Self-supervised Learning of Distance Functions for Goal-Conditioned Reinforcement Learning

Anonymous Authors[1]

Abstract

A crucial requirement of goal-conditioned policies is to be able to determine whether the goal has been achieved. Having a notion of distance to a goal is thus a crucial component of this approach. However, it is not straightforward to come up with an appropriate distance, and in some tasks, the goal space may not even be known a priori. In this work we learn, in a self-supervised manner, a distance-to-goal estimate which is computed in terms of the average number of actions that would need to be carried out to reach the goal. In order to learn the distance estimate, we propose to learn an embedding space such that the distance between points in this space corresponds to the square-root of the average number of timesteps required to go from the first state to the second and back, i.e. the commute time between the states. We discuss why such an embedding space is guaranteed to exist and provide a practical method to approximate it in the online reinforcement learning setting. Experimental results in a number of challenging domains demonstrate that our approach can greatly reduce the amount of domain knowledge required by existing algorithms for goal-conditioned reinforcement learning.

1. Introduction

Reinforcement Learning (RL) is a framework for training agents to interact optimally with an environment. Recent advances in RL have led to algorithms that are capable of succeeding in a variety of environments, ranging from video games with high-dimensional image observations (Mnih et al., 2013; 2015) to continuous control in complex robotic tasks (Lillicrap et al., 2016; Schulman et al., 2015). Meanwhile, innovations in training powerful function approximators have all but removed the need for hand-crafted state representation, thus enabling RL methods to work with minimal human oversight or domain knowledge. However, one component of the RL workflow that still requires significant human input is the design of the reward function that the agent optimizes.

One way of alleviating this reliance on human input is by allowing the agent to condition its behavior on a provided goal (Kaelbling, 1993; Schaul et al., 2015), and training the agent to achieve (some approximation of) all possible goals afforded by the environment. A number of algorithms have recently been proposed along these lines, often making use of curriculum learning techniques to discover goals and train agents to achieve them in a structured way (Narvekar et al., 2017; Florensa et al., 2018). At the end of this process, the agent is expected to be able to achieve any desired goal.

An important component of this class of algorithm is a distance function, used to determine whether the agent has reached its goal; this can also require human input and domain knowledge. In past work, it has been common to assume that the goal space is known and use the $L2$ distance between the current state and the goal. However, this straightforward choice is not satisfactory for general environments, as it does not take environment dynamics into account. For example, it is possible for a state to be close to a goal in terms of L_2 distance, and yet be far from satisfying it in terms of environment dynamics.

We propose a self-supervised method for learning a distance between a state and a goal which accurately reflects the dynamics of the environment. We begin by defining the distance between two states as the square root of the average number of time steps required to move from the first state to the second and back for the first time under some policy π. To make this distance usable as part of a goal-conditioned reward function, we train a neural network to approximate this quantity from data. The distance network is trained online, in conjunction with the training of the goal-conditioned policy.

The contributions of this work are as follows. i) We propose a self-supervised approach to learn a distance function, by learning an embedding with the property that the p-norm

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

between the embeddings of two states approximates the average temporal distance between the states according to a policy π, ii) we show that our method is approximating a theoretically motivated quantity and discuss the connection between our approach and the graph Laplacian, iii) we demonstrate that the learned distance estimate can be used in the online setting in goal-conditioned policies, iv) we develop an automatic curriculum generation mechanism that takes advantage of our distance learning algorithm, and v) we explain a phenomenon that arises due to learning the distance function using samples from the behavior policy. Our method solves complex tasks without prior *domain knowledge* in the online setting in three different scenarios in the context of goal-conditioned policies - a) the goal space is the same as the state space, b) the goal space is given but an appropriate distance is unknown and c) the state space is accessible, but only a subset of the state space represents desired goals, and this subset is known a priori.

2. Related Work

Goal-conditioned RL aims to train agents that can reach any goal provided to them. Automatic goal generation approaches such as (Florensa et al., 2018; 2017) focus on automatically generating goals of appropriate difficulty for the agent, thereby facilitating efficient learning. These methods utilize domain knowledge to define a goal space and use $L2$ distance as the distance function in the goal space. However, in most tasks the goal space is inaccessible or an appropriate distance function in the goal space is unknown. There have been recent efforts on learning an embedding space for goals in an unsupervised fashion using the reconstruction error and the $L2$ distance is then computed in the learned embedding space (Péré et al., 2018; Nair et al., 2018; Sukhbaatar et al., 2018). The main drawback of these approaches is that they do not capture the environment dynamics.

(Andrychowicz et al., 2017) and (Rauber et al., 2019) focus on improving the sample efficiency of the goal-conditioned policies by relabeling or reweighting the reward from a goal on which the trajectory was conditioned to a different goal that was a part of the trajectory. Our method is complementary to these approaches since they rely on prior knowledge of the goal space and use the $L1$ or $L2$ distance in the goal space to determine whether the goal has been reached.

Similar to our work, (Savinov et al., 2018) and (Savinov et al., 2019) trained a network R to predict whether the distance in actions between two states is smaller than some fixed hyperparameter k. However, (Savinov et al., 2018) and (Savinov et al., 2019) were done in the context of supervised learning-based navigation and intrinsic motivation, respectively, in contrast to our work. (Savinov et al., 2018) proposed a non-parametric graph based memory module for navigation where the nodes correspond to landmarks in the environment and the nodes judged similar by the network R are connected by an edge; goal-oriented navigation is performed by using a locomotion network L trained using supervised learning to reach intermediate way-points selected as a result of localization and planning on the learned graph. (Savinov et al., 2019) used the network to to provide agents with an exploration bonus for visiting novel states; given a state s visited by the agent, an exploration bonus was provided if the network judged s to be far from the states in a buffer storing a representative sample of states previously visited by the agent.

(Ghosh et al., 2019) defines the actionable distance between states s_1 and s_2 in terms of expected Jensen-Shannon Divergence between $\pi(a|s, s_1)$ and $\pi(a|s, s_2)$, where $\pi(a|s, g)$ is a fully trained goal-conditioned policy. They then train an embedding such that the $L2$ distance between the embeddings of s_1 and s_2 is equal to the actionable distance between s_1 and s_2. This differs from our approach in that we use a different objective for training the distance function, and, more importantly, we do not assume availability of a pre-trained goal-conditioned policy; rather, in our work the distance function is trained online, in conjunction with the policy.

(Wu et al., 2019) learns a state representation using the eigenvectors of the Laplacian of the graph induced by a fixed policy and demonstrates its suitability to reward shaping in sparse reward problems. Our method aims to learn an embedding of the states without computing the eigenvectors of the graph Laplacian. However, the justification of our approach relies on why eigenvectors of the Laplacian are insufficient when the distance between the states in the embedding space is crucial. We discuss the details of this and the connection to the commute time in sections 4.2 and A.3. Furthermore, our approach differs by not using negative sampling; only the information present within trajectories are used to obtain the embeddings.

3. Background

3.1. Goal-Conditioned Reinforcement Learning

In the standard RL framework, the agent is trained to solve a single task, specified by the reward function. Goal-conditioned reinforcement learning generalizes this to allow agents capable of solving multiple tasks (Schaul et al., 2015). We assume a goal space G, which may be identical to the state space or related to it in some other way, and introduce the goal-augmented state space $S_G = S \times G$. Given some goal $g \in G$, the policy $\pi(a_t|s_t, g)$, reward function $r(s_t, g, a_t)$ and value function $V_\pi(s_t, g)$ are conditioned on the goal g in addition to the current state. The objective is to train the agent to achieve all goals afforded by the environment.

We assume the goal space is either identical to or a subspace of the state space, that all trajectories begin from a single start state s_0, and that the environment does not provide a means of sampling over all possible goals (instead, goals must be discovered through experience). Moreover, we require a distance function $d(s, g)$; agents are given a reward of 0 at all timesteps until $d(s, g) < \epsilon$, for hyperparameter $\epsilon \in \mathbb{R}^+$, at which point a reward of 1 is provided and the episode terminates.

