Reinforcement Learning with options

Ayoub GHRISS

Supervisor:
Dr Masashi Sugiyama
RIKEN AIP

Academic Supervisor:
Dr Alessandro Lazaric
ENS Paris-Saclay

Master Thesis
Mathmatiques, Vision, Apprentissage

Submitted to Department of Applied Mathematics, ENS Paris-Saclay
Performed in RIKEN Advanced Intelligence Project

Tokyo, Japan
Sept, 2018

Abstract

The current thesis aims to explore the reinforcement learning field and build on existing methods to produce improved ones to tackle the problem of learning in high dimensional and complex environments. It addresses such goal by decomposing learning task in hierarchical fashion known as Hierarchical Reinforcement Learning.

We start in the first chapter by introducing with Markov Decision Process framework and presenting some of its recent techniques that the following chapters use. We then proceed to build our Hierarchical Policy learning as an answer for the limitations of using single primitive policy. The hierarchy is composed of a manager agent at the top and employee agents at the lower level.

In the last chapter, we attempt to learn lower-level elements of the hierarchy in an independent way of the manager level in what is known as the "Eigenoption Discovery". Based on the graph structure of the environment, Eigenoptions allows to build agents that are aware of the geometric properties of the environment. Their decision making is invariant to several symmetric transformations of the the environment. Allowing as consequence to reduce the complexity of the learning task.

Contents

Chapter 1

Reinforcement Learning Framework

Abstract

We introduce the Markov Decision Process paradigm, some of the approximate Dynamic Programming methods, their counterpart in continuous space states and lay their fundamentals which will be used in the next chapters.

Introduction

The concept of Reinforcement Learning (Sutton and Barto [1]) introduced a mathematical framework of the active learning from experience. The concept is based on developing agents capable of learning in an environment through trial-and-error. The environment is a set of states S and actions A. The agent interacts with the environment by moving from a state to another and receives a feedback about its decisions through rewards. The goal of the agent is to learn a decision making process, also called a policy, that maximizes the expected accumulated reward.

Based on this definition, we can already distinguish between two different types of Reinforcement Learning, depending on the prior knowledge about the environment,: model-based and model-free. In the first type ofReinforcement Learning (RL), the dynamics of the environment are known prior to the learning process, and the agent incorporate the knowledge of the transition process between states (the laws of physics in a robotics task for example). In contrast, in the second type, the agent has no prior knowledge about the environment dynamics and needs to acquire it while learning the best policy. In this work, we address model-freeRL.

In this chapter, we introduce the Markov Decision Process paradigm (**MPD**) and summaries its fundamentals.

1 Markov Decision Process

In standard RL models, the agent is represented as a time-homogeneous Markov Decision Process (MDP). An MDP is a tuple $\Omega = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$ where $\mathcal{S}$, $\mathcal{A}$ are respectively the state and action spaces, $\mathcal{P}$ is the transition kernel where $P(s_{t+1}|s_t, a_t)$ represents probability of transition from state s_t to a new state s_{t+1} when taking the action a_t. Each decision of taking action a_t at state s_t can yield a certain reward $R(s_t, a_t)$ which we assume is bounded and deterministic given s_t and a_t. In the case of an infinite horizon , future rewards can be discounted with factor γ and the MDP becomes $\Omega = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, which will our MDP on interest in this work.

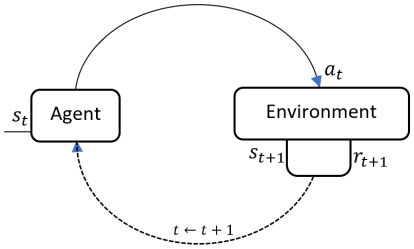


Figure 1.1: MDP structure

The goal in RL is to permit the agent to learn autonomously, through trial and error, a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that allows to maximize the expected accumulated rewards $\eta(\pi)$ defined as:

$$\eta(\pi) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \tag{1.1}$$

where $\mathbb{E}^{\pi}$ means:

- $s_0 \sim \rho_0$, ρ_0 is the initial distribution over the state space
- The action is then distributed following $a_t \sim \pi(.|s_t)$.
- The next states follows $s_{t+1} \sim P(.|s_t, a_t)$

The time homogeneity implies that at any time t we have:

$$P(s_{t+1} = s'|s_t = s, a_t = a) = P(s'|s, a)$$

1.1 Value Functions

Value functions measure the expected return conditioned on starting state and/or actions: the state-action value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and state value function $V_\pi : \mathcal{S} \to \mathbb{R}$ both defined with respect to policy π:

$$Q_\pi(s,a) = \mathbb{E}_{s_t,a_t,\dots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)|s_0 = s, a_0 = a\right], \tag{1.2}$$

$$V_\pi(s) = \mathbb{E}_{a_t,s_t,\dots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)|s_0 = s\right], \tag{1.3}$$

and define the advantage function :

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

The value $Q_\pi(s,a)$ is the expected accumulated reward when taking the action a at state s. When we sample this action a following the policy $\pi(.|s)$, then we expect to accumulate $V_\pi(s)$. The advantage $A_\pi(s,a)$, on the other hand, translates the gain of taking the action a in state s instead of following the policy π. Consequently, the policy is optimal if the advantage is non-positive at each state for any adversary action a.

1.2 State density and expected return

Each policy induces a distribution over the state space which is proportional to the "unnormalized" state density ρ_π defined as

$$\begin{aligned}\rho_\pi(s) &= \mathbb{E}^\pi\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{1}_{s_t=s}\right] \\ &= \mathbb{E}_{s_0\sim\rho_0, a_t\sim\pi(.|s_t)}\sum_{i=0}^{\infty}\gamma^t P(s_{t+1}=s|s_t,a_t).\end{aligned} \tag{1.4}$$

Using ρ_π, the expected return for using the policy π can be formulated as:

$$\eta(\pi) = \int_{\mathcal{S}}\int_{\mathcal{A}}\rho_\pi(s)\pi(a|s)r(s,a)dsda \tag{1.5}$$

We can also simplify V_π using the initial density ρ_0:

$$\eta(\pi) = \mathbb{E}_{s_0\sim\rho_0}V_\pi(s_0), \tag{1.6}$$

This simple property can be used to prove the following important lemma, which links the discounted density and the expected return of two policies (see [2] for proof):

Lemma 1.1. *Let π and $\tilde{\pi}$ be two policies over the states space $\mathcal{S}$, we have the following:*

$$\eta(\pi) = \eta(\tilde{\pi}) + \mathbb{E}^{\tilde{\pi}} \left[\sum_t \gamma^t A_\pi(s_t, a_t) \right], \tag{1.7}$$

and using the density ρ_π :

$$\eta(\pi) = \eta(\tilde{\pi}) + \int_{\mathcal{S}} \int_{\mathcal{A}} \rho_{\tilde{\pi}}(s) \tilde{\pi}(a|s) A_\pi(s_t, a_t) \tag{1.8}$$

This lemma, as discussed in subsection 1.1, implies that a policy update $\pi \leftarrow \tilde{\pi}$ with non-negative expected advantage over π at every state will increases $\eta(\pi)$.

Remarks

In the remaining study, we restrict our interest to the following MDPs

- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is deterministic given the state and action
- The transition mechanism from s_t to s_{t+1} given a_t is deterministic

While deterministic rewards applies to a large set of environments, using a prior distribution on the reward has recently been proven to improve many of the reinforcement learning models, even when the reward is deterministic [3]. It has also proven that in distributional RL, some of the properties of MDP change like the contraction property of Bellmann operator which we'll define in the following section.

2 Dynamic Programming

Dynamic Programming (DP) [4] refers to a mathematical optimization paradigm developed by Richard Bellman in the 1950s. It addresses optimization problems where the sub-problems can be nested recursively inside the general one. Using Bellman equations with specific assumptions on the objective function, we are guaranteed to have a link between the general solution and the sub-problems solutions. In our case, we can notice that for two consecutive states s_t, s_{t+1}, if a policy π maximizes $V_\pi(s_t)$ then it necessarily maximizes $V_\pi(s_{t+1})$, and a similar reasoning can be applied to Q_π.

2.1 Bellman Equations

The Bellman equations applied to the value functions can be written as follows:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(.|s)}\left[r(s,a) + \gamma\mathbb{E}_{s'\sim P(.|s,a)}\left[V_\pi(s')\right]\right] \tag{1.9}$$
$$Q_\pi(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim P(.|s,a)}\left[V_\pi(s')\right]. \tag{1.10}$$

We define the Bellman operators $\mathcal{T}_\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ as:

$$\mathcal{T}_\pi(f)(s) = \mathbb{E}_{a\sim\pi(.|s)}\left[r(s,a) + \gamma\mathbb{E}_{s'\sim P(.|s,a)}\left[f(s')\right]\right], \tag{1.11}$$

and the optimal Bellman operator $\mathcal{T}^* : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ as:

$$\mathcal{T}^*(f)(s) = \max_a\left[r(s,a) + \gamma\mathbb{E}_{s'\sim P(.|s,a)}\left[f(s')\right]\right] \tag{1.12}$$

Both $\mathcal{T}_\pi$ and $\mathcal{T}^*$ are contractions with respect to the L^2 norm on their respective spaces, with Lipschitz constant γ. Considering that $\gamma < 1$, both operators admit unique fixed points [5] and we can conclude that:

- Every policy π defines a unique value functions V_π and Q_π
- There is unique optimal value function V^*, the fixed point of $\mathcal{T}^*$, which verifies:

$$V^*(s) = \max_a \mathbb{E}_{\pi(.|s)}\left[r(s,a) + \gamma\mathbb{E}_{s'\sim P(.|s,a)}\left[V^*(s')\right]\right]$$

The optimal state-action value function Q^* can be induced:

$$\begin{aligned} Q^*(s,a) &= r(s,a) + \gamma V^*(s) \\ &= r(s,a) + \gamma \max_a \mathbb{E}_{s'\sim P(.|s,a)}\left[\max_{a'} Q^*(s',a')\right] \end{aligned}$$

These properties, however, do not imply the uniqueness of the optimal policy π^* associated with the optimal functions :

$$\pi^*(a|s) = \mathbb{1}_{\arg\max_a Q_{\pi^*}(s,a)},$$

Based on the advantage function A^*, if two actions a_1 and a_2 at a certain state s verify $A^*(s,a_1) = 0$ and $A^*(s,a_2) = 0$, then we're indifferent between the two as they both have the same future accumulated discounted rewards.

