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

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

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? Why do we care?

2.1 Sequential decision problems

A MDP is defined as a tuple, $\{\mathcal{S}, \mathcal{A}, P(s_{t+1} \mid s_t, a_t), R(s_t, a_t, s_{t+1}), \gamma\}$[1]. 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, ...*), $P(s_{t+1} \mid s_t, a_t)$ 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, $r(s_t, a_t, s_{t+1})$ is the reward function, *whether you won (+1) or lost (-1) the game* . The objective when solving a MDP is to find a policy, $\pi(a_t|s_t)$, that maximises the discounted cumulative reward, $R = \sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})$.

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.

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

[1] Why is the discount factor a part of the definition of the MDP? By defining the discount it ensures the MDP has a unique solution.

about the longer term aggregates.

2.1.1 The Markov property

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. The next transition step depends only on the current state and action. It is invariant to any and 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.

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?

2.1.2 Optimality

What does it mean to solve an MDP?

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 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?

How hard is it to solve an MDP?

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

2.2 How do MDPs relate to RL?

Reinforcement learning refers to the set of solutions to a type of problem. This general, reinforcement learning, problem, has two main properties; *"trial-and-error search and delayed rewards"* [1]. 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 student 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."*)

Ontop of terse teachers, many actions may be taken before any evaluation is received, thus requiring credit to be assigned to actions, often leaving the learner wondering: "what did I do to deserve this?" (see pigeon superstition for an amusing example of credit assignment gone wrong [2]).

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

- Observable or un-observable
- Deterministic or stochastic
- Synchronous or asynchronous
- Terminating or infinite
- Discrete versus continuous
- Given knowledge of the underlying model

2.3 The simplest RL setting

What is the minimally complex setting we can consider that still poses an interesting challenge to the ML / RL communities?

2.3.1 A tabular representation of MDPs

Imagine a tabular MDP, where we can describe the MDP with tables. 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.

Consider a tabular MDP with deterministic actions, where $P(s_{t+1}|s_t, a_t) \in \{0, 1\}$. This RL problem can be efficiently solved by non-statistical methods: dynamic programming and related planning techniques [3].

But, a tabular MDP with stochastic actions, $P(s_{t+1}|s_t, a_t) \in [0, 1]$, seems to retain much of the complexity we care about. This setting does not al-

low efficient solutions via dynamic programming. And can be approached with algorithms that are used for state-of-the-art DRL such as policy gradients,

Let's more formally define our tabular MDP.

$$\mathcal{M} = \{S, A, P, r, \gamma\} \tag{2.1}$$
$$S = [0 : n-1] \tag{2.2}$$
$$A = [0 : m-1] \tag{2.3}$$
$$P \in [0,1]^{n \times n \times m}, \quad \forall j,k : \sum_i P[i,j,k] = 1 \tag{2.4}$$
$$r \in \mathbb{R}^{n \times m} \tag{2.5}$$

A result of this formulation is that we consisely write and solve the Bellman equation analytically.

$$V = r_\pi + \gamma P_\pi V \qquad \text{(the bellman eqn)}$$
$$V - \gamma P_\pi V = r_\pi \tag{2.6}$$
$$(I - \gamma P_\pi) V = r_\pi \tag{2.7}$$
$$V = (I - \gamma P_\pi)^{-1} r_\pi \tag{2.8}$$

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 $P_\pi[s',s] = \sum_a P[s',s,a]\pi[s,a]$.

2.3.2 The value function polytope

The value function polytope [4] is the

Imagine a two state MDP. Following some initial, ill-informed policy, the value that you might get starting from either state state is v_1, v_2.