3.2. Goal Generation and Curriculum Learning

In order to train agents to achieve all goals in this setting, it is desirable to have a way of systematically exploring the state space in order to discover as many goals as possible, as well as a means of tailoring the difficulty of goals to the current abilities of the agent (a form of goal-based curriculum learning). An algorithmic framework satisfying both of these requirements was proposed in (Florensa et al., 2018). Under this framework, one maintains a working set of goals, and alternates between two phases. In the *policy-learning* phase, the agent is trained (using an off-the-shelf RL algorithm) to achieve goals sampled uniformly from the working set. In the *goal-selection* phase, the working set of goals is adapted to the current abilities of the agent in order to enable efficient learning in the next policy-learning stage. In particular, the aim is to have the working set consist of goals that are of intermediate difficulty for the agent; goals that are too hard yield little reward to learn from, while goals that are too easy leave little room for improvement. 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.

3.3. Multidimensional scaling

The study of multidimensional scaling (MDS) is concerned with finding a low-dimensional configuration of objects by taking as input a set of pairwise dissimilarities between the objects (Borg & Groenen, 2006). The resulting low-dimensional configuration has to be such that the distance in this the low-dimensional configuration between any pair of objects best 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. They must be symmetric, non-negative and obey the triangle inequality. The discussion and notation used in this section follows (Borg & Groenen, 2006).

3.3.1. Classical MDS

MDS in its original form is now called Classical MDS (cMDS) or Toegerson scaling. Classical MDS assumes that the dissimilarities are distances in some Euclidean space. The embeddings produced by cMDS preserve the input distances exactly whenever the inputs are Euclidean distances.

Provided with a matrix of pairwise distances D, cMDS proceeds as follows. First a matrix $D^{(2)}$ of squared pairwise distances is formed. Then a matrix B is obtained by double centering $D^{(2)}$, i.e $B = -\frac{1}{2}JD^{(2)}J$ where $J = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ and $\mathbf{1}$ is a vector of all ones. B is symmetric and positive-semidefinite (details in Appendix A.1). Finally, an eigen-decomposition on B produces $B = Q\Lambda Q^T$, where Λ is a diagonal matrix whose elements are the eigenvalues of B arranged in descending order and the columns of Q are the corresponding eigenvectors. An embedding $X^{'}$ that preserves the Euclidean distance is then obtained using $X^{'} = Q\Lambda^{\frac{1}{2}}$.

3.3.2. Metric MDS

Let δ_{ij} denote the dissimilarity between objects i and j, and let $d_{ij}(X)$ denote a distance metric between the i^{th} and j^{th} rows of X denoted by x_i and x_j respectively. Typically, the distance metric d is the Euclidean distance. x_k is the representation of the object k provided by X. MDS minimizes a quantity called *stress*, denoted by $\sigma_r(X)$, defined as

$$\sigma_r(X) = \sum_{i<j} w_{ij}(d_{ij}(X) - \delta_{ij})^2 \tag{1}$$

where $w_{ij} \geqslant 0$, and $w_{ij} = w_{ji}$. Any meaningful choice of weights that satisfies these constraints can be used.

In equation (1), δ_{ij} can be replaced by $f(\delta_{ij})$. If f is continuous, the approach is then called metric MDS. A generalization of the stress is defined as

$$\sigma_G(X) = \sum_{i<j} w_{ij}(f(d_{ij}(X)) - f(\delta_{ij}))^2 \tag{2}$$

where $f(x) = x$ corresponds to the raw stress σ_r. In general, metric MDS does not admit an analytical solution. Instead, it is solved iteratively, and convergence to a global minimum is not guaranteed.

3.4. Spectral Embeddings of Graphs

Given a simple, weighted, undirected and connected graph G, the Laplacian of the graph is defined as $L = D - W$ where W is the weight matrix and D is the degree matrix. The eigenvectors corresponding to the smallest eigenvalues of the graph Laplacian are used to obtain an embedding for the nodes and have been shown useful in several applications such as spectral clustering (Luxburg, 2007) and spectral graph drawing (Koren, 2003). In spectral embedding methods, a k-dimensional embedding of the node i is obtained by taking the i^{th} components of the k eigenvectors corresponding to the k smallest non-zero eigenvalues.

3.5. Markov Chain based distances

The average first passage time from state i to j is defined as the expected number of steps to reach j for the first time after starting from i. We write the average first passage time $m(j|i)$ recursively as

$$m(j|i) = \begin{cases} 0 & \text{if } i = j \\ 1 + \sum_{k \in S} P(k|i) m(k|i) & \text{if } i \neq j \end{cases}$$

A related quantity, the average commute time $n(i, j)$ is defined as $n(i, j) = m(i|j) + m(j|i)$. Average commute time is a distance metric as noted in (Fouss et al., 2005).

4. Method

In this section we introduce an action-based distance measure for use in trajectory-based reinforcement learning which captures the dynamics of the environment. We then present a method for automatically learning an estimator of that distance using samples generated by a policy π.

4.1. Learned Action Distance

We propose to learn a task-specific distance function where the distance between states s_1 and s_2 is defined as half of the commute-time, which we call the *action distance*. Defining the distance in terms of reachability of the states captures the environment dynamics as experienced by the agent under the policy π. In order to learn a distance estimator we propose to learn an embedding such that the distance between the embeddings of a pair of states is equal to the action distance between the states. Formally, let $s_i, s_j \in \mathcal{S}$, and define the action distance $d^\pi(s_i, s_j)$ as:

$$d^\pi(s_i, s_j) = \frac{1}{2} m(s_j|s_i) + \frac{1}{2} m(s_i|s_j) \tag{3}$$

In general, $d^\pi(s_i, s_j)$ is difficult to compute; this is problematic, since this distance is intended to be used for detecting when goals have been achieved and will be called frequently. Therefore, we propose to train a neural network to estimate it. Specifically, we learn an embedding function e_θ of the state space, parameterized by vector θ, 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_θ is:

$$\theta^* = \arg\min_\theta (||e_\theta(s_i) - e_\theta(s_j)||_p^q - d^\pi(s_i, s_j))^2 \tag{4}$$

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 ρ 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

$$m(s_j|s_i) = E_{\tau \sim \pi, t \sim \bar{t}(s_i, \tau), t' = \min\{\bar{t}(s_j, \tau) \geqslant m\}} [|t - t'|] \tag{5}$$

where $\bar{t}(s, \tau)$ is a uniform distribution over all temporal indices of s in τ and the expectation is taken over trajectories τ sampled from π such that s_i and s_j both occur in τ. If π is a goal-conditioned policy we also average over goals g provided to π. In the next section we discuss the existence of the embedding space that preserves action distance (equation 3), its connection to the graph Laplacian, and a practical approach to approximate it.

When used as part of an algorithm to learn a goal-conditioned policy, 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 distance function will be non-stationary. This can create a number of issues, the most important of which is difficulty in setting the threshold ϵ. Recall that in our setting, the goal is considered to be achieved and the episode terminated once $d^\pi(s, g) < \epsilon$, where ϵ is a threshold hyperparameter. This thresholding creates an ϵ-sphere around the goal, with the episode terminating whenever the agent enters this sphere. The interaction between the ϵ-sphere and the non-stationarity of the on-policy distance function causes a subtle issue that we dub the *expanding ϵ-sphere* phenomenon, discussed in detail in Section 5.3.

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 ϵ-sphere phenomenon of the on-policy distance predictor.

4.2. Existence and Approximation of the Embedding Space

Our approach relies on spectral theory of graphs to obtain a representation of states, similar to (Wu et al., 2019). We note that the Laplacian $L = D - W$ of (Wu et al., 2019) in the finite state setting is given by $W_{uv} = \frac{1}{2}\rho(u)P^\pi(v|u) + \frac{1}{2}\rho(v)P^\pi(u|v)$ and the transition prob-

abilities $P_{uv} = \frac{1}{2}P^{\pi}(v|u) + \frac{1}{2}\frac{\rho(v)}{\rho(u)}P^{\pi}(u|v)$ (details in A.2). The 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 et al., 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 Λ 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 $B_{ij} = L^{\dagger}_{ij}$. Further discussion is provided in A.3.

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 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

$$\sigma_S(X) = \sum_{i<j} w_{ij}(||x_i - x_j||_2^2 - \frac{1}{2}(m(j|i) + m(i|j)))^2 \quad (6)$$

where x_i is the embedding of node i. By noting that the quantity in equation (6) 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 (6) as

$$\sigma_S(X) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(||x_i - x_j||_2^2 - k(i,j))^2 \quad (7)$$

where $k(i,j) = \mathbb{1}m(j|i) + (1 - \mathbb{1})m(i|j)$ and $\mathbb{1} \sim Bern(0.5)$. Thus, sampling $m(j|i)$ and $m(i|j)$ in equal proportion provides the required averaging. In practice, however, it is simpler to sample $\mathbb{1} \sim Bern\left(\frac{\rho(u)}{\rho(u)+\rho(v)}\right)$ than $Bern(0.5)$, motivating our definition in equation (5).