2.2 Iterative Learning

Based on the Bellman equations and the properties of the operators, we introduce in this section different methods used to learn the optimal value functions and/or the optimal policies. The iterative learning is used for finite MDP, but they share many fundamentals with their counterparts in continuous state MDP. It is important to note that the presented methods belong to the approximate Dynamic Programming, where we attempt to approximate value functions and policies. The approximation quality mainly depends on the richness of the functions class in which we optimize (linear, quadratic, neural networks...).

As noted in subsection 2.1, since the Bellman operators are contraction mappings, we have the following result:

Theorem 1.1 (Contraction mapping theorem)**.** *Let $\mathcal{X}$ be a complete metric space, and $F : \mathcal{X} \to \mathcal{X}$ be a contraction. Then F has a unique fixed point $x^* \in \mathcal{X}$, and under the action of iterates of F, all points converge with exponential speed to x^*.*

Using Bellman operators, by starting from any intial function $f_0 : \mathcal{S} \to \mathbb{R}$ and defining $f_n = \mathcal{T}^n(f_0)$, then $(f_n)_n$ converges to the fixed point of $\mathcal{T}$. While this applies to both V and Q functions, it is generally preferable to learn the action value Q (or the advantage A) function since they allow to induce the optimal action at each state. Learning only V, however, requires the knowledge of either the reward or the temporal difference $V(s_{t+1}) - V(s_t)$, implying prior knowledge about the transition mechanism to predict the next state.

Value Functions Iteration

Using the contraction mapping property, we can iteratively approximate the value function V in a discrete infinite time horizon MDP: The same algorithm can be applied

Algorithm 1 Value Iteration Algorithm

Require: An initial value function $V : \mathcal{S} \to \mathbb{R}$
  for $t \leftarrow 1$ to T **do**
    $s_0 \sim \rho_0$
    Sample a trajectory $\tau = (a_0, s_1, s_2 ... s_n)$
    Estimate $V_{t+1} \leftarrow T^*(V_t)$ based on τ
  end for
Return: The greedy policy associated with V_T

on the state-action function Q_t. The updates can be performed in an asynchronous fashion, by updating the value function at each step of the trajectory. The trajectory sampling

can be done in a on-policy manner by using the greedy policy π_t at iteration t, or in an off-policy way. The on-policy method can imply a risk of staying in a local domain of the state space, the common practice is to add stochasticity with $\epsilon-$greedy method: by choosing random actions with probability ϵ. The contraction mapping property guarantees convergence only if all states can be visited infinitely often. This is known as the exploitation-exploration dilemma in RL, since we want to exploit the learned knowledge while being able to explore new states once in a while.

Policy Iteration

In a similar way to the value iteration, the policy iteration starts with an initial policy π_0 then alternates between learning V_π and π. The same thing about exploration-

Algorithm 2 Policy Iteration Algorithm

Require: An initial value policy π_0
  for $t \leftarrow 1$ to T **do**
    Sample a trajectory $\tau = (s_0, a_0, s_1, s_2 ... s_n)$
    Evaluate $V_{\pi t}$
    $\pi_{t+1} \leftarrow \pi^*{}_t$ with $\pi^*{}_t$ the greedy policy of $V_{\pi t}$
    Estimate $V_{t+1} \leftarrow T^*(V_t)$ based on τ
  end for
Return: The learned policy π_{T+1}

exploitation applies to the policy iteration. The PI and VI methods assume that we can efficiently estimate $V_{\pi,t}$ and π_t which are practically challenging, since the value functions are formulated as the expectations the policy and initial distribution. Practically, we sample one or several trajectories and estimate the expectation over the trajectories and incrementally update the previous quantities.

Temporal Difference

When starting from a state s_t and taking action a_t to observe s_{t+1}, the one step temporal difference error is defined as

$$\Delta_t := r(s_t, a_t) + \gamma {V_\pi}^{(n)}(s_{t+1}) - {V_\pi}^{(n)}(s_t), \tag{1.13}$$

with ${V_\pi}^{(n)}$ being the value function estimation at iteration n and $r(s_t, a_t) + \gamma {V_\pi}^{(n)}(s_{t+1})$ the TD target.

For $\lambda \in [0, 1]$, the λ-temporal difference TD(λ) is used to update the estimated function:

$$V_{\pi n+1}(s_t) \leftarrow V_{\pi n} + \alpha_n \sum_{s=0} \lambda^s \Delta_s.$$

The parameter $\lambda \in [0, 1]$ called the trace decay, higher λ reflects lasting reward traces, with $\lambda = 1$ the value function update is performed at the end of the trajectory, while for $\lambda = 0$ the update is performed after each decision step. The update rate (α_i) is generally chosen to satisfy the Robbins-Monro condition:

$$\sum_i \alpha_i = \infty \tag{1.14}$$

$$\sum_i \alpha_i^2 < \infty \tag{1.15}$$

Given enough samples, Robbins-Monro condition guarantees that the approximations $({V_\pi}^{(n)})_n$ converge almost surely to the true V_π.

The temporal difference error plays an important role in continuous and large space state MDP and it is largely used for Q and V function learning (SARSA[6], Q-learning[1]). It's main advantage is that it be applied in high dimension state space MDP using differentiable approximations. In the following section, we will be exposing some of these methods.

3 Gradient Methods

In continuous and high dimensional state space MDP, the Temporal Difference is core of several approximate DP models. The most basic methods use parameterized functions (Least Square, Neural Networks) and attempts to update the parameters to minimize $TD(\lambda)$ using closed form solutions or gradients depending on the parameterization. Neural Networks have gained popularity for their ease to use in gradient descent methods and their ability to scale RL techniques to high dimensional state space.

In this section, we go beyond the simple applications of $TD(\lambda)$ to briefly present a particular class of gradient methods: policy gradient methods. We summarize the work of Neu et al.. In their work on unifying policy gradient methods, they re-introduces 3 popular policy gradient methods from a convex optimization point of view:

- Relative Entropy Search (REPS) (2010)
- Trust Region Policy Optimization (TRPO) (2015)
- Asynchronous Actor-Critic Agents (A3C) (2016)

They demonstrate that these methods aim to solve the same objective using different constraints.

3.1 Regularization functions

We define the Shannon entropy E_S for a policy π as

$$E_S(\pi) := \sum_{s,a} \pi(x,a) \log \pi(x,a), \tag{1.16}$$

and the negative conditional entropy E_c :

$$E_C(\pi) := \sum_{s,a} \pi(s,a) \log \frac{\pi(s,a)}{\sum_b \pi(s,b)}. \tag{1.17}$$

To constraint the policy updates, we use the Bregman divergence D_S and D_C associated to the entropy, respectively, E_S and E_D defined as :

$$D_S(\pi \| \tilde{\pi}) := \sum_{s,a} \pi(s,a) \log \frac{\pi(s,a)}{\tilde{\pi}(s,a)} \tag{1.18}$$

$$D_C(\pi \| \tilde{\pi}) := \sum_{s} \pi(s) \sum_{a} \pi(a|s) \log \frac{\pi(a|s)}{\tilde{\pi}(a|s)} \tag{1.19}$$

E_S and E_C are two strictly convex functions over the space of distributions on the $\mathcal{S} \times \mathcal{A}$. D_S is in fact the Kullback-Leibler divergence while D_C is known as the conditional KL divergence over the state space.

3.2 Convex optimization perspective

With regularization functions defined above, Neu et al. proved that at the t gradient update of the policy, all the three methods attempts to solve the following objective :

$$\pi_{t+1} = \arg\max_{\pi} \Big(\eta(\pi) - \alpha . D(\pi \| \pi_t) \Big), \tag{1.20}$$

where D being the divergence associated with the method.

REPS uses mirror descent with Bregman divergence D_S in 1.20.

$$\pi_{t+1} = \arg\max_{\pi} \Big(\eta(\pi) - \alpha . D_S(\pi \| \pi_t) \Big).$$

In each update t, the trajectories are sampled using the current policy π_t. Using Lagrangian strong duality and the strict convexity of the entropy E_S, REPS is equivalent to the mirror descent algorithm using the Bregman divergence D_S. Consequently, REPS converges to the optimal policy. However, as it is the case for all other methods, the optimality is relative to the class of parameterized functions on which we optimize the objective in (1.20).

TRPO Similarly, it solves 1.20 using the Bregman divergence D_C. TRPO formulates the problem as:

$$\begin{aligned} &\underset{x}{\text{maximize}} \quad \eta(\pi) \\ &\text{subject to} \quad D_C(\pi \| \pi_t) \leq \delta, \end{aligned} \tag{1.21}$$

where δ is a hyper-parameter controlling the update step. TRPO determines the trust region update using conjugate gradient to bound the divergence.

A3C accumulates gradients of the objective (1.20) with regularization D_C. The gradients are provided asynchronously from several agents. Instead of using mirror descent, it uses the dual averaging method by making updates provided by the agents. However, in their unified framework, Neu et al. argue that A3C doesn't present any convergence guarantees. The interesting result is that the optimal update in the case of A3C has a closed form:

$$\pi_{t+1} \propto \exp \frac{A_{\pi_k}}{\beta_k}, \tag{1.22}$$

where (β_k) an increasing sequence to insure convergence.

