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

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?!)

Understanding Markov decision problems

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

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. . . .

1.2 Related work

MDPs Dynamic programming, linear programming, . . . ?

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

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.

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

1.2.1 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 $\tau : S \times A \rightarrow \mathcal{D}(S)$. 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 $\hat{f} : \hat{S} \to \hat{S}$?

https://arxiv.org/pdf/1903.00374v3.pdf https://arxiv.org/abs/1907.02057

1.2.2 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
- ???

Chapter 2

MDPs

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?

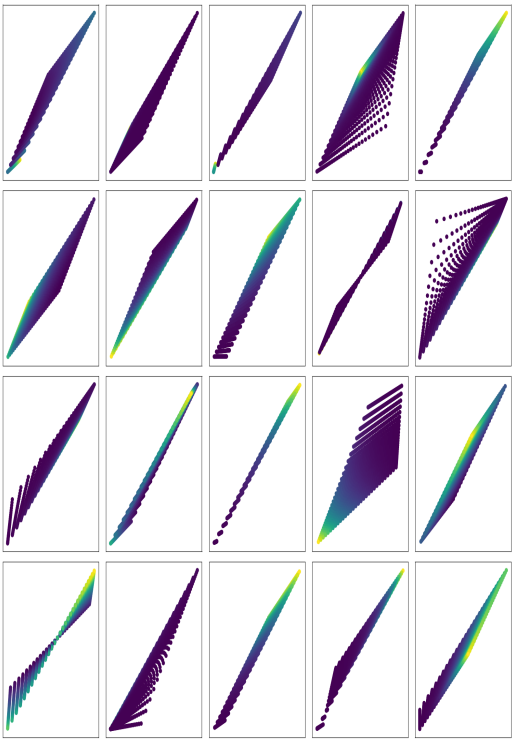


Figure 2.1: '2-state 2-action MDPs. We have visualised the likelihood of values under a uniform on policies. They are coloured by density. Lighter colour is higher probability'

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

Derivation of derivative

$$
\begin{aligned}
V(\pi) &= (I - \gamma P_\pi)^{-1} r_\pi && (2.1)\\
&= (I - \gamma P \cdot \pi)^{-1} r \cdot \pi && (2.2)\\
\frac{\partial V}{\partial \pi} &= \frac{\partial}{\partial \pi}((I - \gamma P_\pi)^{-1} r_\pi) && (2.3)\\
&= (I - \gamma \pi P)^{-1} \frac{\partial \pi r}{\partial \pi} + \frac{\partial (I - \gamma \pi P)^{-1}}{\partial \pi} \pi r && \text{(product rule)}\\
&= (I - \gamma \pi P)^{-1} r + -(I - \gamma \pi P)^{-2} \cdot -\gamma P \cdot \pi r && (2.4)\\
&= \frac{r}{I - \gamma \pi P} + \frac{\gamma P \cdot \pi r}{(I - \gamma \pi P)^2} && (2.5)\\
&= \frac{r(I - \gamma \pi P) + \gamma P \pi r}{(I - \gamma \pi P)^2} && (2.6)\\
&= \frac{r}{(I - \gamma P \pi)^2} && (2.7)\\
& && (2.8)
\end{aligned}
$$

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.

$$M \to \{P, r, \gamma\} \quad \text{(a MDP)}$$

$$H(M) := \mathop{\mathbb{E}}_{\pi \sim \Pi} \Big[-\log p(V(\pi)) \Big] \tag{2.9}$$

$$= \mathop{\mathbb{E}}_{\pi \sim \Pi} \Big[-\log(|\det \frac{\partial V(\pi)}{\partial \pi}|^{-1} p(\pi)) \Big] \tag{2.10}$$

$$= \mathop{\mathbb{E}}_{\pi \sim \Pi} \Big[-\log(|\det \frac{r}{(I - \gamma P \pi)^2}|^{-1} p(\pi)) \Big] \tag{2.11}$$

$$\tag{2.12}$$

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 $s \leq \frac{\sigma_{MDP}}{D}$ 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?
- 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).