For the choice of w_{ij}, the discussion of (Wu et al., 2019) in Appendix A.2 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 frequency of visiting i and j together. In A.4, we discuss the weights for the case when $m(.|.)$ is not available and has to be estimated using the trajectories.

To summarize, (Fouss et al., 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 3) 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 F.2 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.

4.3. Action Noise Goal Generation

Our algorithm maintains a working set of goals for the policy to train against. The central challenge in designing a curriculum is coming up with a way to ensure that the working set contains as many goals of intermediate difficulty (GOID) as possible. The most straightforward way of generating new GOID goals from old ones is by applying perturbations to the old goals. The downside of this simple approach is that the noise must be carefully tailored to the environment of interest, which places a significant demand for domain knowledge about the nature of the environment's state space. Consider, for instance, that in an environment where the states are represented by images it would be difficult to come up with any kind of noise such that the newly generated states are feasible (i.e. in S). Another option is to train a generative neural network to generate new GOID goals, as proposed by GoalGAN (Florensa et al., 2018); however, this introduces significant additional complexity.

A simple alternative is to employ action space noise. That is, to generate new goals from an old goal, reset the environment to the old goal and take a series of actions using a random policy; take a random subset of the encountered states as the new goals. The states generated in this way are guaranteed to be both feasible and near the agent's current ability. Moreover, applying this approach requires only knowledge of the environment's action space, which is typically required anyway in order to interact with the environment. A similar approach was used in (Florensa et al., 2017), but in the context of generating a curriculum of start states growing outward from a fixed goal state.

If implemented without care, action space noise has its own significant drawback: it requires the ability to arbitrarily reset the environment to a state of interest in order to start taking random actions, a strong assumption which is not satisfied for many real-world tasks. Fortunately, we can avoid this requirement as follows. Whenever a goal is successfully achieved during the policy optimization phase, rather

than terminating the trajectory immediately, we instead continue for a fixed number of timesteps using the random policy. During the goal selection phase, we can take states generated in this way for GOID goals as new candidate goals. The part of the trajectory generated under the random policy is not used for policy optimization. This combines nicely with the off-policy method for training the distance predictor, as the distance predictor can be trained on these trajectories; this results in a curriculum learning procedure for goal-conditioned policies that requires minimal domain knowledge.

5. Experimental Results

As a test bed we use a set of 3 Mujoco environments in which agents control simulated robots with continuous state/action spaces and complex dynamics. The first environment is called Point Mass, wherein an agent controls a sphere constrained to a 2-dimensional horizontal plane which moves through a figure eight shaped room. The state space is 4-dimensional, specifying position and velocity in each direction, while the 2-dimensional action space governs the sphere's acceleration. In the other two environments, the agent controls a quadripedal robot with two joints per leg, vaguely resembling an ant. The 41-dimensional state space includes the center-of-mass of the Ant's torso as well as the angle and angular velocity of the joints, while the action space controls torques for the joints. The Ant is significantly more difficult than point mass from a control perspective, since a complex gait must be learned in order to navigate through space. We experiment with this robot in two different room layouts: a simple rectangular room (Free Ant) and a U-shaped maze (Maze Ant). Further discussion of our environments can be found in (Florensa et al., 2018) and (Duan et al., 2016). For certain experiments in the Maze Ant environment, the canonical state space is replaced by a pixel-based state space giving the output of a camera looking down on the maze from above.

In the first set of experiments we seek to determine whether our online distance learning approach can replace the hand-coded distance function in the GoalGAN algorithm, thereby eliminating the need for a human to choose and/or design the distance function for each new environment. We experiment with different choices for the goal space, beginning with the simplest case in which the goal space is the (x, y) coordinates of the robot's center-of-mass (i.e. a subspace of the state space) before proceeding to the more difficult case in which the goal and state spaces are identical. Next, we present the results of training the distance predictor using pixel inputs in the batch setting in the Ant Maze environment, showing that our method can learn meaningful distance functions from complex observations. Next, we empirically demonstrate the expanding ϵ-sphere phenomenon mentioned in Section 4.1, which results from training the distance predictor with on-policy samples. Finally, we show that the goal generation approach proposed in Appendix 4.3 yields performance that is on par with GoalGAN while requiring significantly less domain knowledge. In Appendix F, we present additional experiments on comparing spectral graph drawing with commute-time preserving embeddings in a tabular setting and discuss the effects of some hyperparameters.

5.1. GoalGAN with Learned Action Distance

Here we test whether our proposed method can be used to learn a distance function for use in GoalGAN, in place of the hard-coded L2 distance. We explore two methods for learning the distance function: 1) the on-policy approach, in which the distance function is trained using states from the trajectories sampled during GoalGAN's policy-learning phase, and 2) the off-policy approach, in which the distance function is trained on states from random trajectories sampled at the end of controlled trajectories during the policy-learning phase. For the embedding network e we use a multi-layer perceptron with one hidden layer with 64 hidden units and an embedding size of 20. As we are interested in an algorithm's ability to learn to accomplish all goals in an environment, our evaluation measure is a quantity called *coverage*: the probability of goal completion, averaged over all goals in the environment. For evaluation purposes, goal-completion is determined using Euclidean distance and a threshold of $\epsilon = 0.3$ for Point Mass and $\epsilon = 1.0$ for Ant. Since the goal spaces are large and real-valued, in practice we approximate coverage by partitioning the maze into a fine grid and average over goals placed at grid cell centers. Completion probability for an individual goal is taken as an empirical average over a small number of rollouts.

5.1.1. XY Goal Space

In this experiment, the goal space is the (x, y) coordinates of the robot's center-of-mass (i.e. a subspace of the state space). We compare our approach with the baseline where the $L2$ distance is used as the distance metric in the goal space. In this setting we are able to achieve performance comparable to the baseline without using any domain knowledge, as shown in Figure 1.

5.1.2. Full State Space as Goal Space

In this setting the goal space is the entire state space, and the objective of the agent is to learn to reach all feasible configurations of the state space. This is straightforward for the Point Mass environment, as its 4-dimensional state space is a reasonable size for a goal space while still being more difficult than the (x, y) case explored in the previous section. In contrast, the Ant environment's 41-dimensional

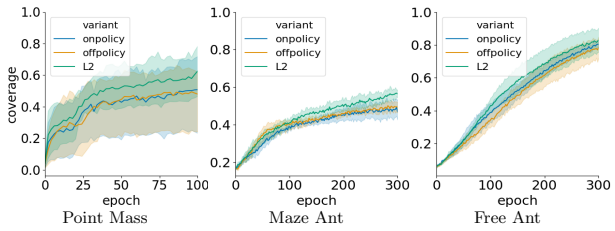


Figure 1: Coverage plots for the (x, y) goal space. Our method does not require domain knowledge unlike the $L2$ distance.

state space is quite large, making it difficult for any policy to learn to reach every state even with a perfect distance function. Consequently, we employ a *stabilization* step for generating goals from states, which makes goal-conditioned policy learning tractable while preserving the difficulty for learning the distance predictor. This step is described in detail in Appendix E.

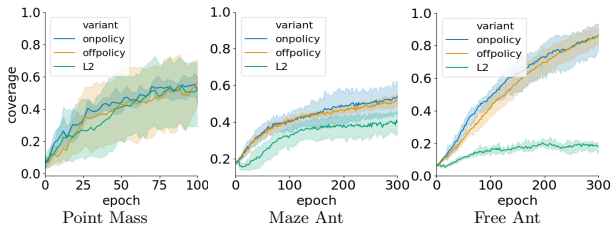


Figure 2: Coverage plots for the full goal space.

Results for these experiments are shown in Figures 1 and 2, where we can see that the agents are able to make progress on all tasks. For the Point Mass agent, progress is slow compared to the (x, y) case, since now to achieve a goal the agent has to reach a specified position with a specified velocity. The learning progress of the Ant agent trained to reach only stable positions suggests that our approach of learning the distance function is robust to unseen states, since the distance predictor can generalize to states not seen or only rarely seen during training. Fig. 3(a,b) shows the visualization of the distance estimate of our predictor on states visited during a sample trajectory in the Point Mass and Maze Ant environments in (x, y) and full goal spaces respectively. Fig. 3(c) shows the visualization 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 5.3 we study the qualitative difference between the on-policy and off-policy methods for training the distance predictor.

