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
2019

Abstract

An abstract of fewer than 500 words must be included.

Acknowledgments

I would like to thank my advisor, Will Browne, for supporting my work and giving me the freedom to explore my interests.

Contents

Chapter 1

Introduction

Blah blah. Challenges of Real-World Reinforcement Learning. - Learning on the real system from limited samples. - High-dimensional continuous state and action spaces.

1.1 Reinforcement learning

> Reinforcement learning (RL) defines a type of problem, closely related to Markov decision problems (MDPs).

A Markov decision problem is defined as the tuple, $\{\mathcal{S}, \mathcal{A}, P, r\}$. Where $s \in \mathcal{S}$ is the set of possible states (*for example arrangements of chess pieces*), $a \in \mathcal{A}$ is the set of actions (*the different possible moves, left, right, diagonal, weird L-shaped thing, ...*), $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0 : 1]$ is the transition function which describes how the environment acts in response to the past (s_t) and to your actions (a_t) (*in this case, your opponent's moves, taking one of your pieces, and the results of your actions*), and finally, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, (*whether you won (+1) or lost (-1) the game*) and $R = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$ is the discounted cumulative reward, or return. The player's goal, is to find a policy π, (which chooses actions, $a_t = \pi(s_t)$) that yields the largest return (max R).

A RL problem is an extension of the MDP definition adove. Where, rather than the learner being provided the state space, action space, transition function and reward function ($\{\mathcal{S}, \mathcal{A}, P, r\}$), the learner recieves samples (s_t, a_t, r_t). From these samples the learner can either; - attempt to infer the transition and reward functions (known as model-based reinforcement learning), or attempt to estimate value directly (model-free reinforcement learning). - collect the samples in memory and use them to find a policy (offline learning), or - on / off policy - bootstrap / not - types of model (fn approximators)

For example _"Dynamic programming is one type of RL. More specifically, it is a value-based, model-based, bootstrapping and off-policy algorithm. All of those traits can vary. Probably the"opposite" of DP is REINFORCE which is policy-gradient, model-free, does not bootstrap, and is on-policy. Both DP and REINFORCE methods are considered to be Reinforcement Learning methods."_ SE

1.1.1 Understanding Theoretical Reinforcement learning

What are its goals. Its definitions. It methods?

- Optimality
- Model based
- Complexity
- Abstraction

Recent work has bounded the error of representation learning for RL. Abel et al. 2017, Abel et al. 2019

But. It is possible that this representation achieves no compression of the state space, making the statement rather vacuous. Further more, it consider how easy it is to find the optimal policy in each of the two representations. It is possible to learn a representation that makes the optimal control problem harder. For example. TODO

Current theory does not take into account the structure within a RL problem.

The bounds are typically for the worst case. But these bounds could be tighter if we exploited the structure tht exists in natural problems. The topology of the transition function; its, sparsity, low rankness, locality, The symmetries of the reward function. ??? (what about both?!)

1.1.2 Understanding Markov decision problems

- Properties of the polytope
- Search dynamics on the polytope
- ??? LPs? Convergence? Exploration? ...?

1.1.3 Abstraction

- Near optimal representations
- Solvable representations (LMDPs)
- Invariant representations (TODO)

Algorithms

We explore four algorithms.

- Memorizer: This learner memorizes everything it sees, and uses this knowledge as an expensive oracle to train a policy.
- Invariant. This learner discovers symmetries in its evironment and uses this knowledge to design an invariant representation.
- Tabular. ...
- MPC. ...

Chapter 2

MDPs

What is a decision problem?

2.0.1 Sequential decision problems

Define. And give example.

If we wanted we could pick our actions before we make observations, reducing the search space to only $|A| \times T$. But this is a bad idea... example.

2.0.2 MDPs

- Observable
- Deterministic
- Synchronous
- Terminating
- Knowledge of the model
- Discrete

MDPs are a subset of sequential decision problem. Define MDPs. Give example.

When actions you have taken in the past can bite you in the butt... Maze with pendulums / doors. When moving through the maze, you must swing the pendulums. In the future you must avoid being hit. (maybe make a picture of this?) also, is there a more general way to think about it?

The general feeling of an MDP. - Actions need to be adapted to new observations and contexts. - While instantaneous results are good, we care about the longer term aggregates.

$$Q^\pi(s_0, a_0) = r(s_0, a_0) + \gamma \max_{a_1} \mathop{\mathbb{E}}_{s_1 \sim p(\cdot|s_0,a_0)} \Bigg[r(s_1, a_1)$$

$$+\gamma \max_{a_2} \mathop{\mathbb{E}}_{s_2 \sim p(\cdot|s_1,a_1)} \Bigg[r(s_2, a_2) + \gamma \max_{a_3} \mathop{\mathbb{E}}_{s_3 \sim p(\cdot|s_2,a_2)} \Big[\dots \Big] \Bigg] \Bigg]$$

The Markov property

What does the M in MDP mean?

When we say a decision problem is Markovian, we mean that the transition function acts as a Markov chain. The next transition step depends only on the current state. It is invariant to any / all histories that do not change the current state.

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 **observable** state.

Can easily make a sequence Markovian by adding information. E.g. time

Optimality

And importantly, existing theory tells us that there is a unique optima to the bellman iterations. And that this optimal policy(ies) is(are) necessarily deterministic.

(why does this make sense?)

How do MDPs relate to RL?

Reinforcement learning set of solutions to a general type of problem. This general, reinforcement learning problem, has the properties;

- evaluation, not feedback. Learners are never provided information about what makes a good policy, rather they told whether a policy is good or not.
- delayed credit assignment.

MDPs have these properties, so are considered within RL. They are also within the fields of Operational Research, Optimal Control, Mathematical Optimisation, Stochastic Programming.