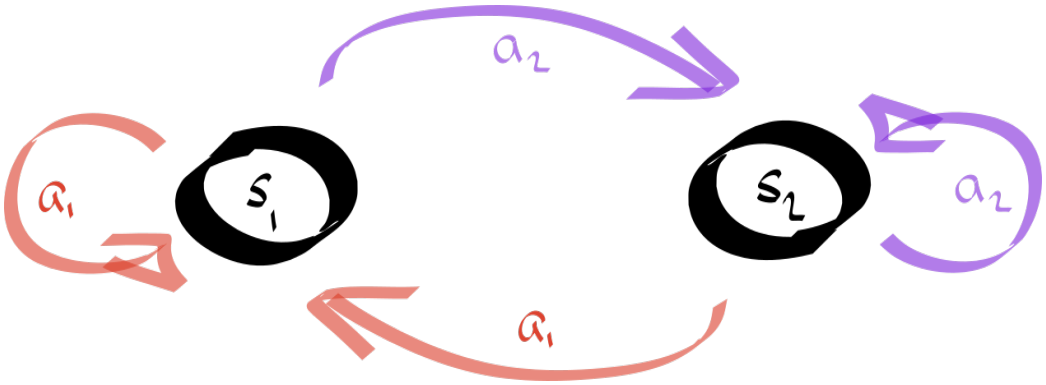


Figure 2.1: Consider the simplest possible MDP, with two states and two actions. (Any simpler setting is entierly uninteresting. A single state means actions do nothing. And a single action means all policies are the same...).

2.3.3 What are the properties of the polytope?

How is the structure of the RL problem reflected in geometry?

How is the shape of the polytope determined by the Bellman equation? If we look at the polytope, we see that;

- All the edges have positive gradients
- All the edges are linear
- Non-convex

$$\hat{V}(s_1) = V(s_1)P(s_1|s_1, a)\pi(a|s_1) + V(s_2)P(s_2|s_1, a)\pi(a|s_1) \tag{2.9}$$

- If $V(s_1)$ increases then $\hat{V}(s_1)$ must increase...
- If $V(s_2)$ increases then $\hat{V}(s_1)$ either increases or stays the same. As $P(s_2|s_1, a)\pi(a|s_1) \geq 0$

2.3.4 Policies in high dimensions

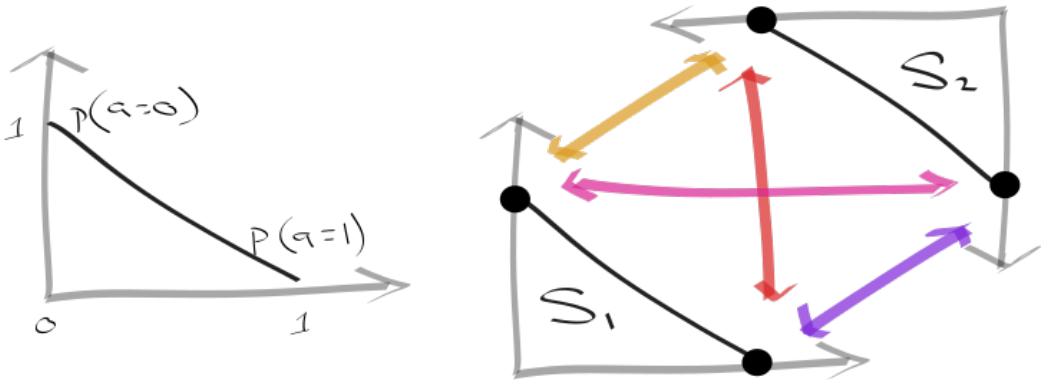


Figure 2.2: Imagine what the geometry of the space of policies in the two state, two action MDP. A policy tells us what actions should be taken when in a given state. Therefore, there will be $|A| \times |S|$ entries in the policy. However, because the policy returns a distribution over actions, the true dimensionality of the policy is $(|A| - 1) \times |S|$. Which in the two state, two action case equals 2D.

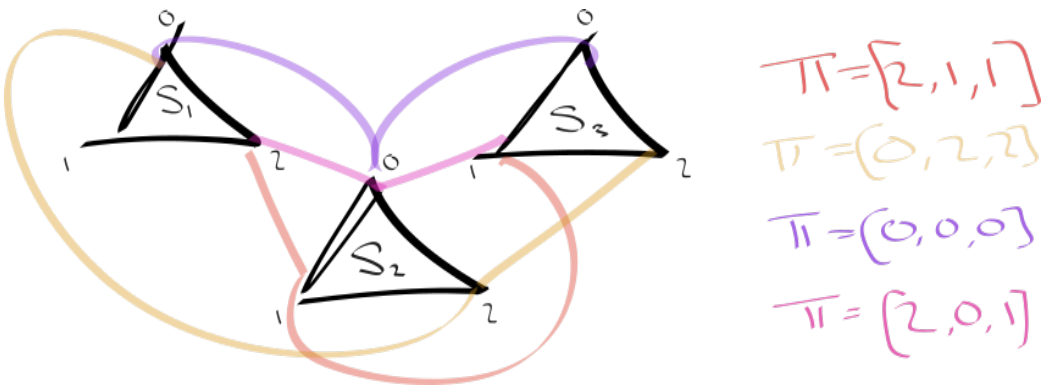


Figure 2.3: Let's *try* to gain some intuition about the space of policies in higher dimensions. For each state, we have a distribution (on a simplex), over the possible actions.

