olver, "Classical Invariant Theory," *London Mathematical Society Student Texts). Cambridge: Cambridge University*, 1999.

[81] T. Dean and R. Givan, "Model minimization in Markov decision processes," *Proceedings of the National Conference on Artificial Intelligence*, pp. 106–111, 1997.

[82] S. NARAYANAMURTHY, "Abstraction Using Symmetries in Markov Decision Processes," *October*, no. October 2007.

[83] S. Chen, E. Dobriban, and J. H. Lee, "Invariance reduces Variance : Understanding Data Augmentation in Deep Learning and Beyond," pp. 1–44, 2019.

[84] F. Abdolhosseini and X. B. Peng, "On Learning Symmetric Locomotion,"

[85] F. Anselmi, G. Evangelopoulos, L. Rosasco, and T. Poggio, "Symmetry-adapted representation learning," vol. 86, pp. 201–208, 2019.

[86] S. Ravanbakhsh, J. Schneider, and B. Poczos, "Equivariance through parameter-sharing," *34th International Conference on Machine Learning, ICML 2017*, vol. 6, pp. 4416–4428, 2017.

[87] A. Mahajan and T. Tulabandhula, "Symmetry Learning for Function Approximation in Reinforcement Learning," pp. 1–12, 2017.

[88] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Advances in Neural Information Processing Systems*, vol. 2018-December, no. 5, pp. 8571–8580, 2018.

[89] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen, "Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules," 2019.

[90] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, "Fast AutoAugment," 2019.

[91] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning Augmentation Policies from Data," 2018.

[92] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "RandAugment: Practical automated data augmentation with a reduced search space," 2019.

[93] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," pp. 1–4, 2016.

[94] P. Dhariwal, Prafulla and Hesse, Christopher and Klimov, Oleg and Nichol, Alex and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, Yuhuai and Zhokhov, "OpenAI Baselines," 2017.

As these appendices make up a significant amount of this thesis, it may be worth having a guide.

- In A.1 we give an alternative derivation of the value functional, a tabular form of the state valued Bellman equation.
- In A.2 we give some intuition about the abstract structure of policies.
- In A.3 we explore the density of policies and effect of discounting on the value polytope.
- In A.4 we demonstrate a new (so far as we know) type of model based reinforcement learner.
- In A.5 we present a method of visualising the search for the optimal policy in high dimensions.
- In A.6 we ask questions about over-parameterisation and re-parameterisation in the context of policy gradients.
- In B.1.1 we offer a modified derivation of a linear Markov decision problem.
- In B.2.1 we explore related work combining symmetries and Markov decision problem, and add a new definition of a temporal homomorphism.
- In B.2.3 we construct a set of symmetric reinforcement learning examples, in the hope of understanding how to identify relevant symmetries from invariants.
- In B.3.1 we review related work on symmetries in machine learning and draw connections between some of these methods.
- In B.3.2 we provide an example of how there can be many (group) actions for even a simple group.
- In B.3.2 we describe a toy problem used in testing our symmetry biased reinforcement learner.
- In B.4 we describe experiments done to test a learner's ability to exploit symmetries.

Appendix A

MDPs

A.1 Tabular MDPs

It is possible to derive the value functional in another, possibly more enlightening way. But it takes a little more work. It requires a result from real analysis, the Neumann series. This is simply the generalisation of a geometric series to contractive linear operators, such as a matrix.

$$
\begin{aligned}
r &\in (-1, 1) \\
(1-r)^{-1} &= \lim_{n\to\infty} \sum_{i=0}^{n} r^i && \text{(Geometric series)} \\
T &\in \mathbb{X}^k : \det(T) \in (-1, 1) \\
(I-T)^{-1} &= \lim_{n\to\infty} \sum_{i=0}^{n} T^i && \text{(Neumann series)}
\end{aligned}
$$

We can expand the recursion in the Bellman equation to get an (infinite series). We can then use the (Neumann series) (by setting $T = \gamma\tau_\pi$) to give the nice analytic form of the value functional.

$$
\begin{aligned}
V &= r_\pi + \gamma\tau_\pi V && \text{(Bellman eqn)} \\
V &= r_\pi + \gamma\tau_\pi\big(r_\pi + \gamma\tau_\pi V\big) \\
V &= r_\pi + \gamma\tau_\pi\Big(r_\pi + \gamma\tau_\pi\big(r_\pi + \gamma\tau_\pi V\big)\Big) \\
&= r_\pi + \gamma\tau_\pi r_\pi + \gamma^2\tau_\pi\tau_\pi r_\pi + \gamma^3\tau_\pi\tau_\pi\tau_\pi V \\
&= \sum_{t=0}^{\infty}\gamma^t\tau_\pi^t r_\pi && \text{(infinite series)} \\
&= \big(\sum_{t=0}^{\infty}\gamma^t\tau_\pi^t\big)\cdot r_\pi \\
&= (I - \gamma\tau_\pi)^{-1} r_\pi && \text{(value functional)}
\end{aligned}
$$

This proof is more satisfying because we can more clearly see the nature of the value functional. It is a closed form of the infinite sum of discounted future rewards.

A.2 Policies in high dimensions

Let's *try* to gain intuition about the space of policies in higher dimensions. For each state, we have a distribution (on a simplex), over the possible actions.

Imagine what the geometry of the space of policies in the two state, two action MDP. A policy tells us which actions should be taken when in a given state. Therefore, there will be $|A| \times |S|$ entries in the policy. However, because the policy returns a distribution over actions, the true dimensionality of the policy is $(|A| - 1) \times |S|$. Which in the two state, two action case equals 2D.

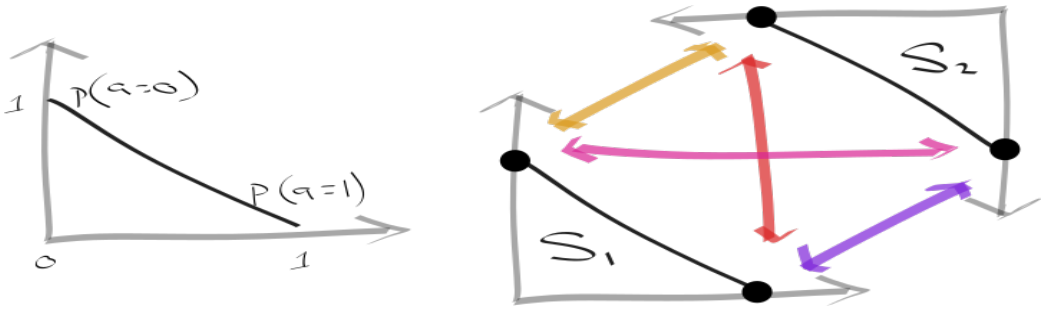

Figure A.1: (left) Given that you are in state, s, we have a simplex over two actions $\{a_1, a_2\}$. (right) The policy must describe distributions over actions for each state, so the policy space must describe the $|S|$ possible combinations of distribution over actions.

A.3 Other properties of the polytope

A.3.1 Distribution of policies

A potentially interesting question to ask about the value polytope is how the values (of the policies) are distributed. We can calculate this density analytically by using the probability chain rule: $p(f(x)) = |\det \frac{\partial f(x)}{\partial x}|^{-1} p(x)$. Where we set f to be our value functional and $p(x)$ to be a uniform distribution over policies. Thus we have;

$$p(V(\pi)) = |\det \frac{\partial V(\pi)}{\partial \pi}|^{-1} \cdot p(\pi) \qquad \text{(density)}$$

As we can see in A.3: for some polytopes, there is high density around the optimal policy. In other polytopes, many of the policies are far away from the optimal policy.

Consider the expected suboptimality of our MDP, which tells us how far away the optimal policy is from the 'center of value' of the polytope.

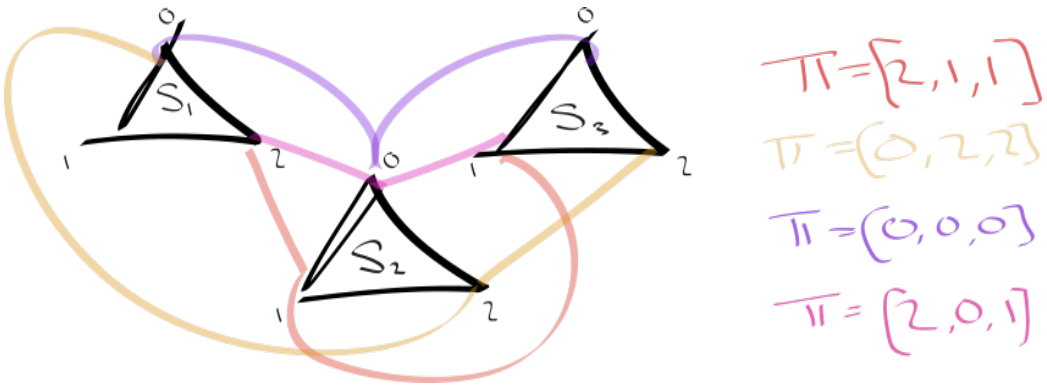

Figure A.2: Here we are visualising the policy space of a three state, three action, MDP. For each state, we must specify a distribution over actions.

$$\mu_M(s) := \mathop{\mathbb{E}}_{\pi \sim \Pi} \left[V_M^\pi(s) \right]$$
$$\epsilon^*(M) = \max_s V_M^{\pi^*}(s) - \mu(M)(s)$$

This suggests that MDPs with low expected suboptimality, $\epsilon^*(M)$, are easier to solve than other MDPs. This is because we can simply sample a random policy which is likely to be close to the optimal policy (or we could sample many policies and pick the best, which would, with high probability, be close(r) to the optimal policy).