A tabular representation of MDPs

Tabular MDPs with deterministic actions are of little interest to the ML community. Not because they are easy, but because they do not involve ...? They can be solved by planning techniques and dynamic programming.

The minimally complex MDP that poses an interesting challenge to the ML community is when the transition function is non deterministic. Alternatives we could add on. Contextua decision problem (transition fn changes with t), stochastic reward function, ...?

Learn a tabular MDP representation of the RL problem.

Why would we want to do this? - Policy evaluation is expensive in the RL setting. The policy must be simulated over all possible states-action pairs. And scales poorly with variance. (how poorly?) - ?

Just quickly, what does a tabular MDP look like? - discrete states and actions - r and P are simply look up functions, indexed by the current state-action.

$$
\begin{aligned}
V &= r_\pi + \gamma P_\pi V && \text{(bellman eqn)} \\
V - \gamma P_\pi V &= r_\pi && (2.1) \\
(I - \gamma P_\pi) V &= r_\pi && (2.2) \\
V &= (I - \gamma P_\pi)^{-1} r_\pi && (2.3) \\
& && (2.4)
\end{aligned}
$$

(finding the optimal policy is still a non-linear problem. how / why is it non-linear?!)

Learning the (tabular) abstraction

Most recent advances in ML have been by finding clever ways to extend supervised learning techniques to unsupervised learning. Similarly, we can use supervised learning techniques, batch training, cross entropy, ... to train reward and transition approximations.

We are provided with examples $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$. We can use these to...

$$
\begin{aligned}
&\mathbf{r} \in \mathbb{R}^{n\times m}, \mathbf{P} \in [0,1]^{n\times m\times n} && (2.5) \\
L_r &= \min \parallel r_t - \mathbf{r}[\phi(s_t), a_t] \parallel_2^2 && \text{(mean squared error)} \\
L_P &= \max_\theta \mathbf{P}[\phi(s_{t+1}), \phi(s_t), a_t] && \text{(max likelihood)} \\
& && (2.6)
\end{aligned}
$$

Why is the discount factor a part of the definition of the MDP? Initially, it didnt make sense to me. By defining the discount, it ensure the MDP has a unique solution.

2.1 The value function polytope

Why is it a polytope?

Imagine a two state MDP. Following some initial, ill-informed policy, the value that you might get starting from each state is ${v_1}^0, {v_2}^0$. Nn the future we learn something new and alter our policy; so the value of (say) the first state is now greater, $v_1^t > v_1^0$. This explains why the edges of the polytope by be "aligned with the positive orthant", they slant upward. An increase in the value of state one, can, at worst, do nothing for state two, aka a flat line, either horizontal or vertical.

What are its properties?

Some simple question to explore;

- How does the distribution of policies on the polytope effect learning?
- How does gamma change the shape of the polytope?
- How do the dynamics of GPI partition the policy / value spaces?

2.1.1 Distribution of policies

A potentially interesting question to ask about the polytopes is how the policies are distributed over the polytope. To calculate this analytically, we can use the probability chain rule: $p(f(x)) =\mid \det \frac{\partial f(x)}{\partial x} \mid^{-1} p(x)$. Where we set f to be our value functional and $p(x)$ to be a uniform distribution.

- **Observation** In some polytopes, many of the policies are close to the optimal policy. In other polytopes, many of the policiesare far away from the optimal policy. **Question** Does this make the MDP harder or easier to solve? **Intuition** If there is a high density near the optimal policy then we could simply sample policies and evaluate them. This would allow us to find a near optimal policy with relative easy.
- **Observation** The density is always concentrated / centered on an edge.

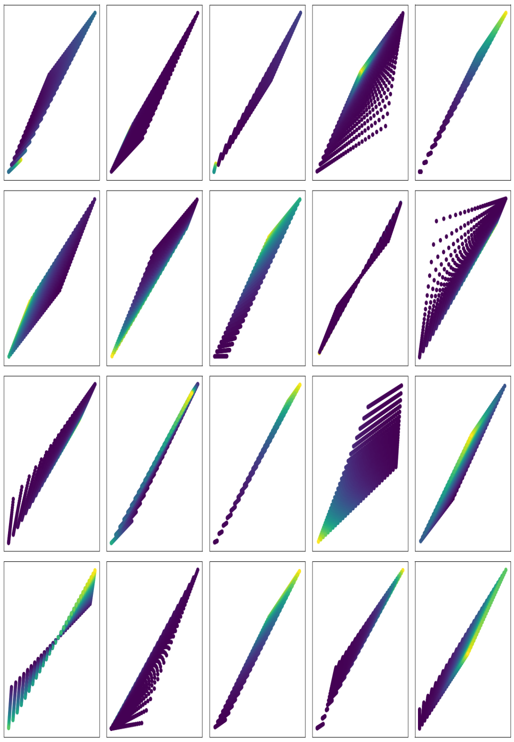


Figure 2.1: '2-state 2-action MDPs. We have visualised the likelihood of values under a uniform on policies. They are coloured by density. Lighter colour is higher probability'

- **Question** how does the entropy of the distribution change under different gamma/transitions/rewards. . . ?