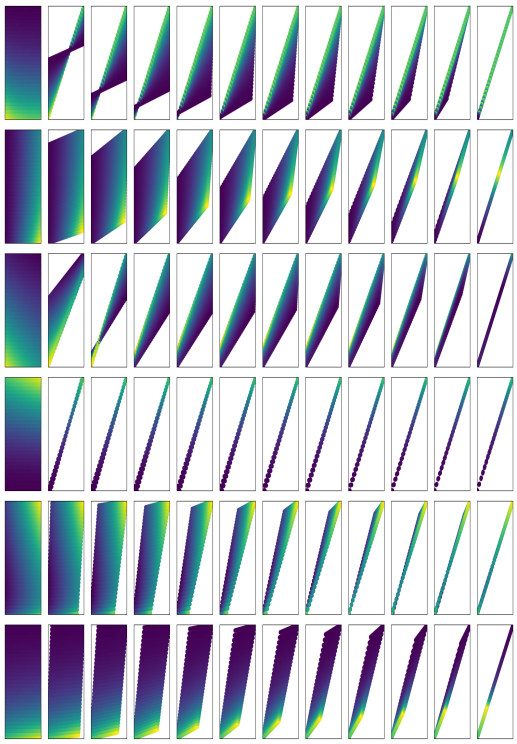


Figure 2.2: '2-state 2-action MDPs. Here we have shown a few different P/r MDPs and how their polytopes change with changes in discount rate.'

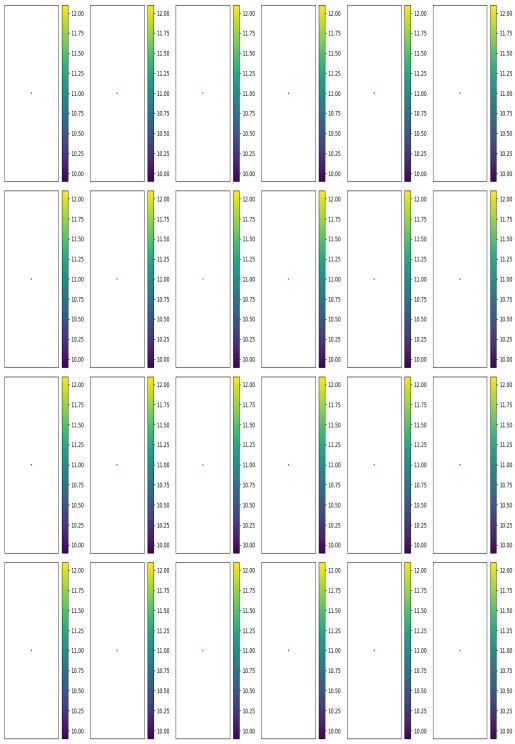


Figure 2.3: '2-state 2-action MDPs. We have visualised the number of steps required for convergence to the optimal policy. The number of steps are show by color.'

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** ???
- **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.

(note this one is different to the others. as we dont assume we know the model) Related to Thompson sampling?!?

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

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.

$$w_{t+1} = w_t - \eta \nabla f(w_t) \tag{2.13}$$

$$\tag{2.14}$$

It's dynamics are dependtent 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.

$$\max_{V} \mathop{\mathbb{E}}_{s\sim D} V(s) \tag{2.15}$$

$$\max_{\pi} \mathop{\mathbb{E}}_{s\sim D} V^{\pi}(s) \tag{2.16}$$

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D} V_{\theta}(s) \tag{2.17}$$

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D} V^{\pi_\theta}(s) \tag{2.18}$$

$$\max_{\phi} \mathop{\mathbb{E}}_{s\sim D} V^{\pi_{\theta_\phi}}(s) \tag{2.19}$$

$$\max_{\varphi} \mathop{\mathbb{E}}_{s\sim D} V^{\pi_{\theta_{\phi_\varphi}}}(s) \tag{2.20}$$

$$\tag{2.21}$$

We can pick the space we optimise in. Why would we want to pick one space over another?

- In which spaces can we 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. $\| \theta_1 - \theta_2 \|_2$ is small. (but that doesnt necessarily help... depends on the landscapce imposed by $\nabla_\theta V$)
- ???

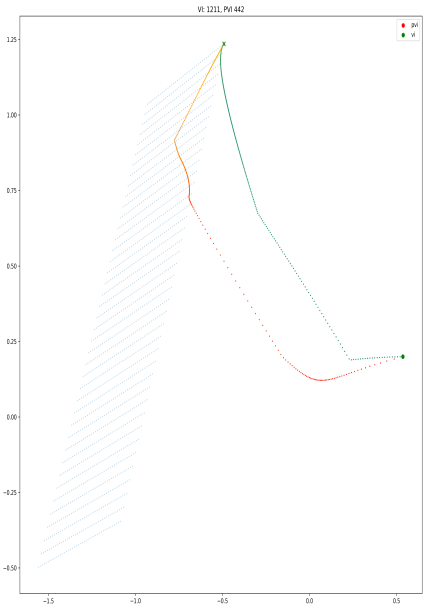


Figure 2.4: The optimisation dynamics of value iteration versus parameterised value iteration.