5.2. Pixel inputs

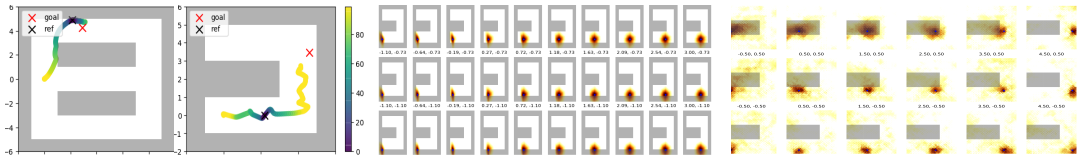
To study whether the proposed method of learning a distance function scales to high-dimensional inputs, we evaluate the performance of the distance predictor using pixel representation of the states. This experiment is performed in the batch setting. Similar to the pretraining phase of (Wu et al., 2019), 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 3 (d) shows the distance estimates of the distance predictor in this setting. For qualitative comparison, we also experimented with the approach proposed in (Wu et al., 2019); these results are shown in section F.3 for various choices of hyperparameters.

5.3. Expanding ϵ-sphere

In this section we show that there are 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 ϵ-sphere of the goal, the episode is terminated when the agent reaches the boundary of the ϵ-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 ϵ-sphere of the corresponding goal. In this scenario, the path to the goal g from any state within the ϵ-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 ϵ-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 ϵ-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. 4. 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 ϵ-sphere centered on g increases in volume. In contrast, the bottom row in Fig. 4 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 ϵ-sphere exhibits significantly less growth.

Since the agent does not receive training for states that are within the ϵ-sphere centered at a goal g, it is desirable to keep the ϵ-sphere as small as possible. One way to do this



(a) (b) (c) (d)

Figure 3: (a, b): Predicted action distance between a reference state and states along a trajectory in Point Mass (a) and Maze Ant (b); (c,d,e): Predicted action distance between stabilized reference states and all other states. (c): Ant Maze environment in full goal space (trained online) and (d): pixel space (batch setting) respectively. Closer states are darker. Each subplot uses a different reference state near the starting state shown as a blue dot. Full heatmap for (d) is shown in Figure 16.

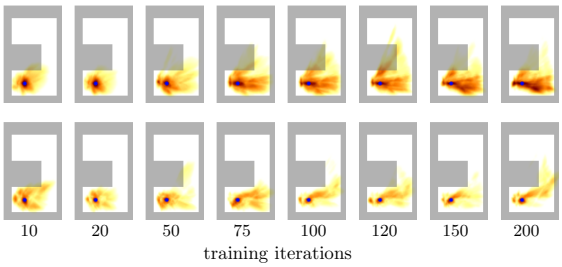


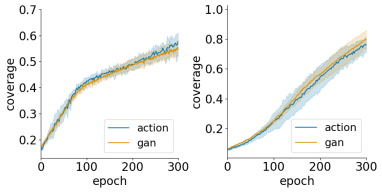
Figure 4: Predictions of the distance predictor trained with on-policy (top) and off-policy (bottom) samples in Maze Ant with (x, y) goal space illustrating how the predictions evolve over time. Darker colors indicate smaller predicted distance and the small blue dot indicates the reference state.

would be to employ an adaptive algorithm for choosing ϵ as a function of g and the agent's estimated skill at reaching g; as the agent gets better at reaching g, ϵ 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.

5.4. Generating Goals Using Action Noise

We perform this comparison in both the Maze Ant and Free Ant environments, using (x, y) as the goal space and the Euclidean distance. The results, shown in Fig. 5, demonstrate that the performance of our approach is comparable 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. 6.

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



(a) Maze Ant (b) Free Ant

Figure 5: Comparing the proposed goal generation algorithm against GoalGAN.

receive 0 reward and will not contribute to learning.

6. Conclusion

We have presented an approach to automatically learn a task-specific distance function without the requirement of domain knowledge, and demonstrated that our approach is effective in the online setting where the distance function is learned alongside a goal-conditioned policy while also playing a role in training that policy. We then discussed and empirically demonstrated the expanding ϵ-sphere phenomenon which arises when using the on-policy method for training the distance predictor. This can cause difficulty in setting the ϵ hyperparameter, particularly when the final performance has to be evaluated using the learned distance function instead of using a proxy evaluation metric like the Euclidean distance. This indicates that off-policy distance predictor training should be preferred in general. Finally, we introduced an action space goal generation scheme which plays well with off-policy distance predictor training. These contributions represent a significant step towards making goal-conditioned policies applicable in a wider variety of environments (e.g. visual domains), and towards automating the design of distance functions that take environment dynamics into account without using domain knowledge.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Borg, I. and Groenen, P. Modern multidimensional scaling: Theory and applications. *Journal of Educational Measurement*, 40:277 – 280, 06 2006. doi: 10.1111/j.1745-3984.2003.tb01108.x.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1329–1338, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/duan16.html.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. In Levine, S., Vanhoucke, V., and Goldberg, K. (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 482–495. PMLR, 13–15 Nov 2017. URL http://proceedings.mlr.press/v78/florensa17a.html.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In *ICML*, 2018.

Fouss, F., Pirotte, A., and Saerens, M. 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, pp. 550–556, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2415-X. doi: 10.1109/WI.2005.9. URL https://doi.org/10.1109/WI.2005.9.

Fouss, F., Pirotte, A., Renders, J.-M., and Saerens, M. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. on Knowl. and Data Eng.*, 19 (3):355–369, March 2007. ISSN 1041-4347. doi: 10.1109/TKDE.2007.46. URL https://doi.org/10.1109/TKDE.2007.46.

Ghosh, D., Gupta, A., and Levine, S. Learning actionable representations with goal conditioned policies. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Hye9lnCct7.

Kaelbling, L. P. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, 1993. Morgan Kaufmann. URL http://people.csail.mit.edu/lpk/papers/ijcai93.ps.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Koren, Y. On spectral graph drawing. In *Proceedings of the 9th Annual International Conference on Computing and Combinatorics*, COCOON'03, pp. 496–508, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40534-8. URL http://dl.acm.org/citation.cfm?id=1756869.1756936.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z. URL https://doi.org/10.1007/s11222-007-9033-z.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In *NeurIPS*, pp. 9209–9220, 2018.

Narvekar, S., Sinapov, J., and Stone, P. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, August 2017.

Péré, A., Forestier, S., Sigaud, O., and Oudeyer, P.-Y. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=S1DWPP1A-.

Rauber, P., Ummadisingu, A., Mutz, F., and Schmidhuber, J. Hindsight policy gradients. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg2viA5FQ.

Savinov, N., Dosovitskiy, A., and Koltun, V. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SygwwGbRW.

Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkeK3s0qKQ.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897, 2015.

Sukhbaatar, S., Denton, E., Szlam, A., and Fergus, R. Learning goal embeddings via self-play for hierarchical reinforcement learning. *CoRR*, abs/1811.09083, 2018. URL http://arxiv.org/abs/1811.09083.

Wu, Y., Tucker, G., and Nachum, O. The laplacian in RL: Learning representations with efficient approximations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJlNpoA5YQ.

This is the supplementary for the paper titled "Self-supervised Learning of Distance Functions for Goal-Conditioned Reinforcement Learning"

A. Discussion of MDS and Spectral embedding

A.1. Classical MDS

First we discuss the tools used in cMDS, before describing cMDS itself.

Provided with a matrix $X \in \mathbf{R}^{n\times m}$ of n objects in m-dimensional space, we can form the squared pairwise distances 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 $c \in \mathbf{R}^n$ such that $c_i = ||x_i||^2$ and $\mathbf{1} \in \mathbf{R}^n$ is the all-ones vector of length n. 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.

Recovering X from $D^{(2)}$ is the objective of cMDS. We first 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

$$B = Q\Lambda Q^T = Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^T = X^{'}X^{'T} \tag{8}$$

where Λ_{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. Hence, given a pairwise scalar product matrix B, a configuration $X^{'}$ that preserves the pairwise distances can be recovered. The origin is assumed to be $\mathbf{0} \in \mathbf{R}^n$.