3.3 Trust Region Policy Optimization

TRPO optimizes the objective (1.21) by first replacing the discounted state density $\rho_{\tilde{\pi}}$ in Equation 1.8 with ρ_π. The obtained the quantity is:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a). \tag{1.23}$$

$L_\pi(\tilde{\pi})$ is a first order approximation of $\eta(\tilde{\pi})$. At this stage, we consider parameterized policies $(\pi_\theta)_{\theta \in \mathbb{R}^n}$ and we use θ to design π_θ. Schulman et al. proved the following inequality, in analogy with the Conservative Policy Iteration (CPI):

Lemma 1.2.

$$\eta(\theta) \geq L_{\theta_{\text{old}}}(\theta) - C D^{\max}_{\text{KL}}(\theta_{\text{old}}, \theta). \tag{1.24}$$

Maximizing the right side of the inequality improves (remains constant in the worst case) the surrogate. TRPO proceeds in relaxing the bound based on D^{max}_{KL} and the objective becomes:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta) \\ &\text{subject to } \overline{D}^{\rho_{\theta_{\text{old}}}}_{\text{KL}}(\theta_{\text{old}}, \theta) \leq \delta. \end{aligned} \tag{1.25}$$

$D^{\max}_{\text{KL}}$ is the maximum KL divergence between policies over the sampled states, while $\overline{D}_{\text{KL}}$ is the mean KL divergence.

TRPO recipe

By simulating trajectories following the current policy, we estimate the objective and the bound.

- Expand $L_{\theta_{\text{old}}}$ in Equation (1.25), we replace $\sum_s \rho_{\theta_{\text{old}}}(s)\,[\dots]$ in the objective by the expectation $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}[\dots]$
- Replace $A_{\theta_{\text{old}}}$ by the Q-values $Q_{\theta_{\text{old}}}$ in Equation (1.23),
- replace the sum over the actions by an importance sampling estimator based on $\pi_{\theta old}$.

TRPO problem is hence equivalent to the following one:

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}},a\sim q}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta old}(a|s)}Q_{\theta_{\text{old}}}(s,a)\right] \\ &\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s)\parallel\pi_\theta(\cdot|s))\right]\leq\delta. \end{aligned} \tag{1.26}$$

Algorithm:

We denote by g_θ the gradient of the objective with respect to θ. We write the quadratic approximation of the constraint function :

$$\overline{D}_{\text{KL}}(\theta_{\text{old}},\theta)\approx\frac{1}{2}(\theta-\theta_{\text{old}})^T A(\theta-\theta_{\text{old}})$$

where $A_{ij}=\frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\overline{D}_{\text{KL}}(\theta_{\text{old}},\theta)$. Since A is symmetric semi-definite, we use the conjugate gradient method to solve $s=Ag_\theta$. The solution s is the search direction. We then obtain the maximal step scale $\beta=\sqrt{2\delta/s^TAs}$ that verifies $\delta=\overline{D}_{\text{KL}}\approx\frac{1}{2}(\beta s)^TA(\beta s)=\frac{1}{2}\beta^2s^TAs$ The last step is the line search, which allows to shrink the step exponentially until the objective improves.

3.4 Experiment

We test the TRPO algorithm in the "4 room" environment. The motivation for such choice of environment is the ability to compare results between continuous space, where we use the raw RGB screen as input, and the exact solutions in the tabular case. The agent (white square) goal is to reach the red square to receive a $+1$ reward, in minimal number of steps. The horizon is virtually infinite, but practically sufficient enough to explore all states before episodes end. We can also expand the state space by expanding the grid's size. For a size : $n=36$ and randomly located white and red squares, the total

Algorithm 3 Trust Region Policy Optimization

Require: A parameterized policy π_θ and θ=θ_0
  for $i \leftarrow 1$ to T **do**
    Run policy for N trajectories
    Estimate advantage function at all time steps
    Compute objective gradient g_θ
    Compute A
    Use Conjugate Gradient to compute β and s
    Compute the rescaled update line search
    Apply update to θ
  end for

(a) $n = 16$ (b) $n = 36$

Figure 1.2: 4-room environment for different sizes

number of states is of order 10^6. We can also control the sparsity by varying the number of red squares.

Our implementation of TRPO agent is a Pytorch adaptation of the official Tensorflow implementation from OpenAI baselines [11]. We've tested the implementation on some of the Atari Games, and the scores were similar to the ones reported in [9].
However, for the 4-room environment with large size in the stochastic framework (the target red square is randomly located), the TRPO fails to improve the policy. When reaching the goal, it quickly forgets the experience. This is mainly due to the sparsity of the rewards.

We plot below the mean time to the goal for each room size. We start from the uniform policy. The plots are moving averages of 5 runs. We've also run the algorithm stagnates at the maximum episode length.

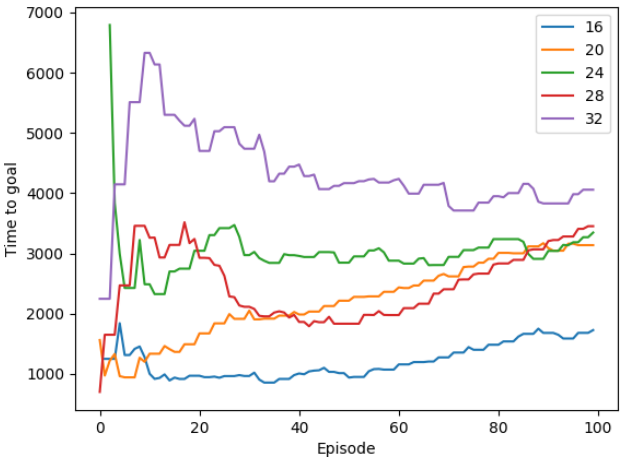


Figure 1.3: TRPO in 4 Rooms

Conclusion

After going through fundamentals of Reinforcement Learning in multi-state space, we summarized Neu et al.'s unifying framework for entropy regularized policy gradient methods. We tested the TRPO algorithm on few environments and demonstrated some of its limitation in a sparse environment.

In the following chapter, we attempt to tackle these limitations in the framework of the Hierarchical Reinforcement Learning (HRL) by extending the TRPO to incorporate task hierarchies.

Chapter 2

Hierarchical Reinforcement Learning

Abstract

We introduce HRL as way to tackle the learning scaling problem and sparsity. We review some recent approaches used to learn decision hierarchies, and conclude with a on-policy hierarchical reinforcement learning method based on the work of Takayuki Osa for an off-policy.

Introduction

As it's the case of the human learning, biological organisms can master tasks from extremely small samples. The fact that a child can acquire a visual object concept from a single unlabeled example is an evidence for preexisting cortical mechanisms that facilitate such efficient learning [12]. It mainly suggests that acquiring new skills is done in a gradual fashion starting with simpler tasks that allow the abstraction of new raw visual objects. While reinforcement learning is rooted in Neuroscience and Psychology[13], Hierarchical Reinforcement Learning was developed in the machine learning field based on the abstraction of either the states or the actions.

State abstraction [14, 15] is based on learning a state representation concentrating features relevant to the decision process and excluding the irrelevant ones, hence, behaviorally similar state should induce the same representation. On the other hand, actions abstraction is based on using temporally extended policies [16] to scale the learning task. Once a temporally abstract action (sub-policy) is initiated, the execution of its associated policy continues until a set of specified termination states is reached. Thus, the selection of a temporally abstract action ultimately results in the execution of a sequence of actions.

In this chapter we focus on the temporal action abstraction for a continuous state MDP. We start by an overview of related work on which we proceed to build our method.

1 Hierarchical Reinforcement Learning Methods

While Hierarchical Learning is based on features hierarchy, hierarchical RL is based on hierarchy of tasks. At the top of the structure, a manager designate an employee with an assigned goal. The employee attempts to accomplish the goal to get the reward before a termination condition is met (it gets fired). Employees can be on their own managers for other sub-employees, which is known as the "Feudal Reinforcement Learning" [17] with the "manager" (Lord) policy at the top and "sub-managers" in low levels of the hierarchy receiving sub-goals from their "super-managers".
Some of the earlier work on HRL used the terms "primitive actions" and "macro-actions". In [18], Hauskrecht et al. introduced the macro-actions as local policies that act in certain regions of state space. The "primitive action" policy then clusters the state space and assigns each macro-action to a specific state space domain.

In this thesis, we will use the terms "gate" or "gating" policy for the primitive actions, and "option" policies for the macro-actions.

1.1 Hierarchical RL models

An option is a tuple (I_o, π_o, T_o), where $I_o \subset \mathcal{S}$ is the initiation set, π_o is the option's policy, and $T_o \subset \mathcal{S}$ the termination set (stopping time). The gating policy, in a similar way to the primitive policies from the first chapter, is a mapping $\pi_g : \mathcal{S} \times \mathcal{O} \to [0, 1]$ from the state space to the distributions over the option set $\mathcal{O} = \{o_i | i \in I\}$ the set of the options. Following the same notations of the first chapter, we denote the gating policy by π_g and $\pi(.|s, o)$ the policy of the option o.