However, this strategy will not scale to higher dimensions. As we increase the state size or action size, the chance that there is high density near the optimal policy (or a randomly sampled MDP) decreases with rate proprtional to $|A|^{|S|}$.

A.3.2 Discounting

Another question you may have is: *how does the shape of the polytope depend on the discount rate?* Given an MDP, we can vary the discount rate from 0 to 1 and visualise the shape of the value polytope.

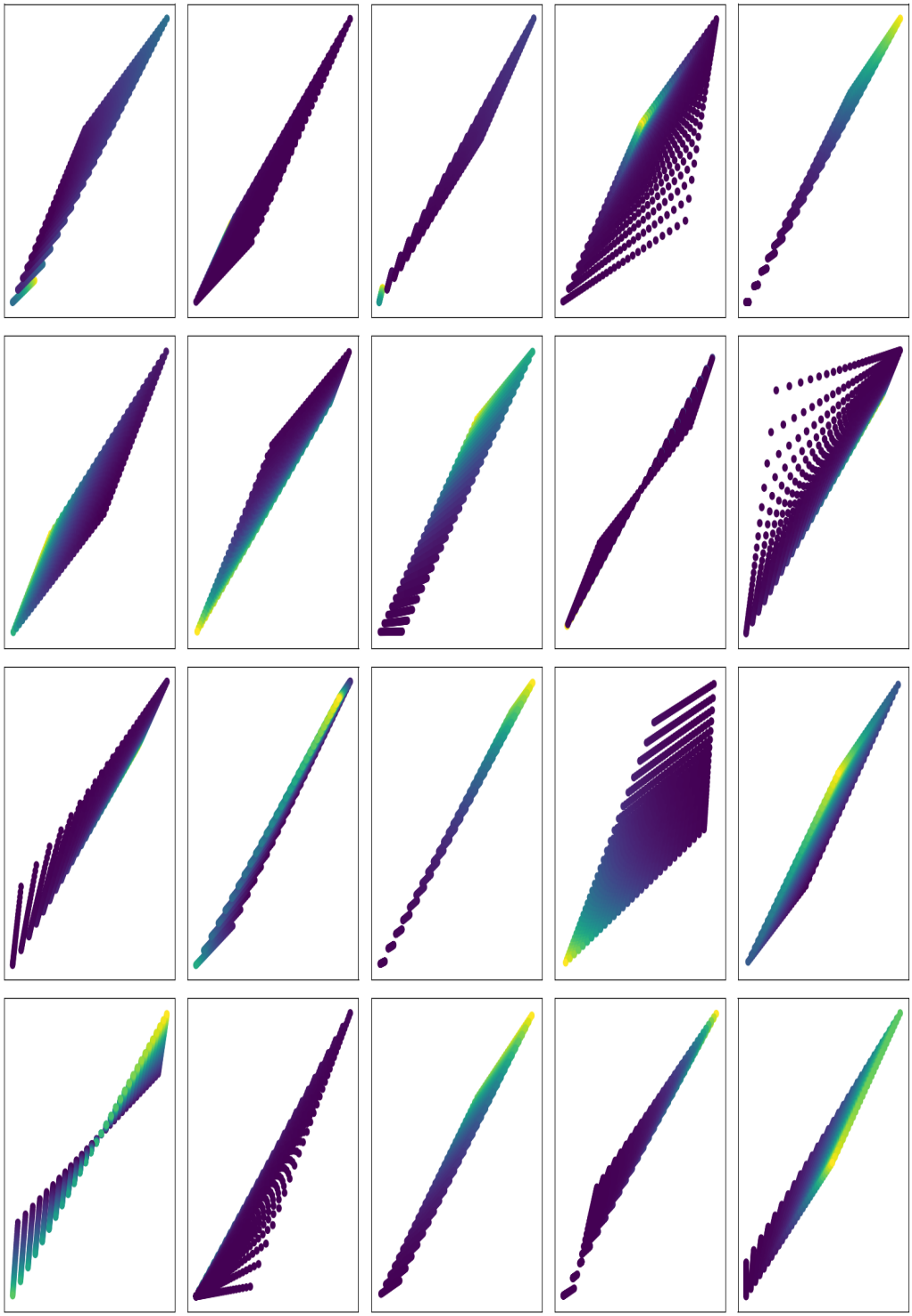

Figure A.3: Here we have visualised the value polytope for 2-state 2-action MDPs. They are colored by the likelihood of each value under a uniform distribution over policies. Lighter color is higher probability.

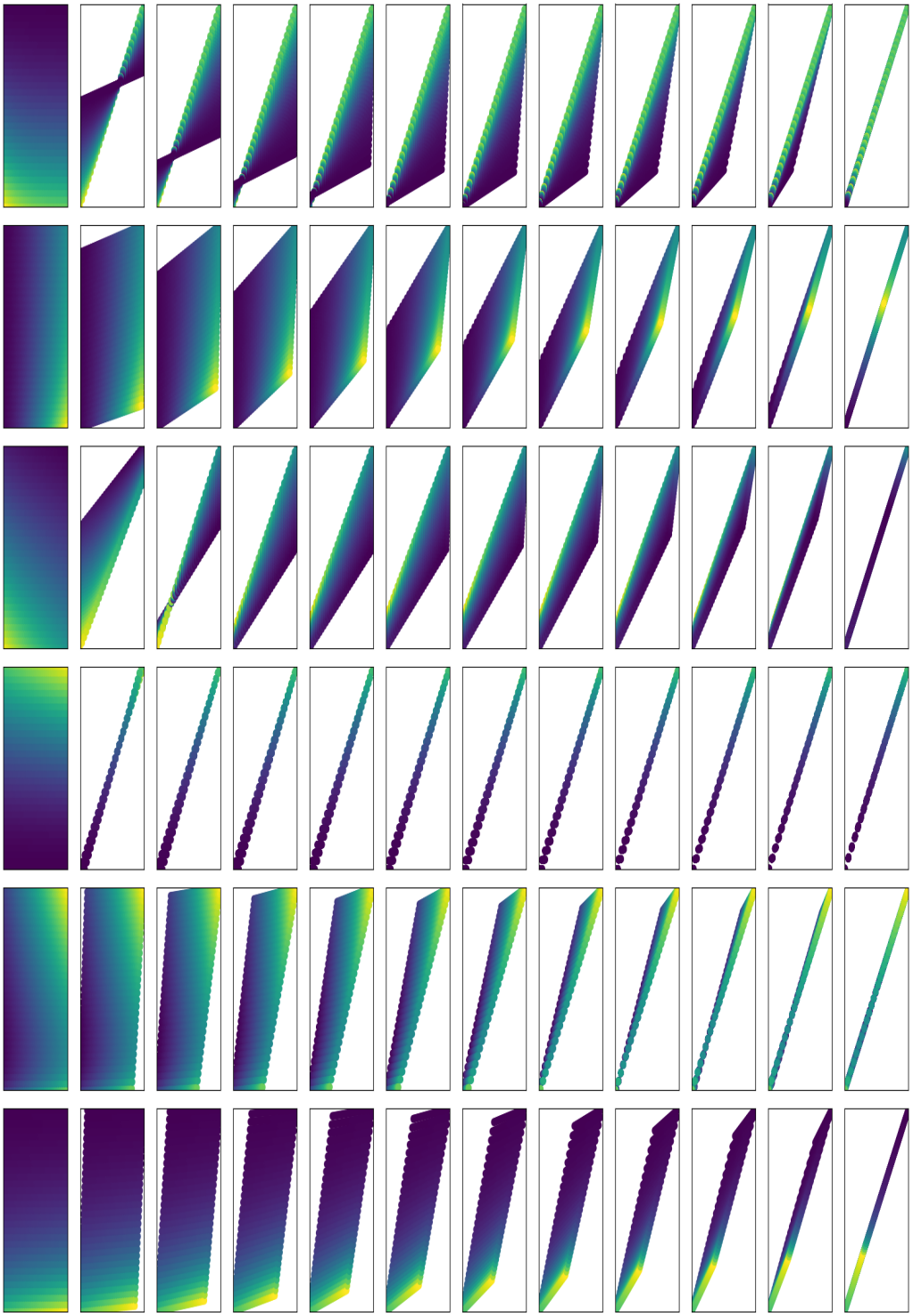

Figure A.4: Here we have visualised the value polytope for 2-state 2-action MDPs. The rows show how a MDP's value polytope change with changes in discount rate, ranging linearly from 0 (left), to 1 (right). Each column is for a different MDP. The color map is represents the density of policies, as in A.3.

As we can see in A.4. As the discount rate tends to 1, all the policies are projected into a 1D space. (is this 1D for 2D problems. is it always 1D?)

A.3.3 Derivation of derivative

We want to derive the derivative of the value functional.

$$
\begin{aligned}
V(\pi) &= (I - \gamma\tau_\pi)^{-1} r_\pi && \text{(value functional)} \\
&= (I - \gamma\tau \cdot \pi)^{-1} r \cdot \pi \\
\frac{\partial V}{\partial \pi} &= \frac{\partial}{\partial \pi}((I - \gamma\tau_\pi)^{-1} r_\pi) \\
&= (I - \gamma\pi\tau)^{-1} \frac{\partial \pi r}{\partial \pi} + \frac{\partial (I - \gamma\pi\tau)^{-1}}{\partial \pi} \pi r && \text{(product rule)} \\
&= (I - \gamma\pi\tau)^{-1} r + -(I - \gamma\pi\tau)^{-2} \cdot -\gamma\tau \cdot \pi r \\
&= \frac{r}{I - \gamma\pi\tau} + \frac{\gamma\tau \cdot \pi r}{(I - \gamma\pi\tau)^2} && \text{(rewrite as fractions)} \\
&= \frac{r(I - \gamma\pi\tau) + \gamma\tau\pi r}{(I - \gamma\pi\tau)^2} && \text{(common demoninator)} \\
&= \frac{r}{(I - \gamma\tau\pi)^2} && \text{(cancel)}
\end{aligned}
$$

A.4 Model search

As noted in 2.5, we can search through policy space or value space, each with their own structure. There is a third search space we could consider, the model parameters.

