# Automatic Goal-Conditioned Reinforcement Learning using Learned Distances and Automatic Curriculum Generation

Srinivas Venkattaramanujam

Computer Science
McGill University, Montreal

November 30, 2020

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Science.

## Acknowledgements

Many thanks to Doina for her continuous support, guidance and encouragement. She gave me the freedom to set my goals and provided me with valuable insights whenever I was stuck. I am grateful for the opportunity she has provided me with. My sincere thanks to my collaborators Eric, Thang and Riashat. I have learned a lot from each of them. Special thanks to Maxime Wabartha for his kind help in translating the abstract. I am very grateful to my parents, sister and friends for their continuous encouragement and support.

# Abstract

Representation learning for reinforcement learning is still an open problem. Prior methods have developed a variety of objectives for representation learning satisfying certain desired properties. In this thesis, we propose an approach to learn embeddings that is suitable for goal-conditioned reinforcement learning. We learn an embedding space with the property that the distance between the states in the embedding space is proportional to square root of the average commute time distance between those states under a fixed policy. Fouss, Pirotte, and Saerens, 2005 show the existence of this embedding space and its connection to the unnormalized Laplacian. Our approach is motivated as providing an approximation to this embedding space using metric Multi-Dimensional Scaling (MDS). Our experiments demonstrate that this learned embedding space can be used learn a task specific distance function required for goal-conditioned policies to determine whether the specified goal has been achieved. A goal is considered achieved when the agent is within an $\epsilon$-ball centered on the goal. As a secondary contribution, we propose an approach to automatically generate a curriculum of goals without using domain knowledge. The two proposed methods, used in conjunction, remove almost all of the domain knowledge required to design goal-conditioned reinforcement learning agents that learn useful behavior on their own.

# Résumé

L'apprentissage de représentations pour l'apprentissage par renforcement est aujourd'hui une question ouverte. Certaines méthodes ont développé une variété d'objectifs pour l'apprentissage de représentations satisfaisant certaines propriétés désirées. Dans cette thèse, nous proposons une approche visant à apprendre des structures utilisables par l'apprentissage par renforcement conditionné par un but. Nous apprenons un espace de structures ayant la propriété que la distance entre les états de l'espace de structures est proportionnelle à la racine carrée du temps de trajet moyen entre ces états, en suivant une stratégie fixe. Fouss, Pirotte, and Saerens, 2005 montre l'existence de cet espace de structures et sa connexion au Laplace non normalisé. Notre approche est motivée par l'approximation de cet espace de structures grâce à la métrique MDS. Nos expériences démontrent que cet espace de structures appris peut être utilisé pour apprendre une distance spécifique pour chaque tâche, ce qui est requis par les stratégies conditionnées par un but, afin de déterminer si le but spécifique a été atteint. Un but est considéré atteint quand l'agent se situe dans une boule-epsilon centrée sur le but. Nous proposons également comme contribution secondaire une approche pour automatiquement générer un curriculum de buts sans utiliser de connaissances spécifiques au domaine d'apprentissage. Les deux méthodes proposées utilisées ensemble permettent de se passer de presque l'entièreté des connaissances spécifiques au domaine d'apprentissage qui sont requises pour concevoir des agents d'apprentissage par renforcement conditionné par un but qui apprennent d'eux-mêmes des comportements utiles.

## Contribution of Authors

- Chapters 1-5 provide the requisite background and is new material written for this thesis.
- Chapters 6-9 are joint work with my co-authors Eric Crawford, Thang Doan and Doina Precup (Venkattaramanujam, Crawford, et al., 2019). I was the main contributor to both the algorithmic ideas and the implementation of the code for the experiments.
- The buffer proposed in section 8.3 is based on the workshop paper (Venkattaramanujam, Islam, and Precup, 2019).
- The code for our experiments is based on the open-source code for (Florensa, Held, Wulfmeier, et al., 2017) and (Florensa, Held, Geng, et al., 2018).

# Contents

# 1

# Introduction

Deep Reinforcement Learning (DeepRL) has seen tremendous success in a variety of tasks such as playing Atari games (Mnih et al., 2015), solving the games of Go and Chess (Silver, Huang, et al., 2016; Silver, Hubert, et al., 2018) and robotic control (Lillicrap et al., 2016; Schulman et al., 2015). Despite these successes, DeepRL is far from solved and it is an active area of research. In this thesis we concern ourselves with the problems arising in the sparse reward setting. There exists no formal definition for what constitutes a sparse reward setting but intuitively it can be understood as the situation where the agent is provided with a feedback only after a long sequence of actions and most sequences lead to the same feedback. Sparse reward problems are typically solved by augmenting the original task with an auxiliary task or by breaking down the original task into simpler tasks. The former approach is known as reward shaping and the latter approach is known as options (Sutton, Precup, and Singh, 1999).

Reward shaping is not a straightforward approach and has been shown to cause spurious behavior in the original task. The foremost reason to use sparse rewards in the original task is because designing a dense reward function is tedious. Reward shaping falls into the same trap. Ng, Harada, and Russell (1999) proved that the optimal policy for the original task remains invariant under augmentation only for a specific class of shaped reward functions which they called potential-based shaping functions. Potential-based shaping functions are based on the state sequence and are agnostic to the actions taken in the states. It is therefore intuitive that they do not modify the optimal policy. Despite its theoretical restrictions, reward shaping has been empirically shown to aid exploration in sparse reward problems (Bellemare et al., 2016; Burda, Edwards, Pathak, et al., 2019; Burda, Edwards, Storkey, et al., 2019). A related application of augmented reward functions is to aid representation learning (Jaderberg et al., 2017).

The reinforcement learning task can be alternatively characterized in terms of achieving goals (Kaelbling, 1993). This characterization is an instance of a hierarchical reinforcement learning (Sutton, Precup, and Singh, 1999) and enjoys the benefit of being compatible with curriculum learning - the technique of solving a task by solving a sequence sub-tasks of progressively increasing difficulty. Curriculum learning was introduced in the supervised-learning setting using hand-designed curriculum (Bengio et al., 2009). Hand-designing a curriculum is cumbersome and approaches to automate this process have been proposed, for example by modelling task sequencing as a multi-armed bandit in the supervised-learning setting (Graves et al., 2017) and as an MDP in reinforcement learning (Narvekar, Sinapov, and Stone, 2017). Florensa, Held, Geng, et al. (2018) proposes an approach that uses a generative adversarial network (Goodfellow et al., 2014) to generate goals for reinforcement learning agents. This approach was shown to be better than alternative approaches proposed in (Sukhbaatar et al., 2018) and (Baranes and Oudeyer, 2013).

The previous approaches to goal generation in reinforcement learning use prior domain knowledge to design the goal space. For tasks of practical importance, it is nontrivial to hand design the goal space. As a result, several recent works propose different approaches to learn the goal space without using prior knowledge. Nair et al. (2018) learn a goal space using a variational autoencoder (VAE) (Kingma and Welling, 2014). The goal space learned using VAE does not capture the task dynamics; it merely captures similarity in the input space, visual similarity for instance. Nachum et al. (2019) learn a goal space that captures task dynamics and is optimal for a specific assumption on the dynamics model. Closely related to the approach we will propose is Wu, Tucker, and Nachum, 2019, in which the goal space is obtained using the eigenvectors of the Laplacian matrix.

The primary contribution of this thesis is to develop an approach to automatically learn the goal space that captures the task dynamics. The learned embedding space approximates the property that the distance between two states in the embedding space is proportional to the square root of the average commute time between these states in the Markov chain induced by the policy used to collect the samples. This embedding space is guaranteed to exist under some assumptions as shown in (Fouss, Pirotte, and Saerens, 2005). Computing this embedding space requires quantities that are not available in the reinforcement learning setting, such as the matrix of pair-wise distances or the pseudo-inverse of the Laplacian of the state connectivity graph. Hence, we propose an approach to approximate this embedding space. Our experiments show that this embedding space is suitable for goal-conditioned reinforcement learning tasks. We use the learned embeddings to define the $\epsilon$-ball around the goals which is used to determine whether the agent has achieved its goal. The embedding space is trained online, in conjunction with the training of the goal-conditioned policy.

As a secondary contribution, we propose an approach to generate goals by applying random actions at the end of the episode to collect a set of new candidate goals. We show that this goal generation approach is on par with (Florensa, Held, Geng, et al., 2018). Notably, unlike prior work, our approach does not require any domain knowledge. The two proposed contributions blend together to give an approach to train goal reaching agents that can automatically pick their next goals, in a self-driven and domain-agnostic fashion.

# 2

# Markov Chains

This chapter summarizes the main definitions and results from the theory of Markov chains which are useful for understanding our work. The presentation is based on (Brémaud, 1999)

A sequence of random variables $(X_n)_{n \geq 0}$ with values in a set $\Omega$ is called a discrete-time **stochastic process** with the state space $\Omega$.

A discrete-time stochastic process with finite state space $\Omega$ is called a discrete-time finite Markov Chain with state space $\Omega$ and transition matrix $P$ if for all $n \geq 0$ and for all $(X_0 = x_0, \cdots, X_{n+1} = x_{n+1})$ $s.t$ $\mathbf{P}(X_0 = x_0, \cdots, X_{n+1} = x_{n+1}) > 0$ the Markov property $\mathbf{P}(x_{n+1}|x_0, \cdots, x_n) = \mathbf{P}(x_{n+1}|x_n) = P(x_n, x_{n+1})$ holds. Here, $P(x_n, x_{n+1})$ denotes the entry of matrix $P$ with row $x_n$ and column $x_{n+1}$.

The rows of the transition matrix $P$ represents a distribution over the $x \in \Omega$ and sum to 1. Hence, $P$ is row stochastic. The joint distribution of the Markov Chain is fully specified given the specification of the initial state distribution $d_0$, defined on $\Omega$, which can be represented as a row vector with size equal to $|\Omega|$. Using the Markov property we obtain that for any $\mathbf{P}(x_0, \cdots, x_{n+1}) = \mathbf{P}(x_{n+1}|x_n) \cdots \mathbf{P}(x_1|x_0) d_0(x_0)$, where $d_0(x_0)$ is the element of $d_0$ with column index $x_0$.

Starting from the initial state distribution $d_0$, the state occupation probabilities after one step are given by $d_1 = d_0 P$. Hence, $d_t = d_{t-1} P$ and by expanding the recurrence we obtain $d_t = d_0 P^t$. Note that $P^t$ is the $t$-step transition probability.

**Definition 2.1.** A distribution $d$ over $\Omega$ is called a stationary distribution if and only if $d = dP$.

Hence, for any stationary distribution $d$ we have $d(x) = \sum_{y \in \Omega} d(y)P(y, x), \forall x \in \Omega$. This set of $|\Omega|$ equations is known as **balance equations**.

Once the transition probabilities have been defined, we might want to characterize the behavior of the Markov chain over the long-run. The first time a state $x \in \Omega$ is visited during a rollout of the Markov chain is called the hitting time of $x$. Since the rollout is stochastic, we expect the hitting time to be a random quantity. Formally, the **hitting time** $T_x$ is a random variable defined as $T_x := \min\{n \geq 0 : X_n = x\}$. Similarly, the **first return time** of a state $x \in \Omega$ is random variable defined as $T_x^+ := \min\{n \geq 1 : X_n = x\}$.

We denote by $\mathbf{P}_x(.)$ and $\mathbf{E}_x[.]$ respectively the probability and expectation of an event under the Markov chain starting at $x \in \Omega$, *i.e* the initial distribution is defined as $d(y) = \delta_{y=x}$, where $\delta$ is the indicator function. Let $\rho_{x,y} := \mathbf{P}_x(T_y^+ < \infty)$ denote the probability of reaching $y$ from $x$ where $x, y \in \Omega$. A state $x \in \Omega$ is called **recurrent** if $\rho_x = 1$ and **transient** if $\rho_x < 1$. Intuitively this means that when the chain is started at state $x$, this state is seen an infinite number of times if it is recurrent and only a finite number of times, after which it is never visited again, if it is transient. A finer distinction among the recurrent states is obtained with the following definitions. A state $x$ is called **positive recurrent** if $\mathbf{E}_x[T_x^+] < \infty$. A state $x$ is called **null recurrent** if $\mathbf{E}_x[T_x^+] = \infty$.

We say that two states $x, y \in \Omega$ **communicate** if $\rho_{x,y} > 0$. A **communicating class** is a maximal set of states where all pairs of states communicate. A property that is shared by all the states in a communicating class is called a **class property**.

**Theorem 2.1.** *Recurrence is a class property.*

Proof can be found in *Theorem 1.2 in Chapter 3* of Brémaud, 1999.

**Definition 2.2.** A Markov chain is called **irreducible** is if consists a single communicating class.

An irreducible Markov chain is called positive recurrent, null recurrent or transient if any of the states is positive recurrent, null recurrent or transient respectively.

**Theorem 2.2.** *A Markov chain is irreducible and positive recurrent if and only if it has a unique stationary distribution.*

**Theorem 2.3.** *Suppose that an irreducible Markov chain has a stationary distribution* $d$*. Then* $d(x) = \frac{1}{\mathbf{E_x}[T_x^+]}$

The proof for Theorems 2.2 and 2.3 can be found in *Theorem 3.1* and *Theorem 3.2 in chapter 3* of Brémaud (1999).

**Definition 2.3.** The period of a state $x \in \Omega$ is defined as $d(x) := gcd\{t \in \mathbf{Z}_+ : P^t(x, x) > 0\}$.

A state $x \in \Omega$ is called **aperiodic** is $d(x) = 1$ and **periodic** otherwise.

**Theorem 2.4.** *For an irreducible positive recurrent aperiodic Markov chain with stationary distribution* $d$*,* $d_n \overset{n\to\infty}{\longrightarrow} d$*.*

The proof can be found in *Theorem 2.1* of *Chapter 4* in Brémaud (1999).

After generating a rollout of a Markov chain, we are interested in knowing whether there is any relationship between looking at the chain in the reverse direction starting from some time $N$ and looking at the chain in the forward direction until time $N$. Intuitively, there must be a relationship between the two views since they correspond to the same process and $P(k, k) = \hat{P}(k, k)$ for all $k \in \Omega$ and $\hat{P}(x, y) \propto P(y, x)$ where $\hat{P}$ is the transition matrix of the reverse chain. The following theorem formalizes this intuition.

**Theorem 2.5.** *Let $(X_n)_{n\geq 0}$ be an irreducible Markov chain with transition matrix $P$ and stationary distribution $d$. Let $Y_n = X_{N-n}$ for any $N \in \mathbf{N}$. The time reversed Markov chain $(Y_n)_{n\geq 0}$ is an irreducible Markov chain with transition matrix $\hat{P}(x,y) = \frac{d(y)P(y,x)}{d(x)}$ $\forall x, y \in \Omega$ and $d$ is a stationary distribution for $(Y_n)_{n\in\mathbf{N}}$.*

**Definition 2.4.** A Markov chain is **time-reversible** if $P(x,y) = \hat{P}(x,y)$ $\forall x, y \in \Omega$.

A time-reversible Markov chain satisfies the set of equations $d(x)P(x,y) = d(y)P(y,x)$ $\forall x, y \in \Omega$. This set of equations is called the **detailed balance** equations. A time-reversible Markov chain looks indistinguishable in the forward and reverse direction when $d_0 = d$.

**Definition 2.5.** The **mean first passage time** from state $x$ to state $y$, denoted by $m(y|x)$, is defined as the average number of steps to reach $y$ when the chain is started in $x$.

The mean first passage time satisfies the following recurrence relation

$$m(y|x) = \begin{cases} 0, \text{ if } x = y \\ 1 + \sum\limits_{\substack{u\in\Omega \\ u\neq y}} P_{xu} m(y|u), \text{ otherwise} \end{cases}$$

**Definition 2.6.** The **mean commute time** $n(x,y)$ is defined as $n(x,y) := m(x|y) + m(y|x)$.

The mean commute time is symmetric by definition and it will play a fundamental role in our approach of developing the state embedding.

# 3

# Reinforcement Learning

Reinforcement Learning (RL) is the problem of learning goal-oriented behavior by interacting with an environment. Markov Decision Processes (MDPs) provide a mathematical abstraction of the RL problem. MDPs can be considered a generalization of Markov Chains to include notions of actions and rewards; the transition distribution is conditioned on the actions in addition to the states and rewards assign utilities to transitions. Solving a reinforcement learning problem corresponds to solving MDPs but without assuming the knowledge of dynamics of the MDP. When the dynamics of the environment is known apriori, the approach to solve MDPs is called Dynamic Programming. In this chapter we introduce some important reinforcement learning notions and solution methods. The notation and discussion in this chapter follows (Sutton and Barto, 2018).

## 3.1 Preliminary Definitions

Let $S$, $A$ and $R$ be the set of states, actions and rewards respectively, which we assume for simplicity to be finite. The set of actions available to an agent could depend on the state. For notation convenience, we assume that all the actions are available in all the states, but the definitions and the concepts presented apply in the general case with only appropriate change in notations.

The **reward function** is a distribution over $R$ from $S \times A$. The **dynamics model** $p$ of the MDP is a probability distribution $p(s', r|s, a)$ over $S \times R$ conditioned on $S \times A$. A sequence of states, actions and rewards experienced by the agent by interacting with the environment, $S_0, A_0, R_1, S_1, A_1, R_2, \cdots$, is known as a trajectory. The length of the trajectory is called the horizon.

RL problems naturally fall into two categories: where the interaction with the environment terminates in a finite number of steps, and where the interaction continues for infinite steps. The former is known as the **finite horizon** setting and the latter is known as the **infinite horizon** or the **continuing** setting.

The **return** is defined as a function of the sequence of rewards experienced. In the finite horizon setting, the **total return** is the sum of rewards starting from time $t$, defined as $G_t = \sum_{k=t}^{T-1} R_{k+1}$, where $T$ is the horizon. In the infinite horizon setting, the return as defined above might become infinite. The **discounted return** starting from $t$ is defined as $G_t = \sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1}$ where $\gamma \in (0, 1)$ is the discount factor. Discounting the rewards ensures that the return is finite as long as the rewards are bounded. To unify the notations for the finite and the infinite horizon cases, we write the return as $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ where either $T = \infty$ or $\gamma = 1$. The goal of a RL agent is to find a way of behaving that optimizes the **expected return**, as we detail below.

The sequence of rewards, and hence the return, depends on the behavior of the agent, known as its policy. Formally, a **policy** $\pi$ is a distribution over actions at every state; the agent takes an action $a$ at state $s$ with probability $\pi(a|s)$. The expected return starting from a state $s$ and following a policy $\pi$ is termed the **value** of $s$, denoted by $v_\pi(s)$. Therefore, the **value function of a policy** is $v_\pi : S \to \mathbb{R}$. The **state-action value function** $q_\pi(s, a)$ is defined as the expected return starting from state $s$ and taking an action $a$ and following $\pi$ afterwards. $v_\pi(s)$ can be written in terms of $q_\pi(s,a)$ as follows

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a) \tag{3.1}$$

The state-action value function can be written in terms of the state value function as follows

$$\begin{aligned} q_\pi(s,a) &= \mathbf{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a] \\ &= \sum_{s' \in S} \sum_{r \in R} p(s', r|s,a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3.2}$$

Using equations 3.1 and 3.2, the value function can be recursively written as

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} \sum_{r \in R} p(s', r|s,a)[r + \gamma v_\pi(s')] \tag{3.3}$$

and the state-action value function as

$$q_\pi(s,a) = \sum_{s' \in S} \sum_{r \in R} p(s', r|s,a)[r + \gamma \sum_{a' \in A} \pi(a'|s') q_\pi(s', a')] \tag{3.4}$$

The equations (3.3) and (3.4) are called the **Bellman equations** for $v_\pi$ and $q_\pi$ respectively. The set of Bellman equations (3.3) forms a linear system of $|S|$ equations in $|S|$ unknowns. This system of equations has a unique solution if $\gamma < 1$ and the Markov chain induced by $\pi$ has a single recurrent class, or if the MDP is finite-horizon. This solution is the value function $v_\pi$ resulting from following the policy $\pi$.

The objective of solving an MDP is to find an optimal behavior that maximizes the value of all the states. The value function, thus, induces a definition for what it means for a policy to be better than another. Using the value function, we define a partial order on $\Pi$, the space of all possible policies for the given MDP. For policies $\pi'$ and $\pi$, $\pi' \geq \pi$ if and only if $v_{\pi'}(s) \geq v_\pi(s), \forall s \in S$. $\pi_*$ is said to be an **optimal policy** if $\pi_* \geq \pi, \forall \pi \in \Pi$. Optimal policies are not necessarily unique. The probabilities among actions with equal state-value functions in a state can be distributed in any way without affecting the value of the state.

The value function under an optimal policy $\pi_*$ is denoted by $v_*$. From the definition of optimal policies given above, we can see that $v_*(s) = \operatorname{argmax}_{\pi \in \Pi} v_\pi(s)$ and

$v_*$ is unique. The optimal state-action value functions $q_*$ is defined as $q_*(s,a) = \max_\pi q_\pi(s,a)$. Like the Bellman equations, $v_*$ and $q_*$ can be defined recursively, called the **Bellman optimality equations**. We first define $v_*$ in terms of $q_*$ and $q_*$ in terms of $v_*$, similar to equations (3.1) and (3.2) to obtain the recursive definition. The optimal state value is given by

$$v_*(s) = \max_a q_*(s,a) \tag{3.5}$$

and the optimal state-action value function is given by

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s)] \tag{3.6}$$

Using equations (3.5) and (3.6), the optimal state and state-actions value functions are written as follows

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

and

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \tag{3.8}$$

Since the objective of solving an MDP is to find $\pi_*$, it might seem unnecessary to define or compute $v_*$, a quantity derived from $\pi_*$. The Bellman optimality equations (3.1) and (3.8) define a non-linear system of $|S|$ equations with $|S|$ unknowns and $|S \times A|$ equations with $|S \times A|$ unknowns respectively. Solving this system of equations provides the optimal state and state-action value functions without the knowledge of an optimal policy. An optimal policy $\pi_*$ can be recovered from $v_*$ using a one-step search, or it can be obtained directly by being greedy with respect to $q_*(s,a)$ in state $s$, $\forall s \in S$. Thus, $\pi_*$ can be computed from $q_*$ and $v_*$.

## 3.2 Dynamic Programming

Dynamic Programming (DP) methods are used to solve MDPs when the dynamics of the MDP are known. Ideas from DP methods lay the foundations for reinforcement learning methods, even though DP methods are not applicable in the usual reinforcement learning setting. In this section, we discuss how the value function for a given policy can be computed, and how this value function can then be used to improve the policy. A repeated application of this process returns an optimal policy which is guaranteed to be better than the policy with which we started, by the *policy improvement theorem.* The combination of policy evaluation and policy improvement can be used obtain an optimal policy. These two steps can be combined in several ways, leading to a framework called the Generalized Policy Iteration (GPI), which is a recurring theme in the DP and reinforcement learning methods. The methods discussed below require that all the states continue to be updated to guarantee convergence to the correct values. The same is also true for the reinforcement learning methods.

### 3.2.1 Policy Evaluation

Policy evaluation is the process of computing or estimating the value function $v_\pi$ for a given policy $\pi$. $v_\pi$ is well defined when all the episodes starting from every state $s \in S$ terminate in a finite number of steps when behaving according to $\pi$, or when $\gamma < 1$. As noted in the previous section, $v_\pi$ may be obtained by solving the linear system of $|S|$ equations. In this section and in the subsequent section, iterative approaches for computing $v_\pi$ are favored instead, mainly due to the flexibility that an iterative approach offers in terms of obtaining meaningful partial computations and asynchronous updates. After all, the reason for computing the value function of a given policy is to obtain an optimal policy; exact computation of the value function is not a goal in itself to obtain optimal behavior.