We now proceed to describe the procedure to obtain 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 (mean is zero). Similarly for $x \in \mathbf{R}^{1\times n}$, xJ is a row vector balanced on the origin. Since $D^{(2)} = c\mathbf{1}^T + \mathbf{1}c^T - 2XX^T$ we get

$$\begin{aligned} -\frac{1}{2}JD^{(2)}J &= -\frac{1}{2}J(c\mathbf{1}^T + \mathbf{1}c^T - 2XX^T)J \\ &= 0 - 0 + JXX^TJ \end{aligned} \tag{9}$$

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 balanced on 0. Therefore $JX = X$. Hence, equation (9) is written as $-\frac{1}{2}JD^{(2)}J = XX^T = B$. Following equation (8), 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 $X = Q_+\Lambda^{\frac{1}{2}}_+$. Note that the reconstruction produces only a configuration that preserves the pairwise distance exactly and does not necessarily recover the original configuration (differs by rotation).

An alternate characterization of cMDS is given by the loss function $L(X) = ||XX^T - B||^2_F$ known as *strain* where F is the Frobenius norm. Due to the Frobenius norm, strain can be written as

$$L(X) = \sum_{i<j}(x_ix_j^T - B_{ij})^2 \tag{10}$$

showing that cMDS can be used in an iterative setting.

A.2. Spectral Embeddings of Graphs

Given a simple, weighted, undirected and connected graph G, the laplacian of the graph is defined as $L = D - W$ where W is the weight matrix and D is the degree matrix. The eigenvectors corresponding to the smallest positive eigenvalues of the graph laplacian are used to obtain an embedding for the nodes and have been shown useful in several applications such as spectral clustering (Luxburg, 2007) and spectral graph drawing (Koren, 2003).

The discussion here in the finite state setting but the definition of the Laplacian used here is same as that in (Wu et al., 2019) and hence we refer the reader to (Wu et al., 2019) for a discussion in continuous state spaces.

First we begin by translating the equations in (Wu et al., 2019) to the finite state setting. In (Wu et al., 2019), 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 case we write $\mathbf{P}$ and $\mathbf{W}$ to denote the use of D for transition matrix and weight matrix respectively. P^π denotes the transition probabilities of the markov chain induced by the policy π. $\hat{P}^\pi$ denotes the transition probabilities of the time-reversed markov chain of policy π. We assume that the stationary distribution ρ of P^π exists.

The definition of density $D(u, v)$ in (Wu et al., 2019) is given by

$$D(u, v) = \frac{1}{2}\frac{P^\pi(v|u)}{\rho(v)} + \frac{1}{2}\frac{P^\pi(u|v)}{\rho(u)} \tag{11}$$

Since $D(u, v)$ is integrated with respect to measure $\rho(v)$ and since integrating with respect to a measure is analogous to weighted sum, equation (11) is multiplied by $\rho(v)$ to

obtain

$$\mathbf{P}_{uv} = \frac{1}{2}P^{\pi}(v|u) + \frac{1}{2}\frac{\rho(v)}{\rho(u)}P^{\pi}(u|v) \tag{12}$$

$$= \frac{1}{2}P^{\pi}(v|u) + \frac{1}{2}\hat{P}^{\pi}(v|u) \tag{13}$$

For obtaining $\mathbf{W}$, notice that $D(u, v)$ is integrated with respect to $\rho(u)$ and $\rho(v)$ in equation (2) of (Wu et al., 2019). Therefore, by multiplying equation (11) by $\rho(u)$ and $\rho(v)$ we obtain

$$\mathbf{W}_{uv} = \frac{1}{2}\rho(u)\rho(v)\frac{P^{\pi}(v|u)}{\rho(v)} + \frac{1}{2}\rho(u)\rho(v)\frac{P^{\pi}(u|v)}{\rho(u)} \tag{14}$$

$$= \frac{1}{2}\rho(u)P^{\pi}(v|u) + \frac{1}{2}\rho(v)P^{\pi}(u|v) \tag{15}$$

Note that the transition probabilities $\mathbf{P}$ is given by forward transition probabilities P^{π} 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^{π}. Hence, we assume that P^{π} is reversible. Note that this assumption is a special case of the transition matrix $\mathbf{P}$ given in equation (12). Hence, the definition of $\mathbf{P}_{uv}$ is given by $\mathbf{P}_{uv} = P^{\pi}(u|v)$ and $\mathbf{W}$ is given by $\mathbf{W_{uv}} = \rho(u)P^{\pi}(v|u)$.

A.3. Euclidean Commute Time Distance

Given the Laplacian $L = D - W$, (Fouss et al., 2005; 2007) show that the average first passage time and the average commute times can be expressed as

$$m(j|i) = \sum_{k=1}^{n}(l^{\dagger}_{ik} - l^{\dagger}_{ij} - l^{\dagger}_{jk} + l^{\dagger}_{jj})d_{kk} \tag{16}$$

and

$$n(i, j) = V_G(l^{\dagger}_{ii} + l^{\dagger}_{jj} - 2l^{\dagger}_{ij}) \tag{17}$$

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

$$\begin{aligned} n(i, j) &= V_G(l^{\dagger}_{ii} + l^{\dagger}_{jj} - l^{\dagger}_{ij} - l^{\dagger}_{ij}) \\ &= V_G(e_i - e_j)^T L^{\dagger}(e_i - e_j) \\ &= V_G(e_i - e_j)^T Q\Lambda Q^T(e_i - e_j) \\ &= V_G(e_i - e_j)^T Q\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}Q^T(e_i - e_j) \\ &= V_G(e_i - e_j)^T Q\Lambda^{\frac{1}{2}T}\Lambda^{\frac{1}{2}}Q^T(e_i - e_j) \\ &= V_G(x_i - x_j)^T(x_i - x_j) \end{aligned} \tag{18}$$

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) in (Fouss et al., 2005).

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

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

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 (18) 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 F.1 in a tabular setting.

A.4. Approximation using Metric MDS

The mean first passage times $m(.|.)$ are not available and has to be estimated from the trajectories. First note that $m(j|i) = \sum_{t=0}^{\infty} P^t_{ij} t$ where we use the notation P^t_{ij} 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

$$\sigma(X) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(||x_i - x_j||_2^2 - k)^2 \tag{19}$$

where $k \sim P^t_{ij}$. The quantities in equation (19) can be obtained from trajectories drawn under a fixed policy, thus providing a practical approach to learn the embedding as

given in equation (4) 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 F.2, q in equation (4) 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.

B. Training the distance predictor

The distance predictor has to be trained prior to being used in determining whether the goal has been reached in order to produce meaningful estimates. To produce a initial set of samples to train the distance predictor, we use a randomly initialized policy to generate a set of samples. This provides a meaningful initialization for the distance predictor since these are states that are most likely to occur under the initial policy.

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.

C. Goal Generation Buffer

To generate goals according to the proposed approach we store the states visited under the random policy after reaching the goals in a specialized buffer and sample uniformly from the buffer to generate the goals for each iteration. The simplest approach of storing the goals in a list suffers from the two following issues: i) all the states visited under the random policy from the beginning of the training procedure will be considered as a potential goal for each iteration and ii) the goals will be sampled according to the state visitation distribution under the random policy. The issues i) and ii) are problematic because the goal generation procedure has to adapt to the current capacity of the agent and avoid the goals that have already been mastered by the agent; sampling goals according to the visitation distribution will bias the agent towards the states that are more likely under the random policy. To overcome these issues we use a fixed size queue and by ensure that the goals in the buffer are unique.

To avoid replacing the entire queue after each iteration, only a fixed fraction of the states in the queue are replaced in each iteration. In our experiments, the queue size was set 500 and 30 goals were updated after each iteration.

We observe that our goal generation or off-policy distance predictor are agnostic to the policy being a random policy and hence the random policy can be replaced with any policy if desired.

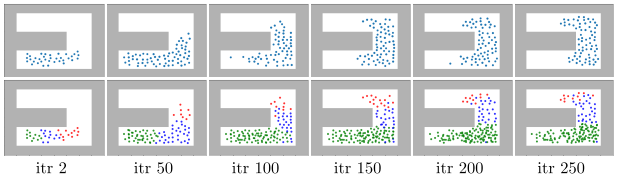


Figure 6: Evolution of the goals generated by our goal generation approach (top). A sample of goals so-far encountered (bottom), color-coded according to estimated difficulty: green are easy, blue are GOID and red are hard.