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.

Intuition: Every point is closer, under some measure of distance?!? But. Momentum seems like it might be a bad thing here?

Accleration and parameterisation

Intuition. Something weird happens with momentum in overparameterised spaces.

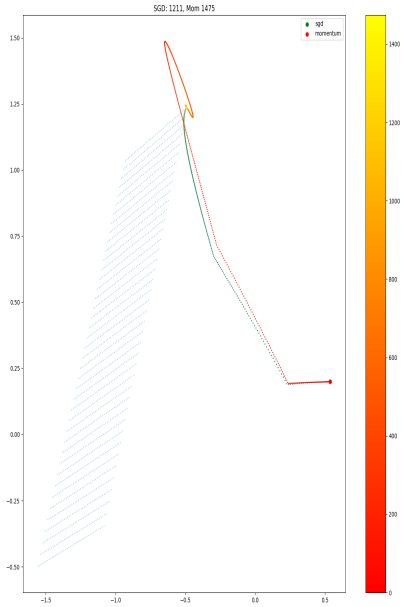
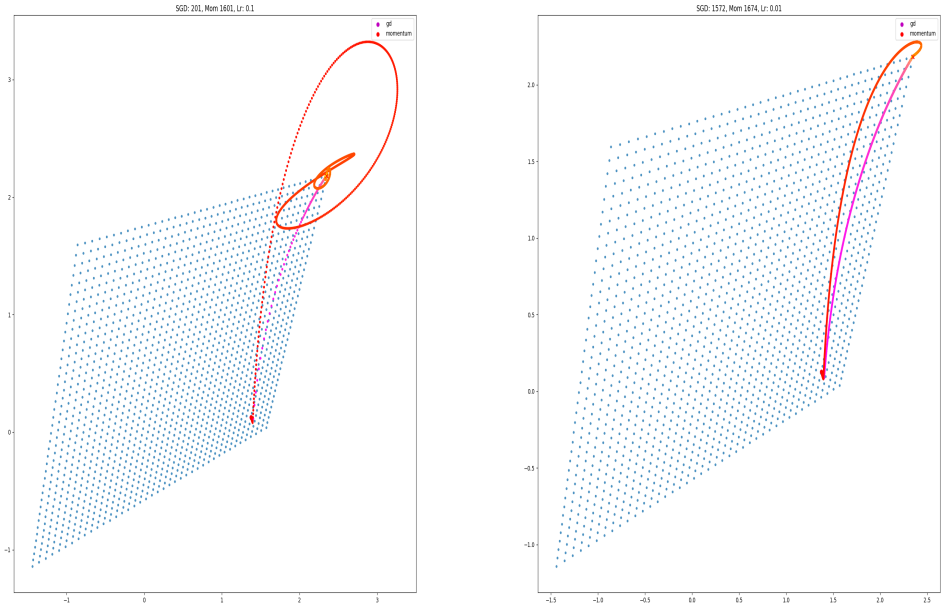


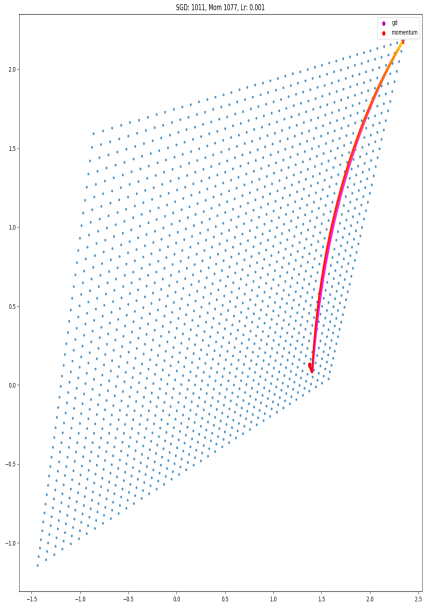
Figure 2.5: The optimisation dynamics of value iteration versus value iteration with momentum.

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.

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. **??**On symplectic optimization.

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

$$m_{t+1} = m_t + \gamma \nabla f(w_t) \tag{2.22}$$

$$w_{t+1} = w_t - \eta(1-\gamma) m_{t+1} \tag{2.23}$$

$$\tag{2.24}$$

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

TODO, prove it.

Chapter 3

Abstraction

3.0.1 Solveable representations

Representations with structure that is easily solvable.

While there are other ways to add exploitable structure, here we only consider linearity.