Iterative policy evaluation produces a sequence of value functions $\{v_0, v_1, v_2, \cdots\}$ where $v_0$ is initialized arbitrarily with the exception that the terminal states, if any, have values of 0. This sequence converges to $v_\pi$. $v_{k+1}$ is computed from $v_k$ as follows

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma v_k(s')] \tag{3.9}$$

Equation (3.9) is the Bellman equation for the state value function, but turned into an update rule. If the sequence $\{v_k\}$ converges to $v$, then $v$ satisfies the Bellman equation for the value function under $\pi$. Since $v_\pi$ is unique, $v$ must be $v_\pi$. Thus, the iterative process can be used to compute the value function $v_{pi}$ for a given policy $\pi$.

### 3.2.2 Policy Improvement Theorem

Assume $v_\pi$ of a policy $\pi$ can be used to obtain a policy $\pi' \geq \pi$. Then, $v_{\pi'}$ can be used to obtain $\pi'' \geq \pi'$ and so on. Thus, if there is a mechanism to obtain a better policy $\pi'$ from a given policy $\pi$ using its value function $v_\pi$, a sequence of better policies can be obtained. We will now show that a better policy $\pi'$ given a policy $\pi$ can be obtained by being greedy with respect to $v_\pi$ at every state $s \in S$. Let $\pi$ and $\pi'$ be arbitrary policies such that

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \quad \forall s \in S$$

where we use the notation $q_\pi(s, \pi'(s)) = \sum_{a \in A} \pi'(a|s) q_\pi(s, a)$. Now we will prove $v_{\pi'}(s) \geq v_\pi(s)$ $\forall s \in S$ where $\pi$ and $\pi'$ are as defined above.

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbf{E}_{\pi'}[q_\pi(s, A_t)|S_t = s] \\
&= \mathbf{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&\leq \mathbf{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] \\
&= \mathbf{E}_{\pi'}[R_{t+1} + \gamma \mathbf{E}_{\pi'}[q_\pi(S_{t+1}, A_{t+1})]|S_t = s] \\
&= \mathbf{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s] \\
&= \mathbf{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})|S_t = s] \\
&\vdots \\
&= \mathbf{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}$$

Hence, the value function $v_{\pi'}$ of a policy $\pi'$ is at least as good as $v_\pi$ whenever selecting actions to $\pi'$ in $s$ and following $\pi$ afterwards is better than following $\pi$ at $s$, $\forall s \in S$ i.e whenever $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, $\forall s \in S$. Therefore, $\pi' \geq \pi$. This is known as the **policy improvement theorem**.

From this, it is easy to see that a new policy $\pi'$ obtained by being greedy with respect to $v_\pi$ is a better policy than $\pi$, since $max_{a \in A} q_\pi(s,a) \geq \sum_{a \in A} \pi(a|s) q(s,a)$, $\forall s \in S$.

Let $\{\pi_k\}$ be the sequence of policies obtained using the policy improvement theorem starting from some policy $\pi_0$. Then $\pi_{i+1} \geq \pi_i, \forall i$. Let $\pi_{k+1} = \pi_k$ for some $k$. Then, the following equations hold

$$v_{\pi_{k+1}}(s) = v_{\pi_k}(s) \quad \text{(since } \pi_{k+1} = \pi_k\text{)} \tag{3.10}$$

$$v_{\pi_{k+1}}(s) = \sum_{s',r} p(s',r|s,\pi_{k+1}(s))[r + \gamma v_{\pi_{k+1}}(s')] \quad \text{(using Bellman equation)} \tag{3.11}$$

$$\pi_{k+1}(s) = \text{argmax}_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi_k}(s')] \quad \text{(greedy action selection)} \tag{3.12}$$

$$v_{\pi_k}(s) = \max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi_k}(s')] \quad \text{(using 3.10, 3.11 and 3.12)}$$

Therefore, $v_{\pi_k}$ satisfies the Bellman optimality equation. So, $v_{\pi_k} = v_{\pi_{k+1}} = v_*$, the optimal value function and $\pi_k$ and $\pi_{k+1}$ are optimal policies. Therefore, policy improvement by being greedy with respect to current value function results in strictly better policies, until an optimal policy is obtained.

### 3.2.3 Policy Iteration

Policy iteration is a procedure that combines policy evaluation and policy improvement to obtain optimal policies, starting from an arbitrary policy $\pi$. First, the value function $v_\pi$ is computed using policy evaluation described in the previous section. Then, the improved policy $\pi'$ is obtained by being greedy with respect to $v_\pi$. The process is terminated if $\pi' = \pi$. If not, the process is repeated with $\pi$ set to $\pi'$ until $\pi = \pi'$.

### 3.2.4 Value Iteration

In each iteration of policy iteration, the policy evaluation is performed until convergence for the corresponding policy. Since the value functions of intermediate policies are only used to improve the current policy, the exact convergence is not needed and so the policy evaluation step of policy iteration can be terminated before the convergence of $v_\pi$. Value iteration performs policy improvement and terminates the policy evaluation step in exactly one step. The update for value iteration is given by

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Thus, value iteration is obtained by turning the Bellman optimality equation into an update rule. The optimal policy is the greedy policy of the value function after convergence.

## 3.3 Reinforcement Learning

Reinforcement learning (RL) methods aim to achieve the same effect as DP methods: both DP and RL methods strive to compute the optimal policy using value functions. RL methods differ from DP methods in their approach to policy evaluation. Two equivalent definitions of value functions, one in terms of expected cumulative rewards and other in terms of Bellman equations, give rise to two approaches to estimating the value functions. The two approaches are called Monte Carlo (MC) and Temporal Difference learning (TD) respectively.

Unlike DP methods which compute the value function using the knowledge of the dynamics of the MDP, RL methods estimate the value function from the samples obtained by interacting with the environment. In order to guarantee convergence to the correct value function, both DP and RL methods require that all the states are continued to be updated. Thus, the RL agent must interact with the environment in a manner that guarantees that all the states or state action pairs are visited infinitely often. The update rule in MC and TD methods can be thought of as updating the value function towards an approximation of the return from some particular trajectory, or an approximation of such a return. MC and TD differ in how they compute this approximation.

### 3.3.1 Monte Carlo

Monte Carlo methods estimate the value functions by averaging the sample returns. Hence, Monte Carlo returns are defined only for the finite horizon setting.

Let $P^\pi$ denote the Markov process induced by the transition dynamics $p$ and the policy $\pi$: $P^\pi(s'|s) = \sum_{a \in A} \pi(a|s)p(s'|s,a)$. For a state $S_t$ in a trajectory $\tau \sim P^\pi$, Monte Carlo updates the estimate $V$ of the value function $V^\pi$ as:

$$V(S_t) = V(S_t) + \alpha[G_t - V(S_t)] \tag{3.13}$$

where $\alpha \in (0,1)$ is the learning rate, which controls the proportion of the current estimate $V(S_t)$ and the current return $G_t$ used to compute the new estimate. A sample return $G_t$ in equation (3.13) is the cumulative sum of rewards $R_{t+1} + R_{t+2} + \cdots R_T$ observed in $\tau$ (discounting is not necessary if the horizon is finite, but can be included if desired too).

Unlike the DP setting, where systematic sweeps are performed, in reinforcement learning only those states that have non-zero probability of being visited under $\pi$ will have their values updated. This is problematic; policy evaluation is a step towards policy improvement. The value for each action in every state must be evaluated to perform policy improvement. The additional mechanisms that make Monte Carlo approaches suitable for policy improvement are discussed below.

In order to do policy improvement when the environment dynamics are not assumed to be known, estimating the state-action value function instead of the state value function side-steps the need for one-step search to select the action maximizing the return. Since we are interested in control and performing policy improvement, we assume that all the methods discussed below estimate the state-action value function.

#### 3.3.1.1 Monte Carlo with exploring starts

Monte Carlo with exploring starts is the simplest, but not the most practical setup that ensures that all pairs of states and actions have non-zero visitation probability. In this setup, it is assumed that the agent starts in a state-action pair, where the state-action pair is chosen by the environment. In the real world, this assumption is limiting. More reasonable alternatives are discussed in the next section.

#### 3.3.1.2 On-policy and Off-policy methods

To estimate the value function of all the states or state-action pairs, all actions must continue to be selected. This can be achieved by the following two approaches. The first approach is to ensure that the policy assigns non-zero probability to each action in every state. This class of policies is known as soft-policies. In the policy improvement phase, the current soft-policy is improved to obtain a new soft-policy. This approach is known as on-policy approach, as the samples collected from the policy that was executed are used to improve the policy. The second approach, known as the off-policy approach, is to collect samples under a different policy, known as the behavior policy $b$, to improve a different policy, known as the target policy $\pi$.

In the on-policy setting, the policy improvement theorem is used to show that the policy improvement process produces a sequence of monotonically better policies, until convergence to an optimal soft-policy, depending on how the softness is guaranteed in policy improvement.

In the off-policy setting, since the behavior policy $b$ and the target policy $\pi$ are different, averaging the returns obtained by following $b$ will return the value function estimate for the policy $b$. To compute the correct expectation of the returns under $\pi$, the returns are re-weighted using the importance sampling ratios. The return $G_t$ following time step $t$ for state $S_t$, is re-weighted using the ratio $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$. For the state-action value function, the return $G_t$ is re-weighted using $\rho_{t+1:T-1} = \prod_{k=t+1}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$. Since the target policy is different from the behavior policy, the target policy can be made greedy with respect to the current value estimates during policy improvement.

#### 3.3.1.3 Off-policy Monte Carlo without Importance Sampling

Off-policy methods can be viewed as an exploring starts method but with a significant difference; the agent performs exploring starts for itself instead of relying on the environment. This can be understood as follows. When the importance sampling ratio is 0 at some time step $t$, the ratio is 0 for all the time steps occurring before $t$. As a result, the return for all the states up to $S_t$ is 0. Let $t$ be the last time step for which $\pi(A_t|S_t)/b(A_t|S_t) = 0$ in a trajectory $\tau$. When the target policy $\pi$ is deterministic, all the actions $A_{t+1}, A_{t+2}, \cdots A_{T-1}$ must be exactly the same as those given by $\pi$. Hence, the behavior policy $b$ can be thought of as implicitly doing exploring starts, starting from an initial state for a random number of steps $k$ and following the target policy $\pi$ afterwards. The time $k$ is determined by when the ratio is zero.

However, since the actions $A_{t+1}, A_{t+2}, \cdots A_{T-1}$ are sampled from $b$, this action sequence has a different probability under the behavior and target policies and therefore the returns have to be weighted by importance sampling. Alternatively, if the actions are selected according to $\pi$ starting from $S_{t+1}$ instead of sampling from $b$, the returns need not be re-weighted. As a result, in the interactive settings, the need for importance sampling can be overcome in the following manner: apply the behavior policy $b$ for a random number of time steps, say $t$, and follow the target policy $\pi$ afterwards until termination and update the value of states $S_t, S_{t+1}, \cdots S_{T-1}$ using the sample returns under $\pi$.

### 3.3.2 Temporal Difference Learning

Temporal Difference (TD) prediction approximates the return $G_t$ by bootstrapping from the estimated value of the next state. Bootstrapping allows the TD prediction to be applicable for both finite horizon and continuing tasks. Furthermore, bootstrapping allows the return from one trajectory to improve the value estimates for even those states that did not appear in the trajectory. As a result, TD prediction can be intuitively seen to allow for efficient use of experience compared to MC methods, even though there is no formal guarantee over MC. Formally,The TD update performed at every state $S_t$ is:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.14}$$

The quantity $\delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the temporal difference prediction error, or TD error in short, and gives the error in the estimate of the value function from one time step to the next.

The TD prediction differs qualitatively from the MC policy evaluation in the batch setting. Given a batch of experiences, TD prediction implicitly estimates the maximum likelihood (MLE) model of the MDP for the observed batch and computes the true value function for this approximate model. MC evaluation, unlike TD methods, minimizes the mean squared error between the average observed returns and predicted values. This partially informs the qualitative difference between TD and MC methods in the online setting.

TD prediction is used in policy evaluation to obtain the value estimates for states or state-action pairs. Using the value functions estimated using TD predictions for policy improvement has the same considerations as the value functions estimated using MC methods; all the states/state-action pairs must be continued to be updated. As a result, TD methods used for control can be categorized into on-policy or off-policy. In the control case, the state-action value function is estimated, for the same reason as in MC methods. The update for state-action value estimates, similar to equation (3.14), is given by:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - Q(S_t, A_t)] \tag{3.15}$$

Whether TD methods for control are on-policy or off-policy depends on how $V(S_{t+1})$ is chosen in the above equation. This is similar to MC methods, since $G_t = R_{t+1} + \gamma V(S_{t+1})$, and depending on how $G_t$ was computed, the MC method was categorized into on-policy or off-policy.

#### 3.3.2.1 Sarsa

Sarsa in an on-policy TD control method and hence $V(S_{t+1})$ in (3.15) is estimated according to the behavior policy. Therefore, $V(S_t) = \mathbf{E}_\pi[Q(S_t, A_t)]$ where $\pi$ is the behavior policy. $\mathbf{E}_\pi[Q(S_t, A_t)]$ can be computed either by computing the average of $Q(S_t, A_t)$ where $A_t \sim \pi(.|S_t)$ or by computing the expectation $\sum_{a \in A} \pi(a|S_t)Q(S_t, a)$. The former approach is called Sarsa and the latter approach is called expected Sarsa. The update equations for Sarsa and expected Sarsa respectively are given below:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.16}$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_{a \in A} \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.17}$$

Since Sarsa is an on-policy method, the behavior policy $\pi$ must be a soft-policy.

#### 3.3.2.2 Q-Learning

Q-Learning is an off-policy method and hence the return $V(S_{t+1})$ in equation (3.15) is computed according to the target policy. Since the target policy is the greedy policy with respect to the current $Q$ estimates, $V(S_t) = \max_{a \in A} Q(S_t, a)$ where $\pi$ is the target policy. Therefore, the return is bootstrapped using $G_t = R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a)$.

Similar to the discussion in 3.3.1.3, the behavior policy provides a mechanism of exploring starts. However, MC methods estimate the returns from the sample trajectories and thus require that either the target policy is followed until the end of the episode, or the sample return from the behavior policy is re-weighted using the importance sampling ratios. In contrast, since TD methods bootstrap, $max_{a \in A} Q(S_{t+1}, a)$ provides a simulation of the expected return under the target policy. Thus, the behavior policy of Q-Learning can be interpreted as providing exploring starts and at each state, the expected return from following the behavior policy is simulated to obtain the targets. The update equation is written as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.18}$$

### 3.3.3 Policy Gradients

Policy Gradient (PG) methods learn a parameterized policy to map directly from states to actions instead of estimating a value function from which to derive the policy indirectly. The policy parameters $\theta$ are learned so as to maximize the performance measured by the expected return, according to some distribution. In the episodic case, the performance is measured in terms of the value of an initial state $s_0$. Policy gradient methods are favored in settings where the policy is easier to learn compared to the value functions. These methods learn the policy parameters using gradient ascent. Computing the gradient of the value function with respect to $\theta$ does not require knowledge of the environment dynamics, as shown by the policy gradient theorem. We derive the policy gradient theorem below. We use $\pi$ to denote $\pi_\theta$ and $P_\pi(S_{t+k} = s' | S_t = s)$ to denote the probability of transitioning form $s$ to $s'$ in $k$ steps under the Markov chain defined by the policy $\pi$ and the dynamics $p$.

$$\begin{aligned}
\nabla v_\pi(s) &= \nabla \sum_{a\in A} \pi(a|s) q_\pi(s,a) \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{a\in A}\pi(a|s)\nabla q_\pi(s,a) \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{a\in A}\pi(a|s)\nabla\sum_{s',r}p(s',r|s,a)[r+v_\pi(s')] \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{s'}\sum_{a\in A}\pi(a|s)p(s'|s,a)\nabla v_\pi(s') \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)\nabla v_\pi(s') \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s)+ \\
&\quad \sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)\nabla[\sum_{a'\in A}q_\pi(s',a)+\sum_{s''}P_\pi(S_{t+2}=s''|S_{t+1}=s')\nabla v_\pi(s'')] \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)\sum_{a'\in A}\nabla q_\pi(s',a)+ \\
&\quad \sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)\sum_{s''}P_\pi(S_{t+2}=s''|S_{t+1}=s')\nabla v_\pi(s'') \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)\sum_{a'\in A}\nabla q_\pi(s',a)+ \\
&\quad \sum_{s''}\sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)P_\pi(S_{t+2}=s''|S_{t+1}=s')\nabla v_\pi(s'') \\
&= \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) + \sum_{s'}P_\pi(S_{t+1}=s'|S_t=s)\sum_{a'\in A}\nabla q_\pi(s',a)+ \\
&\quad \sum_{s'}P_\pi(S_{t+2}=s'|S_t=s)\nabla v_\pi(s') \\
&\vdots \\
&= \sum_{s'}\sum_{k=0}^{\infty}P_\pi(S_{t+k}=s'|S_t=s)\sum_{a\in A}q_\pi(s',a)\nabla\pi(a|s') \\
&= \sum_{s'}\sum_{k=0}^{\infty}\mathbf{E}_\pi[\mathbb{1}_{(S_{t+k}=s'|S_t=s)}]\sum_{a\in A}q_\pi(s',a)\nabla\pi(a|s')
\end{aligned} \tag{3.19}$$

The gradient of the value function starting from the initial state $s_0$ is given by setting $s = s_0$ in equation. Thus, 3.19

$$\begin{aligned}
\nabla v_\pi(s_0) &= \sum_s \sum_{k=0}^{\infty} \mathbf{E}_\pi[\mathbb{1}_{(S_{t+k}=s|S_t=s_0)}] \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) \\
&= \sum_s \mathbf{E}_\pi[\sum_{k=0}^{\infty} \mathbb{1}_{(S_{t+k}=s|S_t=s_0)}] \sum_{a\in A} q_\pi(s,a)\nabla\pi(a|s) \\
&= \mathbf{E}_\pi\left[\sum_{a\in A} q_\pi(S_t,a)\nabla\pi(a|S_t)\Big|S_0 = s_0\right] \qquad (3.20)
\end{aligned}$$

Thus, the gradient of the performance $\nabla v_\pi(s_0)$ can be estimated from the trajectories sampled under $\pi$. A more accurate description of the policy gradient theorem is that, the gradient of the dynamics with respect to the policy parameters is 0; the effect of $\theta$ on $v_\pi(s)$ is only through actions taken at the states seen in the trajectory and parameters are updated to increase the probability of actions leading to higher value. That is, the parameter vector $\theta$ has control over only some aspects (selecting actions) of the dynamical system and the parameters are updated to improve these controllable aspects in a manner that improves the overall performance.

#### 3.3.3.1 REINFORCE and Actor Critic

The quantity in equation (3.20) requires summation over all the actions. It will be beneficial to find a way to replace the summation with some equivalent quantity that only relies on the information contained within a trajectory.

$$\begin{aligned}
\mathbf{E}_\pi\left[\sum_{a\in A} q_\pi(S_t,a)\nabla\pi(a|S_t)\Big|S_0 = s_0\right] &= \mathbf{E}_\pi\left[\sum_{a\in A} q_\pi(S_t,a)\frac{\pi(a|S_t)}{\pi(a|S_t)}\nabla\pi(a|S_t)\Big|S_0 = s_0\right] \\
&= \mathbf{E}_\pi\left[\sum_{a\in A} \pi(a|S_t)q_\pi(S_t,a)\frac{\nabla\pi(a|S_t)}{\pi(a|S_t)}\Big|S_0 = s_0\right] \\
&= \mathbf{E}_\pi\left[q_\pi(S_t,A_t)\frac{\nabla\pi(A_t|S_t)}{\pi(A_t|S_t)}\Big|S_0 = s_0\right] \\
&= \mathbf{E}_\pi\Big[q_\pi(S_t,A_t)\nabla\ln\pi(A_t|S_t)\Big|S_0 = s_0\Big] \qquad (3.21)
\end{aligned}$$

Since $\mathbf{E}[G_t|S_t = s, A_t = a] = q_\pi(s, a)$, equation (3.21) can be written as

$$\begin{aligned}\mathbf{E}_\pi\Big[q_\pi(S_t, A_t)\nabla \ln \pi(A_t|S_t)\Big|S_0 = s_0\Big] &= \mathbf{E}_\pi\Big[\mathbf{E}_\pi[G_t]\nabla \ln \pi(A_t|S_t)\Big|S_0 = s_0\Big] \\ &= \mathbf{E}_\pi\Big[G_t\nabla \ln \pi(A_t|S_t)\Big|S_0 = s_0\Big] \qquad (3.22)\end{aligned}$$

The expectation in equation (3.22) does not change if $G_t$ is replaced with $G_t - b(s)$ where $b(s)$ can be any function that does not depend on the actions. $b(s)$ is called a baseline. A typical choice of $b(s)$ is the value of the state $v_\pi(s)$. $G_t - v_\pi(s)$ is called the advantage function $A_\pi(s, a)$. The return $G_t$ starting at time $t$ can be estimated either using a Monte Carlo approach or using bootstrapping. The former approach is called **REINFORCE** and the latter is called **Actor Critic**.

#### 3.3.3.2 TRPO

The algorithm we will propose in order to learn the embedding space is agnostic to the algorithms used for policy evaluation or control. In our experiments, we use Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) to learn the policy used for control. TRPO is a deep reinforcement learning algorithm, that is, the policy is parameterized by a neural network whose parameters are denoted by $\theta$. Let $\theta_{old}$ be the current policy parameters. The objective function of TRPO is given by:

$$\begin{aligned}&\max_\theta \mathbf{E}_{s\sim\rho_{old}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)}Q_{\theta_{old}}(s, a)\right] \\ &\text{subject to } \mathbf{E}_{s\sim\rho_{old}}[D_{KL}(\pi_{\theta_{old}}(.|s)\ ||\ \pi_\theta(.|s))] \leq \delta \qquad (3.23)\end{aligned}$$

Equation (3.23) is an instance of natural gradient descent (Amari, 1998). Let $g$ be the gradient of the objective function with respect to $\theta$. The direction of steepest descent/ascent in the above problem is given by $s = F^{-1}g$ where $F$ is the Fisher information matrix. The Fisher information matrix can be defined as the expected covariance matrix of the gradient of the log likelihood; the expectation is with respect to the distribution induced by the current parameters. This solution is obtained using a first-order approximation to the objective and second-order approximation of the

KL constraint using the gradient and Hessian evaluated at the current parameters. In TRPO, this corresponds to evaluating at $\theta_{old}$. To solve $F^{-1}g$, TRPO uses conjugate gradient, as storing and inverting $F$ is computationally intensive. The optimal learning rate is given by $\alpha = \sqrt{\frac{2\delta}{s^T Fs}}$. Due to the approximation of objective and the constraints, updating using $\alpha$ might violate the constraints. So, the correct update is obtained by performing line search, exponentially decaying the current step size until a value that guarantees improvement is found.