Chapter 3

Search spaces and their complexity

MDPs are not very cheap to solve. We are interested in exploring the complexity of some different approaches to solving MDPs.

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

In any optimisation problem, we have a set of potential solutions, also known as the search space, and a way to evaluate these potential solutions, a cost or loss function. For example, we

What do I mean by a search space? By optimisation dynamics? Why should anyone care?

3.1 Efficient search for RL

Using tabular MDPs and the value function polytope as our microscope, let explore how different search spaces can er.

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 possible value functions, $\mathbb{R}^{|S|}$,

or maybe some other space. But which space is best?

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.

3.1.1 Value search

What is going on here? We are searching through values of policies, for a value that is stationary under the Bellman optimality operator. Once we have found this value, we know how to easily construct the optimal policy.

$$T(V) = \max_a \left[r + \gamma PV \right] \tag{3.1}$$

$$V_{t+1} = V_t + \alpha(T(V_t) - V_t) \tag{3.2}$$

Intution about why it converges!?

Observe that the value iterations are not constrained to refer to any policy, and thus can go outside of the polytope. [4]

We are constrained to move around by only using the Bellman optimality operator. To move proportionally to value improvement of the best actions ... Why is this good / bad. What dynamics can / cannot be achieved?

3.1.2 Policy search

Searching through the space of potential policies supports a couple of modes of travel. Policy iteration jumps between deterministic policies, guided by ...? Policy gradients ...?

$$\pi^* = \underset{\pi}{\text{argmax}}\, \mathbb{E}[V(\pi)] \tag{3.3}$$

$$\pi_{t+1} = \pi_t + \eta \nabla \mathbb{E}[V(\pi)] \tag{3.4}$$

add ref

3.1.3 Model search

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...

$$\theta^* = \underset{\theta}{\text{argmin}} \int_{\Pi} \parallel V^{\pi}_{P,r} - V^{\pi}_{\theta} \parallel_2 \tag{3.5}$$

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.

Related to AWR? https://arxiv.org/pdf/1910.00177.pdf

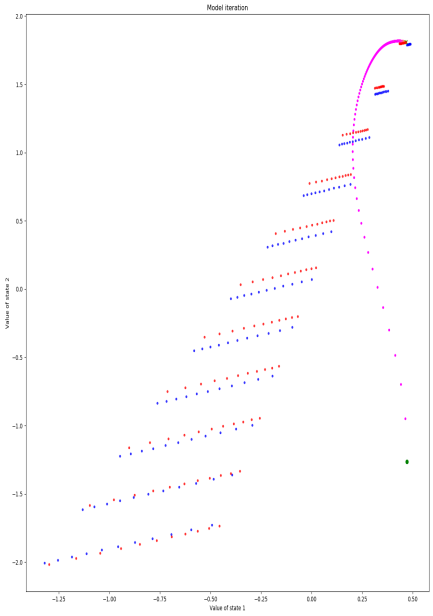


Figure 3.1: Blue shows the value of policies when evaluated under the true model, P, r, and Red shows the estimated value of policies when evaluated under the learned model at convergence, $\tilde{P}, \tilde{r}$.

So there seem to be three different classes of search space: each imbued with special structure from the Bellman equation. What are their properties? And which spaces support efficient search for the optimal policy?

- In the value space we can use the temporal difference operator,
- in the policy space we can estimate policy gradients,
- in the model space we can solve directly for the optimal policy.

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).

For each search space, we need to constrain the solutions to be possible within the original MDP.

3.2 Gradient based search

Let's focus on searching with gradient descent.

$$w_{t+1} = w_t - \eta \nabla f(w_t) \tag{3.6}$$

The dynamics of the search are dependent on the topology of a problems loss surface. The loss surface is determined by the combination of the search space and the loss function.

$$\max_{V} \mathop{\mathbb{E}}_{s\sim D} V(s) \tag{3.7}$$

$$\max_{\pi} \mathop{\mathbb{E}}_{s\sim D} V^{\pi}(s) \tag{3.8}$$

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D} V_{\theta}(s) \tag{3.9}$$

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D} V^{\pi_\theta}(s) \tag{3.10}$$

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D} V_{\theta}^{\pi_\phi}(s) \tag{3.11}$$

$$\max_{\phi} \mathop{\mathbb{E}}_{s\sim D} V^{\pi_{\theta_\phi}}(s) \tag{3.12}$$

Value iteration and related methods like (deep) Q learning. (3.9) (3.11)

Ok. So when ϕ are the parameters of a deep neural network, ...? The search space has certain (which) properties. Euclidean geometry.