However, this problem is not a control problem like the search for values or policies. This is an inference problem, we are trying to infer the model parameters. Once we know these parameters, we can apply control. In general, this is known as model-based RL.

We could search through models (τ, r) using supervision from next step prediction. This is a common approach to model-based RL [71]. Rather, we consider using the value estimation error to guide the search for model parameters.

$$\tilde{P}^*, \tilde{r}^* = \underset{\tilde{\tau},\tilde{r}}{\operatorname{argmin}} \int_{\Pi} \parallel V_{\tau,r}(\pi) - V_{\tilde{\tau},\tilde{r}}(\pi) \parallel_{\infty}$$

There is a main advantage and some disadvantages to this framework. The advantage is that this approach focuses its resources only on 'relevant' features of the model. The disadvantages are that it requires many policy evaluations (from the environment), and many estimates of a policy's value (simulated evaluations).

A.4.1 Relevant features of the model

Model iteration via value prediction error only focuses on 'relevant' features of the model [1]. Relevant in the sense that they are useful for accurately predicting state values.

Consider a problem where the reward is only determined by the first feature of the state. We can add arbitrarily many extra features. A model-

[1] Model free approaches to RL also focus on 'relevant' features, however, they do not explicitly represent the underying model.

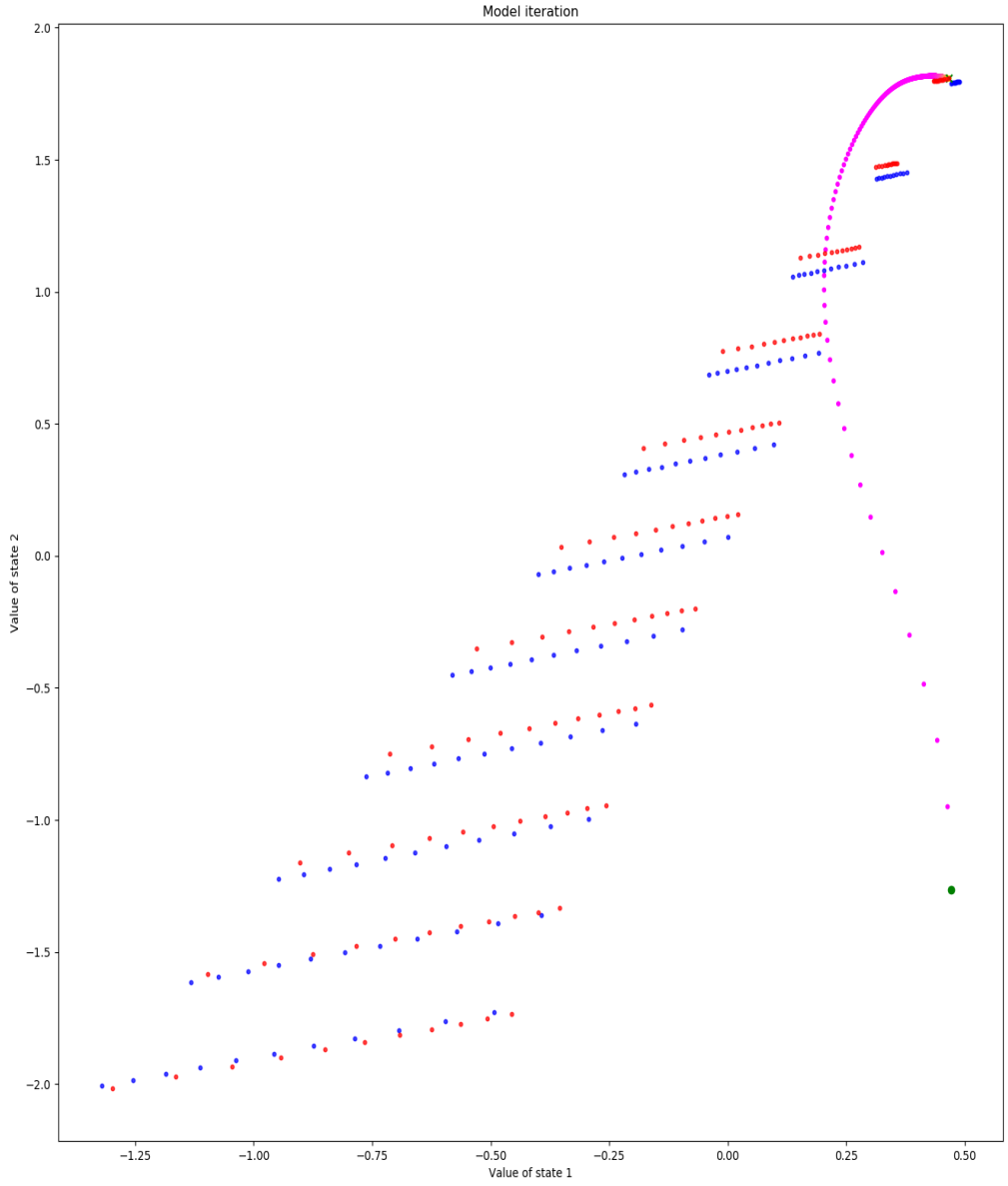

Figure A.5: Model iteration applied to a 2-state, 2-action MDP. Blue shows the value of policies when evaluated under the true model, τ, r, and Red shows the estimated value of policies when evaluated under the learned model at convergence, $\tilde{\tau}, \tilde{r}$. The purple trajectory shows the value of the policy as it is learned.

based learner that learns through next step prediction will attempt to accurately predict all of these features. This unnecessarily spends resources.

More formally; let the state space be a subset of the reals $S \subset \mathbb{R}^d$. And construct a reward function that is only dependent on the first feature of the state space, $r(s,a) = f(s^0, a)$. While the transition function is constructed so that, there exists a k dimensional subspace (that includes the first feature) that is independent of the other $d-k$ features, $\tau(s'|s,a) = [g_1(s^{0:k}, a), g_2(s^{k:d}, a)]$.

An example with this structure, could be the addition of $d-k$ 'noise' features, which play no part in the transition dynamics, but only describe inessential features (such as features that describe the positions of leaves on a windy day).

A.4.2 Policy evaluations

We receive $V_{P,r}(\pi)$ from the environment. But we want to minimise the number of these calls to the environment, as it can be expensive to evaluate a policy with high accuracy.

Also, for each iteration of $\tilde{\tau}_t, \tilde{r}_t$, we need to estimate the value of many policies, $\forall \pi; V_{\tilde{\tau}_t, \tilde{r}_t}(\pi)$. This requires lots of compute.

Rather than integrating over all policies (thus requiring us to evaluate all of them), we only need the deterministic policies. As, Bellamare et al. 2019 show that *"the maximal approximation error measured over all value functions is the same as the error measured over the set of extremal vertices [aka deterministic policies]"*[72]. However, this still requires us to evaluate $|A|^{|S|}$ policies, and to simulate $T|A|^{|S|}$ policies. [2]

[2]It should be possible to use off policy estimation techniques to further reduce the required samples / compute. But we leave this for future work.

A.5 Visualising higher dimensional MDPs

So this value polytope works well for 2D. And could be extended to 3D. But, what about higher dimensions?

> *"To deal with a 14-dimensional space, visualize a 3-D space and say 'fourteen' to yourself very loudly. Everyone does it."* Geoff Hinton

Is there intuition we might gain from visualising optimisation on MDPs with the number of states and / or actions being greater than 3?

How to construct them? Decompose the value of a policy as a convex combination of the values of the deterministic policies.

$$\alpha(\pi) = \underset{\alpha \in \Delta^n}{\operatorname{argmin}} \parallel V(\pi) - \sum_i (V(\pi_i) \cdot \alpha_i) \parallel_2^2 + H(\alpha)$$

The entropy term ensures we prefer convex combinations with lower entropy.

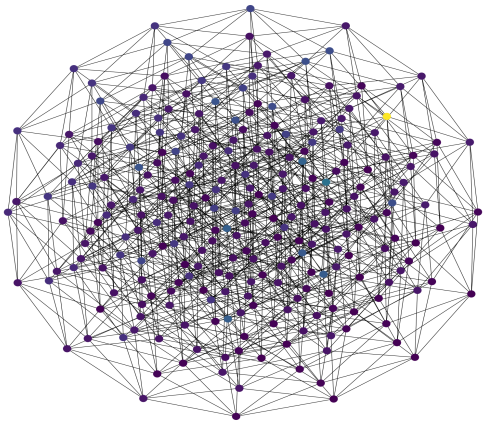

Figure A.6: PI bounces between the nodes of the graph. So the search of the optimal policy using PI can be visualised as the walk along a graph. Note: the current policy is shown in yellow (close to the top right).