The bellman equation is a non-linear optimisation problem. It does have some nice properties, like having a unique optima under the bellman operator. But, in general, it isn't very friendly. Is there a way to turn this into a linear problem? What sacrifices need to be made to achieve this?

Why linearity?

- it has many mathematical tools for analysis.
- we know linear systems can be solved efficiently.
- ?

Linearity is a nice property that makes optimisation simpler and more efficient.

- Linear programming (see appendix: LP)
- Linear markov decision processes

Solving a system of linear relationships. Has a complexity of ???.

In fact. MDPs can actually be solved via LP. see [appendix].

A closer look at LMDPs

(the Todorov ones...)

The three steps of abstraction - relaxation (transform to a new domain) - linearisation (and solve) -

LMDPs; more formally Pick $a \in A$, versus, pick $\Delta(S)$. $f : S \to A$ vs $f : S \to \Delta(S)$.

In the original Todorov paper, they derive the LMDP equations for minimising a cost function. This maximisation derivation just changes a few negative signs around. Although there is also a change in the interpretation of what the unconstrained dynamics are doing. ...?

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

$$\tag{3.1}$$

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

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

$$\tag{3.2}$$

By definition, an LMDP is the optimisation problem in (1). (3) Define a new variable, $z(s) = e^{v(s)}$. (5) Define a new variable that will be used to normalise $p(s'|s)z(s')^\gamma$. (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. (11) Rewrite the equations for the tabular setting, giving a z vector, uncontrolled dynamics matrix.

(see appendix [] for a full derivation)