We define the hierarchical policy as:

$$\pi(a|s) = \sum_{o \in \mathcal{O}} \pi_g(o|s)\pi(a|s, o). \tag{2.1}$$

Using the discounted state distribution ρ_π as in equation (1.5), the objective becomes:

$$\eta(\pi) = \int_{\mathcal{S}} \int_{\mathcal{A}} \rho_\pi(s) \sum_{o \in \mathcal{O}} \pi_g(o|s)\pi(a|s, o)r(s, a)dsda \tag{2.2}$$

Several methods used this structure as in the [19] and[20] (now in ICLR19 [View], Takayuki was my main source for this chapter, I use on-policy (TRPO) while his work is with off-policy nethod),

where, besides the the raw state, the gate provides the option with a latent variable o called "goal" or "sub-goal". In the case of a sub-goal, the options maximizes the expected accumulated intrinsic reward returned from an internal critic.

In a more general framework, the hierarchical policy picks an option at times $S_t = \sum_{l=1}^{t} T_{o_l}$, where o_l is the option drawn at round l. Between S_t and S_{t+1}, the actions are chosen following option o_t. We can write the objective $\eta(\pi)$ as :

$$\eta(\pi) = \mathbb{E}^{\pi}\left[\sum_{t=1}^{\infty}\gamma^{S_{t-1}}\sum_{u=0}^{T_t}\gamma^{u}r(s_t, a_t)\right]$$

Ultimately, the HRL model should also learn the stopping time for each option.

1.2 Related work

In their hierarchical adaptation of the DQN[21], Kulkarni et al. used a gate (meta-controller) and options (controllers) to attain handcrafted goals. While the gate tries to maximize the accumulated reward from the environment (extrinsic reward), the options maximize an intrinsic reward from internal critic by achieving the assigned goals. The abstract action terminates when the episode ends or the goal is reached. Their method succeeded in solving the "MontezumaRevenge Atari" game, in which the non hierarchical methods fail to score. However, the hDQN goals where handcrafted and hence needed input that can be expensive.

For an on-policy variant, Daniel et al.[19] introduced several Hierarchical REPS variants (see section 3 in chapter 1). The option policies in Hierarchical Relative Entropy Search (HREPS) were allowed to share experiences (inter-option learning), the chosen option was a hidden latent representation that model tries to infer using Expectation-Maximization. The inference is based on sampled trajectory, in which each policy o_i attempts to "claim" its responsibility $p(o_i|s, a)$ for action a at state s. To avoid concurrence between policies, HREPS constraints the entropy of options' responsibilities.

Among the recent research on HRL, Harutyunyan et al. in [22] addressed the learning of option's stopping time T_o. In the call-and-return HRL model, as in h-DQN, an option is run until completion. In their paper, they argued for the efficiency of shorter option policies and derived a method to learn the termination condition in an off-policy. In our work however, this aspect is not treated and we restrict our study to fixed option duration.

As pointed out in [18], the gating policy performs a clustering of the state space domain. While in HREPS[19], option policies could achieve same performance on some domains of the state space. This indicates that choice of the optimal gating policy π_g, if achievable, isn't unique o. Since we treat o as a latent variable, in analogy with HREPS, we need to impose additional constraints to obtain a preferable solution. For the choice of constraint, we opt for the Regularization via Information Maximization (RIM).

2 Hierarchical Reinforcement Learning via Information Maximization

To learn the hierarchical policy, we impose a regularization on the latent variable o. The hierarchical policy π attempts then to optimize the objective:

$$\mathcal{L}(\pi) = \eta(\pi) + \mathcal{R}(\pi), \tag{2.3}$$

where $\eta(\pi)$ was defined in Equation 2.2.

We choose to impose mutual information constraint on the gating policy. In Regularised Information Maximization (RIM)framework, the $\eta(\pi)$ is considered the regularization term of π.

2.1 Regularized Information Maximization

In Unsupervised Learning, Mutual Information measures proved to produce interpretable latent representations, especially in the clustering task ([23], [24], [25]). Following the previosuly discussed models, learning the gating policy in our model is similar to performing a clustering of the state space and assigning each options to a specific domain. It is hence justified to leverage Mutual Information (MI) performance in the HRL task.

The gating policy, a discrete distribution over $\mathcal{O}$, is the representation O and $X = (s_i, a_i)_i$ the observed variable. we define the empirical mutual information:

$$\hat{I}(O, X) = \hat{H}(O) - \hat{H}(O|X), \tag{2.4}$$

where $\hat{H}(O)$ the empirical estimate of entropy $H(O) = \int p(o)\log(p(o)) = \mathbb{E}\left[\log(p(O))\right]$ using the empirical distribution:

$$\hat{p}(o) = \frac{1}{n}\sum_{i=1}^{n} p(o|s_i, a_i),$$

and $\hat{H}(O|X)$ the empriical esimtate of $H(O|X)$

$$\hat{H}(O|X) = \frac{1}{n}\sum_{i=1}^{n} p(o_i|s_i, a_i)\log(p(o_i|s_i, a_i),$$

with:$p(o|s, a) = \frac{\pi(a|s,o)\pi_g(o|s)}{p(a|s)}$, and $p(a|s) = \sum_{o\in\mathcal{O}} \pi_g(o|s)\pi(a|s, o)$

Adding the regularizing quantity $\hat{\mathcal{R}}(\pi)$ to the objective η_π incentivize the gating policy to uniformly samples option policies by increasing $H(O)$, while at the same time, to clearly distinguish between each option's domain by decreasing $H(O, X)$.

2.2 Objective function

We fix the number of options policies $|O| = k$ and we consider a deterministic termination set $\mathcal{T}_\iota = \tau$. We consider the hierarchical policy as in (2.1). The hierarchical policy's objective :

$$\eta(\pi) = \int_{\mathcal{S}} \int_{\mathcal{A}} \rho_\pi(s) \sum_{o \in \mathcal{O}} \pi_g(o|s) \pi_o(a|s,o) r(s,a) ds da - \lambda I(O,S), \tag{2.5}$$

for a given parameter λ.

Since we will be using parametric approximation, we note π_θ the policy with parameter θ. In a similar way to TRPO notations, we use θ indices to refer to quantities defined using π_θ.

2.3 On-Policy HRL:

The TRPO model incorporates an estimation of the advantage function A_π, which can be used to model the gating policy using the softmax probabilities. Then the probability of choosing the option i:

$$\pi(o|s,a) = \frac{exp[A(s, \pi_i(s)|o_i)]}{\sum_j exp[A(s, \pi_j(s)|o_j)]} \tag{2.6}$$

This would allow to decouple the learning of the latent variable o and the gating policy, since the distribution on options only depend on their estimation advantages. From the HREPS and hierarchical Deep Q Learning (h-DQN) methods introduced, we can either

- In a similar way to REPS, allow inter-option learning
- Make each policies value estimation hidden from the gating policy. This implies that the gating policy should estimate its own value function as well. This similar to the h-DQN method.

In this work, we present a different method where each option learns only from its own experience while the gating policy is oblivious to the advantages estimation (we won't use softmax Q).

2.4 Hierarchical Kullback-Leibler bound

To extend the TRPO to hierarchical policies, we use the following lemma (see proof 3.2 in Appendix).

Lemma 2.1. *Let $n \geq 1$ be the number of option policies, and π_g the gating policy (distribution over the options set). If π and $\tilde{\pi}$ are two hierarchical policies such that $\tilde{\pi}$ is absolutely continuous w.r.t π, then at any state s, we have*

$$D_{KL}(\pi(.|s)|\tilde{\pi}(.|s) \leq D_{KL}(\pi_g(.|s), \tilde{\pi}_g(.|s)) + \sum_{o \in \mathcal{O}} \pi_g(o|s) D_{KL}(\pi(.|s,o)|\tilde{\pi}(.|s,o))). \tag{2.7}$$

The right side's second term is in fact the Bregman divergence for the conditional entropy but with expectation over the options set instead of the state space as seen in Chapter 1.

In a similar work [26], Michalewski used a hierarchy of parameters. Instead of drawing an option, the gating policy chooses a the subspace for the option's parameters. The options then belong to the family of multivariate Gaussians. This choice of structure turns the inequality in 2.7 into an equality, as the divergence between two hierarchies over the same option vanishes.

Based on 2.7, to bound the KL divergence, it's sufficient to bound both the gate and the options. We can also use adaptive bounds for the options by bounding $\pi_g(o|s)\mathrm{D_{KL}}(\pi(.|s,o)|\tilde{\pi}(.|s,o)))$ instead of $\mathrm{D_{KL}}(\pi(.|s,o)|\tilde{\pi}(.|s,o)))$.

2.5 Algorithm and Experiments

Algorithm 4 Trust Region Hierarchical Policy Optimization

Require: A parameterized gate policy π_θ and option policies $(\pi_{o_i,\theta})$
  for $i \leftarrow 1$ to T **do**
    Run policy for N trajectories
    Estimate the advantage
    Calculate the Mutual Information Measure
    Use Trust Region method to get the update step wrt to the gate
    Update Step
    for each option i **do**
      Get the Objective after last update
      Use the trust region method for the option i
      Apply update
    end for
  end for

The trust region method mentioned in the algorithm is the same one used in standard TRPO, by using quadratic approximation of KL divergence, then exploiting the fisher vector product and use conjugate gradient. The objective needs to be calculated

after each update and also incorporate gradient graphs for each policy when using neural networks. In practical terms, it requires a large GPU memory when naively implemented. A more tricky way would imply stopping gradient from policies uninvolved in the optimization step.

We compare the performance of the TRHPO method with the TRPO in the 4 rooms environment. We use the same neural network architectures for the gating and option policies and the TRPO policy with the same KL bounds.