We can pick any space we we like to search with in. But, why would we want to pick one space over another? What properties are we looking for?

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

3.2.1 Dynamics and complexity

> So. We want to investigate the properties of gradient based search in different search spaces.

Let start with trying to understand the different dynamics and iteration complexities.

How can you quantify a trajectory?

What about the vector fields? Do some spaces have gradients that most globally point in approximately the right direction?

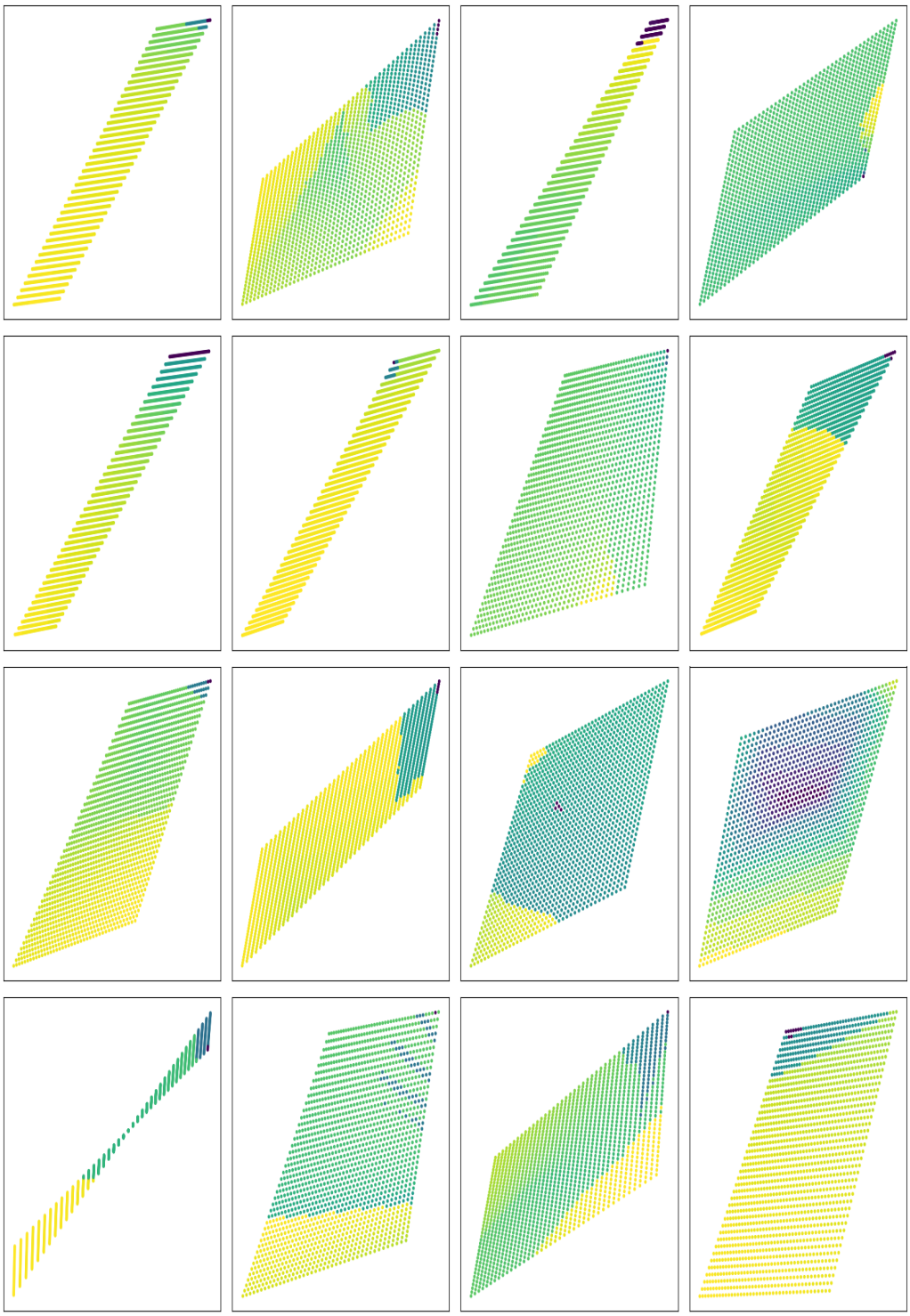


Figure 3.2: Above you see: various MDPs where the value of each policy is colored by the number of iterations it took to converge to the optimum policy. Yellow is many iterations, purple is few iterations.