Derivation of derivative

$$
\begin{aligned}
V(\pi) &= (I - \gamma P_\pi)^{-1} r_\pi && (2.7)\\
&= (I - \gamma P \cdot \pi)^{-1} r \cdot \pi && (2.8)\\
\frac{\partial V}{\partial \pi} &= \frac{\partial}{\partial \pi}((I - \gamma P_\pi)^{-1} r_\pi) && (2.9)\\
&= (I - \gamma \pi P)^{-1} \frac{\partial \pi r}{\partial \pi} + \frac{\partial (I - \gamma \pi P)^{-1}}{\partial \pi} \pi r && \text{(product rule)}\\
&= (I - \gamma \pi P)^{-1} r + -(I - \gamma \pi P)^{-2} \cdot -\gamma P \cdot \pi r && (2.10)\\
&= \frac{r}{I - \gamma \pi P} + \frac{\gamma P \cdot \pi r}{(I - \gamma \pi P)^2} && (2.11)\\
&= \frac{r(I - \gamma \pi P) + \gamma P \pi r}{(I - \gamma \pi P)^2} && (2.12)\\
&= \frac{r}{(I - \gamma P \pi)^2} && (2.13)\\
& && (2.14)
\end{aligned}
$$

An MDPs Entropy (*the goal is to understand what makes some MDPs harder to solve than others*)

We can visualise polytopes in 2D, but we struggle in higher dimensions. However, it is possible to use lower dimensions to gain intuition about metrics and carry that intuition into higher dimensions. A potential metric of interest here is the entropy of our distribution, (and / or the expected distance from the optima) to give intuition about unimaginable MDPs.

$$
\begin{aligned}
M &\to \{P, r, \gamma\} && \text{(a MDP)} \\
H(M) &:= \mathop{\mathbb{E}}_{\pi\sim\Pi}\Big[-\log p(V(\pi))\Big] && (2.15) \\
&= \mathop{\mathbb{E}}_{\pi\sim\Pi}\Big[-\log(|\det\frac{\partial V(\pi)}{\partial \pi}|^{-1} p(\pi))\Big] && (2.16) \\
&= \mathop{\mathbb{E}}_{\pi\sim\Pi}\Big[-\log(|\det\frac{r}{(I-\gamma P\pi)^2}|^{-1} p(\pi))\Big] && (2.17) \\
& && (2.18)
\end{aligned}
$$

What does this tell us? **???** A MDP with a low entropy tells us that many of the policies are in a corner of the polytope. But the 'hardness' of the MDP depends on which corner these policies are concentrated in. Rather we could use the value of each policy to give information about the location of the policy.

$$
\mu(M) := \mathop{\mathbb{E}}_{\pi\sim\Pi}\Big[V(\pi)\Big]
$$

What does this tell us? The expected value of a policy. Thus, a quantity of interest might be the expected suboptimality of a policy, $s = V(\pi^*) - \mu(M)$. This tells us how far away the optimal policy is from the center of mass of the polytope.

Conjecture: If an MDP has suboptimality $s \le \frac{\sigma_{MDP}}{D}$ then it is possible to find a ϵ optimal policy with $\mathcal{O}(n)$ samples. (but sampling in high dimensions always scales badly?!)

Experiment: Correlate the properties of P, r with entropy. Or find derivative wrt P, r. What properties of P, r yield easily solvable MDPs?

NOTE:

- What about the variance of the MDP? What does that tell us?

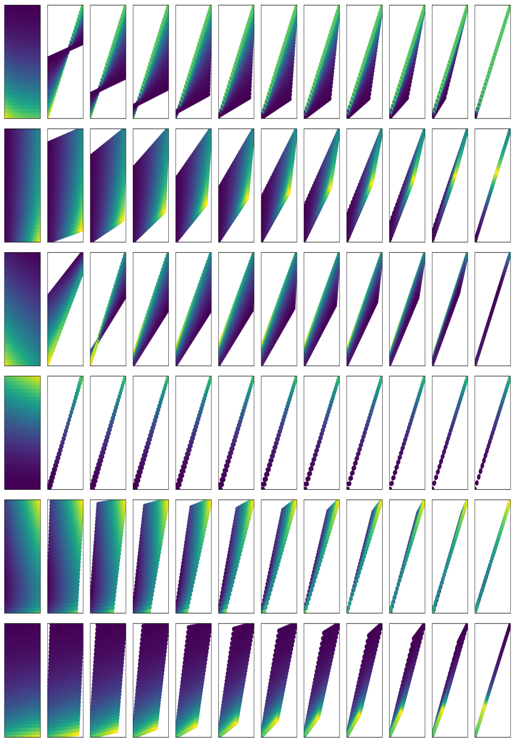


Figure 2.2: '2-state 2-action MDPs. Here we have shown a few different P/r MDPs and how their polytopes change with changes in discount rate.'

- How does a uniform distribution on a simplex behave in high dimensions? Does it become more likely to sample from the center? Less likely to sample from vertices??
- In most cases, this is unlikely to work. A high dimensional polytope … low density everywhere!?

2.1.2 Discounting

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 explore how the shape of the value polytope changes.

- **Observation** As $\gamma \to 1$, all the policies are projected into a 1D space? **Question** Does this make things easier to learn? **Intuition** Orderd 1D spaces are easy to search.

- **Observation** The tranformation that changing the discount applies is quite restricted. They are not generally non-linear, but appear 'close to linear', but not quite. **Question** What is the set of functions /transformations that the discount can apply?

2.2 Search spaces

2.2.1 Dynamics and complexity

TODO Complexity. How many iterations!!! Look up from literature and do some empirical tests.

(we want to know how much it costs to find the optima)

For each initial policy, we can solve / optimise it to to find the optimal policy (using policy iteration). Here we count how many iterations were required to find the optima (from different starting points / policies).

Policy iteration can be summarised easily as an iteration between evaluation and updates, see below.

```
pi = init
while not converged:
  value = evaluate(pi)
  pi = greedy_update(value)
```

- **Observation** Two policies can be within ϵ yet requires more iterations of GPI. **Question** Why are some initial points far harder to solve than others, despite being approximately the same?

- **Observation** With only 2 states and 2 actions, it is possible for 3 partitions to exist. (2,3,4 steps), (2,3,2 steps). **Questions** ???