The TRPO equation (3.23) is an approximation of

$$\begin{aligned} \pi_{i+1} &= \text{argmax}_\pi \, L_{\pi_i} - C D_{KL}^{max}(\pi_i, \pi) \\ \text{where } C &= \frac{4\epsilon\gamma}{(1\gamma)^2} \text{ and } L_{\pi_i} = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s,a) \end{aligned} \tag{3.24}$$

where $\epsilon = \max_{s,a} |A_\pi(s,a)|$ and $\eta(\pi)$ is the expected discounted return under $\pi$. Schulman et al. (2015) show that optimizing equation (3.24) results in monotonically better policies.

## 3.4 Goal Conditioned Reinforcement Learning

A reinforcement learning task has the purpose of learning goal-directed behavior in which the goal is specified by the reward function. Breaking down a complex task into a set of simpler tasks can be beneficial to learning. These simpler tasks themselves are characterized using suitable reward functions. A simple reward function for a sub-task is a binary reward function which assigns 1 when the goal is achieved and 0 otherwise. The policies of the different sub-tasks are then composed to achieve the overall task. Instead of learning independent policies, the Universal Value Function Approximator (UVFA) (Schaul et al., 2015) highlights the benefits of sharing the parameters across these policies. The input to a UVFA is the concatenation of the state and the goal.

The goal space can be same as the state space, or it can be derived from the state space. In some of our experiments, the objective is to reach all the feasible goals, and in other experiments, a subset of the goal space is chosen to be desired goals. We consider the goal space to be the same as the state space.

# 4

# Graphs

The approach that we will propose for learning goals relies heavily on analyzing the connectivity of the graph underlying the MDP of interest. In this chapter, we review the necessary background from graph theory that will allow us to present our algorithm.

Graphs are often used to model the relationships between objects, corresponding to nodes/vertices. The relationships between the objects are represented by the edges of the graph. Graphs primarily fall into two categories: directed and undirected graphs. Directed graphs are useful to model asymmetric relationships and undirected graphs are used to model symmetric relationships. In this chapter, we give a brief overview of the preliminary definitions and theorems necessary to introduce a concept called the graph Laplacian. The graph Laplacian is the fundamental object in techniques such as spectral clustering and spectral graph drawing. In such methods, the eigenvectors of the graph Laplacian are used to obtain a low-dimensional representation of the vertices of the graph, which are then used for down-stream tasks such as clustering or visualization. We assume that the graphs of interest are simple graphs, that is, there are no self-loops and there is at most a single edge between any pair of nodes. The discussion in this chapter follows (Gallier and Quaintance, 2017) (Chapter 20) and (Luxburg, 2007).

## 4.1 Preliminary definitions

### 4.1.1 Directed Graphs

**Definition 4.1.** A **directed graph** $G = (V, E)$ is a pair of collections, where $V = \{v_1, \cdots v_m\}$ is the set of vertices and $E$ is a collection of ordered pairs of distinct vertices, called the edges.

**Definition 4.2.** The **degree** of a node $u$ is defined as the number of incoming and outgoing edges incident on $u$. A **walk** between nodes $u$ and $v$ is a sequence of nodes $\{v_0, \cdots, v_k\}$ where $v_0 = u$, $v_k = v$ and $(v_i, v_{i+1}) \in E$ for all $i \in [0, k-1]$.

**Definition 4.3.** A pair of nodes $(u, v)$ is said to be **connected** if there exists a walk from $u$ to $v$ and vice-versa. A binary relation that defines two nodes to be related if they are connected is an equivalence relation and the corresponding equivalence classes are called the **strongly connected components** of the graph, that is, there exists a walk for any pair of nodes in the same connected component. The graph is said to be **strongly connected** if there is a single equivalence class.

Next, we present two of the possible approaches to represent directed graphs.

**Definition 4.4.** If the directed graph has $m$ vertices and $n$ edges, the **incidence matrix** $B$ is a $m \times n$ matrix whose entries are defined as follows

$$B_{ij} = \begin{cases} +1 & \text{if node } v_i \text{ is the source vertex in edge } e_j \\ -1 & \text{if node } v_i \text{ is the destination vertex in edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

**Definition 4.5.** In directed graphs, a node $v$ is said to be **adjacent** to a node $u$, if there is an edge from $u$ and $v$. The **adjacency matrix** is a $m \times m$ matrix whose entries are given by

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge from node } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

### 4.1.2 Undirected Graphs

**Definition 4.6.** An **undirected graph** $G = (V, E)$ is a pair of collections, where $V = \{v_1, \cdots v_m\}$ is the set of vertices and $E \subseteq V \times V$ is a collection of sets of distinct vertices called the edges.

**Definition 4.7.** The **degree** of a node $u$ is defined as the number of incoming (or equivalently outgoing) edges incident on $u$. A **walk** between nodes $u$ and $v$ is a sequence of nodes $\{v_0, \cdots, v_k\}$ where $v_0 = u$, $v_k = v$ and $(v_i, v_{i+1}) \in E$ for all $i \in [0, k-1]$.

**Definition 4.8.** A pair of nodes $(u, v)$ is said to be **connected** if there exists a walk from $u$ to $v$ and vice-versa. A binary relation that defines two nodes to be related if they are connected is an equivalence relation. The corresponding equivalence classes are called the **connected components** of the undirected graph. If there is only one equivalence class, the undirected graph is said to be **connected**.

Two possible approaches to represent undirected graphs are presented below.

**Definition 4.9.** If the undirected graph has $m$ vertices and $n$ edges, the **incidence matrix** $B$ is a $m \times n$ matrix whose entries are defined as follows

$$B_{ij} = \begin{cases} +1 & \text{if node } v_i \text{ is a vertex in edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

**Definition 4.10.** In undirected graphs, a node $v$ is said to be **adjacent** to a node $u$, if there is an edge between $u$ and $v$. The **adjacency matrix** is a $m \times m$ matrix whose entries are given by

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between nodes } v_i \text{ and } v_j \\ 0 & \text{otherwise} \end{cases}$$

In the case of undirected graphs, the adjacency matrix is symmetric.

### 4.1.3 Weighted Graphs

The notion of edges can be extended to weighted edges to include the strength of the connection between the nodes or to denote the cost of moving from one node to another. Informally, graphs whose edges have weights associated with them are called weighted graphs and have many real world applications such as modelling the routes in a city. The formal definitions for the weighted and unweighted cases are given below.

**Definition 4.11.** A **weighted directed graph** $G = (V, W)$ is a pair where $V = \{v_1, \cdots v_m\}$ is the collection of vertices and $W$ is a $m \times m$ matrix with non-negative entries. An edge is said to be present from node $v_i$ to $v_j$ if $W_{ij} > 0$. The **degree** of a node $v_i$ is the sum of the weights of the incoming and outgoing edges incident on $v_i$.

**Definition 4.12.** Given a weighted directed graph $G = (V, W)$ with $m$ nodes and $n$ edges, the **incidence matrix** $B$ is a $m \times n$ matrix whose elements are given by

$$B_{ij} = \begin{cases} +\sqrt{w_{ik}} & \text{for edge } e_j = (v_i, v_k) \\ -\sqrt{w_{ik}} & \text{for edge } e_j = (v_k, v_i) \\ 0 & \text{otherwise} \end{cases}$$

**Definition 4.13.** A **weighted undirected graph** $G = (V, W)$ is a pair where $V = \{v_1, \cdots v_m\}$ is the collection of vertices and $W$ is a $m \times m$ symmetric matrix with non-negative entries. An edge is said to be present from node $v_i$ to $v_j$ if $W_{ij} > 0$. The **degree** of a node $v_i$ is the sum of the weights of the incoming (or equivalently, the outgoing) edges incident on $v_i$.

**Definition 4.14.** Given a weighted undirected graph $G = (V, W)$ with $m$ nodes and $n$ edges, the **incidence matrix** $B$ is a $m \times n$ matrix whose elements are given by

$$B_{ij} = \begin{cases} \sqrt{w_{ik}} & \text{for edge } e_j = \{v_i, v_k\} \\ 0 & \text{otherwise} \end{cases}$$

In both the directed and the undirected cases, the weights $W$ can be viewed as a generalization of the adjacency matrix.

## 4.2 Graph Laplacian

We provide the definition of the unnormalized and normalized graph Laplacian and then remark on some properties.

**Definition 4.15.** An **orientation** of a weighted undirected graph $G = (V, W)$ is function $\sigma : E \to V \times V$ such that $\sigma(\{u, v\} \in E) = (u, v)$ or $(v, u)$.

**Definition 4.16.** Given a weighted undirected graph $G = (V, W)$, the **oriented graph** $G^\sigma = (V, W^\sigma)$ is obtained by applying an orientation $\sigma$ on $G$.

From the previous definition, we can see that for any weighted undirected graph $G = (V, W)$, $BB^T = D - W$ where $B$ is the incidence matrix of any orientation $G^\sigma$ of $G$ and $D$ is the diagonal degree matrix.

**Definition 4.17.** The **unnormalized Laplacian** of a weighted undirected graph $G = (V, W)$ is defined as $L = BB^T = D - W$ where $B$ is the incidence matrix of any orientation $G^\sigma$ obtained from $G$ and $D$ is the degree matrix.

Let $G = (V, W)$ be an weighted undirected graph. In order to discuss the properties of the Laplacian as defined above, we first interpret the vectors in $\mathbf{R}^m$ and $\mathbf{R}^n$ as real valued functions of the vertices and edges respectively. Hence, $L$ can be viewed as a linear operator on $V$ and $B$ can be viewed as a linear transformation $B : E \to V$.

Note that $B$ is the incidence matrix obtained from any orientation $\sigma$ of $G$ and hence, if $(v_i, v_j)$ is an edge, then $(v_j, v_i)$ is not an edge in $G^\sigma$. For $x \in \mathbf{R}^m$, $(B^T x)_k = (x_i - x_j)$ such that $e_k = (v_i, v_j)$; the component of $B^T x \in \mathbf{R}^n$ is the difference between the value of $x$ at the source vertex and the destination vertex of the corresponding edge. Hence, it can be verified that for $x \in \mathbf{R}^m$, $(Lx)_i = \sum_{j=1}^{m} w_{ij}(x_i - x_j)$.

For any $x \in \mathbf{R}^m$ in the nullspace of $B^T$, the values of $v_i$ and $v_j$ are equal if there is an edge $v_i, v_j$ in $G$. As a result, all the nodes connected to $v_i$ have the same value as $v_i$. This implies that the nodes in the same connected component have the same value. Let $c$ be the number of connected components of $G$. Hence, we can see that a basis for the nullspace of $B^T$ is $z_1, \cdots, z_c$ where $z_i^k = 1$ if the node $v_k$ is in the $i^{th}$ connected component of $G$, otherwise $z_i^k = 0$. Therefore, the nullity of $B^T = c$. Using the rank-nullity theorem, we obtain the $rank(B^T) = m - c$ and therefore $rank(B) = m - c$, since the row rank of a matrix is equal to its column rank.

From the definition $L = BB^T$, we can also deduce that the nullspace of $L$ is same as that of $B^T$ and rank($L$)=rank($B^T$) =rank($B$) $= m - c$. If the graph $G = (V, W)$ is connected, then the null space of $L$ is spanned by $\mathbf{1}$, a vector of all ones. From the definition of $L$, it follows that the unnormalized Laplacians are symmetric and positive semi-definite. Hence, the eigenvalues of the unnormalized Laplacian are all real and non-negative.

Since the orthogonal complement of the null-space of the linear transformation is invertible and a vector of all ones $\mathbf{1}$ spans the nullspace of $L$, the psedudo-inverse of the Laplacian is intuitively given by $L^\dagger = (L + \frac{1}{m}\mathbf{1}\mathbf{1}^T)^{-1} - \frac{1}{m}\mathbf{1}\mathbf{1}^T$.

Though the approach to obtain state embedding introduced in this thesis is based on the unnormalized Laplacian, we present the definitions of the normalized Laplacian and its properties for completeness. The two variants of the normalized Laplacians are as defined as $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$ and $L_{rw} = D^{-1}L = I - D^{-1}W = D^{-\frac{1}{2}} L_{sym} D^{\frac{1}{2}}$.

Suppose $\lambda$ is an eigenvalue of $L_{rw}$ and $u$ is a corresponding eigenvector, then $\lambda u = L_{rw} u = D^{-1} L u = \lambda D^{-1} u$ and therefore, $Lu = \lambda D u$. Thus, the eigenvectors of $L_{rw}$ are the solutions to the generalized eigenvalue problem $Lu = \lambda Du$. The eigenvectors of $L_{rw}$, referred to as degree normalized eigenvectors in (Koren, 2003), are discussed in section 6.

Suppose $\lambda$ is an eigenvalue of $L_{rw}$ and $u$ is the corresponding eigenvector, then $D^{-\frac{1}{2}}L_{sym}D^{\frac{1}{2}}u = \lambda u$ implies $L_{sym}D^{\frac{1}{2}}u = \lambda D^{\frac{1}{2}}u$. Therefore, $\lambda$ is also an eigenvalue of $L_{sym}$ and $D^{\frac{1}{2}}u$ is the corresponding eigenvector. Conversely, if $\lambda$ is a eigenvalue of $L_{sym}$ and $u$ is the corresponding eigenvector, then $L_{sym}u = \lambda u$ implies $D^{-\frac{1}{2}}L_{sym}u = D^{-\frac{1}{2}}L_{sym}D^{\frac{1}{2}}D^{-\frac{1}{2}}u = L_{rw}D^{-\frac{1}{2}}u = \lambda D^{-\frac{1}{2}}u$. Therefore, the eigenvalues of $L_{sym}$ and $L_{rw}$ are the same and the eigenvectors one can be obtained from that of the other.

If $x \in \mathbf{R}^m$ is in the nullspace of $L$, then $x$ is also in the nullspace of $L_{rw} = D^{-1}L$. Conversely, if $L_{rw}x = D^{-1}Lx = 0$, then $Lx = 0$. Therefore, the nullspace of $L_{rw}$ is the same as that of $L$. Thus, the nullspace of $L_{sym}$ is $D^{\frac{1}{2}}x$ for $x$ in nullspace of $L_{rw}$.

Since $L_{sym}$ is symmetric and since the eigenvalues of $L_{sym}$ and $L_{rw}$ are the same, the eigenvalues of $L_{sym}$ and $L_{rw}$ are real valued. Since $L$ is positive semi-definite, $L_{sym}$ is also positive semi-definite since $yLy \geq 0$ for any $y \in \mathbf{R}^m$ including $D^{-\frac{1}{2}}x$ for any $x \in \mathbf{R}^m$.

# 5

# Dimensionality Reduction

Developing algorithms to visualize vast quantities of data is an important area of research and is the object of study of areas such as multidimensional scaling(MDS) and graph drawing. A crucial component in this process of visualization is representing the high-dimensional raw data in low dimensions preserving the properties of interest and the goodness of a low-dimensional representation is measured with respect to a task-specific objective. In this chapter, we explain the basic tools in Multi-Dimensional Scaling (MDS) and graph drawing. Our approach to learning state embeddings in MDPs draws extensively on the ideas presented in this chapter.

## 5.1 Multidimensional Scaling

The study of MDS is concerned with finding a low-dimensional configuration of objects by taking as input a set of pairwise dissimilarities between the objects. The resulting low-dimensional configuration has to be such that the distance in this the low-dimensional configuration between any pair of objects preserves the corresponding pairwise dissimilarities provided as input. As the dissimilarities are mapped to distances in the low-dimensional space, the input dissimilarities cannot be arbitrary and must satisfy certain constraints for MDS to be applicable. It is crucial that the dissimilarities are symmetric, non-negative and obey the triangle inequality. The output low-dimensional configuration is typically taken to be in the Euclidean geometry and sometimes uses the Manhattan-distance. Other geometries have been explored but are not as popular.

MDS in its original form is now called **Classical MDS**(CMDS) or Toegerson scaling. CMDS is simply the eigen-decomposition of a matrix derived from the dissimilarity matrix. The procedure of CMDS is explained in more detail in Section 5.1.2. The more modern approach to MDS minimizes an objective function called **stress** and does not have an analytic solution. In some settings, it is only desired to preserve the order or the rank of the pairwise distances of the objects. This type of ordinal constraints is the subject of **non-metric MDS**. In contrast, the methods that preserve the pairwise distances are classified as **metric MDS** methods. Our application of MDS is to preserve the pairwise dissimilarities and not just the ordinal information and hence we do not discuss non-metric MDS in this thesis. The discussion and notation follows (Borg and Groenen, 2006).

### 5.1.1 Metric MDS

To introduce the framework of MDS we use the notation $X \in \mathbf{R}^{n \times m}$ for a configuration of $n$ objects in $m$-dimensional space, $\delta_{ij}$ denotes the pairwise dissimilarities between the objects $(i, j)$ and $d_{ij}(X)$ denotes the Euclidean distance between objects $(i, j)$ given by the configuration $X$. Sometimes the shorthand notation $d_{ij}$ is used instead of $d_{ij}(X)$. The error between the input dissimilarity and the distance according to the configuration $X$ for any pairs of objects $i$ and $j$ is denoted by $e_{ij} = (d_{ij}(X) - \delta_{ij})^2$. Since it may not always be possible to obtain a configuration $X$ such that $d_{ij}(X) = \delta_{ij}$ for every pair of objects $i$ and $j$, the error terms $e_{ij}$ are weighted by the non-negative $w_{ij}$ proportional to the importance of the pair $(i, j)$. The weights of pairs $(i, j)$ for which $\delta_{ij}$ are missing is set to $w_{ij} = 0$. The weights $w_{ij}$ are assumed to be **irreducible**, which means that there does not exist a partition of the objects such that $w_{ij} = 0$

whenever $i$ and $j$ belong to a different partition. An MDS problem can be decomposed into multiple sub-problems whenever the weights are not irreducible, such that each sub-problem has an irreducible set of weights. Therefore without loss of generality, we assume that the weights are irreducible. Using the terms defined above, the problem of finding a configuration $X$ can be expressed as an optimization problem of the form:

$$\sigma_r(X) = \sum_{i<j} w_{ij}(d_{ij}(X) - \delta_{ij})^2 \text{ where } w_{ij} \geq 0 \tag{5.1}$$

The quantity in equation (5.1) is known as **raw stress**. The dissimilarities $\delta_{ij}$ can be transformed by a function $f : \mathbf{R} \rightarrow \mathbf{R}$ given by $\hat{d_{ij}} = f(\delta_{ij})$. Different choices of $f$ correspond to different MDS models. The class of MDS models induced by restricting $f$ to be continuous corresponds to metric MDS. For example $f$ could be a scaling function defined as $f(x) = bx$ for some constant $b > 0$. Such models are known as **ratio MDS** and are special case of metric MDS. The raw stress $\sigma_r(X)$ defined in 5.1 is dependent on the scale of input $\delta_{ij}$, which causes difficultly in making task agnostic guidelines of how small of a stress is considered good enough. In order to make such general guidelines, the raw stress $\sigma_r$ is normalized by $\sum d_{ij}^2(X)$. We are are simply concerned with a single task and are not concerned about making any general recommendations for the value of the stress. We therefore proceed to use the raw stress $\sigma_r(X)$, with the modification of using $\hat{d_{ij}}$ instead of $\delta_{ij}$ to consider the general case of allowing a continuous transformation of $\delta_{ij}$. Hence, for a continuous function $f$, the raw stress is redefined as

$$\sigma_r(X) = \sum_{i<j} w_{ij}(d_{ij}(X) - \hat{d}_{ij})^2 \text{ where } w_{ij} \geq 0 \text{ and } \hat{d}_{ij} = f(\delta_{ij}) \tag{5.2}$$

In general it is not possible to solve metric and non-metric MDS problems analytically and hence we have to resort to iterative approaches to solve MDS. Kruskal (1964) solved MDS using gradient descent but a more popular approach uses a technique called **majorization**. Majorization is an iterative procedure for minimizing a function $f(x)$ by iteratively upper bounding $f$ by a function $g(z, x)$ whose minimum is easy to compute, and such that $g(x, x) = f(x)$ for some fixed $x$. The minimum $x^*$ of $g(z, x)$ is chosen as $x$ for the next iteration and this procedure is repeated until convergence. Majorization produces a sequence of non-increasing solutions but it is not guaranteed to converge to the global minimum. Most of the effort to minimize a function $f$ using majorization is spent on finding the function $g$ which satisfies the properties mentioned above. We employ neural networks and stochastic gradient descent to minimize stress. Expanding equation (5.2) results in:

$$\begin{aligned}\sigma_r(X) &= \sum_{i<j} w_{ij}(d_{ij} - \hat{d}_{ij})^2 \\ &= \sum_{i<j} w_{ij} d_{ij}^2 + \sum_{i<j} w_{ij} \hat{d}_{ij}^2 - 2\sum_{i<j} w_{ij} d_{ij} \hat{d}_{ij}\end{aligned}$$

The second term in the above equation can be expressed succinctly in matrix notation using $||.||$ to represent the 2-norm: $e_i \in \mathbf{R}^{1\times n}$ such that $i^{th}$ index is 1 and 0 otherwise, $A^{ij} \in \mathbf{R}^{n\times n}$ is matrix such that $A^{ij}_{ii} = A^{ij}_{jj} = 1$ and $A^{ij}_{ij} = A^{ij}_{ji} = -1$ and $x_i$ to denote the $i^{th}$ row of $X$

$$\begin{aligned}\sum_{i<j} w_{ij}\hat{d}_{ij}^2 &= \sum_{i<j} w_{ij}||x_i - x_j||^2 \\ &= \sum_{i<j} w_{ij}(e_iX - e_jX)(e_iX - e_jX)^T \\ &= \sum_{i<j} w_{ij} tr((e_iX - e_jX)^T(e_iX - e_jX)) \\ &= \sum_{i<j} w_{ij} tr(X^T(e_i^T e_i + e_j^T e_j - e_i^T e_j - e_j^T e_i)X) \\ &= \sum_{i<j} w_{ij} tr(X^T A^{ij} X) \\ &= tr(X^T(\sum_{i<j} w_{ij} A^{ij})X) \\ &= tr(X^T V X) \qquad (5.3)\end{aligned}$$

The entries in $V$ in equation (5.3) are of the form $V_{ii} = \sum_{j\neq i} w_{ij}$ and $V_{ij} = -w_{ij}$ for $i \neq j$. Hence, we can write $V = D - W$ where $D_{ij} = 0$ for $i \neq j$ and $D_{ii} = \sum_{i\neq j} w_{ij}$ and $W_{ij} = w_{ij}$ for $i \neq j$ and $W_{ii} = 0$.

The weights $w_{ij}$ are set to emphasize the importance of accurately representing a $\delta_{ij}$. In the case where representing the small and large dissimilarities accurately is equally important, the stress function is modified as:

$$\sigma_{EL} = \sum_{i<j}\left(1 - \frac{d_{ij}(X)}{\hat{d}_{ij}}\right)^2 = \sum_{i<j}\hat{d}_{ij}^2(d_{ij} - \hat{d}_{ij})^2 \tag{5.4}$$

$\sigma_{EL}$ in equation (5.4) is called elastic scaling. This shows that the raw stress $\sigma_r$ emphasizes larger dissimilarities when $w_{ij} = 1$ for all $(i, j)$. The loss function of Sammon mapping (Sammon, 1969) is obtained by setting $w_{ij} = \hat{d}_{ij}^{-1}$. Other choices for metric MDS models are presented in (Borg and Groenen, 2006).

### 5.1.2 Classical MDS

The objective of CMDS is to find a configuration of the objects by assuming that the dissimilarities are distances in some Euclidean space. First, we discuss the tools used in CMDS, followed by the explanation of CMDS.