D. Hyperparameters

The GoalGAN architecture and its training procedure and the policy optimization procedure in our experiments are similar to (Florensa 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 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 ϵ 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 ϵ and the learning rate were determined by performing grid search with ϵ 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 ϵ 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 et al., 2018), located at `https://github.com/florensacc/rllab-curriculum`.

E. Goal Stabilization for Ant Tasks

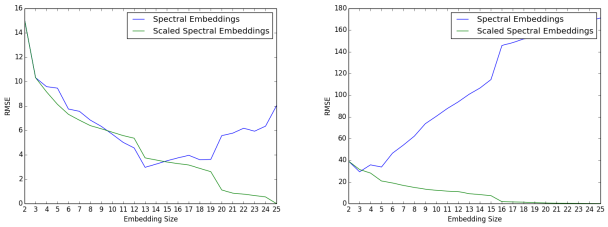
For Ant tasks we used a modified setup for obtaining goals from states. We first identified a stable pose for the Ant's body, with body upright and limbs in standard positions. Then, to create a goal from a state, we take the position component from the state, but take all other components (joint angles and angular velocities) from the stable pose. This stabilization step is used for all goals; however, the distance predictor is still trained on the full state space.

F. Further experiments

F.1. Effect of scaling the eigenvectors of the Laplacian

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 8) and, north and east with 0.375 and south and west with 0.125 probabilities in the second maze (figure 9). 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 8 and 9 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 (14) and D is the diagonal matrix with stationary probabilities on the diagonal.

As seen in both the Figures 8 and 9, 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 8), 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), 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 7. 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.



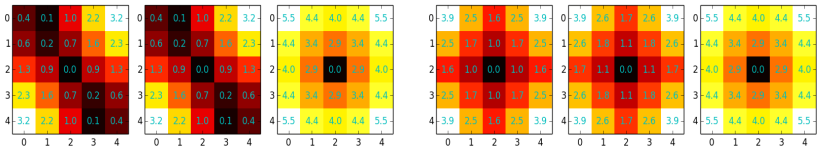
(a) (b)

Figure 7: 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.

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.

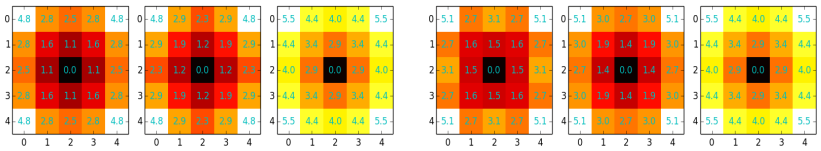
F.2. Effect of q

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 & Groenen, 2006) (section 11.3). Increasing q increases the



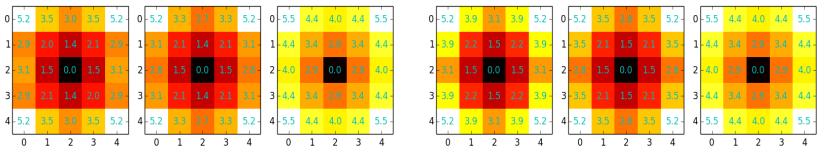
(a) 1

(b) 3



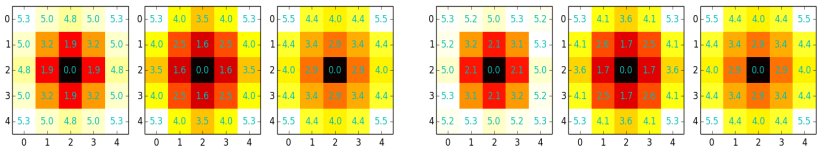
(c) 5

(d) 7



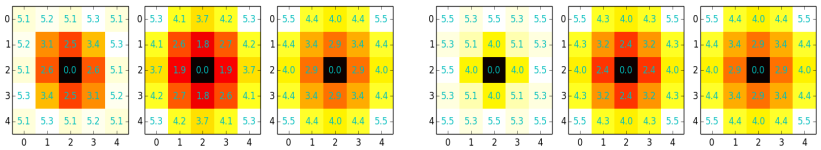
(e) 9

(f) 11



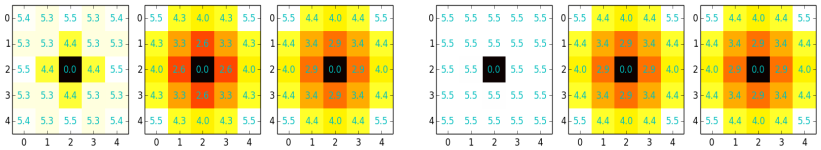
(g) 13

(h) 15



(i) 17

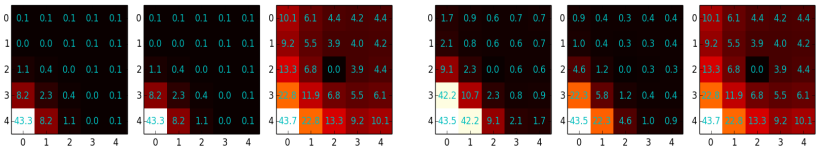
(j) 19



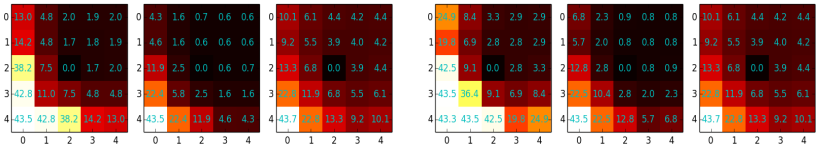
(k) 21

(l) 24

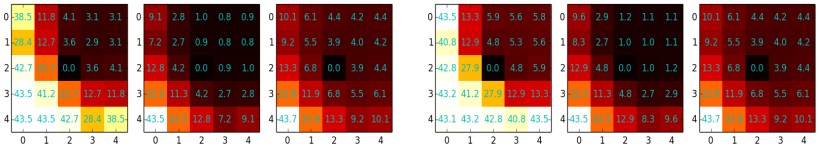
Figure 8: Visualization of the distance from state $(2, 2)$ to all other states using different embedding sizes produced by spectral embedding(first column) and scaled spectral embedding(second column) along with the ground-truth computed analytically(third column). The transition from each state to 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.