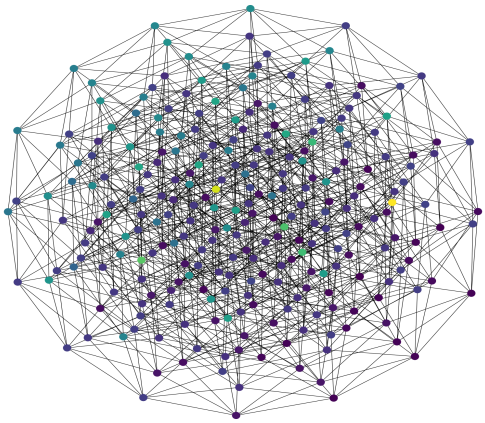

Figure A.7: Policy gradient traverses through non-deterministic policies. So the search of the optimal policy using PG can be visualised the dynamics of a graph signal.

A.6 Deep policy gradients

From the search spaces section 2.5, we are left wondering;

- In which spaces can we (efficiently) calculate gradients?
- In which spaces can we do convex optimisation?
- In which spaces does momentum work (well)?

Currently, the most successful deep RL approaches are variants of policy gradient methods [73, 74]. Despite their efficacy in practice, we still don't have a good understanding of how policy gradient methods combine with deep learning methods to give performant agents.

A key feature of deep learning is over parameterisation (among the other features; hierarchies, non-linearity, smoothness, locality). There has been recent work attempting to understand the effect of overparameterisation [75].

Here we explore the effects of; overparameterising the seach space and then using policy gradient methods, and reparameterising the loss function, thus yielding some other space to search through.

A.6.1 Overparameterisation

Recently there has been work investigating the properties of overparameterised search spaces. Arora et al. 2018 [75] prove that overparameterisation yields acceleration, however, their explanation of the acceleration is not entirely convincing (as it requires first order assumptions).

More applied to RL, we want to know: how does overparameterisation effect the search for optimal policies?

In figures A.8 and A.8 we can see that parameterisation can yield vastly different training trajectories[3]to gradient descent,

[3]Is there any advantages to certain types of trajectory?

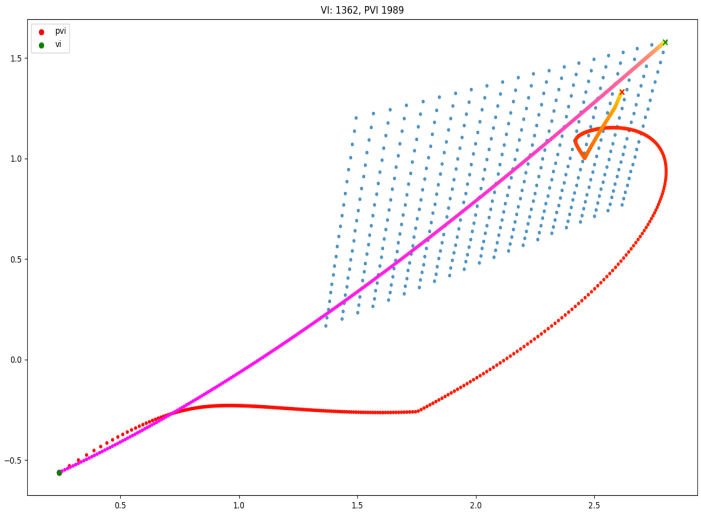

Figure A.8: The optimisation dynamics of value iteration versus parameterised value iteration. Green is VI's trajectory, and orange is parameterised VI's trajectory.

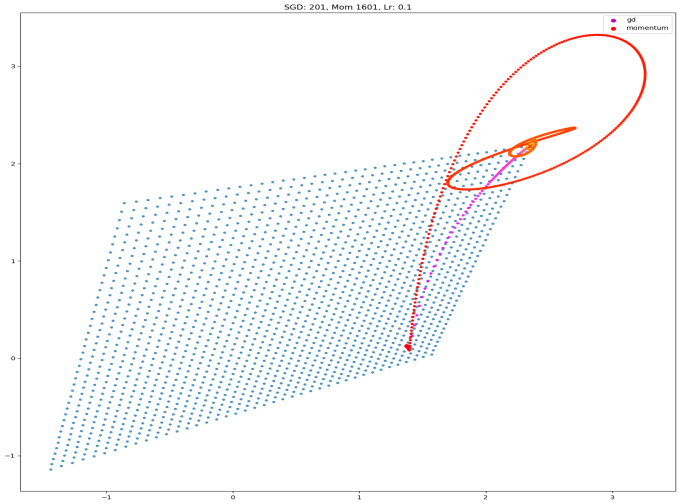

Figure A.9: The optimisation dynamics of value iteration versus value iteration with momentum. Green is SGD's trajectory, and orange is momentum's trajectory.

In A.10 we can observe that there can be sharp boundaries between neighboring policies. A kind of phase transition, between the optimal policy being easy to find, and not. (despite the fact that the we started from neighboring points).

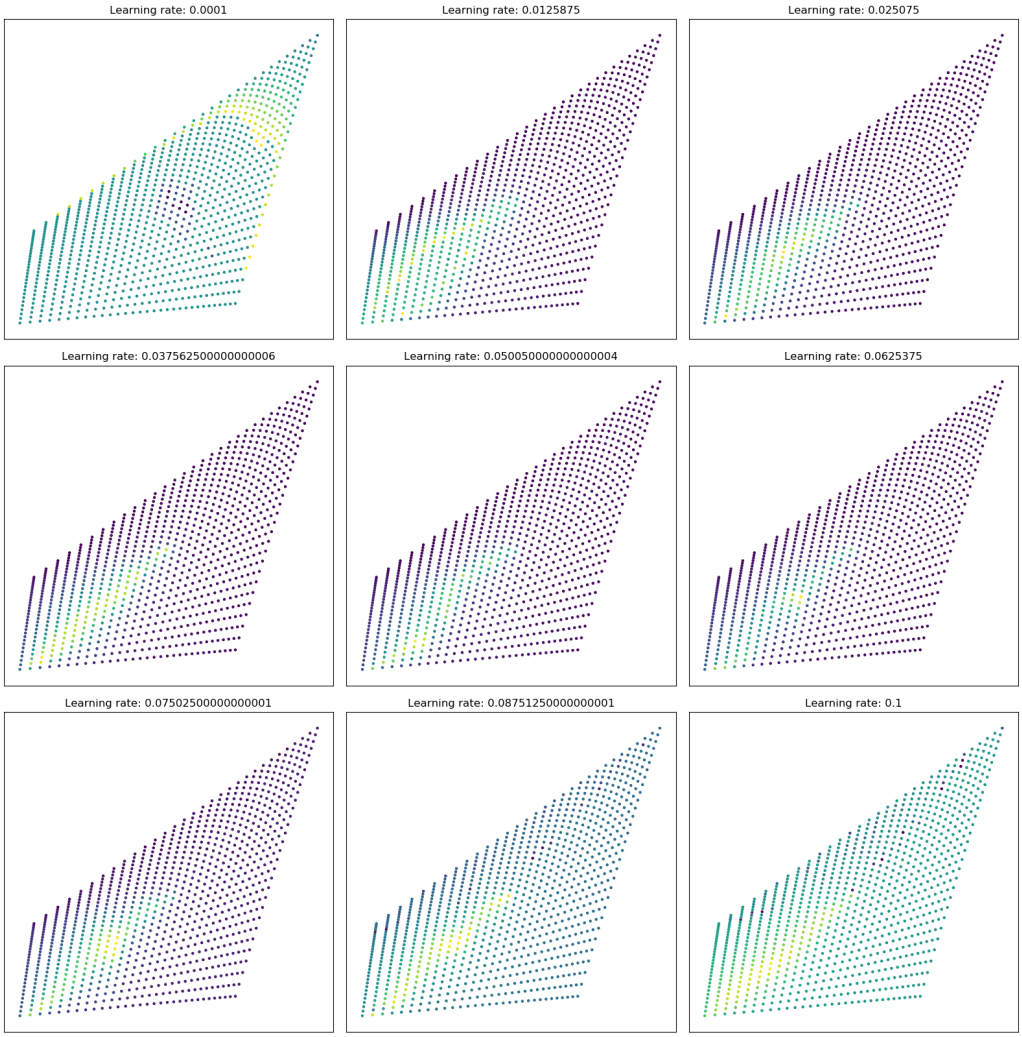

Figure A.10: Here we have visualised the value polytope a 2-state, 2-action MDP. These value polytopes are colored by the iteration complexity (many iterations - yellow, few iterations - purple) of parameterised policy gradients. Each polytope was generated using a different learning rates.

A.6.2 Reparameterisation

If we are allowed to arbitrarily pick our optimisation space, which should we pick?

Here we raise the question: should, and if so, how can, we rationally pick the search space given knowledge of the loss function and likely starting points?

Consider the simple convex problem;

$$x^* = \underset{x}{\text{argmin}}\, x^2$$

We can pick some other space, Z, that we want to search in. And we can map Z on to X using $f : Z \to X$.

$$z^* = \underset{z}{\text{argmin}}\, f(z)^2$$
$$x^* = f(z^*)$$

If we pick some simple functions for f. What happens? Well, if f is non-linear then we lose the guarantees convergence of convex search. However, what is gained?

When z is the parameters of a deep neural network, the search space has certain properties: for example it's euclidean. An alternative could be to search through a hyperbolic space, yielding 'hyperbolic neural networks' (explored by [76]).