Provided with a matrix $X \in \mathbf{R}^{n\times m}$ of $n$ objects in $m$-dimensional space, we can form the squared pairwise distance matrix denoted by $D^{(2)} \in \mathbf{R}^{n\times n}$ such that $D^{(2)}_{ij} = ||x_i - x_j||^2$ and can be expressed as $D^{(2)} = c\mathbf{1}^T + \mathbf{1}c^T - 2XX^T$ where $c \in \mathbf{R^m}$ such that $c_i = ||x_i||^2$ and $\mathbf{1} \in \mathbf{R}^n$ because

$$\begin{aligned} d^{(2)}_{ij}(X) &= ||x_i - x_j||^2 \\ &= ||x_i||^2 + ||x_j||^2 - 2x_i x_j^T \end{aligned}$$

Let $B = XX^T$. The squared pairwise distances matrix $D^{(2)}$ and the scalar product matrix $B$ can be obtained from the configuration matrix $X$. A natural question that arises is whether the we can recover $X$ given $D^{(2)}$ or $B$. Recovering $X$ from $D^{(2)}$ is the objective of CMDS. First, let us consider the simpler case of recovering $X$ from the scalar product matrix $B$.

Since $B$ is symmetric and positive semi-definite, $B$ admits an eigen-decomposition

$$B = Q\Lambda Q^T = Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^T = X'X'^T \tag{5.5}$$

where $\Lambda_{ii}$ is the $i^{th}$ largest eigenvalue of $B$ and $Q$ is an orthogonal matrix whose columns consist of eigenvectors of $B$ ordered by their corresponding eigenvalues in descending order. The positive semi-definiteness of $B$ ensures that all the eigenvalues are greater than or equal to 0. Since $(X'X'^T)_{ij} = B_{ij}$, the pairwise distances on $X'$ preserves pairwise distances in $D^{(2)}$. Hence, given a pairwise scalar product matrix $B$, a configuration $X'$ that preserves the pairwise distances can be recovered. Due to the use of Euclidean distance, the origin is assumed to be $\mathbf{0} \in \mathbf{R}^n$.

We now proceed to describe the procedure for obtaining a configuration that preserves squared pairwise distances given in $D^{(2)}$. Let $J = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T \in \mathbf{R}^{n\times n}$. For any $x \in \mathbf{R}^n$, $y = Jx$ is a column vector in $\mathbf{R}^n$ balanced on the origin. Similarly for $x \in \mathbf{R}^{1\times n}$, $xJ$ is a row vector balanced on the origin. Multiplying a matrix $X$ by $J$ from both right and left is known as double-centering $X$, since the rows and columns of $X$ are balanced on the origin. Since $D^{(2)} = c\mathbf{1}^T + \mathbf{1}c^T - 2XX^T$ we get

$$\begin{aligned} -\frac{1}{2}JD^{(2)}J &= -\frac{1}{2}J(c\mathbf{1}^T + \mathbf{1}c^T - 2XX^T)J \\ &= 0 - 0 + JXX^TJ \end{aligned} \tag{5.6}$$

because $J\mathbf{1} = 0$. Since we are only interested in a configuration $X$ that preserves the distance by the application of CMDS, we assume that the columns of $X$ are centered on 0. Therefore $JX = X$. Hence, equation (5.6) is written as $-\frac{1}{2}JD^{(2)}J = XX^T = B$. Therefore, given $D^{(2)}$, $B$ is derived using the double-centering operation and multiplication by $-0.5$. Following Equation (5.5), an eigen-decomposition is performed on $B = Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^T$. Let $\Lambda_+^{\frac{1}{2}}$ be the matrix containing columns corresponding to the positive eigenvalues and $Q_+$ be the corresponding eigenvectors. A configuration that preserves the pairwise distances is then obtained $X = Q_+\Lambda_+^{\frac{1}{2}}$. Since the dissimilarities are only assumed to be in Euclidean space and the original Euclidean space is unknown, the notation $\Delta^{(2)}$ is used instead of $D^{(2)}$.

To summarize, given a squared pairwise dissimilarity matrix $\Delta^{(2)}$, CMDS first computes $B = -\frac{1}{2}J\Delta^{(2)}J$ and then an eigen-decomposition of $B = Q\Lambda Q^T$ and then

$X = Q_+\Lambda_+^{\frac{1}{2}}$ is computed to the be the configuration of objects. If $\Delta$ in fact corresponds to Euclidean distance, then an exact configuration is obtained. This shows that the solution to CMDS can be computed analytically. An alternate characterization of CMDS is given by the loss function $L(X) = ||XX^T - B||_F^2$ known as **strain** where $F$ is the Frobenius norm. Due to the Frobenius norm, strain can be written as:

$$L(X) = \sum_{i<j} (x_i x_j^T - B_{ij})^2 \tag{5.7}$$

showing that CMDS can be used in an iterative setting but minimizing strain does not correspond to minimizing stress.

## 5.2 Graph Drawing

Graph drawing is the study of representing a graph $G = (V, E)$ in $\mathbf{R}^n$ for $n << |V|$. Each node $v \in V$ is mapped to $\rho(v)$ and a line segment is drawn between $\rho(v_i)$ and $\rho(v_j)$ whenever $(v_i, v_j) \in E$. The graph drawing problem can be generalized to consider other geometries. In this thesis we are concerned with approaches to drawing undirected graphs. See Kobourov, 2012 for a survey of algorithms for drawing undirected graphs whose edges represented by straight lines. The methods described in this section corresponds to an approach called *force directed* drawing where there is a tension between attractive forces attempting to place neighbouring nodes closer and repulsive forces that pushes away all the nodes apart from each other. The attractive forces usually correspond to some graph theoretic distance such as the shortest path between nodes. In this section we review the graph drawing approaches relevant for our approach to learn state embeddings. Both of the approaches presented below assume that the graph is connected. In the case of a disconnected graph, the graph drawing problem can be solved for each of the connected components. Hence, in this section we assume that the graphs are connected without loss of generality.

### 5.2.1 Stress Minimization

Kamada and Kawai, 1989 model the graph drawing problem as a dynamical system where the nodes are connected by springs. The length of the spring connecting a pair of nodes corresponds to the desired distance between this pair of nodes. Their approach to draw undirected graphs is by minimizing an energy function given by

$$E = \frac{1}{2}\sum_{i<j} k_{ij}(||x_i - x_j|| - l_{ij})^2 \tag{5.8}$$

where $l_{ij} = L \times d_{ij}$ and $L$ is a constant and $k_{ij}$ are the strengths of the spring. Kamada and Kawai, 1989 minimize this energy function $E$ using Newton-Raphson. The energy function $E$ in equation (5.8) is the stress function of metric MDS introduced in section This is precisely the *stress* function introduced in (5.1.1). Gansner, Koren, and North, 2004 note this connection and found that minimizing the energy function $E$ using majorization leads to better layouts and reduces the running time.

### 5.2.2 Spectral Graph Drawing

Spectral graph drawing is the approach of using eigenvectors and eigenvalues of matrices related to an undirected graph $G = (V, E)$ for the graph drawing problem. Koren, 2003 highlights the advantages of using spectral properties of the Laplacian matrix associated with the given graph in the context of graph drawing. Since the graph is connected by assumption, the null space of the Laplacian $L = D - A$ is 1-dimensional spanned by $\mathbf{1}$ and hence the multiplicity of the eigenvalue 0 is 1.

Let $n = |V|$. Koren, 2003 introduce the graph drawing problem in 1-dimension as the following constraint optimization problem of the energy function

$$E(x) = \sum_{i<j} w_{ij}(x_i - x_j)^2 \quad \text{s.t } Var(x) = 1 \tag{5.9}$$

where $x \in \mathbf{R}^n$. This approach is known as the **eigen-projection** method. The energy function $E(x) = \sum_{i<j} w_{ij}(x_i - x_j)^2$ can be succinctly represented as $E(x) = x^T Lx$ us-

ing equation (5.3). An alternate derivation is shown here by expanding the terms

$$\begin{aligned}\sum_{i<j} w_{ij}(x_i - x_j)^2 &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}(x_i - x_j)^2 \\ &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}(x_i^2 + x_j^2) - 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_ix_j \qquad (5.10)\end{aligned}$$

We solve the first and second terms separately and combine them to obtain the desired result. First we show $2\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_ix_j = x^TAx$.

$$\begin{aligned}2\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_ix_j &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_ix_j + \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_ix_j \\ &= \sum_{i=1}^{n} x_i \sum_{j=i+1}^{n} w_{ij}x_j + \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ji}x_ix_j \quad \text{(since } w_{ij} = w_{ji}\text{)} \\ &= \sum_{i=1}^{n} x_i \sum_{j=i+1}^{n} w_{ij}x_j + \sum_{j=1}^{n}\sum_{i=1}^{j-1} w_{ji}x_ix_j \\ &= \sum_{i=1}^{n} x_i \sum_{j=i+1}^{n} w_{ij}x_j + \sum_{i=1}^{n}\sum_{j=1}^{i-1} w_{ij}x_ix_j \quad \text{(by change of variable)} \\ &= \sum_{i=1}^{n} x_i \sum_{j=i+1}^{n} w_{ij}x_j + \sum_{i=1}^{n} x_i \sum_{j=1}^{i-1} w_{ij}x_j \\ &= \sum_{i=1}^{n} x_i \sum_{j=1}^{n} w_{ij}x_j \quad \text{(since } w_{ii} = 0\text{)} \\ &= x^TAx \qquad (5.11)\end{aligned}$$

Similarly, we show $\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}(x_i^2 + x_j^2) = x^TDx$ below

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}(x_i^2 + x_j^2) &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_i^2 + \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ji}x_j^2 \\ &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}x_i^2 + \sum_{i=1}^{n}\sum_{j=1}^{i-1} w_{ij}x_i^2 \quad \text{(similar to above proof)} \\ &= \sum_{i=1}^{n} x_{ii}^2 \sum_{j=1}^{n} w_{ij} \\ &= \sum_{i=1}^{n} x_{ii}^2 deg(i) \\ &= x^TDx \qquad (5.12)\end{aligned}$$

Using equations (5.11) and (5.12) in equation (5.10), we obtain $E(x) = x^TLx$. Further, by assuming that the mean of $x$ is zero and $Var(x) = \frac{1}{2}$, the following equation for energy is obtained

$$\min_x E(x) = \sum_{i<j} w_{ij}(x_i - x_j)^2 \quad \text{given } x^T x = 1 \tag{5.13}$$

Koren, 2003 prove that $x$ corresponds to the eigenvector corresponding to the lowest eigenvalue. The lowest eigenvalue is 0 and the corresponding eigenvector is $\mathbf{1}$ resulting in a degenerate solution. To overcome this uninteresting solution, an additional constraint that $x$ is orthogonal to $\mathbf{1}$ is added. As a result, the energy is minimized by choosing the eigenvector corresponding to the lowest positive eigenvalue.

Often it is desired to represent the nodes in more than one dimension. The $m$-dimensional representation is obtained by minimizing the energy under the constraint that $x^T y = 0$ where $y = \sum_{i<m} x_i$ in which $x_i$ are the eigenvectors corresponding to $i^{th}$ lowest positive eigenvalue. This constraint is added in addition to the usual constraints of $x^T x = 1$ and $x^T \mathbf{1} = 0$. The result presented in Koren, 2003 show that $m^{th}$ dimension corresponds to the eigenvector of $m^{th}$ smallest positive eigenvalue.

The choice of setting $Var(x) = \frac{1}{n}$ is arbitrary and any fixed positive value may be used for the variance. Using a fixed variance on all dimensions results in scattering the nodes equally along axis. While minimizing the energy for a fixed variance produces aesthetically pleasing graph drawing is it suitable for learning state embeddings? Intuitively the answer must be no since not all directions of variations must be weighted equally and more so when the embedding is used to compute distances. This intuition is formally shown to be correct chapter 7.

An intuitive interpretation of the solution of the eigen-projection method is provided in Koren, 2003. To understand the interpretation let us first characterize the solution to the energy minimization problem. By setting the derivative of $E(x)$ to be zero with respect to a fixed $x_i$ we obtain

$$\begin{aligned}
\sum_{x_j \in \mathbf{N}(x_i)} w_{ij}(x_i - x_j) = 0 \\
\sum_{x_j \in \mathbf{N}(x_i)} w_{ij} x_i &= \sum_{x_j \in \mathbf{N}(x_i)} w_{ij} x_j \\
x_i deg(x_i) &= \sum_{x_j \in \mathbf{N}(x_i)} w_{ij} x_j \\
x_i &= \sum_{x_j \in \mathbf{N}(x_i)} \frac{w_{ij}}{deg(x_i)} x_j \\
x_i &= \sum_{x_j \in \mathbf{N}(x_i)} p_{ij} x_j \qquad (5.14)
\end{aligned}$$

where $p_{ij}$ is the probability of transitioning from node $i$ to $j$ in the random walk on the graph. The transition probability matrix is $P = D^{-1}A$. Equation (5.14) can be interpreted the minimum of energy function is obtained by placing each node at the weighted centroid of its neighbours. This is trivially achieved by assigning all the nodes to the same embedding. The condition that each node can deviate from the centroid of its neighbours proportional to its current location, a constant value $\lambda$ which is common to all the nodes and inversely proportional to its degree can be expressed simultaneously for all nodes as $Lx = \lambda x$ because

$$\begin{aligned}
(I - D^{-1}A)x &= \lambda D^{-1}x \\
(D - A)x &= \lambda x \\
Lx &= \lambda x
\end{aligned}$$

Hence, the eigenvector associated with an eigenvalue $\lambda$ of the Laplacian allows the position of the node $i$ to deviate from the weighted centroid of its neighbours by $\lambda \times deg(i)^{-1} \times x(i)$ where $x(i)$ is the current position of the node $i$. Letting the nodes deviate from its centroid by the inverse of their degrees causes nodes with very small degrees to be situated very far from the centroid of its neighbours leading to a drawing where the nodes that are highly connected are densely drawn in a small area and the nodes with low degrees occupy most of the area and are sparsely located.

Such a drawing results in poor readability. In order to alleviate the dependence on the degree of the nodes, we may consider the following setup to obtain a balanced drawing

$$\begin{aligned}(I - D^{-1}A)x &= \lambda x \\ (D - A)x &= \lambda Dx \\ Lx &= \lambda Dx \qquad (5.15)\end{aligned}$$

The generalized eigenvectors of equation (5.15) are called degree-normalized eigenvectors. The eigen problem in equation (5.15) is the solution to the following optimization problem

$$\min_x x^T Lx \quad \text{s.t } x^T Dx = 1 \text{ and } x^T D\mathbf{1} = 0 \qquad (5.16)$$

The relationship between certain optimization problems and the eigenvalue problems are given in Ghojogh, Karray, and Crowley, 2019. The proof of the claims made in this section can be found in Koren, 2003 or follows directly from equation *(12)* of Ghojogh, Karray, and Crowley, 2019.

### 5.2.3 Comparison of stress minimization and spectral graph drawing

The energy function of stress and that of spectral graph drawing can be seen to be minimizing $||x_i - x_j||^2$ for all pairs of $(i, j)$. Equation (5.14) proves that $||x_i - x_j||^2$ is minimized by placing the nodes on the weighted centroid of its neighbours. The spectral graph drawing approach avoids the trivial solution of placing all the nodes in the same location by requiring that nodes deviate from the weighted centroid of their neighbours by some positive value $\lambda \times x(i)$ and $\lambda \times deg(i)^{-1} \times x(i)$ in degree normalized and eigen-projection methods respectively.

Metric MDS takes as input the dissimilarities in addition to the weights. Hence, intuitively we expect the metric MDS objective given in equation (5.1) to place the nodes such that the deviation from the centroid corresponds to the expected distance from their neighbours. In an $m$-dimensional representation the location of nodes along each axis $k$ are allowed to deviate from the centroid proportional to the importance of the axis and the dissimilarity between $i$ and $j$. Formally, the location of node $i$ along axis $k$ is given by $x_i^k = \sum_{j \in N(i)} p_{ij} x_j^k + \sum_{j \in N(i)} p_{ij} \delta_{ij} \frac{(x_i^k - x_j^k)}{||x_i - x_j||}$. We now prove this fact. First, we compute the derivative of equation (5.1) with respect to the position of a node $i$ along an axis $k$ denoted by $x_i^k$.

$$\begin{aligned}\frac{\partial E}{\partial x_i^k} &= 2 \sum_{j \in N(i)} w_{ij}(\delta_{ij} - ||x_i - x_j||) \frac{1}{2}((x_i - x_j)^T (x_i - x_j))^{-\frac{1}{2}} 2(x_i^k - x_j^k) \\ &= 2 \sum_{j \in N(i)} w_{ij}(\delta_{ij} - ||x_i - x_j||)||x_i - x_j||^{-1}(x_i^k - x_j^k) \\ &= 2 \sum_{j \in N(i)} w_{ij}(\delta_{ij}||x_i - x_j||^{-1} - 1)(x_i^k - x_j^k) \qquad (5.17)\end{aligned}$$

By setting the gradient in equation (5.17) to zero we obtain

$$\begin{aligned}2 \sum_{j \in N(i)} w_{ij}(\delta_{ij}||x_i - x_j||^{-1} - 1)(x_i^k - x_j^k) = 0 \\ \sum_{j \in N(i)} w_{ij}\delta_{ij}||x_i - x_j||^{-1}(x_i^k - x_j^k) &= \sum_{j \in N(i)} w_{ij}(x_i^k - x_j^k) \\ x_i^k &= \sum_{j \in N(i)} p_{ij} x_j^k + \sum_{j \in N(i)} p_{ij}\delta_{ij} \frac{(x_i^k - x_j^k)}{||x_i - x_j||}\end{aligned}$$

The position of $x_k^i$ is exactly $\mathbf{E}[\delta_{ij}]$ from the weighted centroid of the node's neighbours in the 1-dimensional case whenever $sign(x_i^k - x_j^k)$ is either positive or negative for all $j$ for a fixed $i$. However, in the general setting where not all $sign(x_i - x_j)$ are positive or negative the location of node $i$ node deviates from the centroid by the difference between weighted positive and negative terms. The location of the neighbours of $i$ was assumed to be fixed to provide the interpretation of how the position of $i$ is determined. However, in the case of spectral graph drawing the location of all $i$ are simultaneously determined and in the case of stress minimization it depends on the minimization algorithm.

## 5.3 Principal Component Analysis

Principal Component Analysis (PCA) is a well-known dimensionality reduction algorithm used to represent data in smaller number of uncorrelated dimensions of variations. The property of low-dimensional space obtained using PCA can be characterized in three equivalent ways. They are i) maximizes the variance of the projection of the input data on the low-dimensional space, ii) maximizes the Frobenius norm of the projection on the low-dimensional space, or iii) minimizes the Frobenius norm of the residuals of the projection onto the low-dimensional space.

Let $X \in \mathbf{R}^{m\times n}$ be a matrix of $m$ elements in $\mathbf{R}^n$ and the columns of $X$ are zero centered. The singular value decomposition of $X$ is given by $X = U\Sigma V^T$ where $U, V$ and $\Sigma$ such that $U\Sigma^2U^T = XX^T$ and $V\Sigma^2V^T = X^TX$. The representation of $X$ in the basis is obtained by projecting $X$ onto the columns of $V$, $(V^TX^T)^T = XV$. Since $X = U\Sigma V^T$, $XV = U\Sigma$.

By noting that the representation of $X$ in an orthonormal basis of eigenvectors given by $U\Sigma$, we obtain the interpretation that the Frobenius norm of $X$ is the Frobenius norm of $\Sigma$. This can be verified by noting that orthogonal transformations preserve length and $U$ and $V$ are orthogonal matrices. Since, $||X||_F^2 = ||U\Sigma||_F^2 = \sum_{i=1}^n \Sigma_{ii}^2||U_i|| = \sum_{i=1}^n \Sigma_{ii}^2$ where $U_k$ the $k^{th}$ column of $U$. Thus, in order to find a k-dimensional space which maximizes the Frobenius norm of the projection of $X$, the first $k$ columns of $V$ corresponding to the $k$ largest singular values have to be chosen.

### 5.3.1 Equivalence of PCA and cMDS

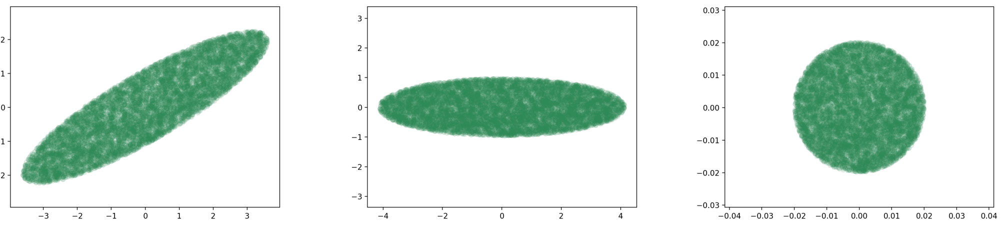
Let $X \in \mathbf{R}^{m\times n}$ be $m$ objects in $\mathbf{R}^n$. When the columns of $X$ is not centered on 0, it is centered on 0 in both PCA and cMDS (equation 5.6). Hence, without loss of generality, we assume that the columns of $X$ are centered on 0. PCA performs the spectral decomposition on the covariance matrix $X^TX \in \mathbf{R}^{n\times n}$ whereas classical MDS performs spectral decomposition on the scalar product matrix $XX^T \in \mathbf{R}^{m\times m}$.

However, the solution produced by PCA on $X$ and cMDS on the squared distances obtained from $X$ is the same. We prove this by first performing the singular value decomposition of $X = U\Sigma V^T$ for $U, V$ and $\Sigma$ such that $U\Sigma^2U^T = XX^T$ and $V\Sigma^2V^T = X^TX$. As discussed in the sections 5.1.2 and 5.3, the solution obtained from cMDS is $U\Sigma$ and the solution of PCA is given by $XV$ or equivalently $U\Sigma$. Thus, the solution of performing PCA on $X$ or performing cMDS on the squared distances matrix obtained from $X$ are the same. Thus PCA can be viewed as an approach to find an embedding space which preserves the pairwise distances.

### 5.3.2 Spectral Graph Drawing and PCA Whitening

In this section, we formalize the intuition for why spectral graph drawing is inadequate to obtain embeddings that preserve given pairwise distances. Let $L \in \mathbf{R}^{m\times m}$. The spectral decomposition of $L$ is given by $L = U\Lambda U^T$. In order to study the properties of the graph drawing obtained from $U$, we construct a different matrix $X$ related to $U$ and use $X$ to describe the properties of $U$.

Let $X \in \mathbf{R}^{m\times n}$ for some $n$ such that $X = U\Sigma V^T$. Thus, $X$ is any arbitrary matrix whose right eigenvectors are given by $U$. PCA on $X$ is given by $U\Sigma$. Therefore, we can write $U$ as $U = U\Sigma\Sigma^{\dagger}$. This corresponds to an operation called PCA whitening or sphering, used to transform the variance along each axis to be 1. Thus, using only the eigenvectors of $L$ causes all the axis to have the same variance. As a result, the scale of axes are discarded. For example, the representation of an ellipse using only the eigenvectors is a circle as shown in Figure 5.1.



(a) Data (b) PCA (c) PCA Whitening

Figure 5.1: This figure illustrates that the representations obtained only using eigenvectors do not preserve the information about the importance of the axes. Figures (b) and (c) show the reconstruction of the data obtained from PCA and only using the eigenvectors, respectively. (c) shows that the variance along both the axes are the same in the reconstruction of the ellipse from the representations obtained by not scaling the eigenvectors by the corresponding singular values.