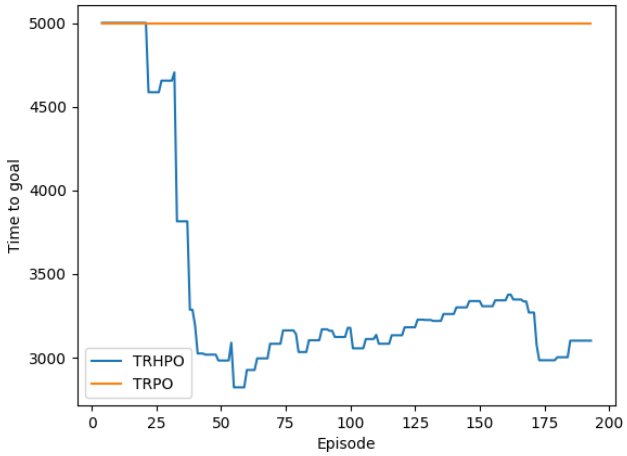


Figure 2.1: TRHPO vs TRPO in 4 Rooms

While the TRHPO performance is far from being optimal, it outperfoms the TRPO method in MDPs with sparse reward, a known advantage of hierarchical methods.

We also plot the exploited options during the learning process:

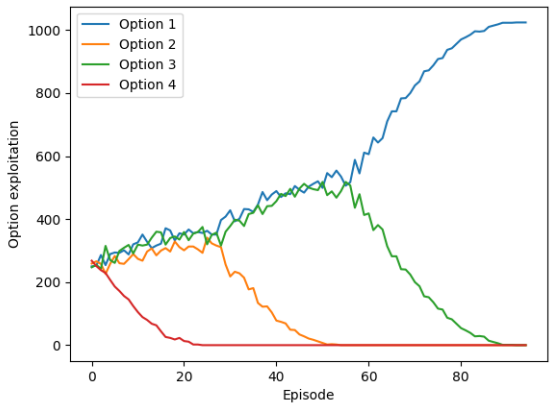


Figure 2.2: TRHPO exploited options

Conclusion

Despite outperforming the simple TRPO, the TRHPO gradually excludes option policies to only exploit one option policy around the 90th episode. At the end of the first 100th episodes, the algorithm reaches a local optima. In our test, the convergence to local optima limits the TRHPO performance in a range of environment, especially where the reward isn't sparse.

After building the hierarchical policy learning method, we aim next to incorporate options that can be learned independently of the top level policies. We aim to achieve such goal by generalizing the Proto-Value Function (PVF) concept to solve the MDP on specific basis functions.

Chapter 3

Spectral Framework for options discovery

Abstract

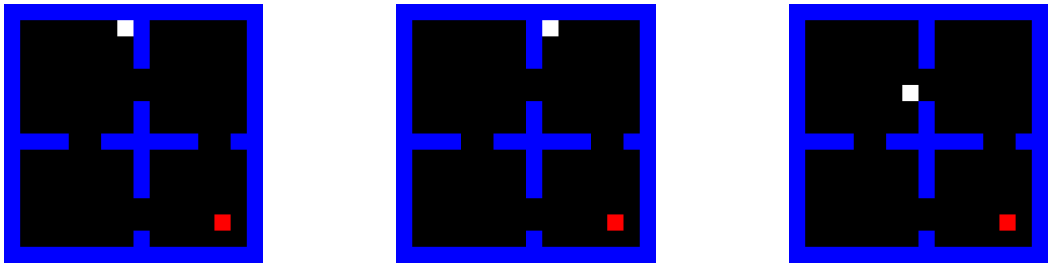
After working on the temporal action abstractions, we tackle the problem of state space abstraction to learn options on basis functions induced from the graph of the environment. We use the spectral basis functions to learn representation that reflect the geometric dynamics of the environment.

Introduction

Another way to learn value functions or policies is to approximate them on a certain basis functions. This approach has several applications in Machine Learning[27] (Fourier Wavelets, Laplacian) and Quantum Physics (Hamiltonian)[28] and yields good results when the right basis is used. The most recent one perhaps is the the 100% accuracy on MNIST using complex neural networks built on the frequency domain features [29].

In the context of RL, the concept of PVF introduced by Mahadevan [30] uses the eigenvectors of the graph Laplacian as basis to learn the optimal policy for a discrete state space MDP. The idea is to exploit the geometry of the environment since spatially close states can be "dynamically" far or even separate. In the 4 room environment in the examples, $state_1$ is visually closer to $state_2$ than $state_3$, but dynamically further than it.

In this chapter, we will introduce PVF paradigm, present the Laplacian operator and its use in Spectral Clustering before integrating it in our study of RL. Since we are inter-



(a) $state_1$ (b) $state_2$ (c) $state_3$

Figure 3.1: Euclidian distance and geodesic distance

ested in applications for high diemnsional state space, we will present and evaluate our model for scaling the introduced techniques.

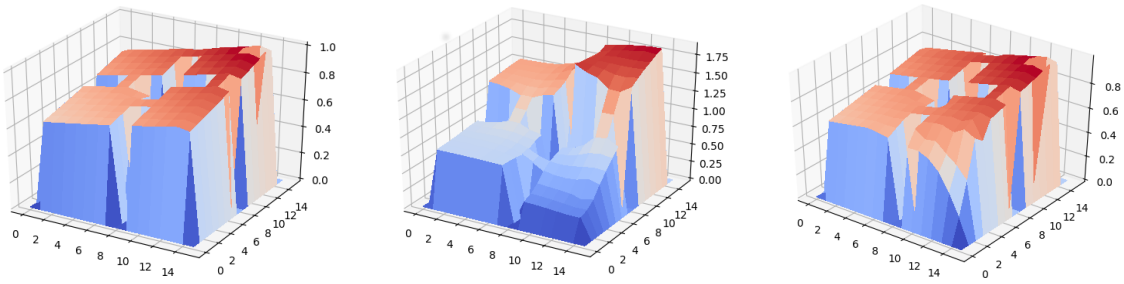
1 Proto-Value Functions and Spectral Clustering

The PVF[30] introduced a spectral framework for approximate dynamic programming. It mainly addressed a finite state space MDP setting in model-free RL. The overall framework can be summarised as follows : the agent constructs the adjacency matrix of the space state graph where the similarity $W_{i,j} = 1$ if the agent could move from state s_i to state s_j. The diffusion model is defined using the normalised symmetric graph Laplacian L_{sym}. Then the basis functions are simply the smoothest eigenvectors of the Laplacian (associated with the lowest eigenvalues).To learn the optimal policy, PVF uses "Representation Policy Iteration", a policy iteration that takes the projection on the smoothest eigenvectors as states.

To illustrate the concept, we use 4-room environment with size 16×16 (177 accessible states), we consider the fixed starting point with absorbing goal state. We have 176 states, for which we plot in (3.2) the optimal value function $V*$ and its approximation on the first 5 and the full Laplacian basis, learned using Least Square Policy Iteration (LSPI).
Contrary to what we might conclude visually from (3.2b), using the smoothest 5 eigenvectors doesn't solve the problem. In fact, the approximated policy has absorbing states different from the target state. Solving this issue requires using more eigenvectors which in its turn creates new modes in the approximated function, hence the need for a sufficiently large basis to nullify the discontinuity. Several works address this problem by using smoother basis as in Sugiyama et al. where the geodesic Gaussian kernels where the similarities are built using the shortest path distance.

In an entirely different perspective, with the concept of "Eigenoptions", Machado et al. learns options that maximise intrinsic rewards induced by the eigenvectors. This allows to create options that reflect the geometric properties translated by each eigen-

vector separately. Afterwards, a policy over the Eigenoptions is used to associate each eigenoption to a certain domain of the state space, in a similar way to the hierarchical RL.



(a) Optimal V^* (b) Using 5 eigenvectors (c) using 177 eigenvectors

Figure 3.2: Optimal V and approximations

In the next section, we go through basic properties of the graph Laplacian that we will use later to introduce eigenoptions.

1.1 Laplacian Operator and Spectral Clustering

Spectral clustering[33] addresses the problem of partitioning an undirected weighted graph $G = (V, E)$ under specific constraints, where V represents the set of vertices and E the set of edges.

Let $V = \{v_1, ..., v_n\}$, and W the $n \times n$ adjacency matrix where $W_{i,j}$ the non-negative weight of the edge linking v_i and v_j, and $W_{i,j} = 0$ means that the vertices v_i and v_j are not linked. The undirected nature of the G means that W is symmetric. We define the degree of the vertex v_i as $d_i = \sum_j W_{i,j}$ and the degree matrix $D = \text{diag}\,(d_1, ..., d_n)$.

Let $A \subset V$, we define the indicator vector $1_A = (f_1, ...f_n)$ as : $f_i = 1_{v_i \in A}$. Similarly, we can measure the similarity between two subsets A and B by the sum of the "similarity" between their vertices:

$$W(A, B) := \sum_{i \in A, j \in B} w_{i,j} = 1_A^T W 1_B \,,$$

and to measure the size of the set A, we can either use:

- The cardinality measure :
$$\text{vol}_c(A) := |A| = 1_A^T 1_A$$
.

- Total of its elements degrees:

$$\text{vol}_\text{d}(A) := \sum_{i \in A} d_i = 1_A^T D 1_A$$

.

Graph cut and Laplacians

To partition the graph G into two separate sets A and $\overline{A}$, Spectral Clustering searches for the least "similar" sets A and B. We can use two types of similarity measures:

$$\text{Ratiocut}(A, \overline{A}) := \frac{W(A, \overline{A})}{|A|} + \frac{W(\overline{A}, A)}{|\overline{A}|} \tag{3.1}$$

$$\text{Ncut}(A, \overline{A}) := \frac{W(A, \overline{A})}{\text{vol}_\text{d}(A)} + \frac{W(\overline{A}, A)}{\text{vol}_\text{d}(\overline{A})} \tag{3.2}$$

We can prove the following properties:

$$\frac{W(A, \overline{A})}{|A|} = \frac{{\mathbb{1}_A}^T (D - W) \mathbb{1}_A}{{\mathbb{1}_A}^T \mathbb{1}_A} \tag{3.3}$$

$$\frac{W(A, \overline{A})}{\text{vol}_\text{d}(A)} = \frac{{\mathbb{1}_A}^T (D - W) \mathbb{1}_A}{{\mathbb{1}_A}^T D \mathbb{1}_A} \tag{3.4}$$

Since we're interested in finding the set A,different from V, the partitioning problem is then formulated as finding the vector of $\mathbb{R}^n$ with $\{0, 1\}$ elements that minimizes the Rayleigh quotient Equation 3.3 or Equation 3.4.