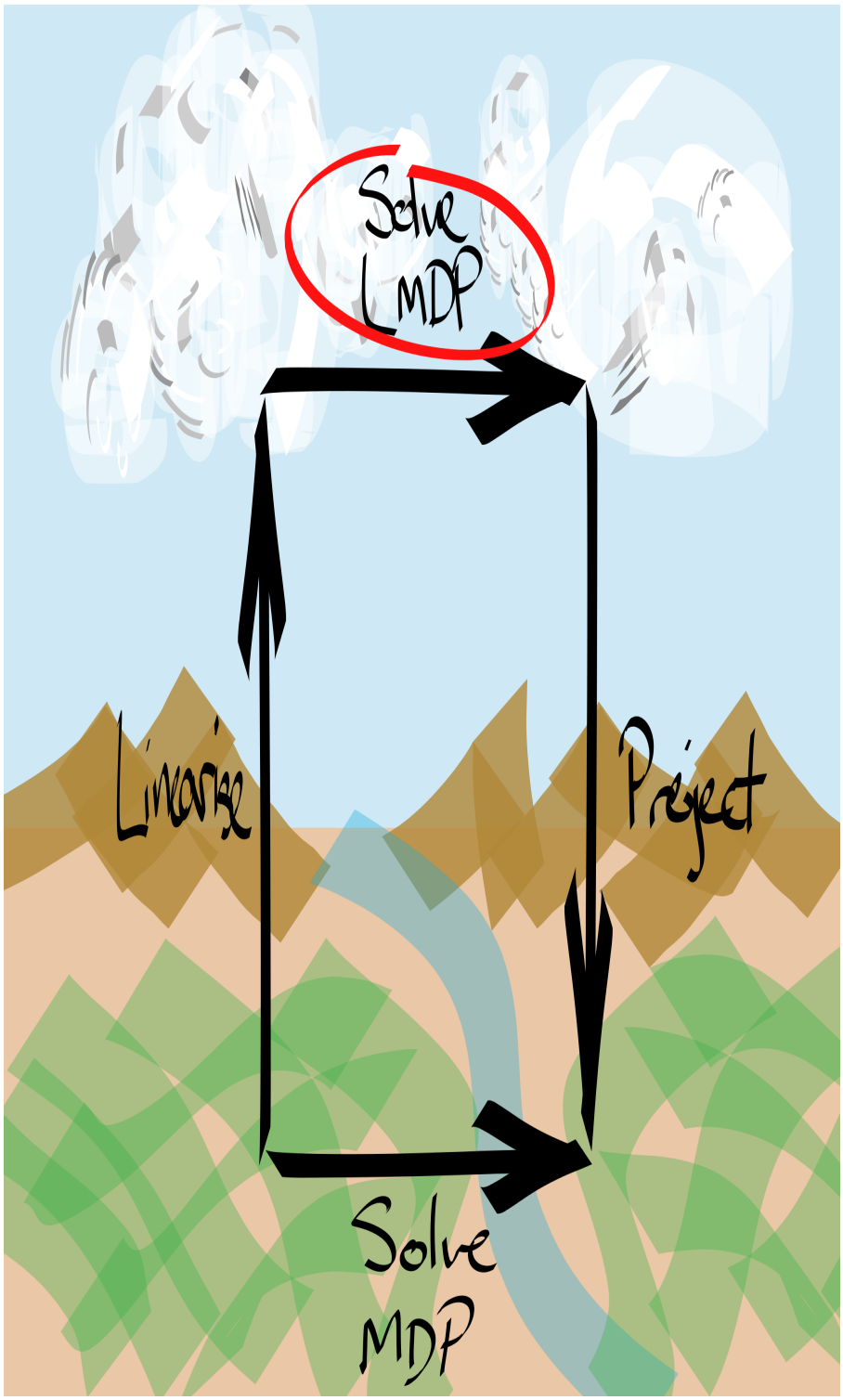


Figure 3.1: ″

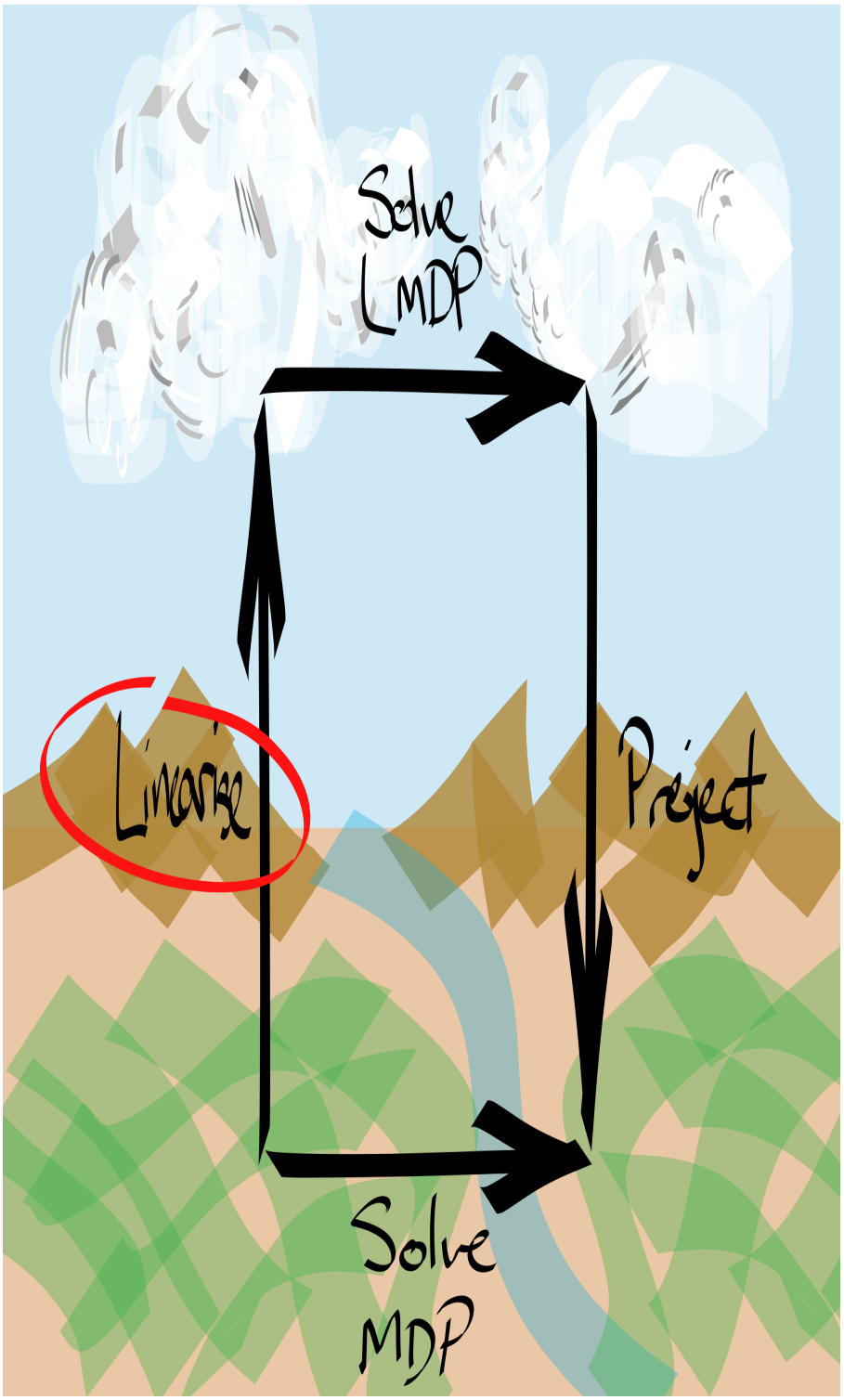


Figure 3.2: ”

A relaxed MDP

> Ok great, we can solve LMDPs. But how does being able to solve an LMDP help us solve MDPs?