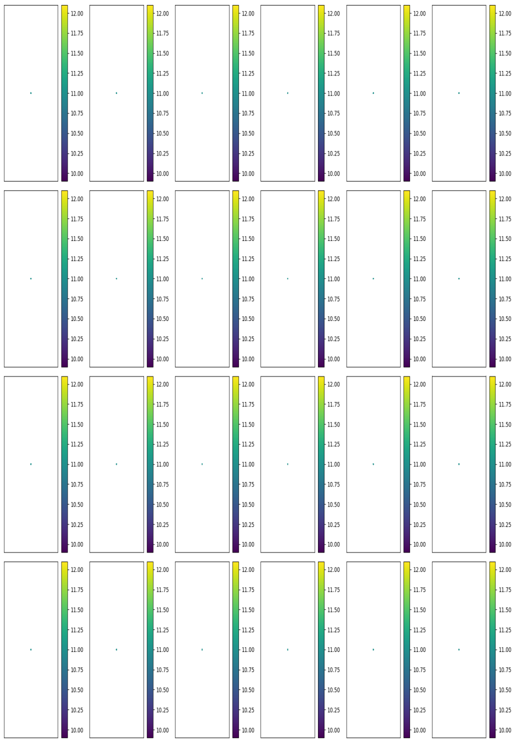


Figure 2.3: '2-state 2-action MDPs. We have visualised the number of steps required for convergence to the optimal policy. The number of steps are show by color.'

- **Observation** Sometimes the iterations don't converge. (a bug in the code?)

NOTES:

- What are the best ways to travel through policy space? (lines of shortest distance?!)
- How does this scale with `n_actions` or `n_states`??
- Is there a way to use an interior search to give info about the exterior? (dual methods?!)
- What if your evaluations are only ϵ-accurate? How does that effect things?!?

reater pleasures, or else he endures pains to avoid worse pains."

2.2.2 Search spaces and gradient descent

We want to find the optimal policy given some MDP. But how should we search for this policy? We could search within set of potentially optimal policies, the $|A|^{|S|}$ discrete policies, or we could search within the set of possible value functions, $\mathbb{R}^{|S|}$, or maybe some other space. Which space allows us to find the optimal policy in the cheapest manner?

Naively, we know that smaller search spaces are better. We would rather search for our keys in a single room, rather than many. But added structure (for example, continuity) can be exploited to yield faster search, even when there are infinitely more states to search.

In RL we know that; - the values must satisfy the bellman optimality criteria. This structure can be exploted. - the policies ...?

Value iteration In RL it is possible to transform the hard combinatorial problem of searching through the $|A|^{|S|}$ possible discrete policies, into an easier (how do we know it is easier?!? PROOF) problem, a search through all possible policies ?!?.

Policy iteration When transforming between two spaces, how does the optimisation space change? Does my abstraction make optimisation easier?

Model iteration Search through possible models, τ, r, calculate the optimal policy $\pi^*_{\tau,r}$ and then update τ, r based on $\parallel V_{\tau,r}(\pi^*) - V(\pi^*) \parallel$.

Search through models while trying to find one that yields similar returns to the oracle when playing the same policy. A supervised problem...

$$\theta^* = \underset{\theta}{\text{argmin}} \int_{\Pi} \parallel V^{\pi}_{P,r} - V^{\pi}_{\theta} \parallel_2 \tag{2.19}$$

Note this solver is different to the others. In the previous optimisations we assumed that we knew the model.

Once we have the model, we can solve for the optimal policy.

Relation to model based RL. This algol only focuses on relevant features of the state space. Where model base learners that attempt to learn by predicting transitions can be made to scale arbitrarily worse. Consider a problem where the reward is only determined by the first feature of the state. We can add n extra, useless, features. The model based learner will spend resources on attempting to build a good predictor of those n features.

Sample efficient. You only need to collect data for the m policies we are matching under. Once that has been done, the optimisation problem is easily solved!?

Model iteration. Model invariant transforms. Pick a policy. Falsify it, and this falsify all models that yield the same optimal policy.

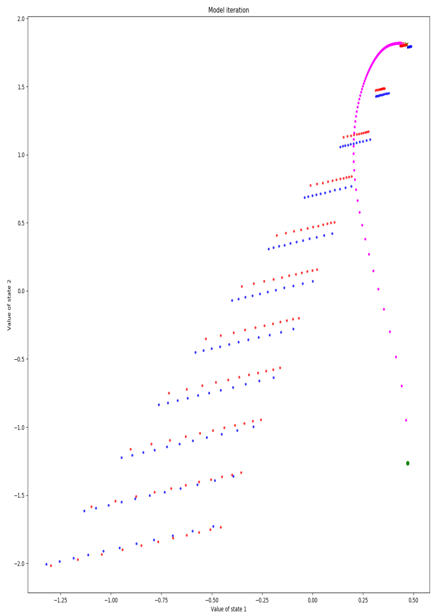


Figure 2.4: Blue shows the value of policies when evaluated under the true model, P, r, and Red shows the value of policies when evaluated under the learned model at convergence.

More generally, I am interested in how searches in different spaces, whether the value, the policy, or some parameters, ...

Let's focus on gradient descent.

$$w_{t+1} = w_t - \eta \nabla f(w_t) \tag{2.20}$$
$$\tag{2.21}$$

It's dynamics are depentdent on the topology of its loss landscpace, which is determined by the search space and .