# 6

# Laplacian in Reinforcement Learning

In this chapter we introduce the definition of the Laplacian in reinforcement learning proposed in Wu, Tucker, and Nachum, 2019. While Wu, Tucker, and Nachum, 2019 discussed the laplacian in continuous state spaces, the discussion here is in the finite state setting for the sake of clearer exposition.

Let $P^\pi$ be the transition probabilities according to the policy $\pi$, and $\rho$ be its stationary distribution which is assumed to exist. In Wu, Tucker, and Nachum, 2019, the Laplacian is defined as $L = I - D$, where $I$ is an identity operator and $D$ is given by $D(u,v) = \frac{1}{2}\frac{dP^\pi(s|u)}{d\rho(v)}|_{s=v} + \frac{1}{2}\frac{dP^\pi(s|v)}{d\rho(s)}|_{s=u}$. The matrix $D$ serves the dual purpose of being the weight $W$ matrix for the Laplacian $L = S - W$ where $S$ is the degree matrix $S_{ii} = \sum_j W_{ij}$, and the density of the transition distributions of the random walk on the graph.

In the finite state setting, we write $\mathbf{P}$ and $\mathbf{W}$ to denote the use of $D$ for transition matrix and weight matrix respectively. Let $\hat{P}^\pi$ denotes the transition probabilities of the time-reversed markov chain of policy $\pi$. The density $D(u,v)$ in Wu, Tucker, and Nachum, 2019 is given by

$$D(u,v) = \frac{1}{2}\frac{P^\pi(v|u)}{\rho(v)} + \frac{1}{2}\frac{P^\pi(u|v)}{\rho(u)} \tag{6.1}$$

$D(u,v)$ is integrated with respect to measure $\rho(v)$ and so, in the finite state setting, equation (6.1) is multiplied by $\rho(v)$ to obtain

$$\mathbf{P}_{uv} = \frac{1}{2}P^{\pi}(v|u) + \frac{1}{2}\frac{\rho(v)}{\rho(u)}P^{\pi}(u|v) \tag{6.2}$$

$$= \frac{1}{2}P^{\pi}(v|u) + \frac{1}{2}\hat{P}^{\pi}(v|u) \tag{6.3}$$

For obtaining $\mathbf{W}$, notice that $D(u,v)$ is integrated with respect to $\rho(u)$ and $\rho(v)$ in equation (2) of Wu, Tucker, and Nachum, 2019. Therefore, by multiplying equation (6.1) by $\rho(u)$ and $\rho(v)$ we obtain

$$\mathbf{W}_{uv} = \frac{1}{2}\rho(u)\rho(v)\frac{P^{\pi}(v|u)}{\rho(v)} + \frac{1}{2}\rho(u)\rho(v)\frac{P^{\pi}(u|v)}{\rho(u)} \tag{6.4}$$

$$= \frac{1}{2}\rho(u)P^{\pi}(v|u) + \frac{1}{2}\rho(v)P^{\pi}(u|v) \tag{6.5}$$

Note that the transition probabilities $\mathbf{P}$ is given by forward transition probabilities $P^{\pi}$ with 0.5 probability and the time-reversed $\hat{P}^{\pi}$ with 0.5 probability. Such a restriction is required since transition probabilities of a random walk on an undirected graph have to be reversible. However, in the RL settting the samples are collected only using $P^{\pi}$. Hence, we assume that $P^{\pi}$ is reversible. Note that this assumption is a special case of the transition matrix $\mathbf{P}$ given in equation (6.2). Hence, the definition of $\mathbf{P}_{uv}$ is given by $\mathbf{P}_{uv} = P^{\pi}(u|v)$ and $\mathbf{W}$ is given by $\mathbf{W_{uv}} = \rho(u)P^{\pi}(v|u)$.

# 7

# State Embeddings

In this chapter, we introduce the proposed embedding learning method that is suitable for goal-conditioned reinforcement learning. First, we introduce the objective function used to learn the embedding space. Next, we discuss the existence of this embedding space and its connection to the graph Laplacian. Finally, we discuss how the objective that we introduced is used to approximate this embedding space.

## 7.1 Proposed Objective Function

In order to be useful in goal-conditioned policies, the distances in the embedding space must correspond to a quantity that depends on the environment dynamics as experienced by the agent. So, we first define the distance between states $s_1$ and $s_2$ as half of the commute-time, which we call the *action distance*. We then propose to learn an embedding space where the distance between the embeddings of a pair of states correspond to the action distance between those states.

Formally, let $s_i, s_j \in \mathcal{S}$, and define the squared action distance $d^\pi(s_i, s_j)$ as:

$$d^\pi(s_i, s_j) = \frac{1}{2}m(s_j|s_i) + \frac{1}{2}m(s_i|s_j) \tag{7.1}$$

In general, the computation of $d^\pi(s_i, s_j)$ requires solving linear systems of equation or equivalently estimating the values of a set of goal-conditioned reward functions. Therefore, computing $d^\pi$ exactly is computation intensive. So, we propose to train a neural network to estimate $d^\pi$. We learn an embedding function $e_\theta$ of the state space, parameterized by vector $\theta$, such that the $p$-norm between a pair of state embeddings is close to the action distance between the corresponding states. The objective function used to train $e_\theta$ is:

$$\theta^* = \arg\min_\theta (||e_\theta(s_i) - e_\theta(s_j)||_p^q - d^\pi(s_i, s_j))^2 \tag{7.2}$$

In general, ensuring $d^\pi(s_i, s_j)$ is computed using equal proportion of $m(s_j|s_i)$ and $m(s_i|s_j)$ leads to practical difficulties. Hence, due to practical considerations, we redefine action distance to

$$d^\pi(s_i, s_j) = \frac{\rho(s_i)m(s_j|s_i)}{\rho(s_i) + \rho(s_j)} + \frac{\rho(s_j)m(s_i|s_j)}{\rho(s_i) + \rho(s_j)}$$

where $\rho$ is the stationary distribution of the Markov chain (we assume the stationary distribution exists) and $m(\cdot|\cdot)$ is estimated from the trajectories as follows

$$m(s_j|s_i) = E_{\tau\sim\pi, t\sim\bar{t}(s_i,\tau), t'=\min\{\bar{t}(s_j,\tau)\geq m\}} \left[|t - t^{'}|\right] \tag{7.3}$$

where $\bar{t}(s, \tau)$ is a uniform distribution over all temporal indices of $s$ in $\tau$ and the expectation is taken over trajectories $\tau$ sampled from $\pi$ such that $s_i$ and $s_j$ both occur in $\tau$. If $\pi$ is a goal-conditioned policy we also average over goals $g$ provided to $\pi$. In the next section we discuss the existence of the embedding space that preserves action distance (equation 7.1), its connection to the graph Laplacian, and a practical approach to approximate it. We call the embeddings learned using the proposed objective function the distance predictor as the distances in the embedding space corresponds to the action distance.

The distance predictor can be trained using the trajectories collected by the behavior policy during the policy-learning phase. We call this case the *on-policy* distance predictor. We emphasize that the on-policy nature of this distance predictor is independent of whether the policies or value functions are learned on-policy. While such an on-policy distance possesses desirable properties, such as a simple training scheme, it also has drawbacks. Both the behavior policy and goal distribution will change over time and thus the distances will be non-stationary.

An important subtlety with the on-policy distance estimators is the difficulty in setting the threshold $\epsilon$. Recall that in our setting, the goal is considered to be achieved and the episode terminated once $d^{\pi}(s, g) < \epsilon$, where $\epsilon$ is a threshold hyperparameter. This thresholding creates an $\epsilon$-sphere around the goal, with the episode terminating whenever the agent enters this sphere. The interaction between the $\epsilon$-sphere and the non-stationarity of the on-policy distance function causes a subtle issue that we dub the *expanding $\epsilon$-sphere* phenomenon, discussed in detail in Section 9.2.

An alternative approach to learning the distance function from the trajectories generated by the behavior policy is to apply a random policy for a fixed number of timesteps at the end of each episode. The states visited under the random policy are then used to train the distance function. Since the random policy is independent of the behavior policy, we describe the learned distance function as *off-policy* in this case. The stationarity of the random policy helps in overcoming the expanding $\epsilon$-sphere phenomenon of the on-policy distance predictor.

## 7.2 Existence of the Embedding Space

A random walk on undirected weighted graphs defines a Markov chain and hence, Markov chain based distances are useful to define dissimilarities between nodes. Specifically, Fouss, Pirotte, and Saerens, 2005 define the similarity of nodes in weighted undirected graphs using commute times and show the existence of a Euclidean embedding space where the distance between embeddings of nodes $i$ and $j$ correspond to the square root of the average commute time between them, $\sqrt{n(i,j)}$. Such a Euclidean space can be computed using the pseudo-inverse of the graph Laplacian $L^{\dagger}$, and the representation of a node $i$ is the $i^{th}$ row of $Q\Lambda^{\frac{1}{2}}$ where the columns of $Q$ are the eigenvectors of $L^{\dagger}$ and $\Lambda$ is a diagonal matrix of the corresponding eigenvalues.

This is also the solution given by cMDS when $D^{(2)}(X)_{ij} = n(i,j)$ or when $B_{ij} = L^{\dagger}_{ij}$ as explained below.

(Fouss, Pirotte, and Saerens, 2005; Fouss, Pirotte, Renders, et al., 2007) show that the average first passage time and the average commute times can be expressed as

$$m(j|i) = \sum_{k=1}^{n} (l^{\dagger}_{ik} - l^{\dagger}_{ij} - l^{\dagger}_{jk} + l^{\dagger}_{jj}) d_{kk} \tag{7.4}$$

and

$$n(i,j) = V_G(l^{\dagger}_{ii} + l^{\dagger}_{jj} - 2l^{\dagger}_{ij}) \tag{7.5}$$

respectively, where $V_G = \sum_{i=1}^{n} D_{ii}$ is the volume of the graph. $n(i,j)$ can be expressed as

$$\begin{aligned} n(i,j) &= V_G(l^{\dagger}_{ii} + l^{\dagger}_{jj} - l^{\dagger}_{ij} - l^{\dagger}_{ij}) \\ &= V_G(e_i - e_j)^T L^{\dagger}(e_i - e_j) \\ &= V_G(e_i - e_j)^T Q\Lambda Q^T(e_i - e_j) \\ &= V_G(e_i - e_j)^T Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}} Q^T(e_i - e_j) \\ &= V_G(e_i - e_j)^T Q\Lambda^{\frac{1}{2}T}\Lambda^{\frac{1}{2}} Q^T(e_i - e_j) \\ &= V_G(x_i - x_j)^T(x_i - x_j) \end{aligned} \tag{7.6}$$

where $x_k$ is $\Lambda^{\frac{1}{2}} Q^T e_k = (e_k^T Q \Lambda^{\frac{1}{2}})^T$, the column vector corresponding to the $i^{th}$ row of $Q\Lambda^{\frac{1}{2}}$. The $L2$ distance in this embedding space between nodes $(i,j)$, $||x_i - x_j||$, corresponds to $\sqrt{n(i,j)}$ up to a constant factor $\frac{1}{\sqrt{V_G}}$, termed euclidean commute time distance (ECTD).

Since the orthogonal complement of the null-space of the linear transformation is invertible and a vector of all ones $\mathbf{1}$ spans the nullspace of $L$, psedudo-inverse of the Laplacian is intuitively given by $L^{\dagger} = (L + \frac{1}{n}\mathbf{1}\mathbf{1}^T)^{-1} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$. Hence, it is easy to verify that $L^{\dagger}$ is symmetric and the nullspace of $L^{\dagger}$ is also $\mathbf{1}$. Therefore, $L^{\dagger}$ is double centered. The resultant matrix of the double centering operation $JAJ$ on any matrix

$A$ are given by

$$\begin{aligned}&(JAJ)_{ij}\\&= a_{ij} - \frac{1}{n}\sum_{k=1}^{n} a_{ik} - \frac{1}{n}\sum_{m=1}^{n} a_{mj} + \frac{1}{n^2}\sum_{m=1}^{n}\sum_{k=1}^{n} a_{mk}\end{aligned}$$

Using these two facts, it is easy to verify that

$$\begin{aligned}&-\frac{1}{2}(JNJ)_{ij}\\&= V_G(L^{\dagger}_{ij} - \frac{1}{n}\sum_{k=1}^{n} L^{\dagger}_{ik} - \frac{1}{n^2}\sum_{k=1}^{n} L^{\dagger}_{kk} + \frac{1}{n^2}\sum_{m=1}^{n}\sum_{k=1}^{n} L^{\dagger}_{mk})\\&= V_G L^{\dagger}_{ij} \quad \text{(since } L^{\dagger} \text{ is double centered)}\end{aligned}$$

where $N$ is the matrix of commute times among all pairs of nodes.

Hence, the solution to the embedding space that preserves ECTD given by equation (7.6) is the same as the one provided by classical MDS by taking $N$ as $D^{(2)}$ or $L^{\dagger}$ as $B$. Finally, since that the eigenvectors of $L$ and $L^{\dagger}$ are the same and the non-zero eigenvalues of $L^{\dagger}$ is the inverse of the corresponding non-zero eigenvalues of $L$. This shows that simply using the eigenvectors of the Laplacian is insufficient to obtain an embedding where the distances are preserved. An empirical comparision of the embeddings obtained from eigenvectors and the scaled eigenvectors of the Laplacian to compute distances is shown in 9.4 in a tabular setting.

## 7.3 Approximation of the Embedding Space

Even though the embedding space with the desired properties exists, neither the matrix of pairwise commute times nor $L^{\dagger}$ are available in the RL setting and hence cMDS cannot be applied. Hence, we approximate the embedding space using metric MDS (equation 5.2) where $\delta_{ij} = \sqrt{n(i,j)}$. Metric MDS provides flexibility in the choices of $f$ and weights $w_{ij}$. We choose $f$ to be the square function i.e $f(x) = x^2$. This choice of $f$ stems from practical considerations - it is easier to estimate the average first passage times between $m(j|i)$ and $m(i|j)$ independently using the trajectories in RL compared to estimating the commute time $n(i,j) = \sqrt{m(j|i) + m(i|j)}$. Hence, our objective function is of the form

$$\sigma_S(X) = \sum_{i<j} w_{ij}(||x_i - x_j||_2^2 - \frac{1}{2}(m(j|i) + m(i|j)))^2 \tag{7.7}$$

where $x_i$ is the embedding of node $i$. By noting that the quantity in equation (7.7) is the mean squared error (MSE) and MSE computes the mean of the labels drawn from a distribution for each input, we can write equation (7.7) as

$$\sigma_S(X) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(||x_i - x_j||_2^2 - k(i,j))^2 \tag{7.8}$$

where $k(i,j) = 1m(j|i) + (1-1)m(i|j)$ and $1 \sim Bern(0.5)$. Thus, sampling $m(j|i)$ and $m(i|j)$ in equal proportion provides the required averaging. In practice, however, it is simpler to sample $1 \sim Bern\left(\frac{\rho(u)}{\rho(u)+\rho(v)}\right)$ than $Bern(0.5)$, motivating our definition in equation (7.3).

For the choice of $w_{ij}$, the discussion of Wu, Tucker, and Nachum, 2019 in Chapter 6 suggests $w_{ij} = \rho(i)P(j|i)$. A better choice would be consider the probabilities over all the time steps instead of the single step transition probability. Hence, we define the weight for the pair $(x_i, x_j)$ by $w_{ij} = \rho(i) + \rho(j)$, the relative mass of visiting $i$ and $j$ together.

We note that the mean first passage times $m(.|.)$ are not available and has to be estimated from the trajectories. In order to do so, first note that $m(j|i) = \sum_{t=0}^{\infty} P_{ij}^t t$ where we use the notation $P_{ij}^t$ to denote the probability of going from state $i$ to $j$ in exactly $n$ steps for the first time. As a result, the objective function is of the form

$$\sigma(X) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(||x_i - x_j||_2^2 - k)^2 \tag{7.9}$$

where $k \sim P_{ij}^t$. The quantities in equation (7.9) can be obtained from trajectories drawn under a fixed policy, thus providing a practical approach to learn the embedding as given in equation (7.2) with $q = 2$. When the quantities $k$ are obtained from the trajectories, in addition to the weights $w_{ij}$ on $(i, j)$, each $k$ is weighted by $P_{ij}^k$. This emphasizes the shorter distances for each pair of $(i, j)$. As show in 9.5, $q$ in equation (7.2) provides a mechanism to control the trade-off between the larger and smaller distances by considering $q$ as a hyperparameter. Thus, setting $w_{ij}^k$ to $\rho(i)P_{ij}^k + \rho(j)P_{ji}^k$ provides a practically convenient set of weights and the trade-off it induces can be mitigated as desired by changing $q$.

To summarize, Fouss, Pirotte, and Saerens, 2005 shows the existence of the embedding space where the distance between points in the embedding space corresponds to the square root of the expected commute times between nodes. Therefore, the existence of the embedding space which preserves the action distance (equation 7.1) is also guaranteed. The computation of this embedding space requires quantities that are unavailable in the reinforcement learning setting. Hence, we propose to approximate the embedding space using metric MDS, and provide a set of values for the weights $w_{ij}$ that are meaningful and practically feasible to compute. In 9.5 we discuss the effect of few choices of $f$ and how it can be used to control the trade-off between faithfully representing smaller and larger distances.

# 8

# Automatic Curriculum Generation

The characterization of reinforcement learning tasks in terms of sub-goals is an attractive alternative to hand-designing reward functions in complex tasks. The need for domain knowledge is partly alleviated by automatically learning the distance functions as described in the previous chapter. In this chapter, we address the question *where do the goals come from?* in order to realize the full potential of goal-conditioned reinforcement in removing the requirement for domain knowledge. In addition to automating the generation of goals, it is desirable to tailor the new goals to the current abilities of the agent to generate a curriculum for the agent. With this motivation, we present a simple approach to automatically generate goals that do not require domain knowledge.

## 8.1 Related Work

Generating a curriculum that is adapted to the agent's current abilities has been proposed in (Sukhbaatar et al., 2018) and (Florensa, Held, Geng, et al., 2018). (Sukhbaatar et al., 2018) proposes an approach that requires two 'minds' for the agent: one to generate goals and the other to achieve those goals, called *Alice* and *Bob* respectively. The reward function for *Bob* is designed to ensure that *Bob* reaches its goal in the least number of steps. On the other hand, the reward function of *Alice* is designed in such a manner so that *Alice's* reward is maximized when the goals given to *Bob* by *Alice* are neither too easy nor too difficult. The reward for *Alice* is maximized when the number of steps taken by *Bob* is $t_M - t_A$ or more, where $t_M$ and $t_A$ are the maximum horizon for the task and the number of steps taken by *Alice* to reach the goal respectively. The reward is minimized when *Bob* reaches the goal in at most as many steps as *Alice* required. A more straightforward and interpretable characterization of the task difficulty is proposed in (Florensa, Held, Geng, et al., 2018).

In (Florensa, Held, Geng, et al., 2018), maintains a working set of goals. The working set of goals are the goals of intermediate difficulty; they are neither too easy nor too difficult to achieve. Formally, given hyperparameters $R_{min}, R_{max} \in (0, 1)$, a goal $g$ is considered to be a *Goal of Intermediate Difficulty* (GOID) if $R_{min} < V_{\pi_\theta}(s_0, g) < R_{max}$, where $V_{\pi_\theta}(s_0, g)$ is the undiscounted return. The reward functions for these goal achieving tasks are sparse binary reward functions; the agent is given a reward 1 when the goal is achieved and 0 otherwise. Hence, the value function $V_{\pi_\theta}(s_0, g)$ can be interpreted as the probability of reaching $g$ from $s_0$. The algorithm alternates between the *policy learning* and *goal-selection* phases. In the *policy-learning* phase, the agent is trained to achieve goals sampled uniformly from the working set of goals. In the *goal-selection* phase, the goals that were sampled in the *policy learning* phase are labeled as easy, *GOID* and difficult goals. The labeling of these goals is either done estimating the probability of achieving those goals successfully either by performing multiple rollouts for each goal or using the trajectories from the *policy learning* phase. Upon labeling the goals, (Florensa, Held, Geng, et al., 2018) train a generative model, a variant of the Generative Adversarial Network (Goodfellow et al., 2014) known as LSGAN (Mao et al., 2017), with the *GOID* goals as the positive examples and the easy and hard goals as the negative examples. This generative model is the working set of goals.

## 8.2 Goal Generation with Action Noise

Using a generative model as proposed in (Florensa, Held, Geng, et al., 2018), introduces additional complexity and may generate infeasible goals. When a learned distance function is used to determine whether the specified goal has been achieved, the incorrect predictions made by the distance predictor due to incorrect generalization may cause infeasible goals to be erroneously marked as achieved. As a result, the generative model is trained to generate a similar set of infeasible goals. This results in wasting samples attempting to train the agent to reach infeasible goals.

In this section, we propose an alternative to using a generative model to generate new goals that does not suffer from the problem of generating infeasible goals. We propose an approach to generate new goals in the following manner. When the specified goals have been achieved in the *policy learning* phase, the a set of states are generated by applying random actions for a fixed horizon without terminating the episode. The candidate set of *GOID* goals are then generated by uniformly sampling from this collection of states. We describe how this candidate set of goals are integrated into the current work set of goals in the next section. (Florensa, Held, Wulfmeier, et al., 2017) uses a similar approach to generate goals, but in the context of generating a set of start states growing outward from a fixed goal state. The part of the trajectory generated under the random policy is not used for policy optimization.

Generating goals in this manner avoids the requirement of arbitrary environment resets, which is a requirement in (Florensa, Held, Wulfmeier, et al., 2017). This makes our method applicable in real-world tasks where arbitrary environment resets are not possible. This approach of goal generation combines with the off-policy distance predictor as the distance predictor can be trained with these trajectories. This results in a curriculum learning procedure for goal-conditioned policies that requires minimal domain knowledge and works nicely with the learned distances.

## 8.3 Goal Buffer

In this section, we describe the mechanism to store the working set of goals and how the new candidate *GOID* goals are integrated into the current working set of goals. First, we begin with describing the desired properties of the mechanism to store the *GOID* goals: i) *Non-Stationarity*: We require a non-stationary process to generate goals that are suited to the agent's current capabilities, and ii) *Stability*: Avoid replacing the entire collection of current working set of *GOID* goals with a new set working after every *policy learning* phase. Otherwise, this will cause the working set of goals to only contain the goals that are more likely under the marginal distribution of current policy. These two properties are satisfied by using a fixed size queue with the restriction that only a random subset of the queue are replaced with the new candidate goals.

# 9

# Experiments

The objective of the experiments is two fold. In the first of experiments, we determine whether the requirement for domain knowledge in the form of distance function can be removed by learning the distance functions using the proposed embedding space. In the second set of experiments, we determine whether the automatic goal generation using only the knowledge of the action space is effective. In the first set of experiments, we experiment in the following 3 scenarios: i) when the goal space is known apriori - $(x, y)$ goal space, ii) the goal space is unknown but only a subset of the state-space is the desired set of goals, and iii) the goal space is unknown and the entire state space is the space of desired goals. These experiments are presented in the control setting i.e when the agent learns the policy. To determine whether the proposed approach to scales to higher dimensional inputs where the goal space is not embedded in the state space, we use test our method with the image inputs. This experiment is performed in the batch setting. In the goal generation experiments, the goals are generated in the $(x, y)$ goal space.