We want a way to transform a MDP into a LMDP, while preserving the 'structure' of the MDP. But what do we mean by a MDP's structure?

The LMDP, $\{S, p, q, \gamma\}$ should;

- be able to represent the same transition dynamics as the original MDP,
- give the the same rewards was the original MDP,
- have the same optima.

(It turns out that (1) and (2) imply (3) given some assumptions. See Optimality)

So, given a reward function, r, and a transition function, P, from the MDP, we must translate them into a p and a q. Thus we have built a LMDP with the same 'structure'.

$$\forall s, s' \in S, \forall a \in A, \exists u_a \text{ such that;} \tag{3.3}$$

$$P(s'|s,a) = u_a(s'|s)p(s'|s) \tag{1}$$

$$r(s,a) = q(s) - \mathrm{KL}(P(\cdot|s,a) \parallel u_a(\cdot|s)) \tag{2}$$

$$\tag{3.4}$$

Which leads to $|A|$ linear equations to solve, for each state in the MDP.

See appendix [] for more details.

Alternative views of linearisation.

- A relaxation of the MDP
- Linelihood interpretation

Unconstrained dynamics and state rewards

Let's try and understand this thing we have contructed.

The state rewards are not capable of giving rewards for actions taken. Rather, the differences in reward, by taking another action, is captured by the KL divergence between the control and the unconstrained dynamics.

- What is their function?
- What do they look like?

Does it make sense to treat the q(s) like rewards?! They reward for bing in state s. But cant capture action specific rewards!?

Decoding Ok, so now we get a glimpse at why LMDPs are an interesting abstraction. THe LMDP has disentangled the search for the behaviour (go to this or that state) and the search for optimal controls (how to actually achieve that behaviour). This can be seen in the decoding step. As we know which states we want to be in, via the optimal control from solving the LMDP, u^*, but, we do not know how to implement those controls using the actions we have available.

Two 'simpler' problems. Easier to solve?

$$P_\pi(\cdot|s) = \sum_a P(\cdot|s,a)\pi(a|s) \tag{3.5}$$

$$\pi = \underset{\pi}{\operatorname{argmin}} \operatorname{KL}\Big(u(\cdot|s)) \parallel P_\pi(\cdot|s)\Big) \tag{3.6}$$

Maybe this isnt enough? Do we need to add a reward sensitive part as well?!? (but what if the actual path we take to get there has a neg rewards?!?)