Thus phrased differently, the original becomes: how does the space we are searching within effect the search dynamics: the rate of convergence and the possible trajectories.

$$\max_V \mathop{\mathbb{E}}_{s\sim D} V(s) \tag{2.22}$$
$$\max_\pi \mathop{\mathbb{E}}_{s\sim D} V^\pi(s) \tag{2.23}$$
$$\max_\theta \mathop{\mathbb{E}}_{s\sim D} V_\theta(s) \tag{2.24}$$
$$\max_\theta \mathop{\mathbb{E}}_{s\sim D} V^{\pi_\theta}(s) \tag{2.25}$$
$$\max_\phi \mathop{\mathbb{E}}_{s\sim D} V^{\pi_{\theta_\phi}}(s) \tag{2.26}$$
$$\max_\varphi \mathop{\mathbb{E}}_{s\sim D} V^{\pi_{\theta_{\phi_\varphi}}}(s) \tag{2.27}$$
$$\tag{2.28}$$

We could pick any space we we like to search with in. But, why would we want to pick one space over another?

- In which spaces can we efficiently do gradient descent?
- In which spaces can we do convex optimisation?
- In which spaces does momentum work well?

Topology and dynamics

Ok, so if we parameterise our search space. We have now changed the topology of our search space.

Q: How can we rationally pick the topology of our search space to accelerate learning?

- A well connected space? For all possible policies, there exists θ_1, θ_2 s.t. $\parallel \theta_1 - \theta_2 \parallel_2$ is small. (but that doesnt necessarily help... depends on the landscapce imposed by $\nabla_\theta V$)
- ???

See these gradient flows for example;

Pics?!?

Here are some examples ...???

If we overparameterise the search space, then we can move between solutions in new ways. We can 'tunnel' from A to B, without crossing C.

Every point (in output space) is closer, when measured in the distance in parameter space needed to be traveled.

Accleration and parameterisation

I think something weird happens with momentum in overparameterised spaces. The intuition is

We have implicit momentum from the parameterisation, and explicit momentum in the accelerated descent.

It is necessary to consider the trajectory to study momentum. It depends on what has happened in the past. Can we construct a space of possible trajectories? What properties do trajectories have? They are connected by the update fn.

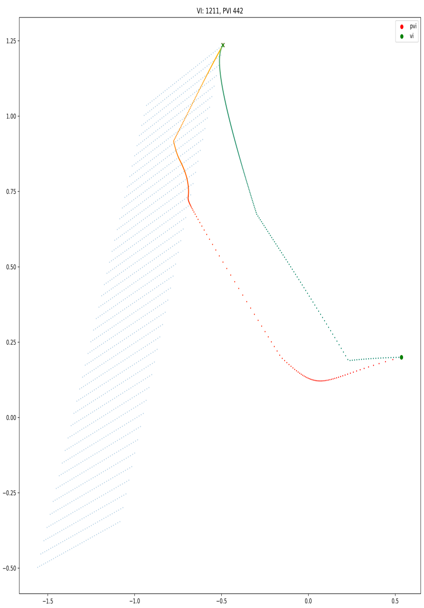


Figure 2.5: The optimisation dynamics of value iteration versus parameterised value iteration.

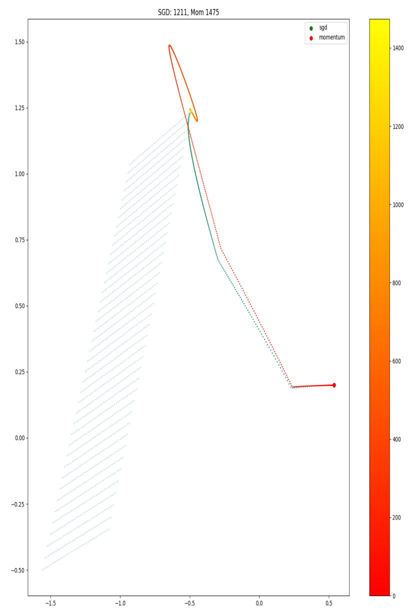
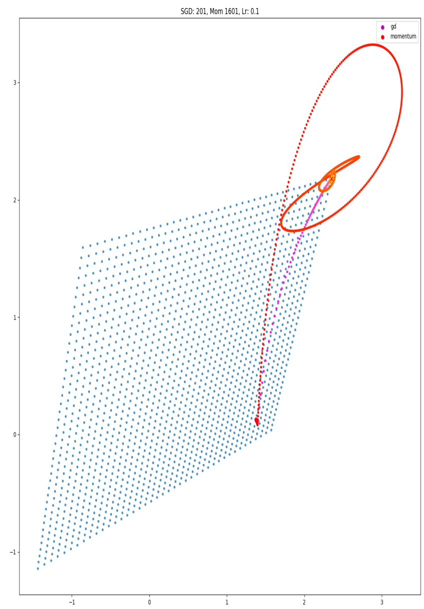
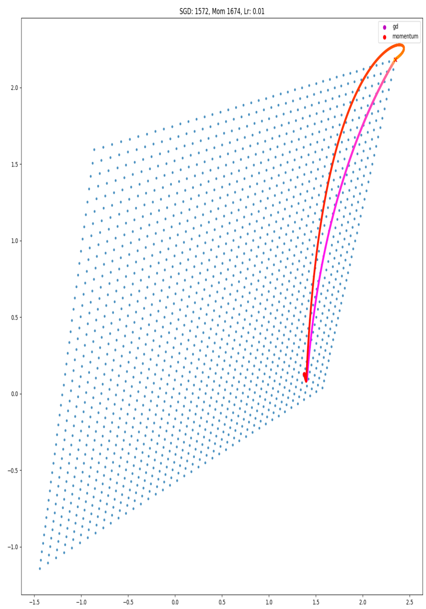


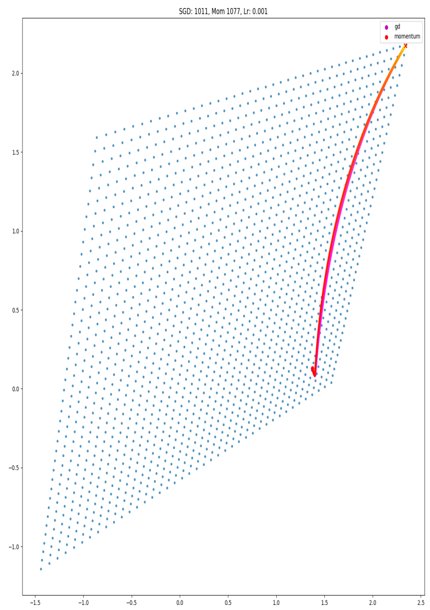
Figure 2.6: The optimisation dynamics of value iteration versus value iteration with momentum.