Graph Laplacians

Unnormalized/Combinatorial Laplacian: The combinatorial Laplacian, appearing in the Ratiocut, is defined as $L := D - W$, which has the following properties:

- for f in $\mathbb{R}^n$

$$f^T L f = \sum w_{i,j} (f_i - f_j)^2$$

- L is symmetric semi definite
- The smallest eigenvector is 1 for the eigen value $\lambda_0 = 0$
- In particular, self-edges in a graph do not change the corresponding graph Laplacian

Normalized Laplacians: Spectral Clustering literature uses two types of normalized Laplacians:

- Random Walk $L_{rw} = I - D^{-1}W$
- Symmetric, appearing in the Ncut, $L_{sym} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$

The matrix $T = D^{-1}W$ can be seen a transition matrix. Although L_{rw} is not symmetric, all its eigenvalues are real, non-negative, and are upper-bounded by 1. The normalized Laplacians have the following properties:

- For $f \in \mathbb{R}^n$:

$$f^T L_{sym} f = \frac{1}{2} \sum_{i,j} (\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}})^2 \tag{3.5}$$

- u is an eigenvector of L_rw with eigen value λ iif $W = D^{\frac{1}{2}}u$ is an eigenvector of L_sym for the same eigen value.

Cuts optimization

Solving Equation 3.3 and Equation 3.4 on the set $\{0,1\}^n$ in an NP-hard problem, we relax the constraint so that the unknown f is dense with specific norm. Minimizing the Ratiocut amounts to finding the minimizer f of the Rayleigh quotient for the matrix L with $f^T f = \sqrt{n}I$ and $f^T \mathbb{1}_V = 0$.

For the Ncut, the target is $g = D^{\frac{1}{2}}\mathbb{1}_A$, the minimizer of the Rayleigh quotient for L_{sym} such that $g^T g = \sqrt{n}I$ and $g^T D^{\frac{1}{2}} \mathbb{1}_V = 0$. The minimizer of the Rayleigh quotient under the mentioned constraints is the smoothest eigenvector of the matrices (excluding the first eigenvector).

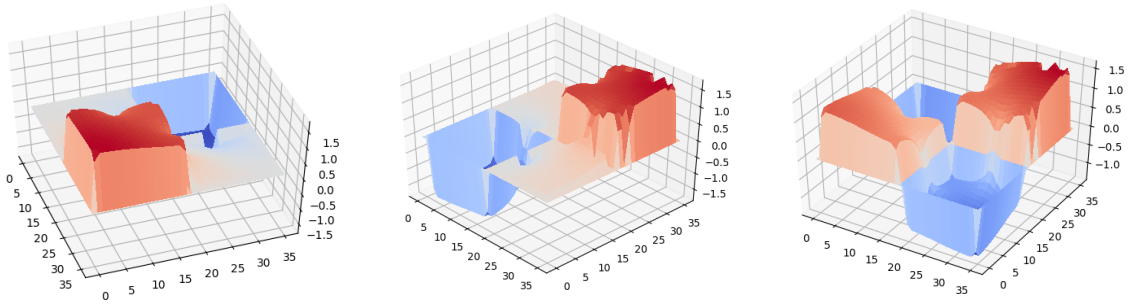
Consequently, vertices in the same subset are expected to have similar projections on the relaxed problem solutions. As we go higher in the Laplacian spectrum, the projection on the eigenvectors tend to represent more specific (higher frequency) differences. Knowing the full spectrum is hence supposed to provide representation that separates all the vertices.

2 Eigenoption Discovery

Eigenoptions[32] are based on a simple property of the eigenvectors. Plotting the first 3 eigenvectors below, the agent starts at position $(3, 3)$ and has to reach $(33, 33)$. Let ϕ_i be the $i - th$ eigenvector, and we define the intrinsic reward r_i for taking action a to transition $s \rightarrow'$

$$r_i(s, a) = \phi_i(s') - \phi_i(s)$$

The i-th Eigenoption is the policy that maximizes the expected accumulated intrinsic reward r_i. In Figure 3.3, the first Eigenoption would lead the agent to the corner of the first room an and exit the third room. For the third Eigenoption, the agent would have to exit the second and fourth room to reach the absorbing state in the corners of the first and third room. So if the task is to exit the first room, we should follow the second eigenoption. Extending the Eigenoption concept to large and continuous state space



(a) 1^{st} eigenvector (b) 2^{nd} eigenvector (c) 3^{rd} eigenvector

Figure 3.3: Eigenvectors of L_{sym}

MDP requires to be able to learn eigen vectors of large, and virtually infinite, graph Laplacian. In this section we go through two recent methods and in the last section we present our own.

2.1 Optimization on Matrix Manifold

For problems involving orthogonality or low-rank constraints, these characteristics are expressed naturally using the Grassmann manifold. There have been growing interests in studying Grassmann manifold and exploiting their structure, especially in computer vision. However, the manifold structure was usually used to model either the input or the parameters of parametric functions. It is only recently that we've started to see works where neural networks are used to output representation on the manifold. More specifically, given a batch sample $(X_i)_{i \leq n}$ in $\mathbb{R}^d$, we want to learn a mapping

$F : \mathbb{R}^d \to \mathbb{R}^k$ that solves:

$$\begin{aligned} \underset{F}{\text{minimize}} \quad & \mathcal{L}((X_i)_i, F) = \sum_{i,j \leq n} K(X_i, X_j) F(X_i)^T F(X_j) \\ \text{subject to} \quad & \frac{1}{n} \sum_i F(X_i)^T F(X_i) = I_k, \end{aligned} \tag{3.6}$$

with K a similarity kernel on $\mathbb{R}^d \times \mathbb{R}^d$.

In other words, the $n \times k$ matrix Y with rows $Y_i = \frac{1}{\sqrt{n}} F(X_i)$ should be on the Stiefel manifold defined as :

$$V_{n,k} := \{A \in \mathbb{R}^n \times \mathbb{R}^k | A^T A = I_k\}$$

However, the problem doesn't have a unique solution. Given an optimal mapping F and taking a matrix $Q \in \mathcal{O}_k$ to define $F'(X) = F(X)^T Q$, then F' is also optimal.

To correctly define the problem, the optimization set should identify element of $V_{n,k}$ whose columns span the same subspace: the Grassmann manifold $G_{n,k}$.

When n is large, evaluating whether F's outputs is on the Grassmann manifold can be only approximated. Informally, if the input X has distribution P, then F is simply transforming X into latent representation such that:

$$\begin{aligned} \underset{F}{\text{minimize}} \quad & \mathbb{E}^P[K(X, X') F(X)^T F(X')] \\ \text{subject to} \quad & \mathbb{E}^P[F(X)^T F(X)] = I_k, \end{aligned} \tag{3.7}$$

To evaluate if the output of batch of size n is on the manifold, we use the distance:

$$d_g(A) = \underset{B \in G_{n,p}}{inf} ||A - B||_2^2$$

Lemma 3.1. *For a matrix $A \in \mathbb{R}^n \times \mathbb{R}^k$:*

$$d_g(A) = \sum_{i=1}^{k} (\sigma_i - 1)^2 \tag{3.8}$$

where $S = \text{diag}(\sigma_1, ...\sigma_k)$ such that $A = USV^T$ is the SVD decomposition of A.

Consequently, the euclidian orthogonal projection of A on Grassmann manifold is $B = UV^T$.

2.2 Neural Network for Eigenvectors learning

Spectral Net

In [34], Shaham et al. used neural networks to learn an approximation of Laplacian eigenvectors for large scale applications. Their 'SpectralNet' $f_\theta : \mathbb{R}^n \to \mathbb{R}_k$, tries to

minimize the "spectral loss" defined as:

$$L_{SpectralNet}(\theta, (x_i)_i) = \frac{1}{m^2} \sum_{i,j=1}^{m} W_{i,j} \left\| f_\theta(x_i) - f_\theta(x_j) \right\|^2,$$

under the constraint :

$$\frac{1}{m} \sum_{i,j=1} f_\theta(x_i)^T f_\theta(x_j) = I_k.$$

The minimization is done with batches and m is the size of the batch. To impose the orthogonality constraint, a "Cholesky" decomposition layer is added on top of the neural network.

Cholesky Layer: Let Y be the neural networks output for a sample X of size m. If we take the Cholesky decompositon L of $S := \frac{1}{m} Y^T Y = LL^T$, where L is a lower triangular matrix, then $Y^* = \sqrt{m} Y L^{-1}$ is on the Grassmann manifold.

The paper's main contribution however is establishing establishes a theoretical result for the minimum number of units required to be able to learn the eigenvectors.

Lemma 3.2.

$$VCdim(F_n^{spectralclustering}) \geq \frac{n}{10}. \tag{3.9}$$

This implies that to learn representation separating n points, the number of weights in the neural net should be in the same order as the number of points n.

They've also demonstrated the convergence to real eigenvectors for some cases. On the other hand, the methods used for learning is based on simple back-propagation. At each iteration, the Cholesky layer weights are freezed to minimize the Spectral loss. Since the minimizer of such loss is the zero matrix, the output approaches zero while the Cholesky layer weight, the inverse of L for Y approaching 0, can explode if the learning rate isn't carefully controlled. Furthermore, the SpectralNet methods doesn't address the overlapping of eigenvectors, hence, the representation learned is in fact a noisy combination of the smoothest eigen vector.