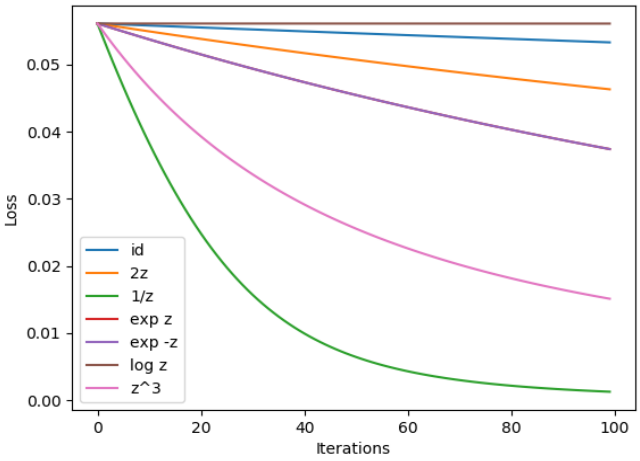

Figure A.11: The convergence rate of different functions. Note that because some of these are non-linear, their convergence rate will be highly dependent on the initialisation and different learning rate.

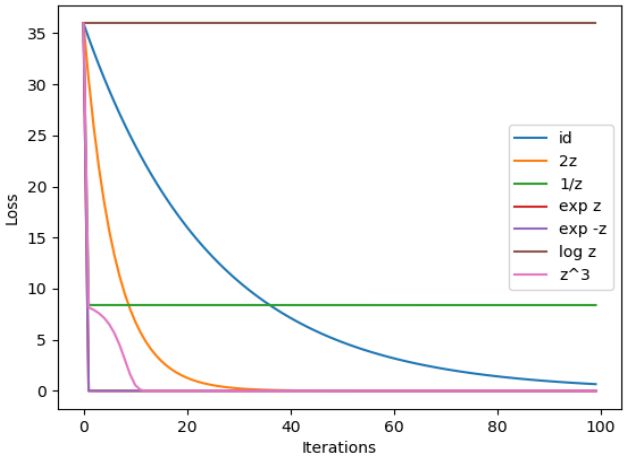

Figure A.12: The convergence rate of different functions. Note that because some of these are non-linear, their convergence rate will be highly dependent on the initialisation and different learning rate.

Appendix B

Abstraction

B.1 LMDPs

B.1.1 LMDP solutions

In the original Todorov paper [57], they derive the LMDP equations for minimising a cost function. This maximisation derivation just changes a few negative signs around.

$$V(s) = \max_u q(s) - \mathrm{KL}(u(\cdot|s) \parallel p(\cdot|s)) + \gamma \mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} V(s') \tag{1}$$

$$= q(s) + \max_u \left[\mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \log(\frac{p(s'|s)}{u(s'|s)} + \gamma \mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \big[V(s')\big] \right] \tag{2}$$

$$\log(z_{u^*}(s)) = q(s) + \max_u \left[\mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \log(\frac{p(s'|s)}{u(s'|s)} + \mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \big[\log(z_{u^*}(s')^\gamma)\big] \right] \tag{3}$$

$$= q(s) + \max_u \left[\mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \log(\frac{p(s'|s) z_{u^*}(s')^\gamma}{u(s'|s)}) \right] \tag{4}$$

$$G(s) = \sum_{s'} p(s'|s) z_{u^*}(s')^\gamma \tag{5}$$

$$= q(s) + \max_u \left[\mathop{\mathbb{E}}_{s' \sim u(\cdot|s)} \log(\frac{p(s'|s) z_{u^*}(s')^\gamma}{u(s'|s)} \cdot \frac{G(s)}{G(s)}) \right] \tag{6}$$

$$= q(s) + \log G(s) + \min_u \left[\mathrm{KL}\big(u(\cdot|s) \parallel \frac{p(\cdot|s) \cdot z_{u^*}(\cdot)^\gamma}{G(s)}) \right] \tag{7}$$

$$u^*(\cdot|s) = \frac{p(\cdot|s) \cdot z_{u^*}(\cdot)^\gamma}{\sum_{s'} p(s'|s) z_{u^*}(s')^\gamma} \tag{8}$$

$$\log(z_{u^*}(s)) = q(s) + \log\big(\sum_{s'} p(s'|s) z_{u^*}(s')^\gamma\big) \tag{9}$$

$$z_{u^*}(s) = e^{q(s)}\big(\sum_{s'} p(s'|s) z_{u^*}(s')^\gamma\big) \tag{10}$$

$$z_{u^*} = e^{q(s)} \cdot P z_{u^*}^\gamma \tag{11}$$

By definition, an LMDP is the optimisation problem in (1). We can move the max in (2), as $q(s)$ is not a function of u. Also in (2), expand the second term using the definition of KL divergence, the negative from the KL cancels the second terms negative. (3) Define a new variable, $z(s) = e^{v(s)}$. Also, use the log rule to move the discount rate. (4) Both expectations are under the same distribution, therefore they can be combined. Also, using log rules, combine the log terms. (5) Define a new variable that will be used to normalise $p(s'|s)z(s')^\gamma$. (6) Multiply and divide by $G(s)$. This

allows us to rewrite the log term as a KL divergence as now we have two distributions, $u(\cdot|s)$ and $\frac{p(\cdot|s)z(\cdot)^\gamma}{G(s)}$. (7) The change to a KL term introduces a negative, instead of maximising the negative KL, we minimise the KL. Also in (7) the extra G(s) term can be moved outside of the expectation as it is not dependent in s'. (8) Set the optimal policy to minimise the KL distance term. (9) Since we picked the optimal control to be the form in (8), the KL divergence term is zero. (10) Move the log. (11) Rewrite the equations for the tabular setting, where z is vector, and the uncontrolled dynamics are a matrix.

B.1.2 MDP Linearisation

The ability to solve LMDPs is great, but it's only useful if we can map MDPs into LMDPs, solve them, and map the solution back. Our goal here is to find a LMDP that has 'similar' structure to the original MDP we were given.[1]

[1]This derivation is not the same as in Todorov. He sets $b_a \neq r, b_a = r - \sum P \log P$.

$$\forall s, s' \in S, \forall a \in A, \exists u_a \text{ such that;} \tag{1}$$

$$\tau(s'|s,a) = u_a(s'|s)p(s'|s) \tag{2}$$

$$r(s,a) = q(s) - \mathrm{KL}(\tau(\cdot|s,a) \parallel u_a(\cdot|s)) \tag{3}$$

$$r(s,a) = q(s) - \mathrm{KL}(\tau(\cdot|s,a) \parallel \frac{\tau(\cdot|s,a)}{p(\cdot|s)}) \tag{4}$$

$$r(s,a) = q(s) - \sum_{s'} \tau(s'|s,a) \log(p(s'|s)) \tag{5}$$

$$m_{s'}[s] := \log p(s'|s) \tag{6}$$

$$D_{as'}[s] := p(s'|s,a) \tag{7}$$

$$c_{s'}[s] := q[s]\mathbf{1} - m_{s'}[s] \text{ such that } \sum_{s'} e^{m_{s'}[s]} = 1 \tag{8}$$

$$r_a = D_{as'}(q\mathbf{1} - m_{s'}) \ \ \forall s \tag{9}$$

$$r_a = D_{as'}c_{s'} \ \ \forall s \tag{10}$$

$$c_{s'} = r_a D^{\dagger}_{as'} \ \ \forall s \tag{11}$$

$$q = \log \sum_{s'} e^{c_{s'}} \ \ \forall s \tag{12}$$

$$m_{s'} = q - c_{s'} \ \ \forall s \tag{13}$$

We want to pick p, q such that the dynamics of every action in the original MDP can be represented with a control (2), and every reward generated by an action, can be given by a combination of the state rewards and the KL-divergence between the true dynamics and a control (3). Combine (2), (3) to yield (4). Expand the definition of KL-divergence to get (5). Now, we move to a tabular representation., where $m_{s'}[s]$ and $c_{s'}[s]$ are vectors, and $D_{as'}[s]$ is a matrix, defined in (6), (7), (8). With these new def-

initions, we can rewrite equation (5) as (9). The expectation can be moved to include q because it sums to one. Substitute equation (8) into (9) to get (10). Solve the linear equation in (11) to get the value of $c_{s'}$. Use the value of $c_{s'}$ to calculate the state rewards and unconditioned dynamics by using equations (12), (13) and (6).

B.2 Symmetries for RL

B.2.1 MDP homomorphisms

As pointed out in 3, the notion of an abstraction is captured by a homomorphism. Given this, it seems natural to extent the definition to MDPs.

A MDP homomorphism is a transformation of a MDP, $\mathcal{H} : \mathcal{M} \to \mathcal{M}$, that preserves the transition and reward function [77]. We can describe this MDP homomorphism as $\mathcal{H} = (f, g)$ such that;

$$\tau(f(s')|f(s), g_s(a)) = \sum_{s'' \in [s']_f} \tau(s''|a, s)$$

$$r(f(s), g_s(a)) = r(s, a)$$

This MDP homomorphism framework yields state-action abstraction, that uses a model based notion of similarity. However, as pointed out in earlier sections, there are many other possible notions of abstraction and similarity that can make sense for RL. Specifically, the MDP homomorphism framework could be generalised in the following ways;

1. approximate symmetries
2. complexity measure / inductive bias
3. inference of symmetries under uncertainty
4. temporal symmetries

Indeed, some work has extended the notion of symmetric model-based abstraction (1) to approximately symmetric model-based abstraction [78].

However, this has yet to be generalised to other types of symmetry with (say) the state-action values.

Section 3.3 is our not-so-humble attempt to achieve a symmetric inductive bias (2). And, as far as we know, little progress has been made on (3).

Finally, there has been at least one attempt [79] to build a framework capable of exploiting temporal symmetries (4). And while their proposal makes sense, we offer an alternative below.