Continuous flow and its discretisation

A linear step of size, α, in parameter space, ie by gradient descent, is not necessrily a linear step in parameter space.







This is consistent with acceleration of gradient descent being a phenomena only possible in the discrete time setting. (see **[?]** for a recent exploration)

This phenomena can be explained by the exponential decay of the momentum terms.

$$m_{t+1} = m_t + \gamma \nabla f(w_t) \tag{2.29}$$
$$w_{t+1} = w_t - \eta(1-\gamma) m_{t+1} \tag{2.30}$$
$$\tag{2.31}$$

As $\eta \to 0$, $(1-\gamma) \cdot m_{t+1} \to \nabla f(w_t)$.

TODO, prove it.

Chapter 3

Abstraction

3.1 Near optimal abstractions

We are working with MDPs (S, A, τ, r), therefore we have a state space, S, an action space, A, a transition function $P : S \times A \times S \to [0, 1]$ and a reward function $r : S \times A \to \mathbb{R}$.

Let's say we have an abstraction, (a road is a road, no real different between them), a natural thing 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 case, of driving 100kph on every road, because they are all prety much the same...)? Or, in other words, which policies are approximately representable within this abstracted MDP.

An abstract MDP is defined as;

???

The metric we are optimising is the representation error of the optimal policy. Given an abstraction, we want to know how well the abstraction can represent the optimal policy.

$$\forall_{s \in S_G, a \in A_G} \mid Q_G^{\pi^*}(s, a) - Q_G^{\pi^*_{GA}}(s, a) \mid \leq 2\epsilon\eta_f$$

We could impose properties on a state abstraction using something like the following;

$$\forall_{s_1, s_2 \in S} \mid f(s_1) - f(s_2) \mid \leq \epsilon \implies \phi(s_1) = \phi(s_2) \tag{3.1}$$

$$\forall. \mid f(\cdot) - f(\cdot) \mid \leq \epsilon \implies g_1(\cdot) = g_2(\cdot) \tag{3.2}$$

$$\tag{3.3}$$

In other words, if there exists an approximate similarity, according to f, then build it into our abstraction.

- **Q:** How should we construct our abstraction?
- **Q:** What properties should it have to achieve 'good' performance?

Using the above method of imposing properties on an abstraction, what should we pick as f?

1. The policy function: $\forall_{\cdot_a,\cdot_b \in D} \mid \pi(\cdot_a) - \pi(\cdot_b) \mid \leq \epsilon$ is approximately the same.
2. The transition function: $\forall_{\cdot_a,\cdot_b \in D} \mid \tau(\cdot_a) - \tau(\cdot_b) \mid \leq \epsilon$ is approximately the same.
3. The reward function: $\forall_{\cdot_a,\cdot_b \in D} \mid r(\cdot_a) - r(\cdot_b) \mid \leq \epsilon$ is approximately the same.

Also,

4. The policy trajectory: $\forall_{\cdot_a,\cdot_b \in D} \mid \sum_{t=0}^{T} \parallel \pi(\cdot_a) - \pi(\cdot_b) \parallel_1 \mid \leq \epsilon$ is approximately the same.
5. The transition trajectory: $\forall_{\cdot_a,\cdot_b \in D} \mid \sum_{t=0}^{T} \parallel \tau(\cdot_{a_t}) - \tau(\cdot_{b_t}) \parallel_1 \mid \leq \epsilon$ is approximately the same.
6. The reward trajectory: $\forall_{\cdot_a,\cdot_b \in D} \mid \sum_{t=0}^{T} \parallel r(\cdot_{a_t}) - r(\cdot_{b_t}) \parallel_1 \mid \leq \epsilon$ is approximately the same.

GVFs

7. The discounted future policy: $\forall_{\cdot_a,\cdot_b \in D} \mid \Pi(\cdot_a) - \Pi(\cdot_b) \mid \leq \epsilon$ is approximately the same.
8. The discounted future transition: $\forall_{\cdot_a,\cdot_b \in D} \mid \Upsilon(\cdot_a) - \Upsilon(\cdot_b) \mid \leq \epsilon$ is approximately the same.
9. The discounted future reward: $\forall_{\cdot_a,\cdot_b \in D} \mid Q(\cdot_a) - Q(\cdot_b) \mid \leq \epsilon$ is approximately the same.

Q: Which is best?

> **Claim 1:** 9.(the value fn) will yield the most compression, while performing well. But, it is a task specific representation, thus it will not transfer / generalise well.

Other types of abstraction We constructed the state abstraction by altering what the policy and value function were allowed to see. Rather than observing the original state space, we gave them access to an abstracted state space.