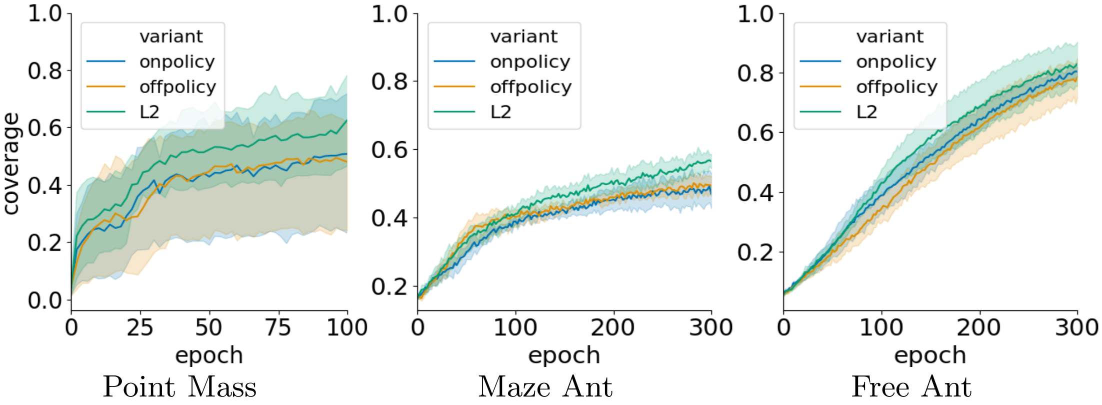
The experiments are presented in 3 Mujoco Todorov, Erez, and Tassa, 2012 environments. The MuJoCo environments are robotics simulations with continuous state and action spaces with complex dynamics. The first environment is a maze that resembles the shape of 8. A point-mass agent placed in this maze learns to navigate the maze using a two-dimensional action space. The state space of the point-mass agent is a 4-dimensional vector representing the agent's $(x, y)$ coordinates along with the velocities in those directions. The actions control the acceleration of the agent in these two directions. In the other two environments, the agent is a four-legged robot that resembles an ant. The state space for the ant agent is 41-dimensional denoting the $(x, y, z)$ position of the center of mass of the torso along with the angles and angular velocities of the joints of the legs. The action-space is 8-dimensions which control the torque applied to each of the joints. We test the ant agent in two environments, the first environment is a 'U' shaped maze and the second environment is an open room. Further discussion of our environments can be found in Florensa, Held, Geng, et al., 2018 and Duan et al., 2016.

## 9.1 Control Tasks

We demonstrate the on-policy and the off-policy approaches to learning the distance predictor. The algorithm alternates between the distance learning and the policy learning phases. In the distance learning phase, the samples collected either on-policy or off-policy is used to train a neural network to learn an embedding space is used to estimate the distances. In the policy learning phase, a policy is learned using policy gradient methods or value function based methods. In the case of off-policy distance predictor, additionally a batch of states are collected using a random policy after the specified goals have been achieved. Using the insight that it is sufficient for the distance predictor to be useful for the regions that are more likely under the current policy, we are able to bootstrap our distance predictor. The distance predictor is learned using a siamese network Bromley et al., 1993 one hidden layer with 64 hidden units. The output embedding size is 20. To evaluate the progress of the agent, we measure a quantity called the *coverage*. The *coverage* measures the fraction of the maze that can be successfully reached. To measure the coverage, the maze is discretized into grids and the goal is set to the center of the grid. During evaluation, the threshold is set in the $L2$ space. The threshold values are set to 0.3 and 1 for the point-mass and the ant agents respectively. To measure the probability of success for each grid, a fixed number of rollouts is used.

### 9.1.1 XY Goal Space

In this experiment, the goal space is the $(x, y)$ coordinates of the agent's center-of-mass (i.e. a subspace of the state space). The results in this experiments are compared to the baseline approach of Florensa, Held, Geng, et al., 2018. The distance function in the baseline is the $L2$ distance. The threshold for termination is the same in the training and evaluations phases in the baseline. As shown in Figure 9.1, the learned distances perform comparably to the baseline without using domain knowledge.



Point Mass

Maze Ant

Free Ant

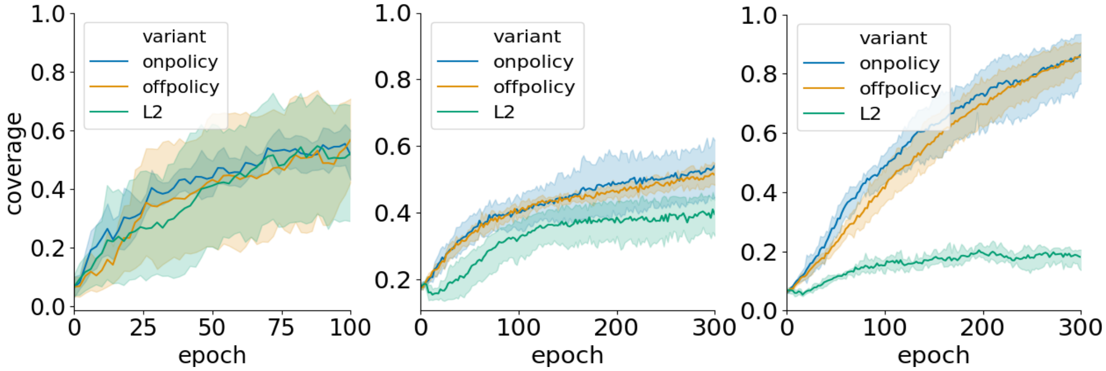
Figure 9.1: Coverage plots for the $(x, y)$ goal space

.

### 9.1.2 Full State Space as Goal Space

In this experiments, we consider the settings where the goal space is not known apriori. Therefore, the distance function is learned using the full state space. With the point-mass agent, the desired set of goals is the same the entire state space. Thus, the agent has to achieve all possible configurations of the state space. Since the ant agent has a higher dimensional state space, we only consider a subset of the state space as the goal space. This subset of desired goals correspond to the stable pose of the ant centered on each of the $(x, y)$ positions on the grid. The distance predictor is trained using the entire states in both of these settings. The results in these two settings are shown in 9.2.

The progress of the point-mass agent is diminished compared to the $(x, y)$ goal space as the agent has to achieve the specified configuration of the velocities for the position. The progress by ant agent suggests that the learned distances are able to generalize to unseen states as the distance predictor is trained on the states visited by a policy and it is unlikely to encounter the stable states which are used as the goals during the evaluation phase.



Point Mass Maze Ant Free Ant

Figure 9.2: Coverage plots for the full goal space

.

Figure 9.3 shows the distances estimated by the distance predictor in the Point Mass and Maze Ant environments in $(x, y)$ and full goal spaces respectively. Figure 9.4 visualizes the distances estimated on a set of reference states in the Maze Ant environment in full goal space. We observe that in all experiments, including the set with the $(x, y)$ goal space, the on-policy and the off-policy methods for training the distance predictor performed similarly. In section 9.2 we study the qualitative difference between the on-policy and off-policy methods for training the distance predictor.

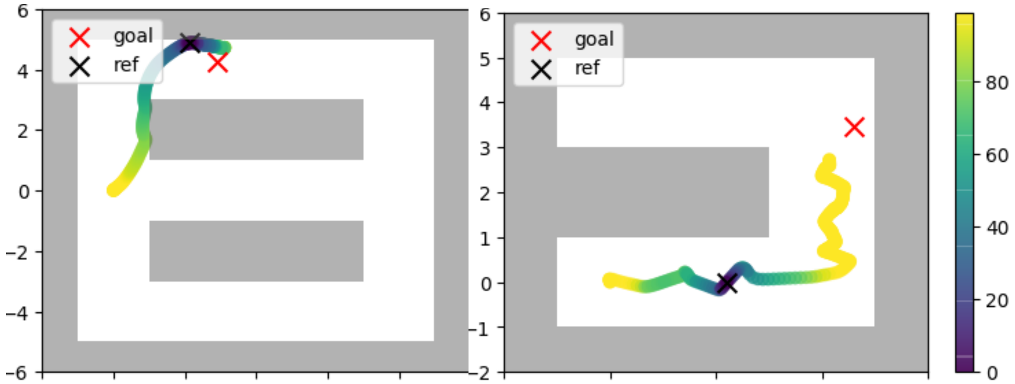


Figure 9.3: Predicted action distance between a reference state and states along a trajectory in Point Mass (left) and Maze Ant (right).

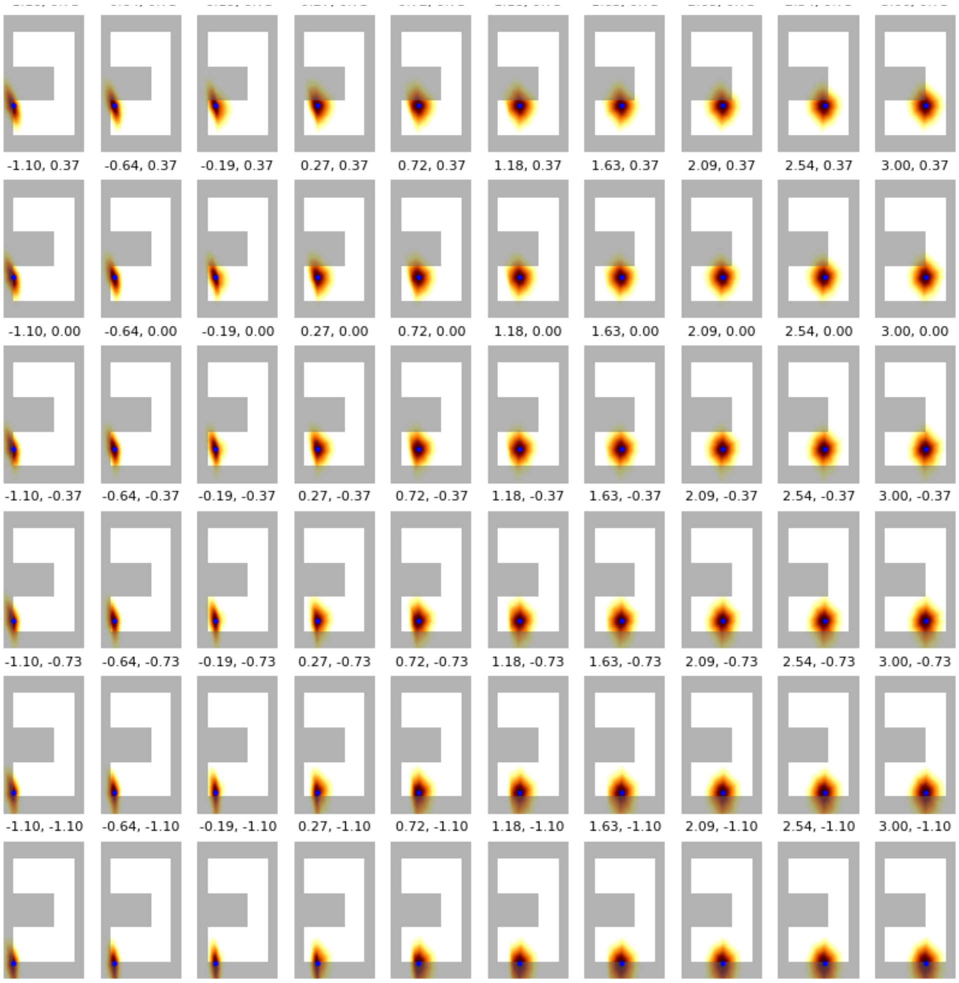


Figure 9.4: Predicted action distance between stabilized reference states and all other states in the Ant Maze environment in full goal space. Each subplot uses a different reference state marked a blue dot. The reference states are chosen near the starting states.

### 9.1.3 Hyperparameters

The distance predictor is a MLP with 1 hidden layer and 64 hidden units and ReLU activation. We initialize distance predictor by training on $100,000$ samples collected according to the randomly initialized policy for 50 epochs. The starting position is not randomized. In subsequent iterations of our training procedure the MLP is trained for one epoch. The learning rate is set to $1e$-5 and mini-batch size is 32. The distance predictor is trained after every policy optimization step using either either off-policy or the on-policy samples. The embeddings are 20-dimensional and we use 1-norm distance between embeddings as the predicted distance.

The GoalGAN architecture and its training procedure and the policy optimization procedure in our experiments are similar to Florensa, Held, Geng, et al., 2018. Similar to the distance predictor, GoalGAN is trained initially with samples generated by a random policy. The GAN generator and discriminator have 2 hidden layers with 264 and 128 units respectively with ReLU non-linearity and the GAN is trained for 200 iterations after every 5 policy optimization iterations. A component-wise gaussian noise of mean zero and variance 0.5 are added to the output of the GAN similar to Florensa, Held, Geng, et al., 2018. The policy network has 2 hidden layers with 64 hidden units in each layer and *tanh* non-linearity and is optimized using TRPO Schulman et al., 2015 with a discount factor of 0.99 and $GAE$ of 1. The $\epsilon$ value was set to 1 and 60 with $L2$ and the learned distance, respectively, in the Ant environments and 0.3 and 20 for $L2$ and learned distance, respectively, for the Point Mass environments. In order to determine the first occurrence of a state, we use a threshold of $1e-4$ in the state space. This value is not tuned.

The hyperparameter for $\epsilon$ and the learning rate were determined by performing grid search with $\epsilon$ values $50, 60$ and 80 (Maze Ant) and $20, 30$ (Point Mass) and the learning rates of $1e$-3, $1e$-4, $1e$-5 in the off policy setting. For the sake of simplicity we use the same $\epsilon$ for the on-policy and off-policy distance predictors in our experiments.

All the plots show the mean and the confidence interval of 95% for all our experiments using 5 random seeds. Our implementation is based on the github repository for Florensa, Held, Geng, et al., 2018, located at https://github.com/florensacc/rllab-curriculum.

## 9.2 Expanding $\epsilon$-sphere

We discuss the qualitative differences between the on-policy and off-policy schemes for training the distance predictor. Since the goal is considered achieved when the agent is within the $\epsilon$-sphere of the goal, the episode is terminated when the agent reaches the boundary of the $\epsilon$-sphere. As the learning progresses and the agent learns a shortest path to the goal, the agent only learns a shortest path to a state on the boundary of the $\epsilon$-sphere of the corresponding goal. In this scenario, the path to the goal $g$ from any state within the $\epsilon$-sphere of $g$ under the policy conditioned on $g$ need not necessarily be optimal since such trajectories are not seen by the policy conditioned on that specific goal $g$. However, the number of actions required to reach the goal $g$ from the states outside the $\epsilon$-sphere along the path to the goal decreases as a result of learning a shorter path due to policy improvement. Therefore, as the learning progresses until an optimal policy is learned, the number of states from which the goal $g$ can be reached in a fixed number of actions increases, thus resulting in an increasing the volume of the $\epsilon$-sphere centered on the goal for a fixed action distance $k$, when using on-policy samples to learn the distance predictor.

This phenomenon is empirically illustrated in the top row in Fig. 9.5. For a fixed state $g$ near the starting position, the distance from all other states to $g$ is plotted. The evolution of the distance function over iterations shows that for any fixed state $s$, $d^{\pi}(s, g)$ gets smaller. Equivalently, the $\epsilon$-sphere centered on $g$ increases in volume. In contrast, the bottom row in Fig. 9.5 illustrates the predictions made by an off-policy distance predictor; in that case, the dark region is almost always concentrated densely near $g$, and the volume of the $\epsilon$-sphere exhibits significantly less growth.

Since the agent does not receive training for states that are within the $\epsilon$-sphere centered at a goal $g$, it is desirable to keep the $\epsilon$-sphere as small as possible. One way to do this would be to employ an adaptive algorithm for choosing $\epsilon$ as a function of $g$ and the agent's estimated skill at reaching $g$; as the agent gets better at reaching $g$, $\epsilon$ should be lowered. We leave the design of such an algorithm for future work, and propose the off-policy scheme as a practical alternative in the meantime. We note that this phenomenon is not observed in the visualization in the full goal space, possibly due to the stabilization of the ant during evaluation.

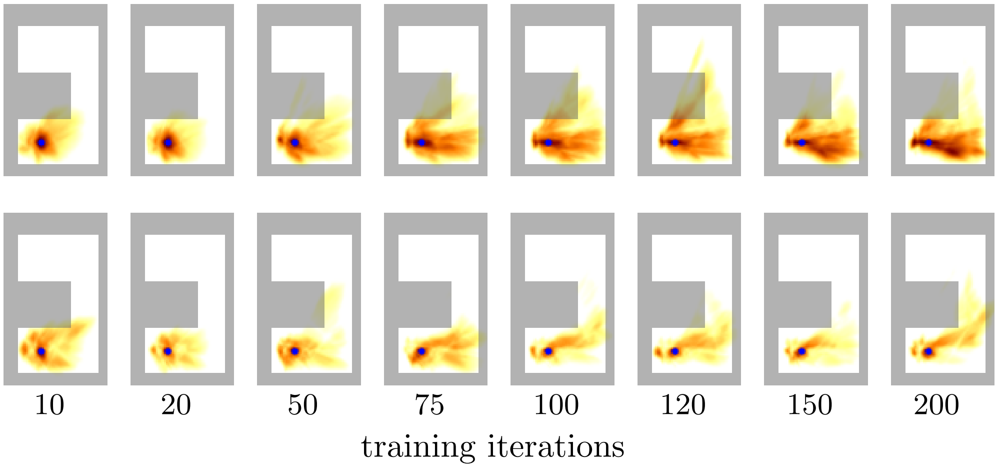


Figure 9.5: Predictions of the distance predictor trained with on-policy (top) and off-policy (bottom) samples in Maze Ant with $(x, y)$ goal space illustrating how the predictions evolve over time. Darker colors indicate smaller predicted distance and the small blue dot indicates the reference state.

## 9.3 Pixel Inputs

To study whether the proposed method of learning a distance function scales to high-dimensional inputs, we evaluate the performance of the distance predictor using pixel representation of the states. This experiment is performed in the batch setting. Similar to the pretraining phase of Wu, Tucker, and Nachum, 2019, the embeddings are trained using sample trajectories collected by randomizing the starting states of the agent. Each episode is terminated when the agent reaches an unstable position or 100 time steps have elapsed. Figure 9.6 shows the distance estimates of the distance predictor in this setting. For qualitative comparison, we also experimented with the approach proposed in Wu, Tucker, and Nachum, 2019 with various choices of hyper-parameters.

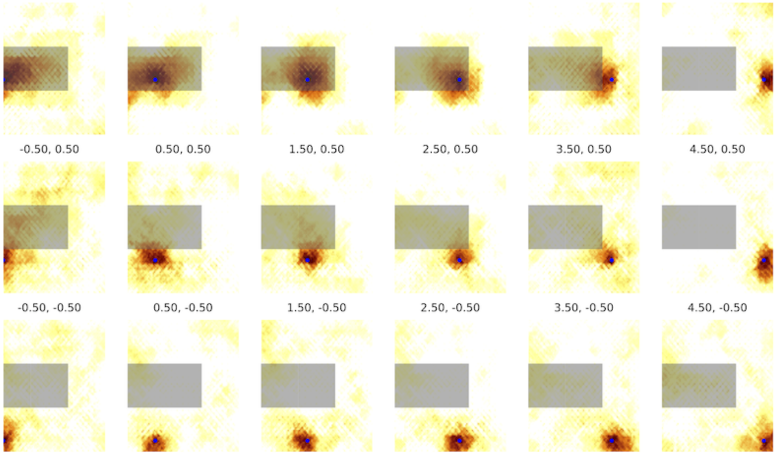


Figure 9.6: Pixel space (batch setting) respectively. Closer states are darker. Each subplot uses a different reference state shown as a blue dot. Full heatmap is shown in Figure 9.10.

The distance predictor is neural network with 4 convolution layers with 64 channels and kernel size of 3 in each layer with strides $(1, 2, 1, 2)$ followed by 2 fully connected layers with 128 units in each layer and the output layer has 32 units. We used *relu* non-linearity along with batchnorm. The learning rate was set to 5$e$-5 and Adam Kingma and Ba, 2014 optimizer. The network was trained for 50 epochs with $p = 2$ and $q = 1$. The top-down view of the maze ant is shown in Figure 9.7. The results

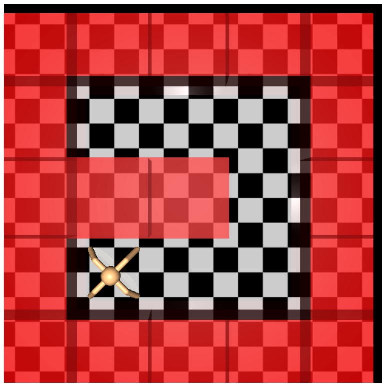
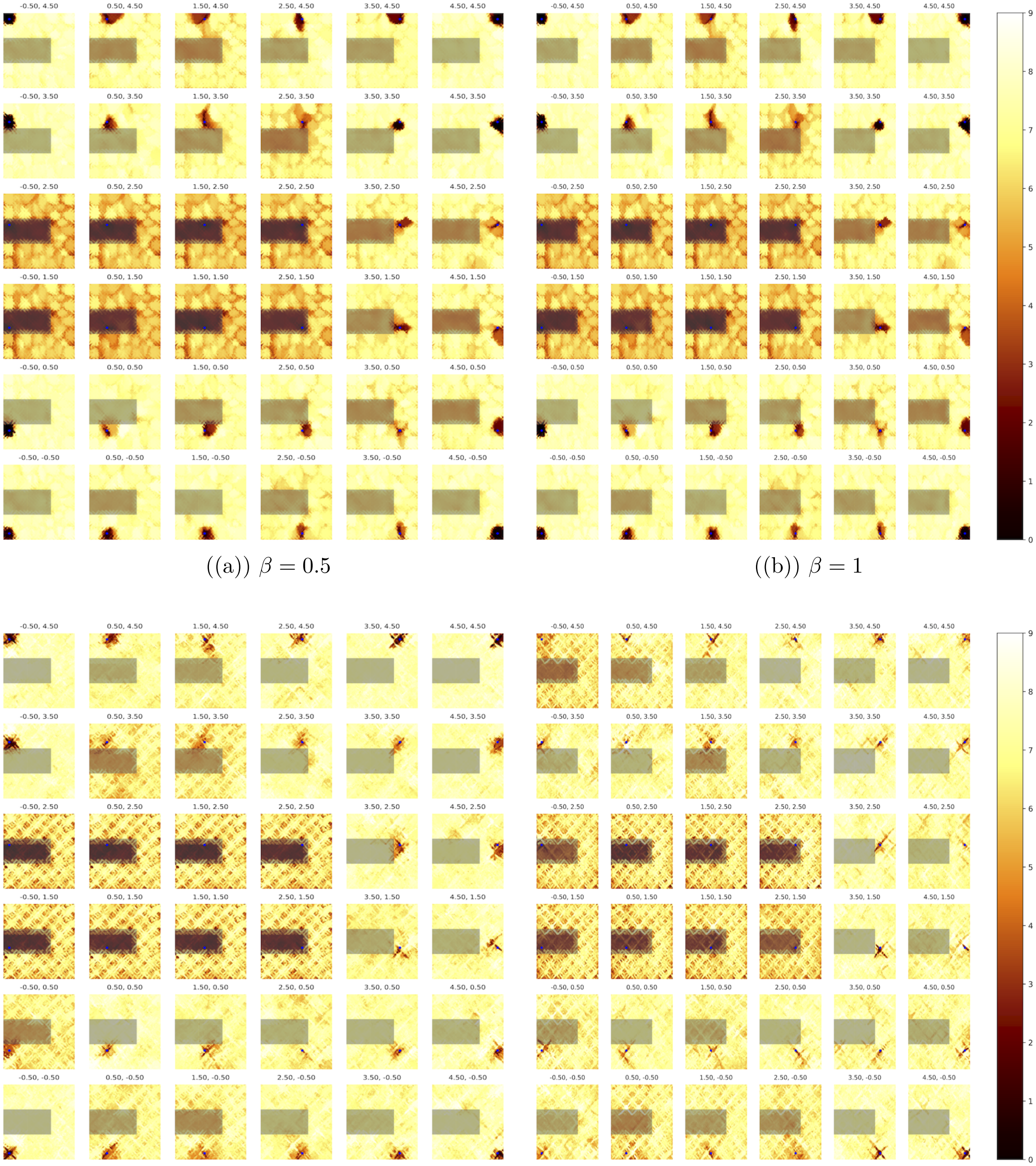


Figure 9.7: Top-down view of the Maze Ant. The RGB image scaled to $32 \times 32$ is the input in the pixel tasks.

with the approach of Wu, Tucker, and Nachum, 2019 are shown in Figures 9.8 and 9.9. The training setup is the same as described in section 9.3. We uniformly sample the states in each trajectory (hence, $\lambda$ is approximately 1). As $\beta$ is increased (better approximation of the spectral objective objective), the points that are predicted close in almost all of the reference points resembles the body of the ant in the stable position (Figure 9.7) centered on nearby points.