B.2.2 Temporal symmetries

Consider a transformation on a MDP that takes sequences of actions, an option, $a_1, a_2, \ldots, a_k$, and relabels them as actions in a new MDP.

$$\begin{aligned}
\Omega_k &: M \to M \\
\Omega_k &: \{S, A, r, P, \gamma\} \to \{S, A^k, \hat{T}_k(r), \hat{T}_k(P), \gamma\} \\
\hat{T}_k(r) &= \sum_{i=t}^{t+k} \gamma^{i-t} r(s_i, a_i) \\
\hat{T}_k(P) &= \prod_{i=t}^{t+k} \tau(s_{i+1}|s_i, a_i)
\end{aligned}$$

It seems easy to see that the transformation of the actions is bijective[2](each option is identified with one action), and therefore we have an isomorphism (which is a type of homomorphism).

We can now define a temporal homomorphism as the composition of our temporal transformation and the MDP homomorphism.

[2]Although, care does need to be taken with the transitions and rewards. We must ensure that the transitions / rewards for an options are the same as the transition / reward for the relabelled option (now a single action).

$$\mathcal{H} : M \to M \qquad \text{(MDP homomorphism)}$$
$$\Omega_k : M \to M \qquad \text{(temporal transformation)}$$
$$\mathcal{H} \circ \Omega_k : M \to M \qquad \text{(temporal homomorphism)}$$

Rather than finding model-based symmetries in the state-actions. This construction will find symmetries in state-options (of a specific length).

B.2.3 Invariants

An important property of a group is that they can be (uniquely) identified by their invariant relations [80]. Together, the invariant relations and generators can be use to construct a group. For example, the cyclic groups can be generated by a single element, which represents an action like $+1$. And the invariant relation is $(+1)^n = e$. This relation says, that after applying the element to itself n times, it loops back to the identity element, e. Thus it makes a cycle. And the group of order n, with only the single invariant relation, $a^n = e$, must be the cyclic group of order n.

So, how can we use invariant relations (and the generators) to help infer symmetries in reinforcement learning problems?

- What does an invariant in the transition function (or reward function) imply about the value function?
- Which symmetries can we identify by using invariants in the value function?
- How hard is (computationally) it to find these invariants?

Cart pole

Let's work through an example, the cart-pole control problem. The goal is to balance a pole on a cart. Where, the cart can be moved left or right.

The states of the cart pole problem are described by $s_i = [p_c^i, v_c^i, p_p^i, v_p^i]$[3]. Where, p_c is the position of the cart, v_c is the velocity of the cart, p_p is the position of the pole, v_p is the angular velocity of the pole. And the actions are -1 for left and 1 for right.

Mirror Symmetry

We can 'flip' the cart pole problem, and it remains the same problem.

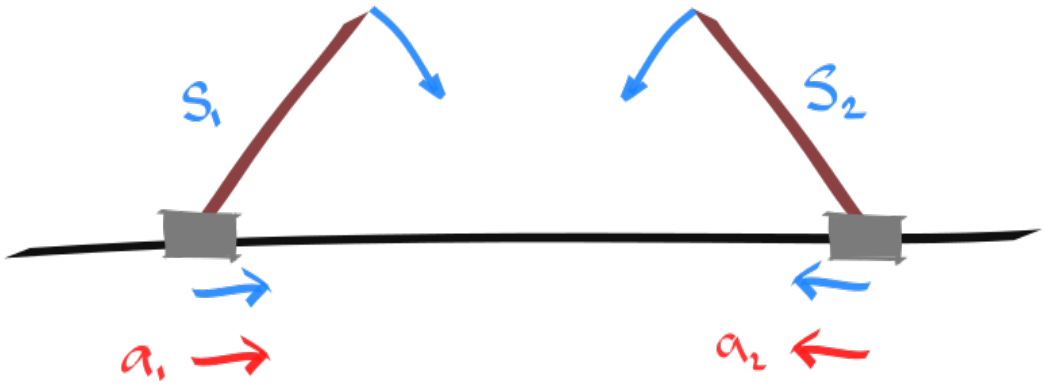

Figure B.1: Two mirror symmetric state-actions of a cart pole.

Let the group $g \in S_2$ act on the

- state space as $g \circ s = -s$
- action space as $g \circ a = -a$

Therefore, we can write the;

- policy as $g \circ \pi(a|s) = \pi(g \circ a | g \circ s) = \pi(a|s)$.
- transition function as $(g \circ \tau)(\cdot|s, a) = \tau(\cdot|g \circ s, g \circ a) = \tau(\cdot|s, a)$.
- reward function as $(g \circ r)(s, a) = r(g \circ s, g \circ a) = r(s, a)$.
- state-action values as $(g \circ Q^\pi)(s, a) = Q^{g \circ \pi}(g \circ s, g \circ a) = Q^\pi(s, a)$.

[3]Where the cart's position is centered around some fixed starting point. And the rotations are measured relative up being upright.

So, what are some invariants we might care about in the RL context?

$$Q^\pi(s,a) = (g \circ Q^\pi)(s,a) \qquad \text{(expected return)}$$

$$T(Q^\pi)(s,a) - Q^\pi(s,a) = T(g \circ Q^\pi)(s,a) - (g \circ Q^\pi)(s,a) \qquad \text{(Bellman residual)}$$

$$\mathop{\mathbb{E}}_{s' \sim \tau(\cdot|s,a)}(s' - s) = g \circ \mathop{\mathbb{E}}_{s' \sim (g \circ \tau)(\cdot|s,a)}(s' - g \circ s) \qquad \text{(change in state)}$$

So, the cart pole problem is invariant to S_2. But, how could this have been identified from the invariants above?

Translational Symmetry

The problem is essentially the same if we move to the left or right.

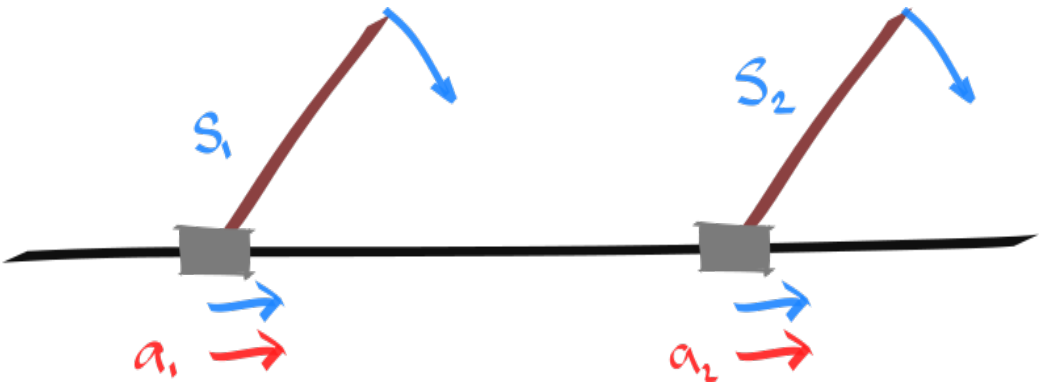

Figure B.2: Two similar cart poles, which are a translation different from each other.

Define the action of $g \in G$ on the

- state space $g \circ s = [g + p_c, v_c, p_p, v_p]$
- action space $g \circ a = a$

Therefore, we can write the;

- policy as $g \circ \pi(a|s) = \pi(a|g \circ s) = \pi(a|s)$.
- transition function as $(g \circ \tau)(\cdot|s,a) = \tau(\cdot|g \circ s, a) = \tau(\cdot|s,a)$.
- reward function as $(g \circ r)(s,a) = r(g \circ s, a) = r(s,a)$.
- state-action values as $(g \circ Q^\pi)(s,a) = Q^{g \circ \pi}(g \circ s, a) = Q^\pi(s,a)$.

So, what are some invariants we might care about in the RL context?

$$Q^\pi(s,a) = (g \circ Q^\pi)(s,a) \qquad \text{(expected return)}$$

$$T(Q^\pi)(s,a) - Q^\pi(s,a) = T(g \circ Q^\pi)(s,a) - (g \circ Q^\pi)(s,a) \qquad \text{(Bellman residual)}$$

$$\mathop{\mathbb{E}}_{s' \sim \tau(\cdot|s,a)}(s' - s) = g \circ \mathop{\mathbb{E}}_{s' \sim (g \circ P)(\cdot|s,a)}(s' - g \circ s) \qquad \text{(change in state)}$$

Despite two different groups acting on the cart pole problem, they have the same invariants (that we considered). Which other invariants should we be measuring, and how would they help us narrow the possible symetries?

We consider some other examples as well. But don't construct their invariants.

Transition Symmetry

> *Despite starting in different states, and applying different actions, we might end up in the same next state.*

If we allow actions to be continuous, where we get to choose the impulse $a \in [-c, c]$. Then we we get a new set of invariants of the transition function $\tau(s'|s,a) = \tau(s'|g \circ (s,a))$. But what do these invariants of the transition function imply about invariants in the state (and action) values?

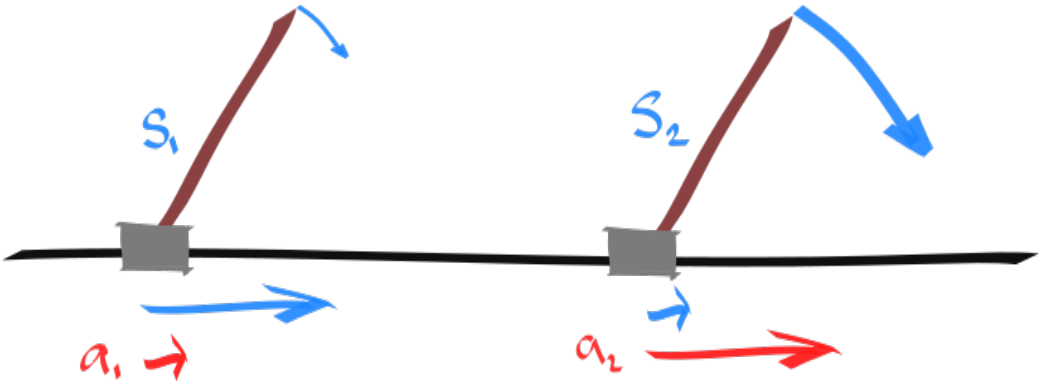

Figure B.3: Invariants of the transition function. Two different actions take two different states to the same one. Roughly, the intuition is; a small push of a fast moving object achieves the same results a large push of a slow moving object.