There are other ways to alter what the policy and value function sees.

$$\begin{aligned}
\phi : S \to X : &\quad \pi(s) \to \pi(\phi(s)) \quad Q(s,a) \to Q(\phi(s),a) && \text{(State abstraction)} \\
\psi : A \to Y : &\quad \pi(s) \to \psi^{-1}(\pi(s)) \quad Q(s,a) \to Q(s,\psi(a)) && \text{(Action abstraction)} \\
\phi, \psi : &\quad \pi(s) \to \psi^{-1}(\pi(\phi(s))) \quad Q(s,a) \to Q(\phi(s),\psi(a)) && \text{(State and action abstraction)} \\
\varphi : S \times A \to Z : &\quad \pi(s) \to \underset{a}{\text{argmax}}\, V(\varphi(s,a)) \qquad Q(s,a) \to V(\varphi(s,a)) && \text{(State-action abstraction)}
\end{aligned} \tag{3.4}$$

Claim 2: The state-action abstraction is the most powerful because it allows the compression of the most symmetries. (want to prove!)

(relationship to Successor features!?)

State abstraction groups together states that are similar. For example, sprinting 100m is equivalent regardless of which track lane you are in.

Action abstraction groups together actions that are similar. For example, X and Y both yeild the state change in state, > Approximation perspective: we have a set of options and we want to use them to approximate the optimal policy. A good set of options can efficiently achieve an accurate approximation.

Motivating example for state and action abstraction: ???

Might want to transfer. But some envs share state space, some share action space. Want to

- Might be teleported to a new environment? (new state space, same action space)
- Might have to drive a new vehicle (same state space, new action space)

Motivating example for state-action abstraction: Symmetric maze

(Some intuition behind claim 2.)

Imagine you are in a mirror symmetric maze. It should not matter to you which side of mirror you are on.

This reduces the state-action space by half! $\frac{1}{2} \mid S \mid \times \mid A \mid$. Note: just using state abstraction it is not possible to achieve this reduction. Mirrored states are not equivalent as the actions are inverted.

While other learners can still solve this problem. They miss out on efficiency gains by abstracting first.

Related work

Other approaches to abstraction for RL focus on . . . ?

Near Optimal Behavior via Approximate State Abstraction [**?**] A Geometric Perspective on Optimal Representations for Reinforcement Learning [**?**] successor representation

3.1.1 Discussion

But can we guarantee that these abstractions do not make it harder to find the optimal policy? Is that even possible?

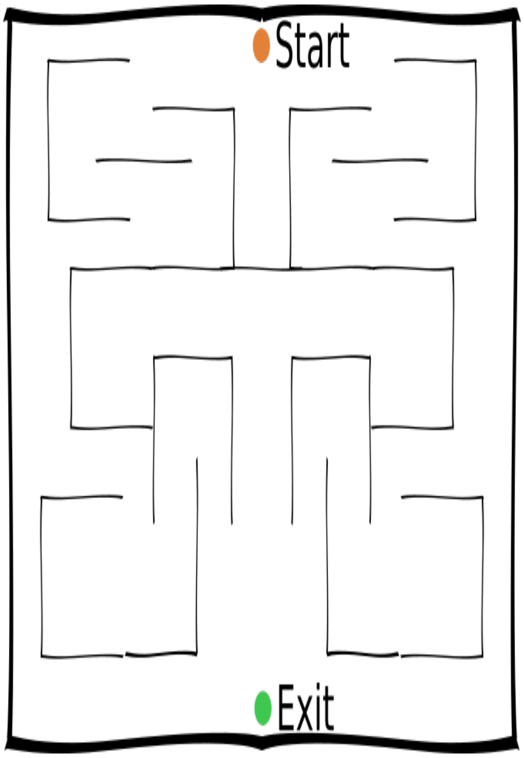


Figure 3.1: maze.png

Want a general way (a function) to take an abstraction of an MDP (defined by certain propreties) and return the difference between its optimal policy and the true optimal policy. Want automated computational complexity to solve this! Actually, we are not considering computational complexity here only approximation error. For that can we just use automatic differentiation!? Want a way to get bounds for all of these combinations!

How do we know one policy is better than another? How do we know a policy is optimal?

$$\forall \pi \;\; V^{\pi^*} \geq V^{\pi}$$

But, this definition of optimality implicitly assumes a uniform distribution over states. This is unlikely. Rather, the distribution is determined by the policy.

$$\mathop{\mathbb{E}}_{s \sim D_\pi} \left[V^{\pi^*}\right] \geq \mathop{\mathbb{E}}_{s \sim D_\pi} \left[V^{\pi}\right] D_\pi(s) = P(s|\pi) = \sum_{\text{all } \tau \text{ with } s_t = s} P(\tau|\pi)$$

Now. How different is this?

I can imagine some degenerate solutions now being possible? Because we can control the distribution we are being evaluated on. We could pick a policy that oscillates between two states, never leaving the cycle. Therefore it would have $p(s_1) = p(s_2) = 0.5$ and $p(s_{i \neq 1,2}) = 0$.

That doesn't seem so bad?

3.2 Symmetry

As recently noted by [?], ...

3.2.1 n-dimensional Cart pole

So, how can we test a learners ability to detect symmetries and exploit them? We propose a simple test, the n-dimensional cart pole.

Many people realise that this problem can be reduce to n, one dimensional cart pole problems. But the learner needs to infer that.