- 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?!?
- how can we pick the topology for better dynamics!?

The momentum polts tell us that if we want to bound the iteration complexity of GD plus momentum then ...? We need a term that caputures the position / oscillation dynamics!?

3.2.2 Acceleration and parameterisation

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

[5] shows that overparameterisation yields acceleration. Consider the dynamics of the parameter w, which we have overparameterised as $w_1 \cdot w_2$.

$$
\begin{aligned}
w^{t+1} &= w_1^{t+1} \cdot w_2^{t+1} && (3.13)\\
&= (w_1^t - \eta \nabla_{w_1} L) \cdot (w_2^t - \eta \nabla_{w_2} L) && (3.14)\\
&= (w_1^t \cdot w_2^t - \eta w_2^t \nabla_{w_1} L - \eta w_1^t \nabla_{w_2} L - \mathcal{O}(\eta^2) && (3.15)\\
&= w^t - ... && (3.16)
\end{aligned}
$$

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

$$
\begin{aligned}
m_{t+1} &= \gamma m_t + g_t && (3.17)\\
w_{t+1} &= w_t - \eta \cdot (1 - \gamma) \cdot m_t && (3.18)
\end{aligned}
$$

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.

Set $g_t = \rho(t)\nabla_w + \sum_{\tau=0} \mu g_\tau$.

$$m_{t+1} = \gamma m_t + \rho(t) g_t + \sum_{\tau=0} \mu_{\tau,t} g_\tau \tag{3.19}$$

$$w_{t+1} = w_t - \eta \cdot (1-\gamma) \cdot m_t \tag{3.20}$$

$$= w_t - \eta \cdot (1-\gamma) \cdot \sum_{k=0}^{t} \gamma^{t-k} \big[\rho_t g_k + \sum_{\tau=0} \mu_{\tau,k} g_\tau\big] \tag{3.21}$$

3.3 Topology and dynamics

What kinds of query and movement does our search space support?

If we parameterise our search space. We my have changed the topology of our search space. Is it possible to arbitrarily change the topology? (probably?!?)

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. $\| \theta_1 - \theta_2 \|_2$ 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.

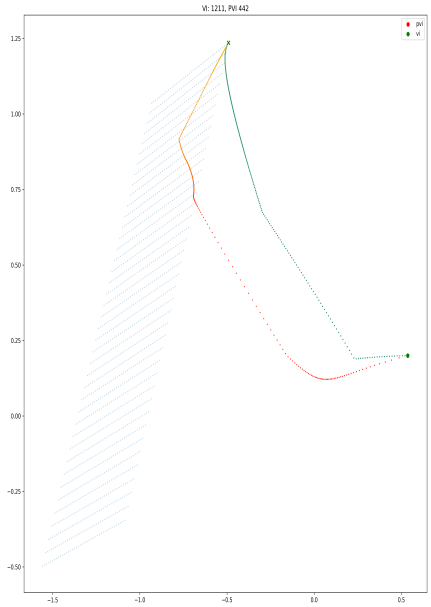


Figure 3.3: The optimisation dynamics of value iteration versus parameterised value iteration.

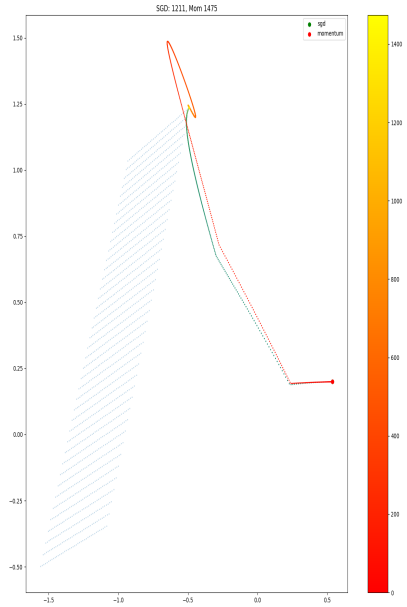
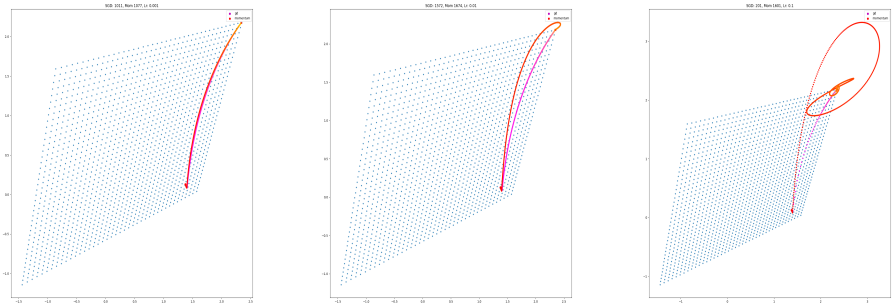