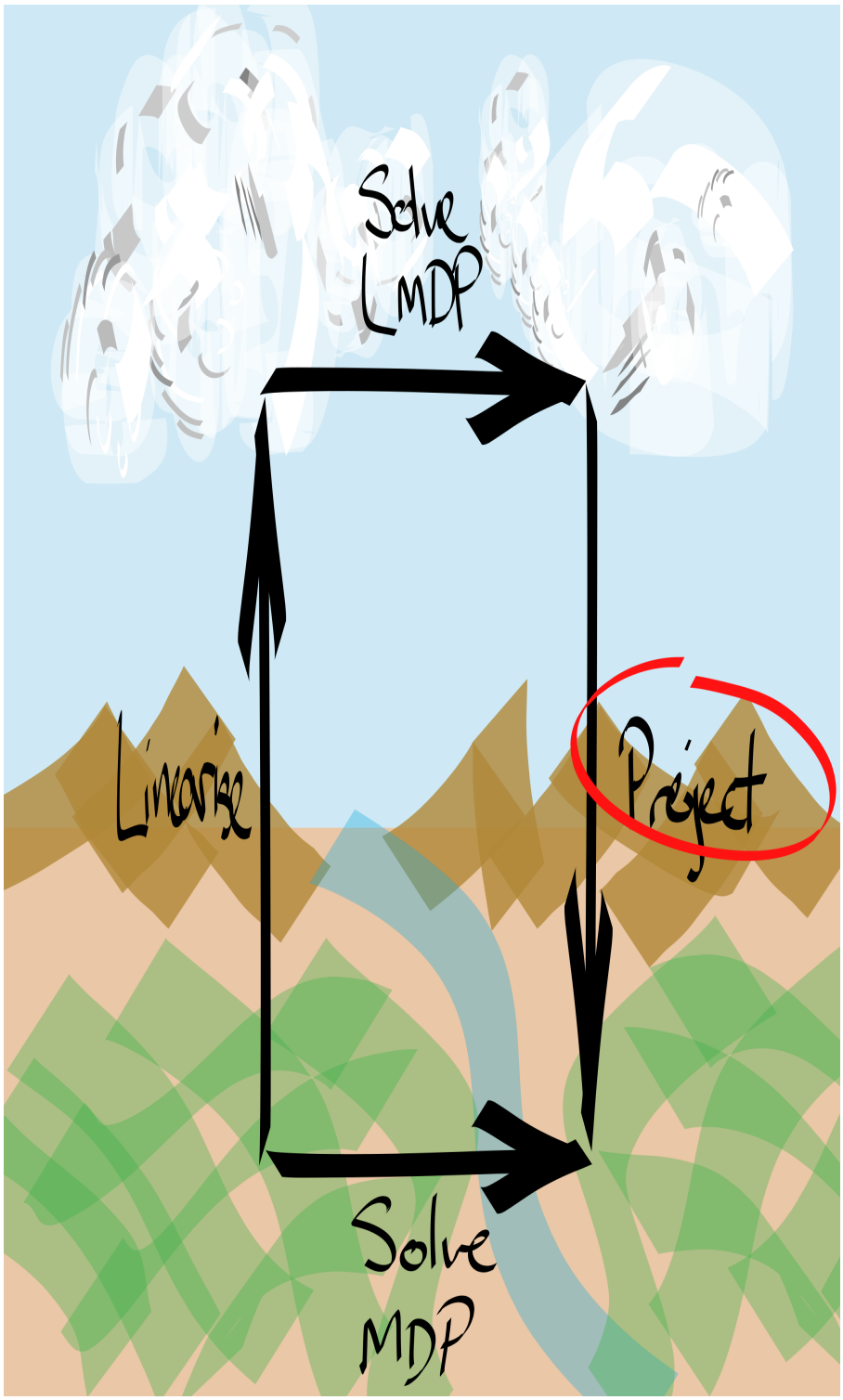


Figure 3.3: ″

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 linearise, solve and project, do we end up in the same place? This is a question about the completeness of our abstraction. Can our abstraction represent (and find) the same solutions that the original can?

$$
\begin{aligned}
\| V_{\pi^*} - \| V_{\pi^*} - V_{\pi_{u^*}} \|_\infty &= \epsilon && (1)\\
&= \| (I - \gamma P_{\pi^*})^{-1} r_{\pi^*} - (I - \gamma P_{\pi_{u^*}})^{-1} r_{\pi_{u^*}} \|_\infty && (2)\\
&\leq \| (I - \gamma P_{\pi^*})^{-1} r - (I - \gamma P_{\pi_{u^*}})^{-1} r \|_\infty && (3)\\
&= \| \left((I - \gamma P_{\pi^*})^{-1} - (I - \gamma P_{\pi_{u^*}})^{-1} \right) r \|_\infty && (4)\\
&\leq r_{\max} \| (I - \gamma P_{\pi^*})^{-1} - (I - \gamma P_{\pi_{u^*}})^{-1} \|_\infty && (5)\\
&= r_{\max} \| \sum_{t=0}^{\infty} \gamma^t P_{\pi^*} - \sum_{t=0}^{\infty} \gamma^t P_{\pi_{u^*}} \|_\infty && (6)\\
&= r_{\max} \| \sum_{t=0}^{\infty} \gamma^t (P_{\pi^*} - P_{\pi_{u^*}}) \|_\infty && (7)\\
&= \frac{r_{\max}}{1 - \gamma} \| P_{\pi^*} - P_{\pi_{u^*}} \|_\infty && (7)
\end{aligned}
$$

(3.7)

(1) We want to compare the optimal policies value and the value achieved by the optimal LDMP solution.

(2) Assume that there exists a policy that can generate the optimal control dynamics (as given by the LMDP). In that case we can set $P_{\pi_{u^*}} = U^*$.

(3) r_{u^*} doesnt really make sense as the reward is action dependent. We could calculate it as $r_{\pi_{u^*}}$, but we dont explicity know π_{u^*}. $(I - \gamma P_{\pi^*})^{-1} r$ represents the action-values, or Q values. By doing this exhange, we

might over estimate the diffference under the infinity norm as two non-optimal actions may have larger difference. Also, use the element wise infinity norm.