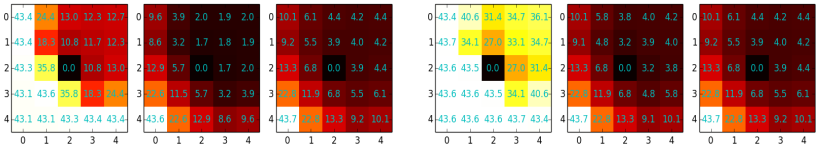
(a) 1 (b) 3



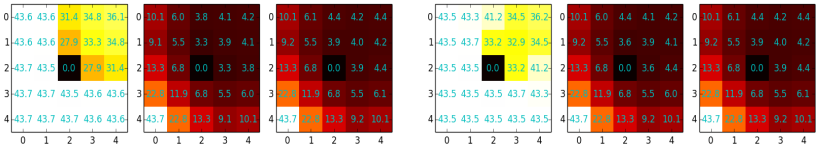
(c) 5 (d) 7



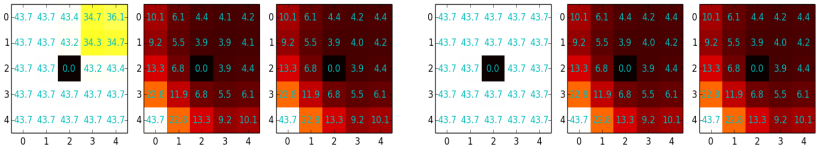
(e) 9 (f) 11



(g) 13 (h) 15



(i) 17 (j) 19



(k) 21 (l) 24

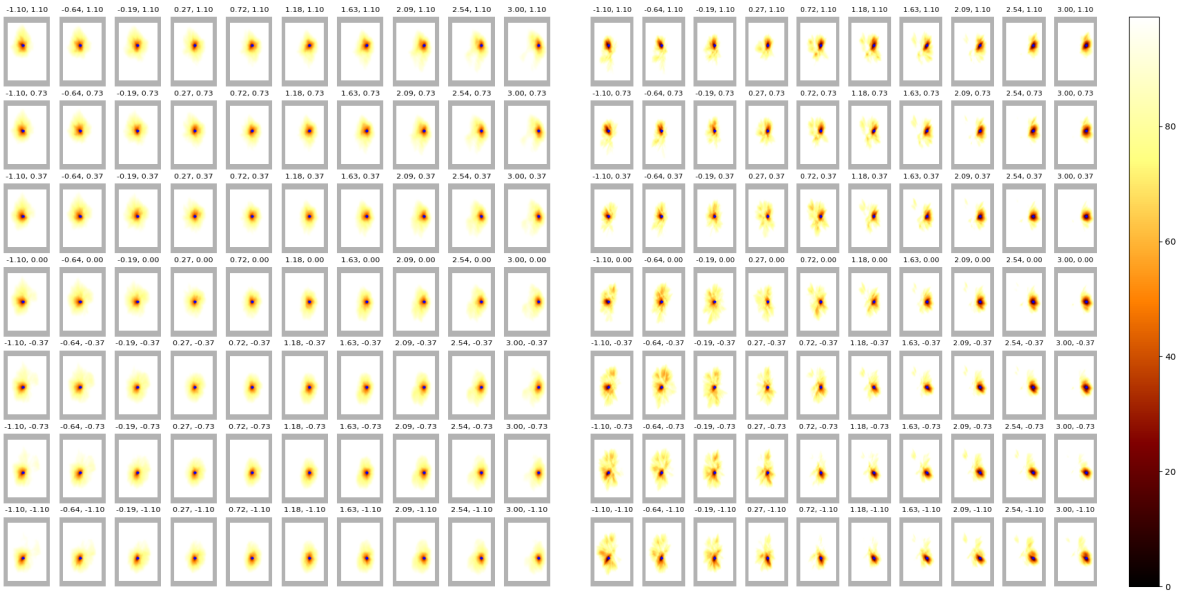
Figure 9: The transition from a state to its neighbours in north and east are with probability 0.375 and in south and west are with probability 0.125. The choice of addition of a dimension to spectral embedding presents a trade-off between preserving the previous estimate and improving the estimate of a closer state. In contrast, the scaled spectral embedding improves with the addition of every dimension.

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 δ_{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 $\delta_{ij} = \sqrt{n(i,j)}^{\frac{1}{4}}$.

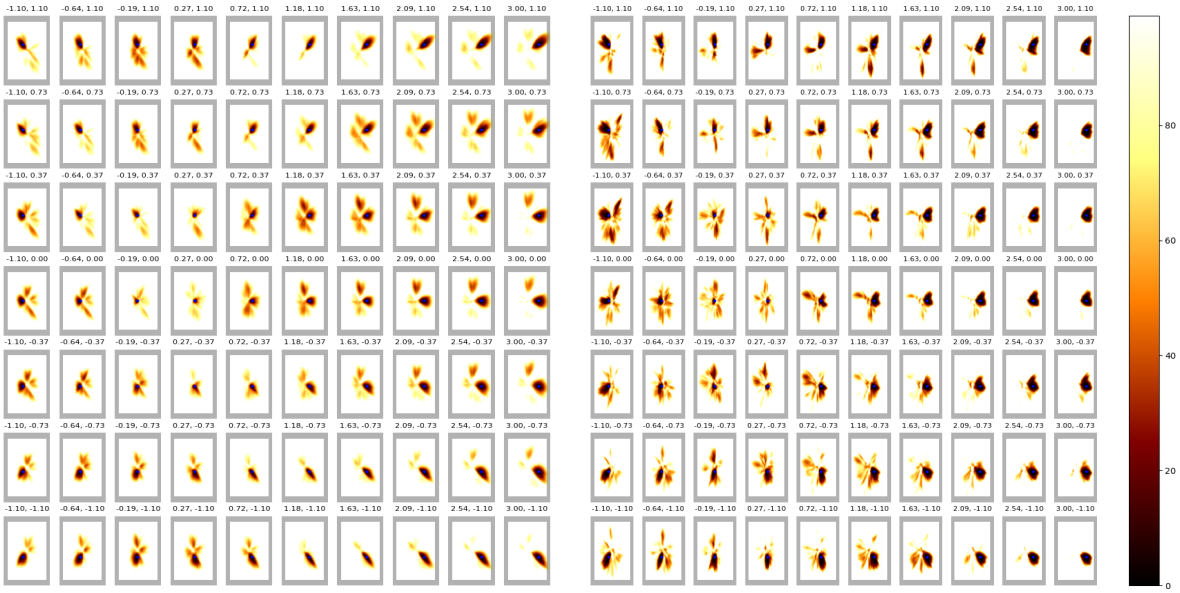
F.3. Prediction with pixel inputs

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 $5e$-5 and Adam (Kingma & 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 11. The results with the approach of (Wu et al., 2019) are shown in Figures 12 and 13. The training setup is the same as described in section 5.2. We uniformly sample the states in each trajectory (hence, λ is approximately 1). As β 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 11) centered on nearby points.

When used in an online setting, the negative sampling in (Wu et al., 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. The discussion in F.1 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 14 (Wu et al., 2019) and 15 (our method).



(a) $q = 0.5$ (b) $q = 1$



(c) $q = 2$ (d) $q = 4$

Figure 10: 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.

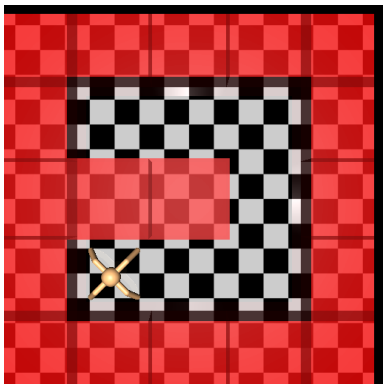
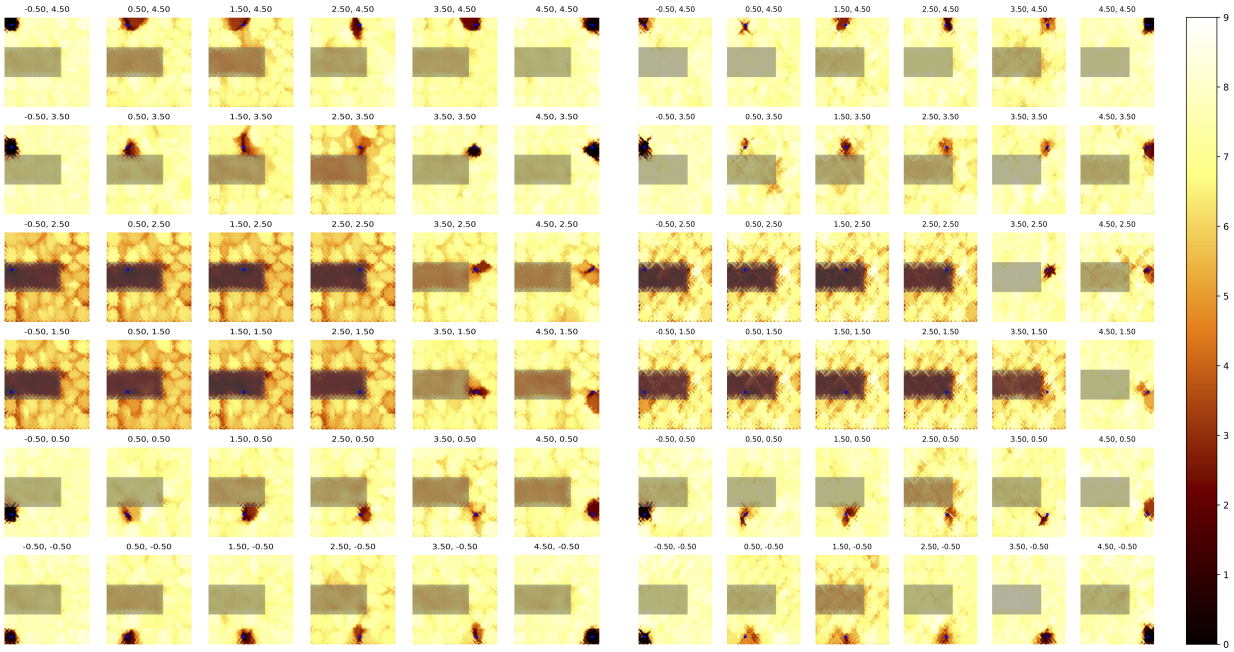
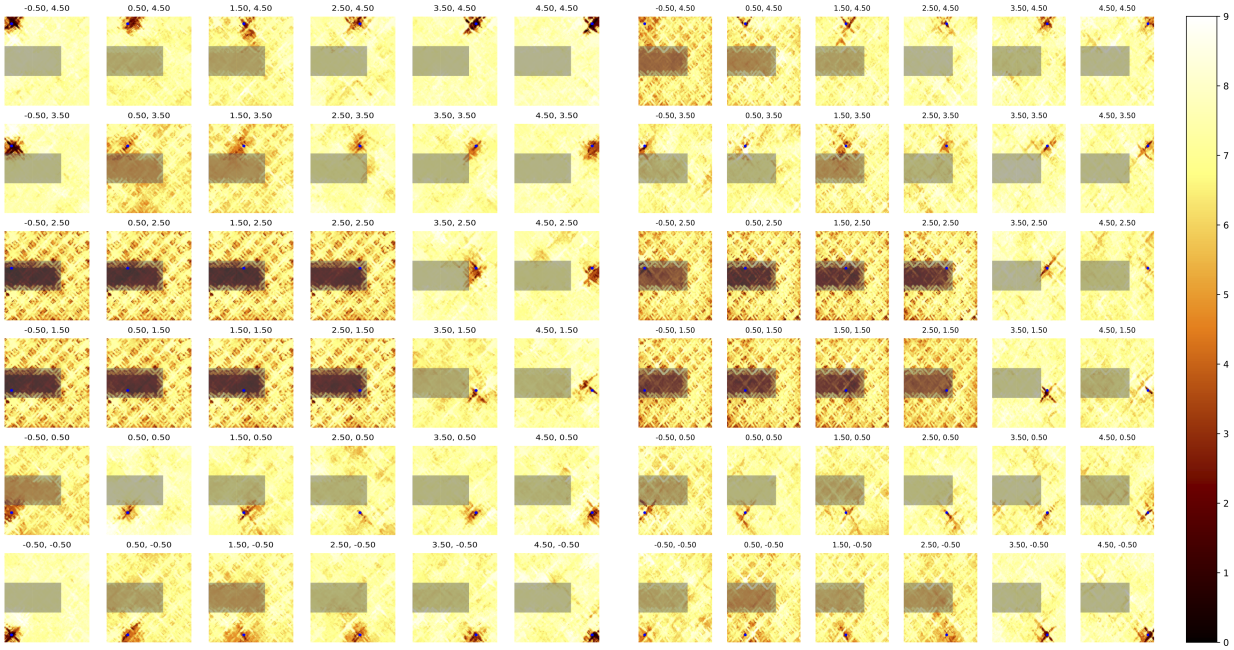