Spectral Inference Networks

Inspired by applications for the Hamiltonian in quantum mechanics, Pfau et al. addresses the general problem of maximizing the Rayleigh quotient for an "infinite" matrix whose elements are defined by a certain similarity kernel.

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \text{Tr}((Y^T Y)^{-1} Y^T A Y) \\ \text{subject to} \quad & Y^T Y = I_k, \end{aligned} \tag{3.10}$$

where $A \in \mathbb{R}^{n,n}$ and $Y \in \mathbb{R}^{n,k}$. Given a input data $(x_i)_{i\leq n}$, $(y_i)_{i\leq n}$ their output using Spectral Inference Network (SpIn), and A their similarity matrix, we define Π and Σ in $\mathbb{R}^{k,k}$as:

$$\Pi := \frac{1}{n}\sum_{i,j} a_{i,j} x_i {x_j}^T \tag{3.11}$$

$$\Sigma := \frac{1}{n}\sum_{i,i} x_i {x_i}^T \tag{3.12}$$

Using these notations,Σ and Π depend on the network parameter θ, and the gradient of the objective can be written as :

$$\text{Tr}(\Sigma \nabla_\theta \Pi) - \text{Tr}(\Sigma^{-1}\Pi\Sigma^{-1}\nabla_\theta\Sigma)$$

To solve the objective, Pfau et al. proceeds by accumulating an empirical estimate of Σ and $\nabla_\theta\Sigma$, which requires calculating k^2 gradients and storing k^2 times the number of network parameters. Based on the theoretical result of SpectralNet, approximating eigenvectors requires a large neural network. Therefore, calculating and storing $\nabla_\theta\Sigma$ is expensive.

On the other hand, SpIn addresses the question of overlapping eigenvectors, and modifies the gradient in a way to make the update sequentially independent : first eigenvectors gradient are independent of gradient coming from eigenvectors of higher eigenvalues. In our implementation of the SpIn, modifying the gradient reduces the update step and hence render the objective increments extremely slow. Furthermore, we believe that the sparsity constraints used in their neural network and the separation of weights of eigenvectors has an important impact on the performance demonstrated in their paper, which was considered a technical detail in their work.

2.3 Proposed Spectral Network

We build on the SpectralNet and SpIn to derive a new method that produces separated eigenvectors, with fewer gradient backprobagations and memory requirements. Using the same notation in the previous subsection, we aim to minimize the "Sequential" Rayleigh quotient. Let

$$R_j(A, Y) = \text{Tr}((Y_j^T Y_j)^{-1} Y_j^T A_{:j,:j} Y_j)$$

Where Y_j is the matrix of the first j vectors of Y and $A_{:j,:j}$ the first $j \times j$ block of matrix A. Then our objective is:

$$\begin{aligned} \underset{Y}{\text{minimizer}} \quad & \sum_j R_j(A, Y) \\ \text{subject to} \quad & Y^T Y = I_k, \end{aligned} \tag{3.13}$$

If the eigenvalues of A are distinct, then the columns of global optimum is guaranteed to be orthogonal.

From this point, The key idea is to take update directions for which the gradient $\nabla_\theta \Sigma$ is small and update the network parameters to minimize the Grassmann distance. This is similar in a way to the sub-gradient method. However, the set on which we incrementally project (Grassmann Manifold) is not convex in our case.

Furthermore, we use large samples to estimate the sequential Rayleigh quotient. Alternatively, we can use the learned covariance matrix as in the SpIn.

At iteration t, where the parameter of the network is θ_t:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & \sum_j R_j(A, Y_\theta) \\ \text{subject to} \quad & ||Y_\theta^T Y_\theta - Y_{\theta_t}^T Y_{\theta_t}||_2^2 = \delta. \end{aligned} \tag{3.14}$$

We note that $||Y_\theta^T Y_\theta - Y_{\theta_t}^T Y_{\theta_t}||_2^2$ is approximately quadratic in θ for θ sufficiently close to θ_t . For a small δ, the update step insures to stay approximately on the same Stiefel manifold as the covariance doesn't changes. After each step minimizing the objective, we take an orthogonal update towards the Grassmann manifold under the constraint of keeping the gained improvement. The learned covariance matrix is used to update the weight of output Cholesky layer. The update is done gradually using a learning step instead of the brutal update as in SpectralNet. Ultimately, the cholesky layer allows the network to output the centered eigenvectors based on the learned covariance matrix Σ.

Algorithm 5 Spectral Network

Require: A parameterized network F_θ, Learning rate α for Σ, Learning rate $\beta < 1$ for gradient updates
for $i \leftarrow 1$ to T **do**
Sample $(x_i)_i$ and get the output $(y_i)_i$
Calculate $\hat{\Sigma}$ and update the Cholesky layer weights with rate α
Get gradient of the Sequential Rayleigh quotient
Use quadratic approximation of the constraint, calculate fisher information matrix
Use Conjugate Gradient to find a search direction g
Get the Grassmann distance gradient p and project it on the normal on g
Use Armijo line search using g
Use a simple line search using p while keeping approximately the gained improvement from previous step
If Rayleigh Quotient doesn't improve after 10 iterations, reduce update rate
end for

3 Evaluating the Spectral Network

We use our network to learn eigenvectors of the combinatorial Laplacian for the 4 rooms environment of size 36. Since the eigenvector associated with the first eigenvalue is the vector 1, we can fix the first component of the output.

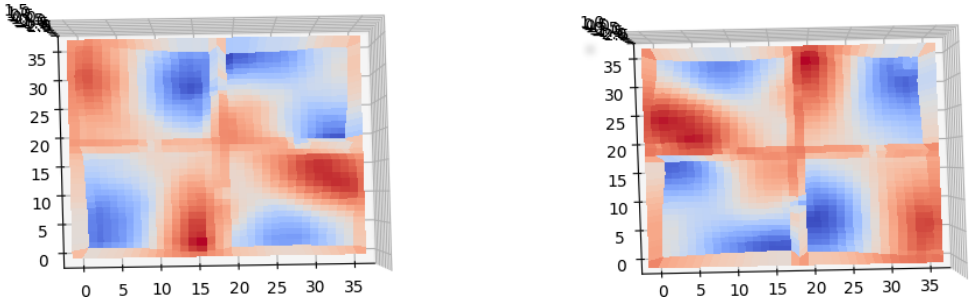
3.1 Eigenvectors of the 4-rooms environment

Two states are similar if the agent transitions from one to another. We consider the grid with size 36 with random locations for the agent and the target point, which sums up to 1202312 possible states. We use the combinatorial Laplacian in our experiment, since all positions have nearly the same degrees, except position near the walls. furthermore, Using normalized Laplacians requires estimating the degree of vertices, which is prohibitively difficult in batch Learning.
To avoid sampling the same state multiple times, we use a graph builder that hashes the states and identifies states using the hash dictionary.

We visualize on the finite set of states for the deterministic 4 room environment. The eigenvectors, however, has been learned for the stochastic 4 room environment, which has a space of over 10^6 states. We display in the figures below the 3rd and the 5th learned eigenvectors for the deterministic environment and a second environment which is a 180 degrees rotation of the initial one.

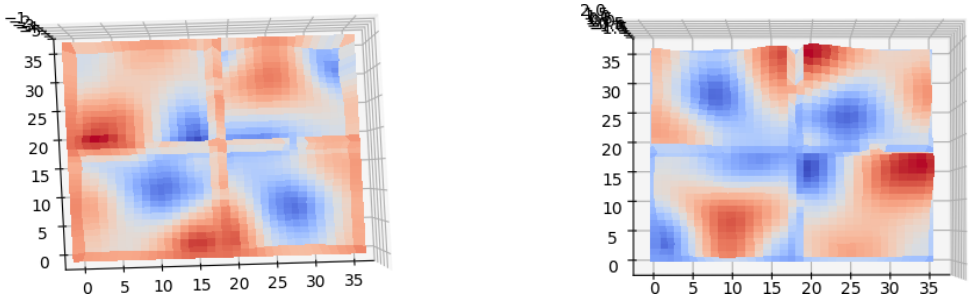
The obtained functions display similar invariance characteristics with spectral eigenvectors. More importantly, learning eigenoptions using these "eigenfunctions" will in-

sure that the learned options will lead to the same relative states. In our work, we attempted to use the TRPO model used in the first 2 chapters to learn eigenoptions, but without success. We believe it is due to the necessary large size of the network.



(a) Initial Environment (b) Rotated Environment

Figure 3.4: 3rd Eigenvector



(a) Initial Environment (b) Rotated Environment

Figure 3.5: 5th Eigenvector

3.2 Clustering

We also compare our method's performance with the baseline in SpectralNet, which uses the spectral network without learning Siamese similarities or using the variational embedding of the dataset.

We define similarities between data points using nearest neighbors for $n = 5$. To reduce the similarity variance between samples, we store the learned distances and keep the closest 1024 neighbors from each point. This allows to stabilize the similarities between data points once enough points are browsed.
Using this method, it is unclear what number of eigenvectors to use, since there is no guarantee to have a fully connected graph or to avoid creating unnecessary cliques.

MNIST For MNIST, we report have the following performance:

Number of eigenvectors	Accuracy	SpectralNet Score
10	69.27%	62.3%
11	71.08%	-
15	74.17%	-
20	67.85%	-

Reuters For Reuters dataset, we learn the first 4 eigenvectors. train on 6.10^5 data points and evaluate on a testing set of size 10^4, we report a test accuracy of 76.18% after 72 iterations and drops around 69% afterwards. The training accuracy stabilizes around 67.24%. Since the training set is large, the reported training accuracy is evaluated between distant iterations. This is to be compared with 64.5% accuracy achieved by the SpectralNet. Hence, we believe that using embedding or Siamese networks as was done by Shaham et al. would outperform the SpectralNet.