When used in an online setting, the negative sampling in Wu, Tucker, and Nachum, 2019 could be problematic especially when bootstrapping (without arbitrary environment resets). In contrast, our objective function does not require negative sampling and only uses the information present within a trajectory, making it more suitable for online learning.



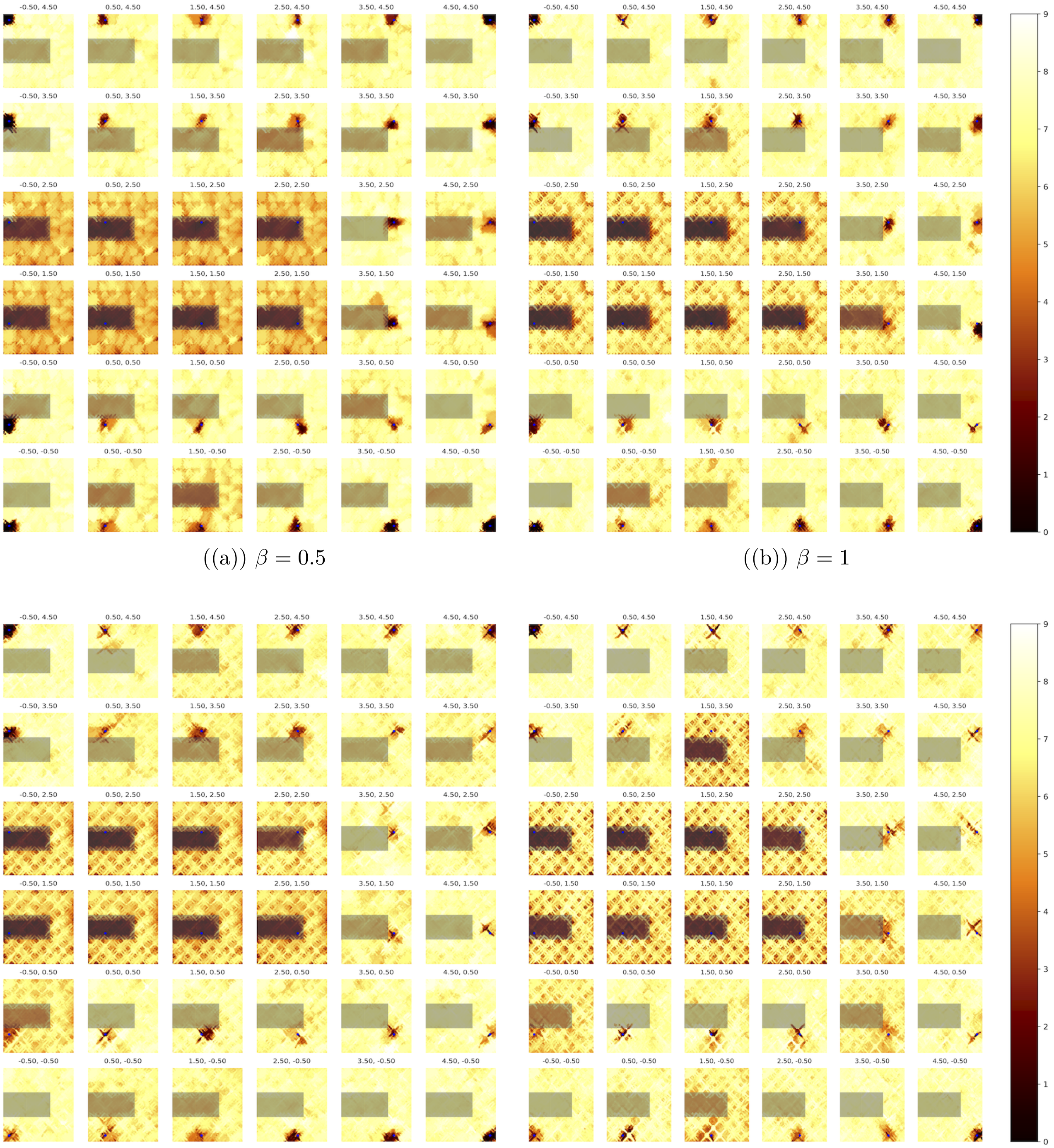
((a)) $\beta = 0.5$

((b)) $\beta = 1$

((c)) $\beta = 2$

((d)) $\beta = 6$

Figure 9.8: Laplacian in RL using pixel inputs. Higher $\beta$ better approximates spectral graph drawing objective. $LR = 1e - 4$



((a)) $\beta = 0.5$

((b)) $\beta = 1$

((c)) $\beta = 2$

((d)) $\beta = 6$

Figure 9.9: Laplacian in RL using pixel inputs. Higher $\beta$ better approximates spectral graph drawing objective. $LR = 5e - 5$

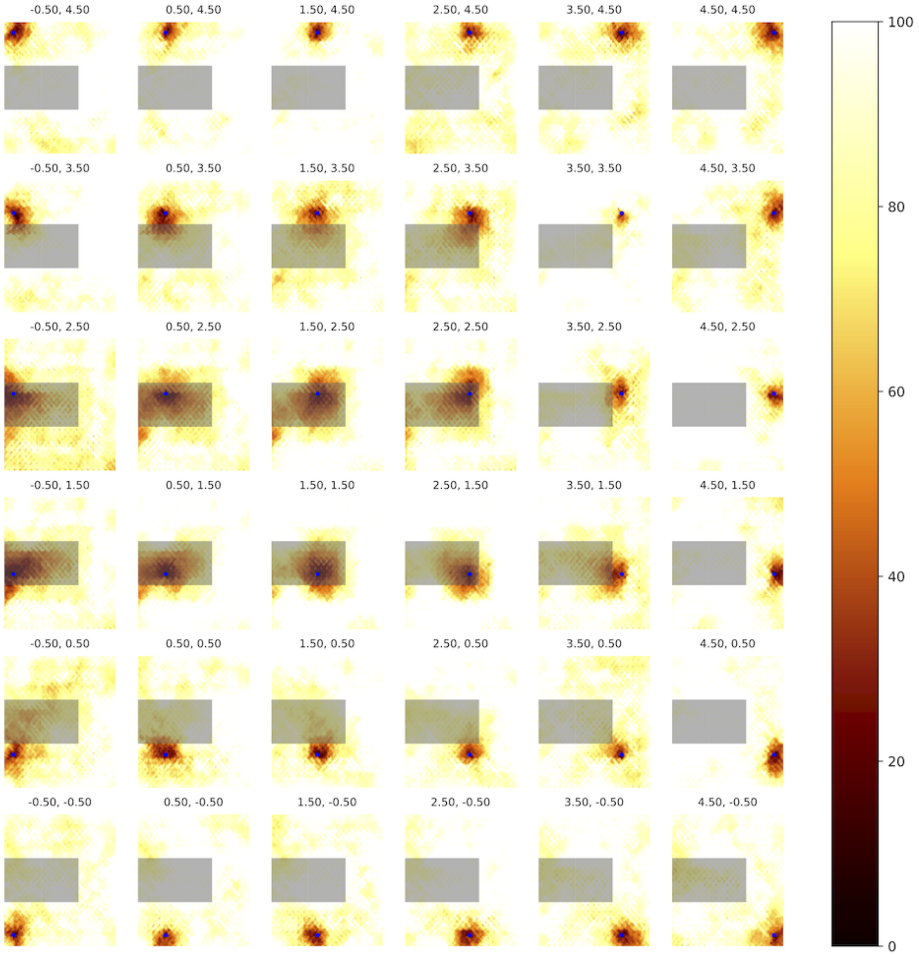


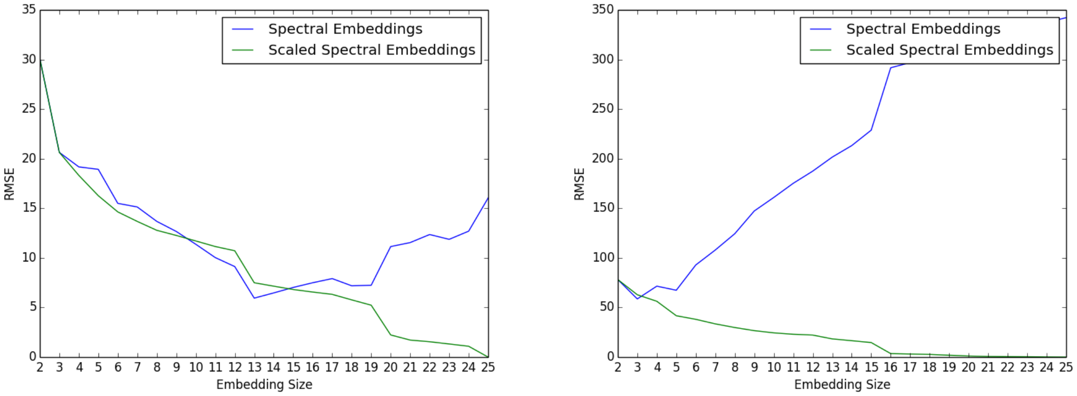
Figure 9.10: Full heatmap of our approach using pixel inputs. Our approach does not require negative sampling.

## 9.4 Comparison of Spectral Embeddings and Scaled Spectral Embeddings

In this section we compare the embeddings obtained using the eigenvectors of the Laplacian (spectral embedding) and the embeddings obtained by scaling the eigenvectors by the inverse of square root of the corresponding eigenvalues (scaled spectral embedding). We perform this comparison in two mazes with 25 states. A transition from each node to the 4 neighbours in the north, south, east and west directions are permitted, with equal probabilities in the first maze (figure 9.12) and, north and east with 0.375 and south and west with 0.125 probabilities in the second maze (figure 9.13). We choose the state corresponding to $(2, 2)$ as the center and the distance from this state to all the other states are plotted. The first two columns in figures 9.12 and 9.13 correspond to spectral embedding and scaled spectral embedding respectively. The third column corresponds to the ground truth $\sqrt{n(i, j)}$ computed analytically from the mean first passage times computed as $M_{ij} = \frac{Z_{jj} - Z_{ij}}{\rho(j)}$ where $Z = (I - \mathbf{P} + \mathbf{1}\rho^T)^{-1}$. The distances produced by spectral embedding are multiplied by the ratio of maximum distance of scaled spectral embedding and the maximum distance of spectral embedding to produce a similar scale for visualization. The Laplacian is given by $L = D - W$ with $W$ given by equation (6.4) and $D$ is the diagonal matrix with stationary probabilities on the diagonal.

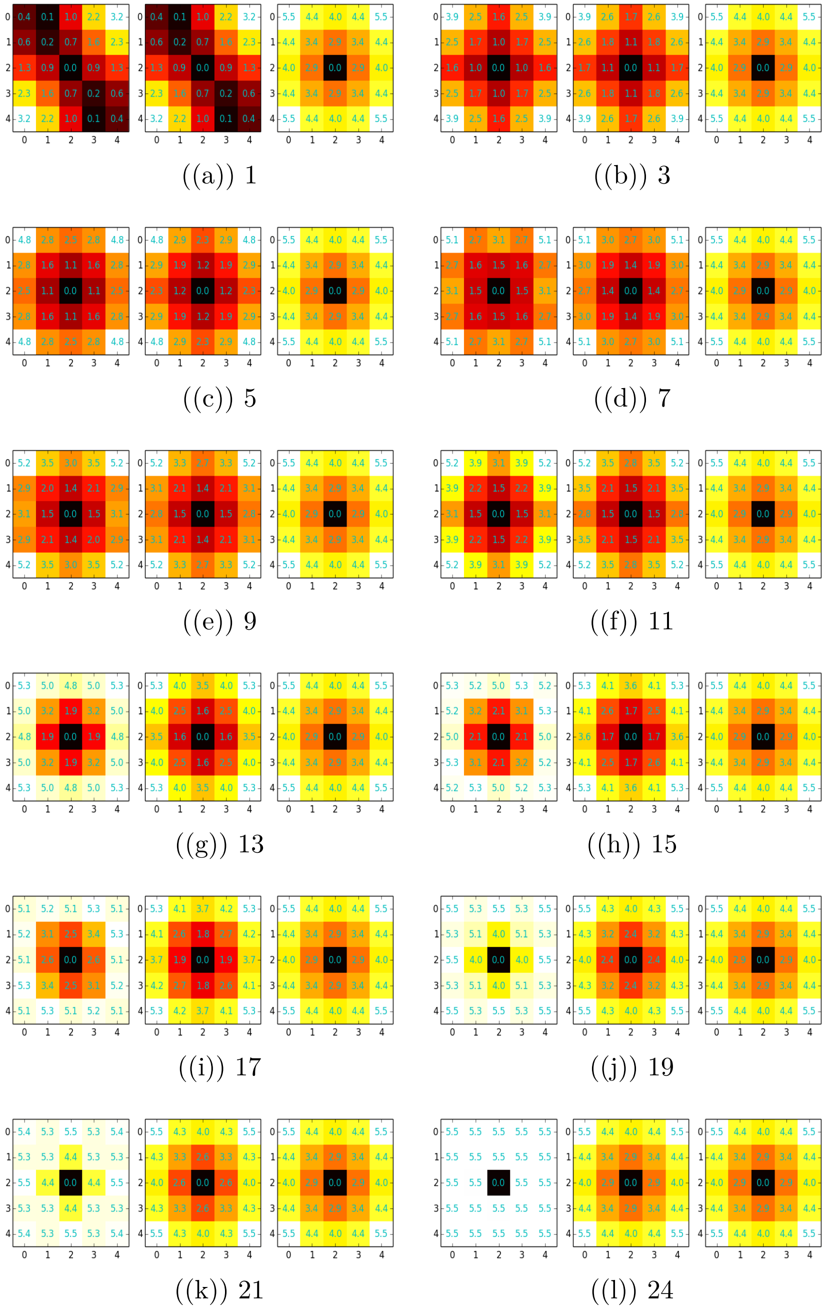
As seen in both the Figures 9.12 and 9.13, increasing the size of the embedding dimensions of scaled spectral embeddings better approximates the commute-time distance. The same cannot be said for the approximation given by spectral embeddings. In the first maze (Figure 9.12), the approximation gets better until the embedding size of 13 and then deteriorates. When the objective is to find an lower-dimensional approximation of the state space, the choice of the embedding size is treated a hyperparameter and hence one might be tempted to consider this as hyperparameter tuning. However, as shown in the second maze (Figure 9.13), when the environment dynamics are not symmetric, the effect is pronounced to the extent that there is no single choice of the embedding size for the spectral embeddings that best preserves the distances of the nearby states and the faraway states. Even in the case when the embedding size is 3, the spectral embeddings are markedly different from scaled spectral embeddings as the states $(0,3),(0,4),(1,4)$ are marked equidistant from the reference state by the spectral embeddings. A comparison of the RMSE error of spectral embeddings and scaled spectral embedding is provided in Figure 9.11. The difference between spectral embeddings and scaled spectral embeddings is blurred in the uniform transitions case since the eigenvalues are similar. In the non-uniform transitions case, the difference between spectral embeddings and scaled spectral embeddings are evident since the eigenvalues are very dissimilar. Note that similar eigenvalues means the corresponding dimensions have similar weights; scaling by a (approximately) constant - scaled spectral embedding approach - doesn't cause significant difference.

We finally note that our objective is not find a low-dimensional embedding of the state space but to find an embedding that produces a meaningful distance estimate. Scaled spectral embeddings are appropriate for this purpose since the accuracy of the distance estimates improves monotonically with an increase in the number of dimensions.



((a)) ((b))

Figure 9.11: The square root of commute times from the center state $(2, 2)$ to all other states is taken as the reference. RMSE of distances obtained from spectral and scaled spectral embeddings is plotted for (a) Maze with uniform transition probabilities and (b) Maze with non-uniform transition probabilities. The distances obtained from spectral embeddings are scaled by the ratio of maximum distance from scaled spectral embeddings and that of spectral embeddings; this is done to map the spectral embeddings to the same scale as square root of commute times.



((a)) 1

((b)) 3

((c)) 5

((d)) 7

((e)) 9

((f)) 11

((g)) 13

((h)) 15

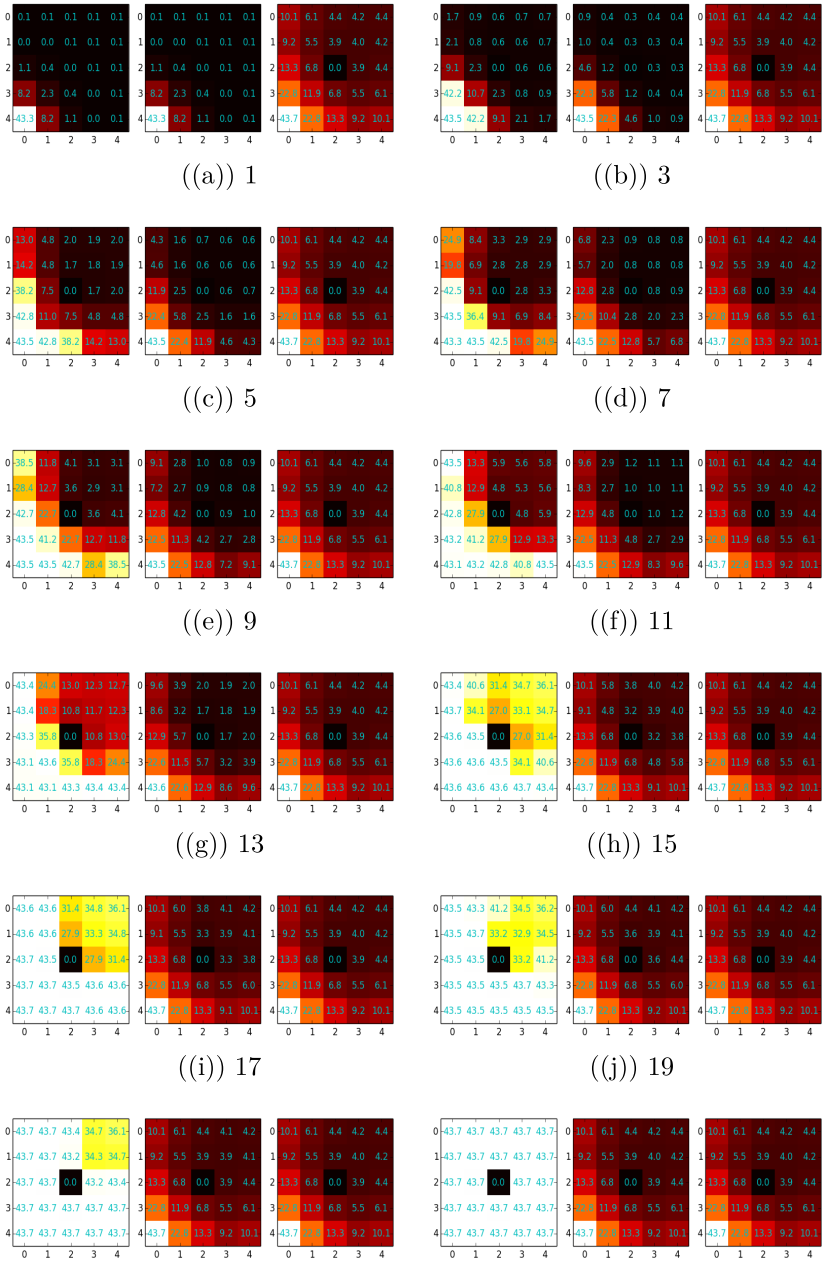
((i)) 17

((j)) 19

((k)) 21

((l)) 24

Figure 9.12: Visualization of the distance from state $(2, 2)$ to all other states using different embedding sizes produced by spectral embedding(first column) and scaled spectral embedding(second column) along with the ground-truth computed analytically(third column). The transition from each state to its neighbours are uniformly random. Spectral embedding and scaled spectral embeddings are reasonably similar upto 11 dimensions. The distance estimate of spectral embeddings deteriorates as many 'less informative' dimensions corresponding to large eigenvalues are added and weighed with as much importance as the 'more informative' dimensions given by small eigenvalues.



((a)) 1

((b)) 3

((c)) 5

((d)) 7

((e)) 9

((f)) 11

((g)) 13

((h)) 15

((i)) 17

((j)) 19

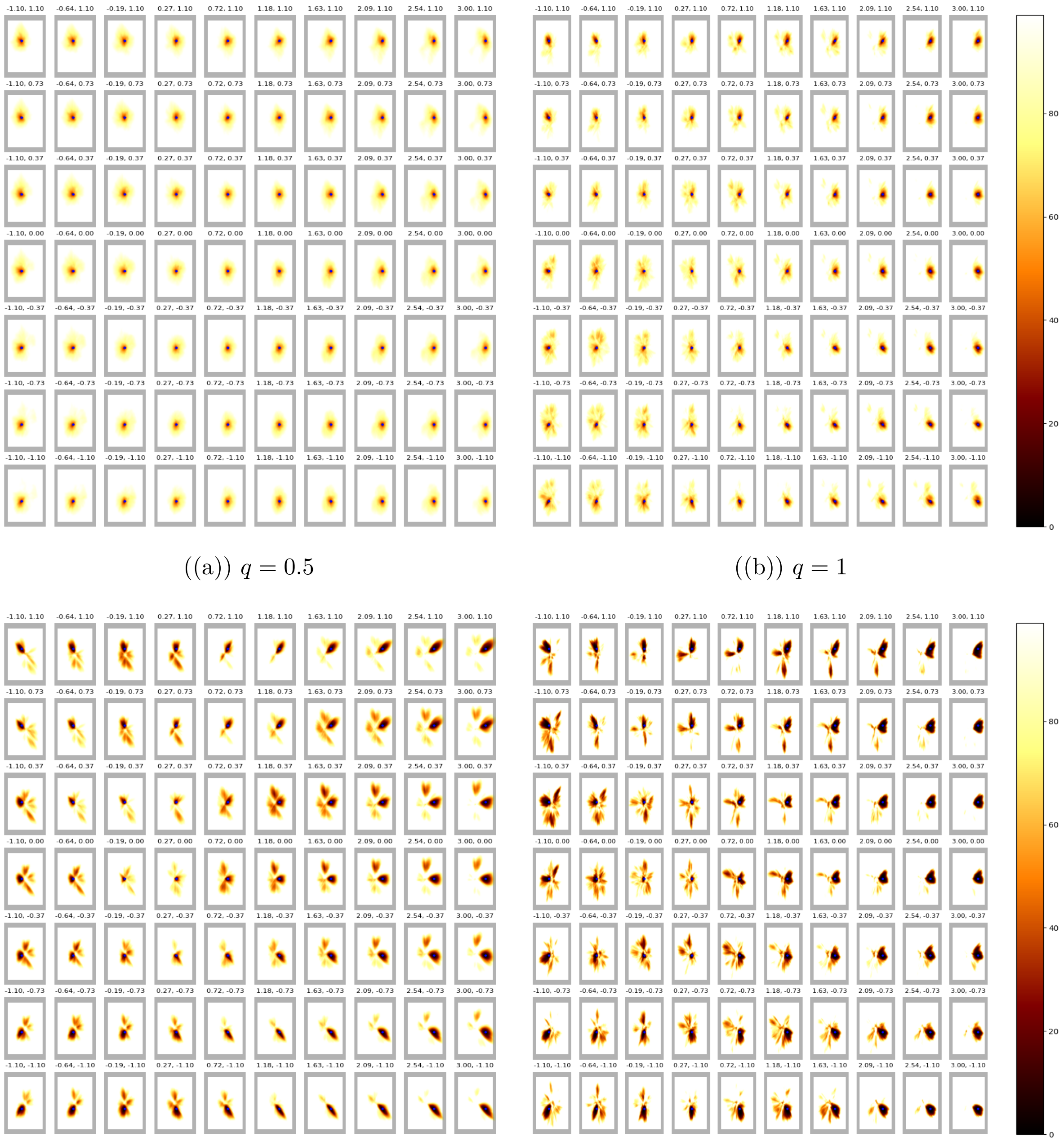
((k)) 21

((l)) 24

Figure 9.13: The transition from a state to its neighbours in north and east are with probability 0.375 and in south and west are with probability 0.125. The choice of addition of a dimension to spectral embedding presents a trade-off between preserving the previous estimate and improving the estimate of a closer state. In contrast, the scaled spectral embedding improves with the addition of every dimension.