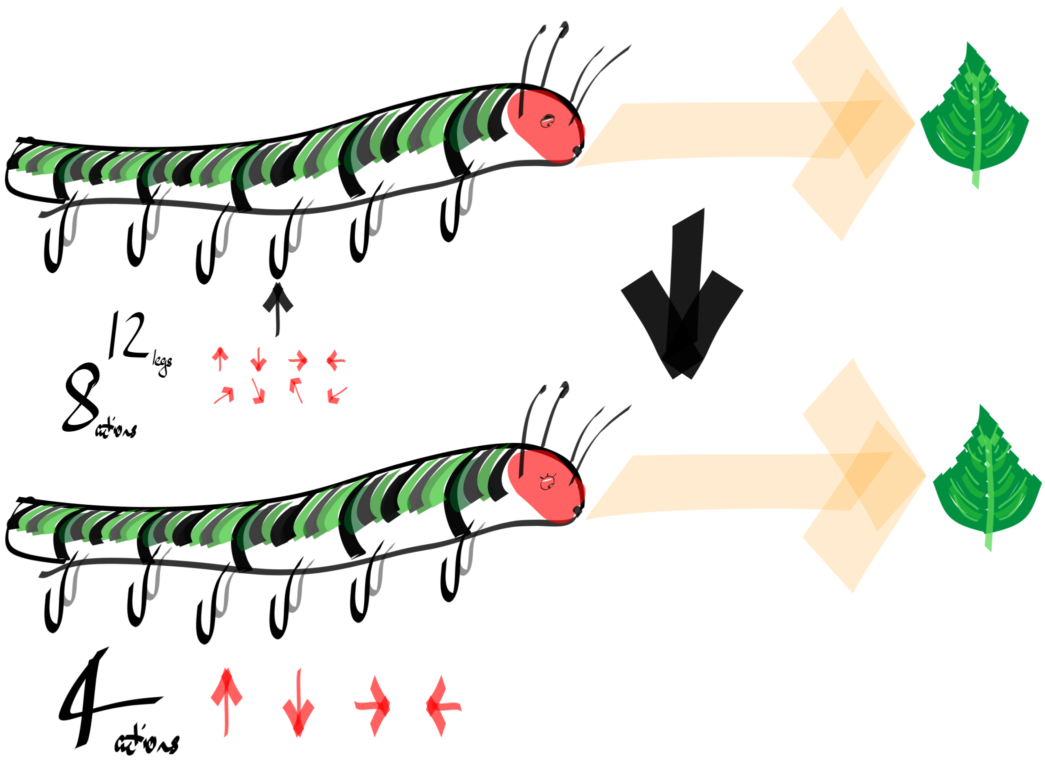


Figure 3.2:

3.3 Action abstractions

A green leaf, is too far, out of reach, What you want, is in front, take the steps.
You move your first leg up You move your second leg left ...
You move your eleventh leg up
You move your twelfth leg right
Many legs burden the act,
Unless coordinated in abstract.
The abstract words in this new language, Must be as few as we can manage.
"left", "right", "forward", "backward"
This time, the act is rather brief,
"forward", to reach the tasty leaf.

Mac Flecknoe
John Dryden

Chapter 4

Conclusions

If all the economists in the world were laid end-to-end they wouldn't reach a conclusion, and neither shall I.

Appendix A

Related work

How do the topics considered in this thesis relate to the work done in the wider scientific community and to society? Which work uses similar tools, which works build on the same foundations, which works have the same goals? How will this help? What can it be used for?

A.1 Academic literature

Decision theory (economics and psychology), ...? Control theory, ..

A.1.1 HRL

Temoral abstractions of actions.(how does this related to a decomposition of rewards) Ok, so we wany a multiscale representation? Understanding how actions combine (this is necessary knowledge for HRL?)

Reasons to do HRL??? (want to verify these claims - and have refs for them)

- credit assignment over long time periods (learning faster in one env)
- exploration

- transfer
- To learn action abstractions they must capture info about the model. How much harder is it to learn action abstractions in model-free vs model-based settings?
- Reward as a function of a subspace of the state space. (this is important for learning abstract representations and actions!?)
- What do cts linear heirarchical actions look like!? and their loss surface!?
- HLMDPs [**?**]
- Modulated policy heirarchies [**?**]
- Model free representations for HRL [**?**]
- Prierarchy: Implicit Hierarchies
- Options
- Near optimal representation learning for heirarchical RL [**?**]

Relation to pretraining / conditioning?

Why does Heirarchy (sometimes) work so well in reinforcement learning?

The authors claim that the benefits of HRL can be explained by better exploration. However, I would interpret their results as saying; "for 2D environments with walls, larger steps / actions result in greater explration". But what if the walls were replaced by cliffs? I imagine this algorithm would do a lot worse!?

They also seem to misunderstand the main problem with HRL, discovery. Once you have discovered a nice set of abstracted actions / a representation, then yeah, you get faster reward propagation, better exploration, ... etc.

A.1.2 Dynamic programming

What is it? Memoized search. Why should we care?

A.1.3 Model-based RL

Pros and cons.

Model-based learning can be bad... There may be many irrelevant details in the environment that do not need to be modelled. A model-free learning naturally ignores these things.

The importance of having an accurate model!

For example, let $S \in R^n$ and $A \in [0,1]^n$. Take a transition function that describes how a state-action pair generates a distribution over next states $\tau : S \times A \to \mathcal{D}(S)$. The reward might be invariant to many of the dimensions. $r : X \times A- > \mathbb{R}$, where $X \subset S$.

Thus, a model mased learner can have arbitrarily more to learn, by attempting to learn the transition function. But a model-free learner only focuses on ...

This leads us to ask, how can we build a representation for model-based learning that matches the invariances in the reward function. (does it follow that the invariances in reward fn are the invariances in the value fn. i dont think so!?)

Take $S \in R^d$ and let $\hat{S} = S \times N, N \in R^k$. Where N the is sampled noise. How much harder is it to learn $f : S \to S$ versus $\hat{f} : \hat{S} \to \hat{S}$?

[?, ?]

A.1.4 Representation learning and abstraction

The goal is to find a representation that decomposes knowledge into its parts.

Another way to frame this is: trying to find the basis with the right properties.

- sparsity,
- independence,
- multi scale,
- locality/connectedness
- ???

Types of abstraction for RL. Abstraction for efficient;

- exploration, [Learning latent state representation for speeding up exploration](https://arxiv.org/abs/1905.12621)
- optimal control,
- ???,

A.1.5 Heirarchical reinforcement learning