Ok, great. Insights from optimality bounds.

Need to be able to approximate the optimal controls. When is it hard to approximate the optimal controls? When our basis set of distributions oer future states (aka our actions) have little weight…?

Potential solution? Use options.

Option decoding What about using options to help solve the optimal control decoding? Does this actually help?!

$$P_\pi(\cdot|s) = \sum_\omega P_k(\cdot|s,\omega)\pi(\omega|s) \tag{3.8}$$

$$\pi = \underset{\pi}{\text{argmin}}\, \text{KL}\Big(u(\cdot|s)) \parallel P_\pi(\cdot|s)\Big) \tag{3.9}$$

Options would allow greater flexibility in the $P_\pi(\cdot|s)$ distribution, making is possible to match $u(s'|s)$ with greater accuracy (and possibly cost).

- First need to demonstrate that action decoding is lossy.
- Then show that using options is less lossy.

This introduces dangers?!? As an option might accumulate unknown rewards along the way!??

The complexity of solutions via LMDPs

Is my path actually shorter?

The whole point of this abstraction was to make the problem easier to solve. So hasit actually made it any easier?

The complexity of solving our abstraction can be broken down into the three steps;

- linearisation: $|S| \times \min(|S|, |A|)^{2.3}$
- solve the LMDP: $\min(|S|, |A|)^{2.3}$
- project back: $???$

Giving a total complexity of ...
Contrasted with the complexity of solving an MDP.

Scaling to more complex problems

Now that we have some evidence that this LMDP solution strategy makes sense, it efficiently (see complexity) yields high value (see optimality) policies. We want to test it out on some real world problems. But the real world isn't as nice as the setting we have been working in. There are a few added complexities;

- sample based / incremental
- large / cts state spaces
- sparse rewards

So now that we have explored LMDPs, how can we extract their nice properties into an architecture that might scale to more complex problems: larger state spaces and action spaces, sparse rewards, ...?

Incremental implementation Generalise to a more complex problem. We are only given samples. A first step to tackling more complex problems.

Model based Learn p, q based on samples.

$$\mathcal{L}(\theta, \phi) = \mathop{\mathbb{E}}_{s,a,} \left[r(s,a) - q_\theta(s) + \mathrm{KL}(p_\phi(\cdot|s) \parallel P(\cdot|s,a)) \right] \tag{3.10}$$

$$\mathcal{L}(\theta, \phi) = \mathop{\mathbb{E}}_{s,r,s'} \left[r - q_\theta(s) - p_\phi(s'|s) \log \frac{1}{p_\phi(s'|s)} \right] \tag{3.11}$$

$$\tag{3.12}$$

Ok. Lets take a different approach. **Q:** Why is it a bad idea to try to do incremental RL with this linearisation trick? Not sure.

Alternative perspective. The high value trajectories are the most likely ones.

Distributions over states

What if we wanted to approximate these distributions? Generalise subgoal methods to work with distributions? The distribution could be constructed via; parzen window / GMM, neural flow, ?!.

Connections to distributional RL?

Questions

- What is p(s′|s)!?!?
- Want some examples of MDPs they cannot solve.
- What is the relationship to other action embedding strategies?
- How does p(s′|s) bias the controls found??? I can imagine the unconstrained dynamics acting as a prior and prefering some controls over others.
- If we have m states and n actions. Where m >> n. Then $u(s'|s)$ is much larger than $\pi(a|s)$. Also, $u(s'|s)$ should be low rank?! $u_{s's} = \sum_a u_a \alpha_a u_a^T$

3.0.2 Other properties

LMDPs have the property that if we have already solved two LMDPs, with the same state space, action space, unconditioned transition dynamics, but different state rewards, q_1, q_2. Then we can solve a new LMDP, again with the same, ..., and state rewards in the span of q_1, q_2, $z_3 = w_1 z_1 + w_2 z_2$, ...

Problem. What does it even mean for two LMDPs to have the same unconditioned dynamics but different state rewards? The MDPs must have been the same up to some additive constant (constant in the actions), $r(s, a) = r(s, a) + c(s)$. Does this really capture what we mean by different tasks?!?

AND HRL!?!?

Refs

- Efficient computation of optimal actions
- Linearly-solvable Markov decision problems
- Moving Least-squares Approximations for Linearly-solvable MDP
- Aggregation Methods for Lineary-solvable Markov Decision Process
- A Unifying Framework for Linearly Solvable Control
- A Stability Result for Linear Markov Decision Processes

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.