## 9.5 Effect of $q$

We empirically demonstrate that increasing $q$ increases the effect of larger distances. In order to obtain the same scale of measurements, the distance in the embedding space are raised to the power $q$. We show the effect of $q$ for values $0.5, 1, 2, 4$. It is clearly evident that increasing $q$ increases the radius and the granularity of distance between points that are near and far are lost. The reason for is suggested in Borg and Groenen, 2006 (section 11.3). Increasing $q$ increases the weight given to larger distances. For instance, when $q = 2$, the stress is approximately $4\delta_{ij}^2(\delta_{ij} - d_{ij})^2$ where $\delta_{ij} = \sqrt{n(i,j)}^{\frac{1}{2}}$ since the target dissimilarity can be rewritten as ${\delta_{ij}^{\frac{1}{q}}}^{q}$. The $q^{th}$ root of the $\delta_{ij}$ decreases the granularity of the difference between near and far states and the larger weighting term results in overweighting sporadic examples causing an increase in radius. Similarly, it can be shown that when $q = 4$, the stress is given by $16\delta_{ij}^6(\delta_{ij} - d_{ij})^2$ where $\delta_{ij} = \sqrt{n(i,j)}^{\frac{1}{4}}$.



((a)) $q = 0.5$

((b)) $q = 1$

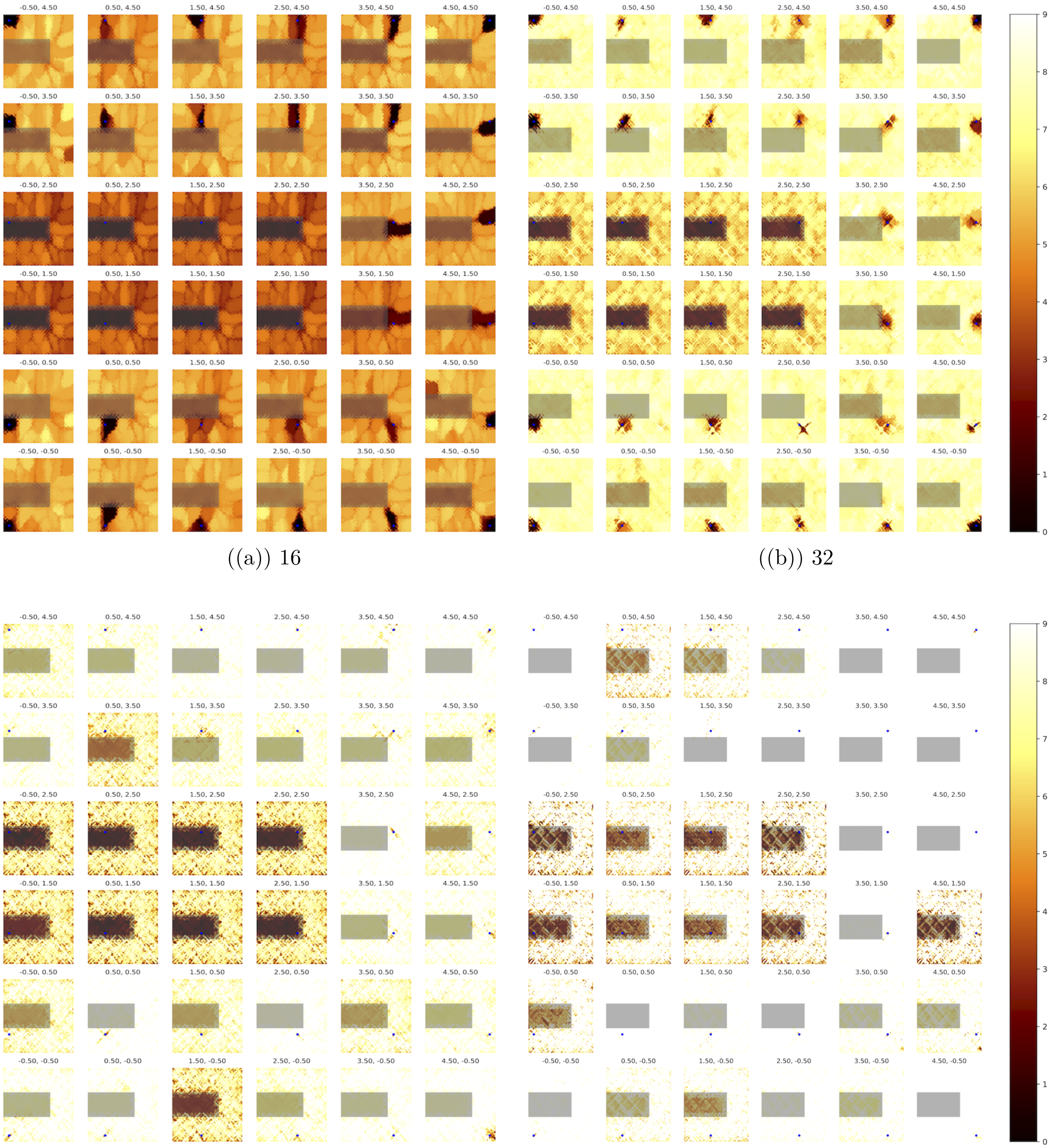
((c)) $q = 2$

((d)) $q = 4$

Figure 9.14: We study the effect of $q$ on the radius and degree of closeness of the ant agent in the free maze. This shows that $q$ is easy to tune and provides a straightforward mechanism to scale the distances. A similar effect is also observed in the other environments and with pixel inputs.

## 9.6 Effect of dimension size

The discussion in section 9.4 suggests that increasing the embedding dimensions monotonically increases the quality of distance estimates in the scaled spectral embeddings unlike spectral embeddings. We show this phenomenon using the pixel inputs in Figures 9.15 Wu, Tucker, and Nachum, 2019 and 9.16 (our method).



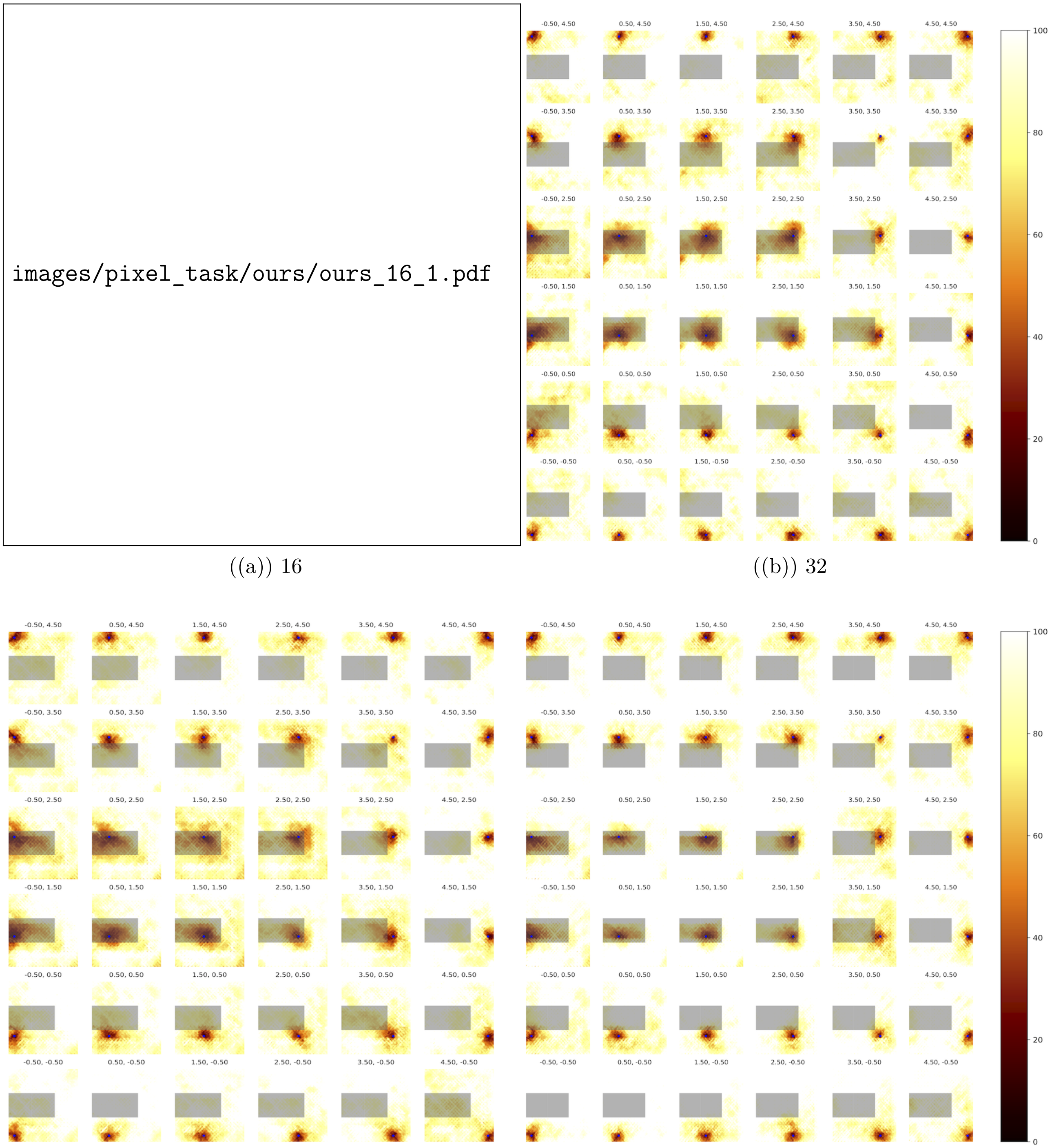
((a)) 16

((b)) 32

((c)) 64

((d)) 128

Figure 9.15: The effect of size of embeddings using the objective of Laplacian in RL with pixel inputs for $LR = 1e - 4$ and $\beta = 1$. The quality of the approximation of the distance drops after 32 dimensions.



((a)) 16

((b)) 32

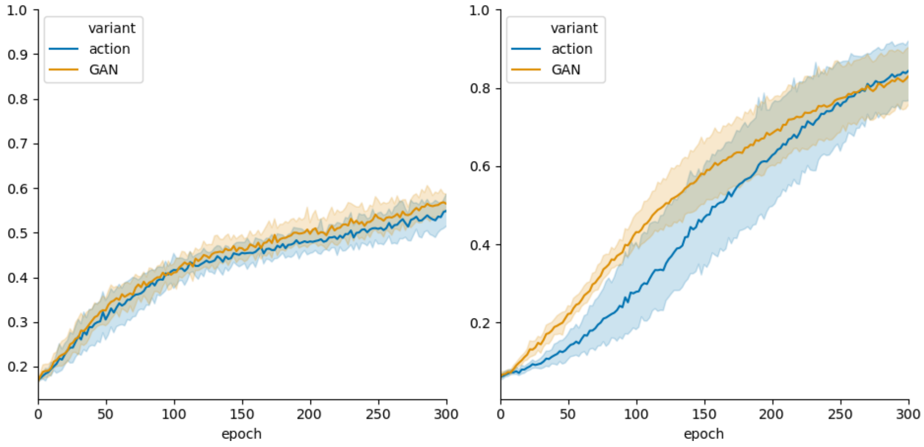
((c)) 64

((d)) 128

Figure 9.16: Increasing the embedding size in our approach improves the distance estimates.

## 9.7 Generating Goals Using Action Noise

We compare the proposed goal generation approach with GoalGAN Florensa, Held, Geng, et al., 2018 in the $(x, y)$ goal space using the $L2$ distance. The results shown in Fig. 9.17 demonstrates that the performance of our approach is on-par with to that of GoalGAN while not requiring the additional complexity introduced by the GAN. The evolution of the working set of goals maintained by our algorithm for Maze Ant is visualized in Fig. 9.18. Though our approach requires additional environment interactions, it does not necessarily have a higher sample complexity compared to GoalGAN in the case of indicator reward functions. This is because the goals generated by GoalGAN are not guaranteed to be feasible (unlike our approach); trajectories generated for unfeasible goals will receive 0 reward and will not contribute to learning.



((a)) Maze Ant ((b)) Free Ant

Figure 9.17: Comparing the proposed goal generation algorithm against GoalGAN.

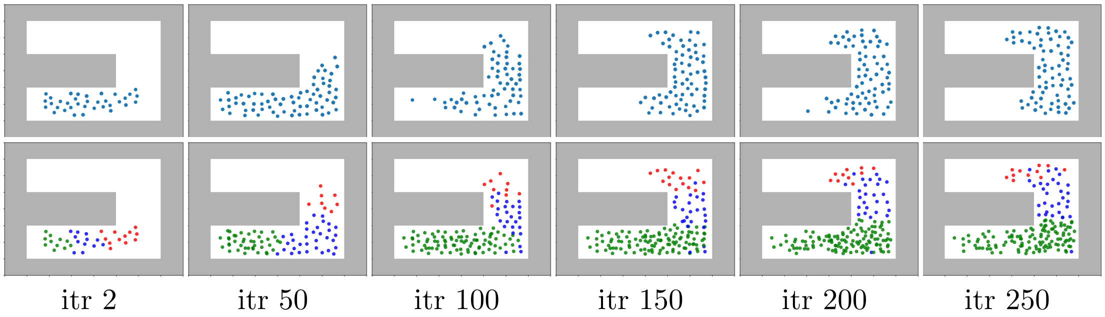


Figure 9.18: Evolution of the goals generated by our goal generation approach (top). A sample of goals so-far encountered (bottom), color-coded according to estimated difficulty: green are easy, blue are GOID and red are hard.

# 10

# Conclusion and future work

We have presented an approach to learn an embedding space for goal states that approximates a theoretically motivated embedding, with the property that the distance between states in the embedding space is proportional to the average commute time between them. We discussed the connection between this embedding space and classical multi-dimensional scaling, and the unnormalized graph Laplacian of the MDP's state transition graph. As seen in the experiments, the approximation of this embedding space using metric MDS leads to emphasis on states that are close. We illustrated how this effect can be mitigated by tuning the algorithm's hyperparameter, $q$. We used the learned embedding space as a goal space in which distance is calculated for goal-conditioned policies and we discussed the phenomenon of expanding the $\epsilon$-sphere. The proposed goal generation procedure results in an effective curriculum generation procedure for the tasks considered. The experiments performed on complex simulated robotic tasks demonstrate the usefulness of our approach. We believe that our approach makes a significant step towards building reinforcement learning agents with minimal domain knowledge.

In this thesis, we have studied the usefulness of the proposed embedding only for goal-conditioned policies. We suspect that the benefits of our method could go beyond this setup: as a representation suitable for credit assignment, as a way to provide a good measure for prioritized sweeping or prioritized sampling from the replay buffer, which has an interesting connection with mutual information. We leave these directions, along with further empirical investigations, for future work.

# Bibliography

Amari, Shun-Ichi (Feb. 1998). “Natural Gradient Works Efficiently in Learning”. In: *Neural Comput.* 10.2, pp. 251–276.

Baranes, Adrien and Pierre-Yves Oudeyer (Jan. 2013). “Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots”. In: *Robot. Auton. Syst.* 61.1, pp. 49–73.

Bellemare, Marc et al. (2016). “Unifying Count-Based Exploration and Intrinsic Motivation”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., pp. 1471–1479.

Bengio, Yoshua et al. (2009). “Curriculum Learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, pp. 41–48.

Borg, Ingwer and Patrick Groenen (June 2006). “Modern Multidimensional Scaling: Theory and Applications”. In: *Journal of Educational Measurement* 40, pp. 277–280.

Brémaud, Pierre (1999). *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. Berlin; New York: Springer-Verlag Inc.

Bromley, Jane et al. (1993). “Signature Verification Using a “Siamese” Time Delay Neural Network”. In: *Proceedings of the 6th International Conference on Neural*

*Information Processing Systems*. NIPS'93. Denver, Colorado: Morgan Kaufmann Publishers Inc., pp. 737–744.

Burda, Yuri, Harri Edwards, Deepak Pathak, et al. (2019). "Large-Scale Study of Curiosity-Driven Learning". In: *ICLR*.

Burda, Yuri, Harrison Edwards, Amos Storkey, et al. (2019). "Exploration by random network distillation". In: *International Conference on Learning Representations*.

Duan, Yan et al. (20–22 Jun 2016). "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1329–1338.

Florensa, Carlos, David Held, Xinyang Geng, et al. (Oct. 2018). "Automatic Goal Generation for Reinforcement Learning Agents". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 1515–1528.

Florensa, Carlos, David Held, Markus Wulfmeier, et al. (13–15 Nov 2017). "Reverse Curriculum Generation for Reinforcement Learning". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, pp. 482–495.

Fouss, Francois, Alain Pirotte, Jean-Michel Renders, et al. (Mar. 2007). "Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation". In: *IEEE Trans. on Knowl. and Data Eng.* 19.3, pp. 355–369.

Fouss, Francois, Alain Pirotte, and Marco Saerens (2005). "A Novel Way of Computing Similarities Between Nodes of a Graph, with Application to Collaborative Recommendation". In: *Proceedings of the 2005 IEEE/WIC/ACM International*

*Conference on Web Intelligence*. WI '05. Washington, DC, USA: IEEE Computer Society, pp. 550–556.

Gallier, Jean and Jocelyn Quaintance (2017). *Algebra, Topology, Differential Calculus, and Optimization Theory For Computer Science and Machine Learning*.

Gansner, Emden R., Yehuda Koren, and Stephen North (2004). "Graph Drawing by Stress Majorization". In: *Proceedings of the 12th International Conference on Graph Drawing*. GD'04. New York, NY: Springer-Verlag, pp. 239–250.

Ghojogh, Benyamin, Fakhri Karray, and Mark Crowley (2019). "Eigenvalue and Generalized Eigenvalue Problems: Tutorial". In: *ArXiv* abs/1903.11240.

Goodfellow, Ian et al. (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2672–2680.

Graves, Alex et al. (2017). "Automated Curriculum Learning for Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, pp. 1311–1320.

Jaderberg, Max et al. (2017). "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: *International Conference on Learning Representations*.

Kaelbling, Leslie Pack (1993). "Learning to Achieve Goals". In: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. Chambery, France: Morgan Kaufmann.

Kamada, T. and S. Kawai (Apr. 1989). "An Algorithm for Drawing General Undirected Graphs". In: *Inf. Process. Lett.* 31.1, pp. 7–15.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Kingma, Diederik P and Max Welling (2014). "Auto-encoding variational bayes". In: *International Conference on Learning Representations*.

Kobourov, Stephen G. (2012). “Spring Embedders and Force Directed Graph Drawing Algorithms”. In: *CoRR* abs/1201.3011.

Koren, Yehuda (2003). “On Spectral Graph Drawing”. In: *Proceedings of the 9th Annual International Conference on Computing and Combinatorics*. COCOON’03. Big Sky, MT, USA: Springer-Verlag, pp. 496–508.

Kruskal, J. B. (Mar. 1964). “Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis”. In: *Psychometrika* 29.1, pp. 1–27.

Lillicrap, Timothy P. et al. (2016). “Continuous control with deep reinforcement learning”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun.

Luxburg, Ulrike (Dec. 2007). “A Tutorial on Spectral Clustering”. In: *Statistics and Computing* 17.4, pp. 395–416.

Mao, X. et al. (Oct. 2017). “Least Squares Generative Adversarial Networks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2813–2821.

Mnih, Volodymyr et al. (Feb. 2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540, pp. 529–533.

Nachum, Ofir et al. (2019). “Near-Optimal Representation Learning for Hierarchical Reinforcement Learning”. In: *International Conference on Learning Representations*.

Nair, Ashvin et al. (2018). “Visual Reinforcement Learning with Imagined Goals”. In: *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*. NIPS’18. Montr&#233;al, Canada: Curran Associates Inc., pp. 9209–9220.

Narvekar, Sanmit, Jivko Sinapov, and Peter Stone (2017). “Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. Melbourne, Australia: AAAI Press, pp. 2536–2542.

Ng, Andrew Y., Daishi Harada, and Stuart J. Russell (1999). “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping”. In: *Proceedings of the Sixteenth International Conference on Machine Learning.* ICML ’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 278–287.

Sammon, J. W. (May 1969). “A Nonlinear Mapping for Data Structure Analysis”. In: *IEEE Trans. Comput.* 18.5, pp. 401–409.

Schaul, Tom et al. (2015). “Universal Value Function Approximators”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1312–1320.

Schulman, John et al. (July 2015). “Trust Region Policy Optimization”. In: *Proceedings of the 32nd International Conference on Machine Learning.* Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1889–1897.

Silver, David, Aja Huang, et al. (Jan. 2016). “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, et al. (2018). “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: 362.6419, pp. 1140–1144.

Sukhbaatar, Sainbayar et al. (2018). “Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play”. In: *International Conference on Learning Representations.*

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book.

Sutton, Richard S., Doina Precup, and Satinder Singh (Aug. 1999). “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In: *Artif. Intell.* 112.1–2, pp. 181–211.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033.

Venkattaramanujam, Srinivas, Eric Crawford, et al. (2019). “Self-supervised Learning of Distance Functions for Goal-Conditioned Reinforcement Learning”. In: *CoRR* abs/1907.02998.

Venkattaramanujam, Srinivas, Riashat Islam, and Doina Precup (2019). “Automatic Curriculum Generation via Task Perturbations in Reinforcement Learning”. In: *TARL workshop, ICLR 2019*.

Wu, Yifan, George Tucker, and Ofir Nachum (2019). “The Laplacian in RL: Learning Representations with Efficient Approximations”. In: *International Conference on Learning Representations*.