Figure 3.4: The optimisation dynamics of value iteration versus value iteration with momentum.

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

3.4 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 [6] for a recent exploration)

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

$$m_{t+1} = m_t + \gamma \nabla f(w_t) \tag{3.22}$$

$$w_{t+1} = w_t - \eta(1-\gamma) m_{t+1} \tag{3.23}$$

$$\tag{3.24}$$

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

TODO, prove it.

3.5 Notes

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

Chapter 4

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?

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

4.1 Search spaces

Recently there has been work investigating the properties of overparameterised search spaces. Many [5] (and others?!?!?) claim that overparameterisation yeilds acceleration, however, their explanation of the acceleration is not entirely convincing.

[5]'s proofs ...??

4.2 Abstraction

4.2.1 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 [7]
- optimal control,
- ???,

Unsupervised State Representation Learning in Atari [8] State Aggregation Learning from Markov Transition Data !!![9]

4.2.2 Linear RL

Recently there have been a few other attempts to exploit linearity for RL. [10] factored

[11] Reward and transitions are a linear function of features of a state-action pair.

Levine et al. [12] build a latent representation such that the transition fn (in the latent space) is approximately linear. This allows ...

These attempts all focus on linearity in the transition function...

4.2.3 Heirarchical reinforcement learning

What is HRL? Really it is just temporal abstraction, using a heirarchy.

As far as I know, there are three main apporaches to HRL. Equip an agent with options, learning to generate sub goals, pretrained low level policies, .

Between MDPs and Semi-MDPs [13] Reinforcement learning with structured hierarchical grammar representations of actions [14] !!! theory for this tho?! Latent Space Policies for Hierarchical Reinforcement Learning [15]

Two main approaches. Subgoals and options. To achieve temporal abstraction.

- Does it help? - Why does it help? - ?

We don't really have a principled approach to HRL?!

- How does the heirarchy help? - When can we expect transfer?

Discovery! And no free lunches!?

> The challenge in the single-task case is overcoming the additional cost of discovering the options; this results in a narrow opportunity for performance improvements, but a well-defined objective. In the skill transfer case, the key challenge is predicting the usefulness of a particular option to future tasks, given limited data. [16]

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 [17]
- Modulated policy heirarchies [18]
- Model free representations for HRL [19]
- Prierarchy: Implicit Hierarchies
- Options
- Near optimal representation learning for heirarchical RL [20]

In "Why does Heirarchy (sometimes) work so well in reinforcement learning?" [4] 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, better exploration (aka larger steps, aka actions that are more temporally abstract), result in greater explration". But what if the walls were replaced by cliffs? I imagine this algorithm might do a lot worse!? Also, they don't seem to consider the main problem with HRL: discovery. Once you have discovered a nice set of abstract actions or states, then yeah, you get faster reward propagation, better exploration, ... etc.

Bibliography

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

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

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

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

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

[6] M. Betancourt, M. I. Jordan, and A. C. Wilson, "On Symplectic Optimization," 2018.

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

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

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

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

[11] Z. Wang and M. I. Jordan, "Provably Efficient Reinforcement Learning with Linear Function Approximation," pp. 1–28.

[12] 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.

[13] S. S. a. Richard S. Sutton a,∗, Doina Precup b, "Between MDPs and semi-MDPs," vol. 1, no. 1999, pp. 1–30, 1998.

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

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

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

[17] A. M. Saxe, A. Earle, and B. Rosman, "Solvable Markov Decision Processes,"

[18] A. Pashevich, J. Davidson, R. Sukthankar, and C. Schmid, "Modulated Policy Hierarchies,"

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

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