Temporal Symmetry

There exist multiple ways of achieving the same thing.

This is a temporally extended version of the Transition Symmetry (above).

$$\Upsilon(s_t|s, a_1, \dots, a_t) = \sum_{s_1, s_2, \dots s_{t-1}} \prod_{i=0}^{t} \tau(s_t|s_{t-1}, a_{t-1})$$
$$\Upsilon(s'|s, a_1, \dots a_t) = \Upsilon(s'|s, g \circ (a_1, \dots a_t))$$

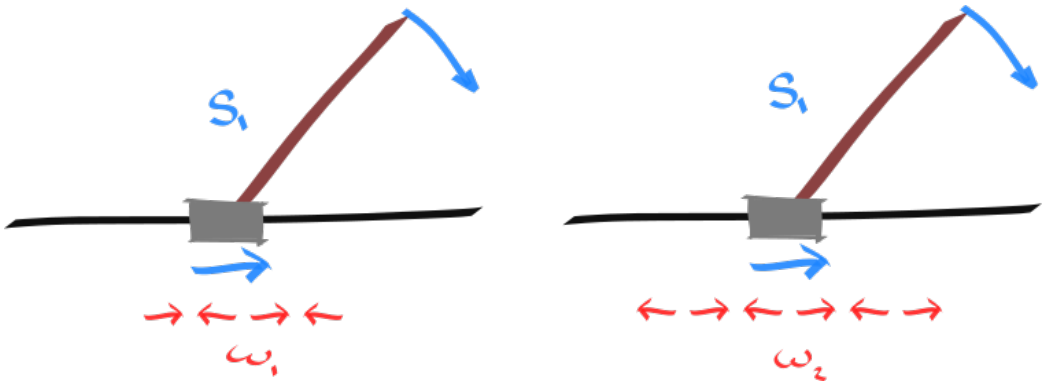

Figure B.4: Temporal invariants. For example, we might be able to reorder actions, without changing the effect (*putting your left sock on before you put on your right sock*). Or we might be able to use different numbers of actions to achieve the same thing (*three left turns makes a right*).

Pong

The states of Pong are described by $s = [p_1, v_1, p_2, v_2, p_b^x, p_b^y, v_b^x, v_b^y]$ (all of these are centered around the middle of the table). Where, p_1 is the position of player 1's paddle, v_1 is the velocity of player 1's paddle, p_b^x is the x position of the ball, v_b^y is the y component of the ball's velocity. And the actions are -1 for left and 1 for right.

Pong also has mirror symmetry in the same sense as the cart pole problem. And, it has another type of mirror symmetry.

Mirror symmetry (player perspective / horizontal) Because Pong is a zero sum, two player game, there is a symmetry of perspective. Whether you are playing as player 1 or player two, you are still playing the same game of Pong, but with inverted pay-offs.

Let $G = (\{e, g\}, \circ)$. And define the action of g on the;

- state space $g \circ [p_1, v_1, p_2, v_2, p_b^x, p_b^y, v_b^x, v_b^y] := [p_2, v_2, p_1, v_1, -p_b^x, p_b^y, -v_b^x, v_b^y]$
- action space $g \circ a := a$
- rewards $g \circ r(g \circ s, g \circ a) := -r(g \circ s, g \circ a)$

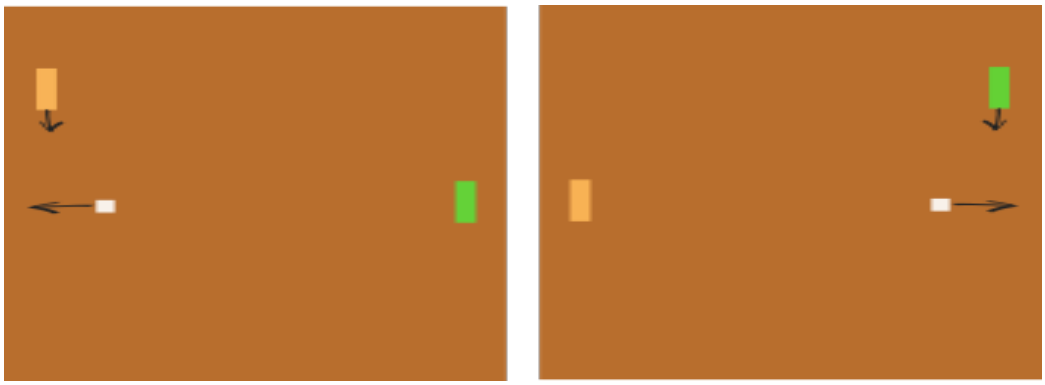

Figure B.5: The invariants of a shift in player perspective.

Therefore, we can write the;

- policy to be $g \circ \pi(a|s) = \pi(g \circ a|g \circ s) = \pi(s, a)$.
- transition function to be $(g \circ \tau)(s'|s, a) = \tau(g \circ s'|g \circ s, g \circ a) = \tau(s'|s, a)$.
- value function is $(g \circ Q^\pi)(s, a) = g \circ r(s, a) + \gamma(g \circ \tau)(s'|s, a)(g \circ Q)^\pi(s, a) = Q^\pi(s, a)$

B.3 Symmetry and machine learning

All of these methods rely on a similar type of method: use the symmetry to construct orbits, and then average or pick a representative.

B.3.1 Exploitation

> *Once we have discovered a symmetry, how might we exploit that discovery?*

Similar to how we considered how to exploit an abstraction in section 3.1.1, let's review some existing methods for exploiting the knowledge of a symmetry.

Exploiting symmetry for efficient control

If we have a MDP, M_1, then solving it via value iteration requires $\mathcal{O}(\epsilon|S|^2|A|)$ iterations. However, if we know that there exists symmetry of order k in the state space, then we can 'minimise the model', by applying the MDP homomorphism $\mathcal{H} : \mathcal{M} \to \mathcal{M}$. This new, minimised, MDP, M_2 has a smaller state space, as $|S_{M_2}| = \frac{|S_{M_1}|}{k}$ and essentially the same dynamics and rewards. Thus we can solve M_2, with cost $\mathcal{O}(\epsilon\frac{|S|^2|A|}{k^2})$ and then lift the solution back to M_1. [81, 82]

Exploiting symmetry for efficient inference

There has been a large amount of work (that we are familiar with) exploring the exploitation of symmetries for faster learning. The essence of the idea is *"invariance reduces variance"* [83].

Possibly the most famous exploitation strategy is data augmentation [61]. But, there are other techniques;

- Use the known symmetries to build invariant network architectures [84, 85]
- By sharing weights according to group structure [64, 86]
- Output coupling [87, 84]
- Gradient coupling

Gradient coupling Inspired by the view of neural network updates as being controlled by a 'neural tangent kernel' [88], here we present another way to exploit symmetries for machine learning.

Let $f : X \to Y$ be some trainable function.

$$\dot{y_j} = \sum_{i \neq j} \alpha_{ij} \nabla_\theta \ell(y_i, \theta)$$

Then neural networks share updates between examples according to the neural tangent kernel $\alpha_{ij} = \langle \nabla_\theta \ell(x_i, \theta), \nabla_\theta \ell(x_j, \theta) \rangle$. However, we could

pick another way to do this update. Possibly using the symmetries to group x_is and x_js.

A note about discovery

The discovery of symmetries within data has had little success. However, with the framing of data augmentation as a form of symmetry exploitation, this implies that automated data augmentation [89, 90, 91, 92] is a form of symmetry discovery.

These methods discover which symmetries apply to a given domain, and at what magnitude. They tend to frame the optimisation problem as picking the probability of a set of given op and their magnitude. For instance, [89] provides a small set of operations (aka symmetries): *Identity, AutoContrast, Equalize, Rotate, Solarize, Color, Posterize, Contrast, Brightness, Sharpness, ShearX, ShearY, TranslateX, TranslateY.* Validation error is then used as a reward for learning.

While this approach does work, like other 'meta-learning techniques', it does not scale well.

B.3.2 Actions

Let G be a group. Where we represent the action of $g \in G$ on the real vectors, $x \in R^n$ as permutation matrices, $\phi(g, x) = P_g \circ x$, where the composition operator $\circ$ becomes matrix multiplication, $\cdot$. Therefore $e = I_n$ and g can be one of many possible actions. For example, consider the representations of the n-gram swaps [4]of g when applied R^4.

[4]There are no others. Proof by construction. Enumerate all permutations and check which ones are idempotent.

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}
$$
$$
\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}
$$
$$
\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}
\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}
$$

The first two rows contain permutations that swaps (two) elements of the vector ($a \leftrightarrow b$), these are the possible actions of S_2. The last row contains permutations that swaps pairs ($(a, b) \leftrightarrow (c, b)$), this entire row is the only representation of $S_2 \times S_2$.

Race grid world

We want to construct a simple symmetric problem to test our learners. The intuition behind this toy problem comes from a $100m$ sprint. It doesn't matter which lane you are in, you should run forwards, not sideways...