Conclusion

After building the HRL model, we attempted to learn options directly without going through the full HRL method. We used the PVF concept and we've built on recent work to approximate eigenvectors using neural networks for large datasets. We've demonstrated that the learned functions present similar invariance characteristics as the Spectral eigenvectors and showed that our method outperforms the recent SpectralNet in the clustering task when used without Siamese distances nor the variational embedding.

On the other hand, we encountered difficulties when learning eigenoptions using TRPO method. Therefore, we envision to continue improving and building on the current achievements and to extend it to the standard use of eigenfucntions : Successor Features estimation.

Conclusion

Starting with the goal of building improved Reinforcement Learning methods, we showed that clustering the states space is an efficient strategy to master complex and large states spaces. First, using action abstraction by building the Hierarchical TRPO method. On the second part, we build on the Eigenoption method, which can be seen as a state abstraction, to learn embedding of the state space incorporating the knowledge about environment dynamics.

We perform extensive tests for our method for learning the eigenvectors for clustering and for Eigenoptions estimation. Our next step was supposed to combine the Hierarchical model with Eigenoptions. As we have demonstrated, learning eigenfunctions requires large neural networks. Similarly, when learning an eigenoption, the option policy should be as rich as the spectral network to have a good approximation of the policy. This has hindered our testing of the HTRPO and limited our reported results for the RL task.

To tackle this problem, the model can be improved by incorporating more sophisticated parameterization as the sparse weight block used in [35]. In the immediate term, we will pursue more extensive tests of the developed model with more theoretical examination of the optimization heuristic.

Acronyms

A3C Asynchronous Actor-Critic Agents. 9, 11

CPI Conservative Policy Iteration. 11

DP Dynamic Programming. 5, 9

h-DQN hierarchical Deep Q Learning. 19

HREPS Hierarchical Relative Entropy Search. 17, 19

HRL Hierarchical Reinforcement Learning. 14–16

LSPI Least Square Policy Iteration. 24

MDP Markov Decision Process. 3, 5, 7, 9, 16, 23, 28

MI Mutual Information. 18

PVF Proto-Value Function. 22–24, 35

REPS Relative Entropy Search. 9, 10, 19

RIM Regularised Information Maximization. 18

RL Reinforcement Learning. 2, 3, 5, 8, 9, 16, 23, 24

SpIn Spectral Inference Network. 31

TRPO Trust Region Policy Optimization. 9, 11, 12, 14, 19

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. The MIT Press, 2017. [View].

[2] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proceedings of 19th International Conference on Machine Learning*, pages 267–274, 2002. [View].

[3] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. [View].

[4] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60, 1954. [View].

[5] Stefan Banach. Sur les oprations dans les ensembles abstraits et leur application aux quations intgrales. *Fundamenta Mathematicae*, 3:133–181, 1922. URL `http://eudml.org/doc/213289`.

[6] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. ., 1994. [View].

[7] Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *CoRR*, abs/1705.07798, 2017. [View].

[8] J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence*, pages 1607–1612, 2010. [View].

[9] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. [View].

[10] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. [View].

[11] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. *GitHub repository*, 2017. [View].

[12] Jacob V. Bouvrie. Hierarchical learning: Theory with application in speech and vision. *Massachusetts Institute of Technology*, 2003. [View].

[13] Matthew M. Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3), 2009. [View].

[14] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 2002. [View].

[15] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018. [View].

[16] Richard S.Sutton, Doina Precupb, and Satinderv Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999. [View].

[17] Peter Dayan and Geoffrey Hinton. Feudal reinforcement learning. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 271–278. Morgan-Kaufmann, 1993. [View].

[18] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L. Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. *CoRR*, abs/1301.7381, 1998. [View].

[19] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17(93):1–50, 2016. [View].

[20] Takayuki Osa and Masashi Sugiyama. Hierarchical policy search via return-weighted density estimation. *CoRR*, abs/1711.10173, 2017. [View].

[21] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057, 2016. [View].

[22] Anna Harutyunyan, Peter Vrancx, Pierre-Luc Bacon, Doina Precup, and Ann Nowé. Learning with options that terminate off-policy. *CoRR*, abs/1711.03817, 2017. [View].

[23] Weihua Hu, Takeru Miyato, Seiya Tokui, Eiichi Matsumoto, and Masashi Sugiyama. Learning discrete representations via information maximizing self augmented training. In *ICML*, 2017.

[24] Masashi Sugiyama, Makoto Yamada, Manabu Kimura, and Hirotaka Hachiya. Information-maximization clustering based on squared-loss mutual information. In *Neural Computation*, 2011.

[25] Daniele Calandriello, Gang Niu, and Masashi Sugiyama. Semi-supervised information-maximization clustering. *CoRR*, 2013. [View].

[26] Maciej Klimek Piotr Mio Henryk Michalewski. Hierarchical reinforcement learning with parameters. *MLR*, 2017. [View].

[27] Stphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, Inc., 3rd edition, 2008. [View].

[28] Jean-Louis Basdevant, Jean Dalibard, and Manuel Joffre. *Quantum Mechanics*. Ecole Polytechnique, 2006.

[29] Igor Aizenberg and Alexander Gonzalez. Image recognition using mlmvn and frequency domain features. *Private*, 2018. [View].

[30] Mahadevan Sridhar and Mauro Maggioni. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05. ACM, 2005. doi: 10.1145/1102351.1102421. [View].

[31] Masashi Sugiyama, Hirotaka Hachiya, Christopher Towell, and Sethu Vijayakumar. Geodesic gaussian kernels for value function approximation. *Autonomous Robots*, 2008. [View].

[32] Marlos C. Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. *CoRR*, 2017. [View].

[33] Ulrike von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007. [View].

[34] Uri Shaham, Kelly Stanton, Henry Li, Ronen Basri, Boaz Nadler, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks. In *International Conference on Learning Representations*, 2018. [View].

[35] David Pfau, Stig Petersen, Ashish Agarwal, David Barrett, and Kim Stachenfeld. Spectral inference networks: Unifying spectral methods with deep learning. *CoRR*, 2018. [View].

TRPO Implementation Parameters

Chapter 2 proofs

Lemma .3. *Let $n \geq 1$ be the number of option policies, and π_g the gating policy (distribution over the options set). If π and $\tilde{\pi}$ are two hierarchical policies such that $\tilde{\pi}$ is absolutely continuous w.r.t π, then at any state s, we have*

$$D_{KL}\left[\tilde{\pi}(.|s)||\pi(.|s)\right] \leq D_{KL}\left[\tilde{\pi}_g(.|s)||\pi_g(.|s)\right] + \sum_{k=1}^{n} \pi_g(o|s) D_{KL}\left[\tilde{\pi}(.|s,o))||\pi(.|s,o)\right] \tag{15}$$

Proof. We fix a state s, and we consider the two distributions p and q:

$$\begin{aligned} p(a,o) &:= \pi(a,o|s) = \pi_g(o|s)\pi(a|s,o), \\ q(a,o) &:= \tilde{\pi}(a,o|s) = \tilde{\pi}_g(o|s)\tilde{\pi}(a|s,o) \end{aligned} \tag{16}$$

and we denote p_A and q_A, p_O and q_O, respectively, the marginal distributions.

Using the conditional KL-divergence, we have th following two identities

$$\mathrm{D_{KL}}\left[q||p\right] = \mathrm{D_{KL}}\left[q_A||p_A\right] + \sum_a q_A(a)\mathrm{D_{KL}}\left[q(.|a)||p(.|a)\right] \tag{17}$$

$$\mathrm{D_{KL}}\left[q||p\right] = \mathrm{D_{KL}}\left[q_O||p_O\right] + \sum_o q_O(o)\mathrm{D_{KL}}\left[q(.|o)||p(.|o)\right] \tag{18}$$

Substracting Equation 18 from Equation 17, we obtain:

$$\mathrm{D_{KL}}\left[q_A||p_A\right] = \mathrm{D_{KL}}\left[q_O||p_O\right] + \sum_o q_O(o)\mathrm{D_{KL}}\left[q(.|o)||p(.|o)\right] - \sum_a q_A(a)\mathrm{D_{KL}}\left[q(.|a)||p(.|a)\right] \tag{19}$$

Since $\mathrm{D_{KL}}\left[q(.|a)||p(.|a)\right] \geq 0$for all $a \in \mathcal{A}$, we have:

$$\mathrm{D_{KL}}\left[q_A||p_A\right] \leq \mathrm{D_{KL}}\left[q_O||p_O\right] + \sum_o q_O(o)\mathrm{D_{KL}}\left[q(.|o)||p(.|o)\right] \tag{20}$$

All is left is to notice that :

$$\begin{aligned} &p_A = \pi(.|s),\ q_A = \tilde{\pi}(.|s) \\ &p_o = \pi_g(.|s),\ q_o = \tilde{\pi}_g(.|s) \\ &p(.|o) = \pi(.|s,o),\ q(.|o) = \tilde{\pi}(.|s,o). \end{aligned}$$

Hence :

$$\mathrm{D_{KL}}\left[\tilde{\pi}(.|s)||\pi(.|s)\right] \leq \mathrm{D_{KL}}\left[\tilde{\pi}_g(.|s)||\pi_g(.|s)\right] + \sum_o \tilde{\pi}_g(o|s)\mathrm{D_{KL}}\left[\tilde{\pi}(.|s,o)||\pi(.|s,o)\right] \tag{21}$$

□