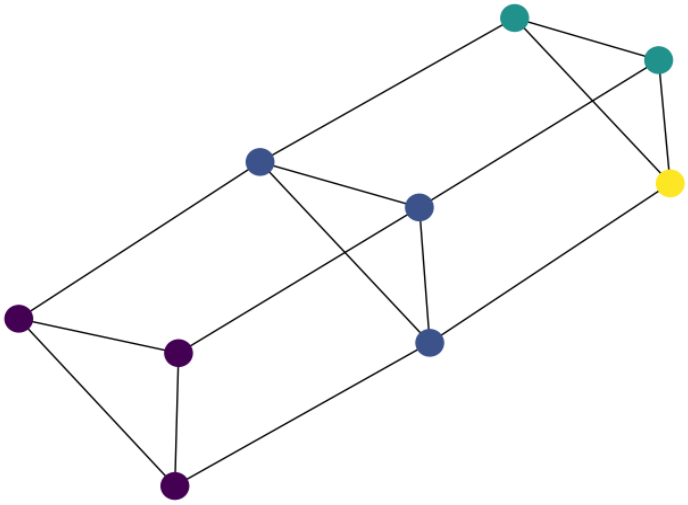

Figure B.6: The purple nodes represent the starting states. The green-teal nodes represent states wth reward 1. There are 5 actions, left, right, up, down, none. Starting from the leftmost purple node: Left moves clockwise to the next purple node. Up moves to the blue node. Down doesn't result in movement. None doesn't result in movement.

B.4 n-dimensional Cart pole

How can we test a learners ability to detect symmetries and exploit them?

We propose a simple test, the n-dimensional cart pole: a generalisation of the cart pole problem to n dimensions. Rather than receiving observations in $\mathbb{R}^4$ (the position, velocity, angle and angular velocity), observations are in $\mathbb{R}^{4\times n}$. And the action space is generalised from $\{0, 1\}$ (left and right), to $\{0, 1\}^n$.

B.4.1 How is this problem symmetric?

The n-dimensional cart pole problem can be reduced to n, one dimensional cart pole problems. Where each of these one dimensional cart pole prob-

lems is easy to solve.

In a more formal sense. This problem is symmetric because the optimal policy and its (Q) values are invariant to the actions of the permutation group of order n, S_n.

$$
\begin{aligned}
g \circ s_j &= g \circ [x_0, \ldots, x_i, \ldots, x_{n-1}] \\
&= [x_1, x_0, \ldots, x_i, \ldots, x_{n-1}] \\
g_i \circ a_k &= g \circ [u_0, \ldots, u_i, \ldots, u_{n-1}] \\
&= [u_1, u_0 \ldots, u_i, \ldots, u_{n-1}] \\
g \circ \tau(s'|s,a) &= \tau(g \circ s' | g \circ s, g \circ a) \\
g \circ R(s,a) &= R(g \circ s, g \circ a) \\
g \circ \pi^*(a|s) &= \pi^*(g \circ a | g \circ s) \\
&= \pi^*(a|s) && \text{(invariance of the optimal policy)} \\
g \circ Q^{\pi^*}(s,a) &= Q^{\pi^*}(g \circ s, g \circ a) \\
&= Q^{\pi^*}(s,a) && \text{(invariance of the optimal values)}
\end{aligned}
$$

We describe a state as $s_j = [x_0, \ldots, x_i, \ldots, x_{n-1}]$, where $x_i = (p_c^i, v_c^i, p_p^i, v_p^i)$. Where, p_c is the position of the cart, v_c is the velocity of the cart, p_p is the position of the pole, v_p is the angular velocity of the pole. We describe actions as $a_k \in \{0,1\}^n$. Let $g \in S_n$ be the pairwise permutation, swapping the first two elements $(0 \to 1)$.

B.4.2 An advantage

What advantage is provided by exploiting symmetries?

If a learner has inferred that the n-dimensional cart pole problem can be decomposed into n identical sub problems, then that means it is gathering n times the data for the one-dimensional cart pole problem. So, we should see a factor of n speed up in learning. This is the same argument made here [quotient groups appendix...].

For a learner that doesn't know of the symmetries. How is this problem hard? The more dimensions there are, the more ways there are to fail. Consider how exploration is done. In a single dimension, actions are taken with probability is taken with some chance of exploring instead. Maybe you correctly balanced the pole in all dimensions except one. To bad, you don't get any reward.

B.4.3 Experiments

We use OpenAI's Gym [93] and Baselines [94] to test this environment.

Note: 'Average mean reward' refers to the fact that we have averaged (n=5) the mean reward (per episode). Also note: This reward is the training performance.

As mentioned in the previous section, we expected learning to become much harder for a learner that doesn't exploit symmetries. These results suggest either of two possibilities: that PPO2 can discover and exploit symmetries, our setting does not test what we think it does.

While investigating this further, we realised that the given action space, *MultiBinary*, provides a large amount of information. We ran another test with a *Discrete* action space. Where the learner gets to choose $a \in \mathbb{Z}_n$. This action then gets (binary) decoded to the *MultiBinary* format.

A learner that exploits the permutation symmetries in the n-dimensional cart pole problem should learn n times quicker. However, the cost of discovering this permutation symmetry is unknown.

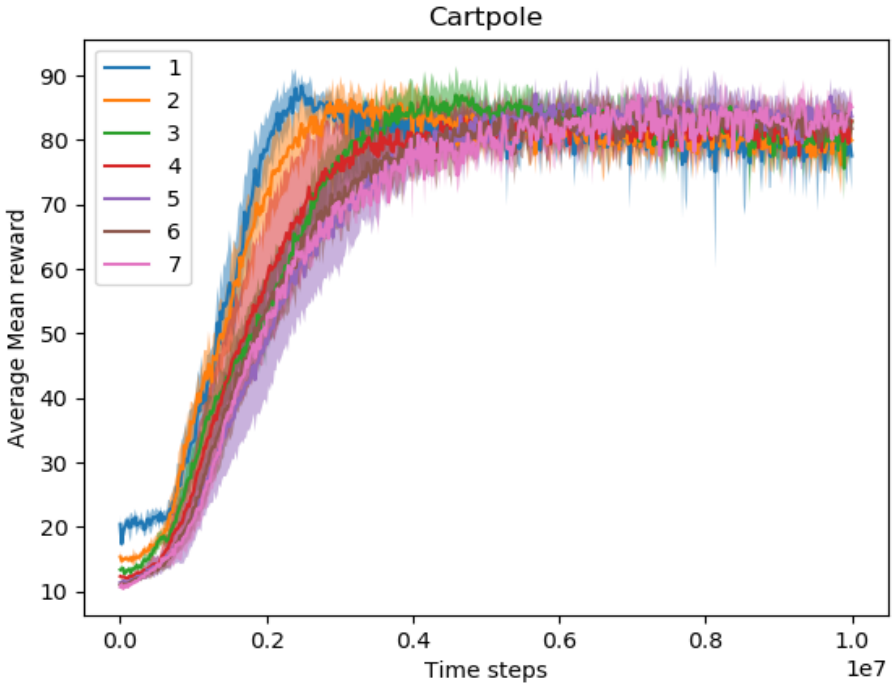

Figure B.7: Default PPO2 solving the nd cartpole problem with access to a *MultiBinary* action space. Each color corresponds to a the average mean return of different, n, the number of repeated cart pole problems.

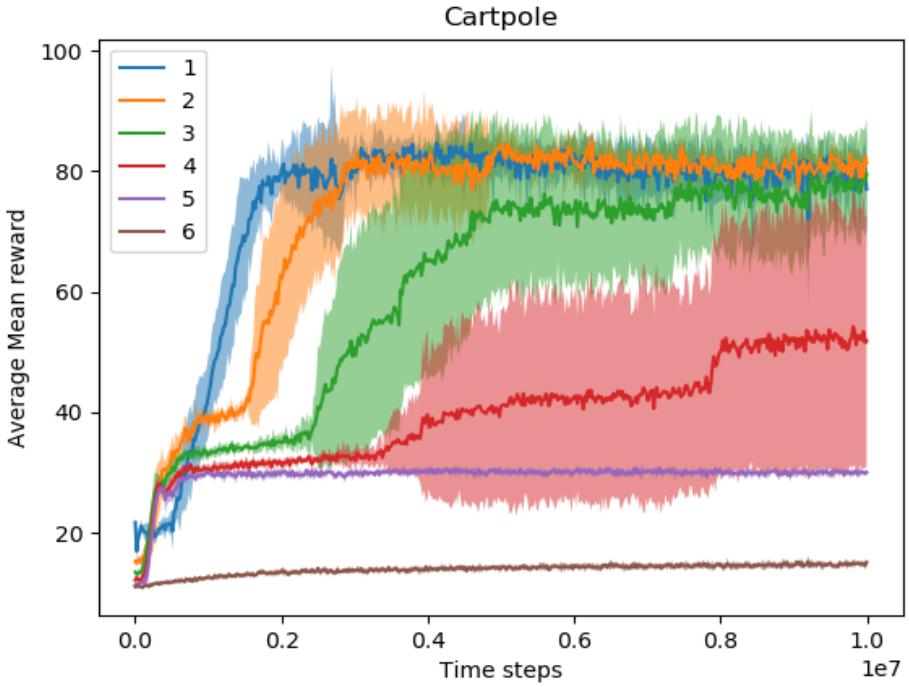

Figure B.8: Default PPO2 solving the nd cartpole problem with access to a *Discrete* action space. Each color corresponds to a the average mean return of different, n, the number of repeated cart pole problems.