Figure 11: Top-down view of the Maze Ant. The RGB image scaled to 32×32 is the input in the pixel tasks.

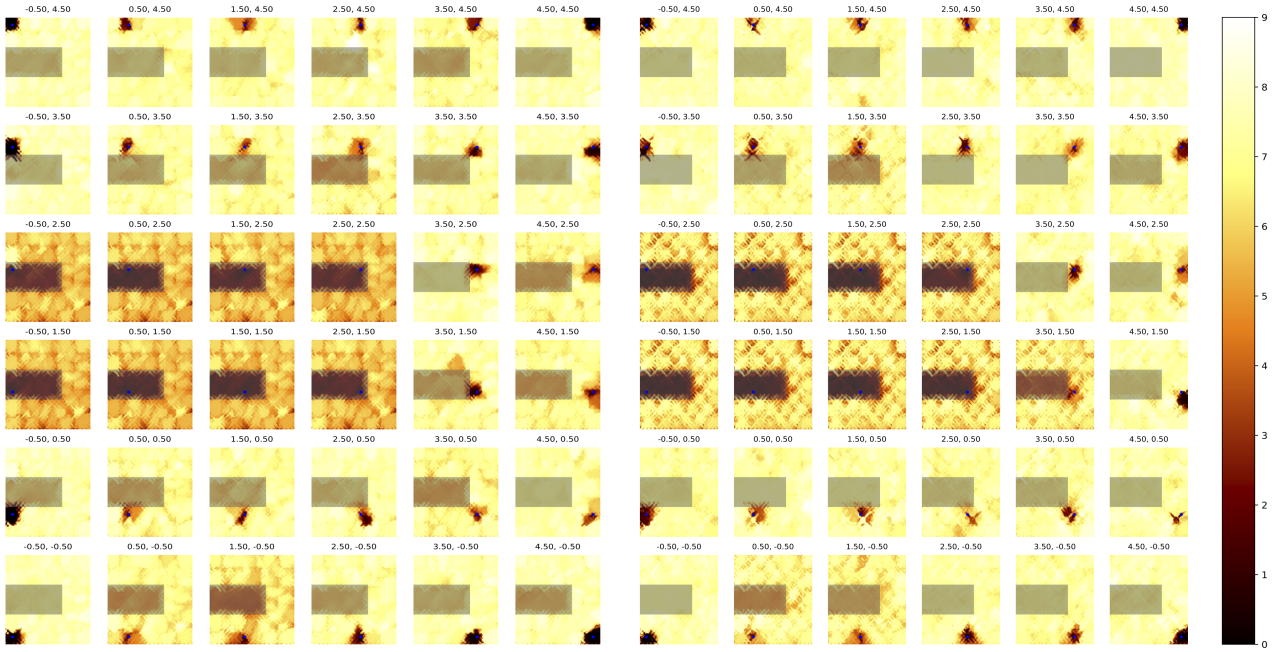


(a) $\beta = 0.5$ (b) $\beta = 1$

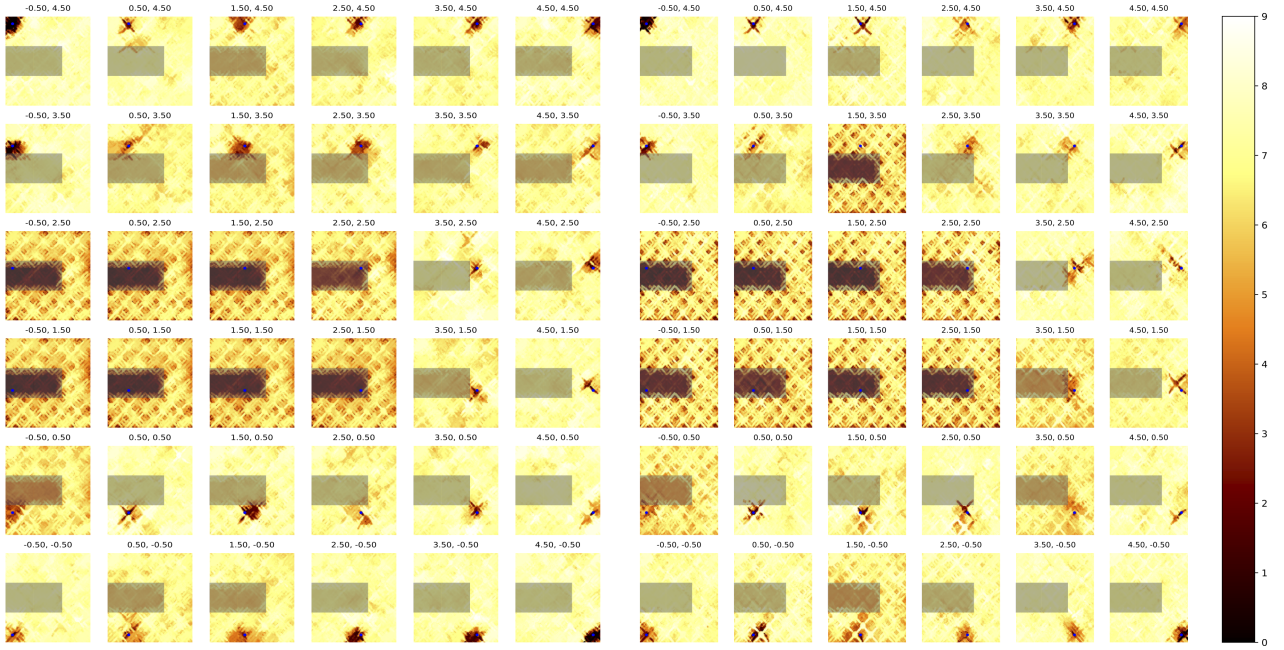


(c) $\beta = 2$ (d) $\beta = 6$

Figure 12: Laplacian in RL using pixel inputs. Higher β better approximates spectral graph drawing objective. $LR = 1e-4$

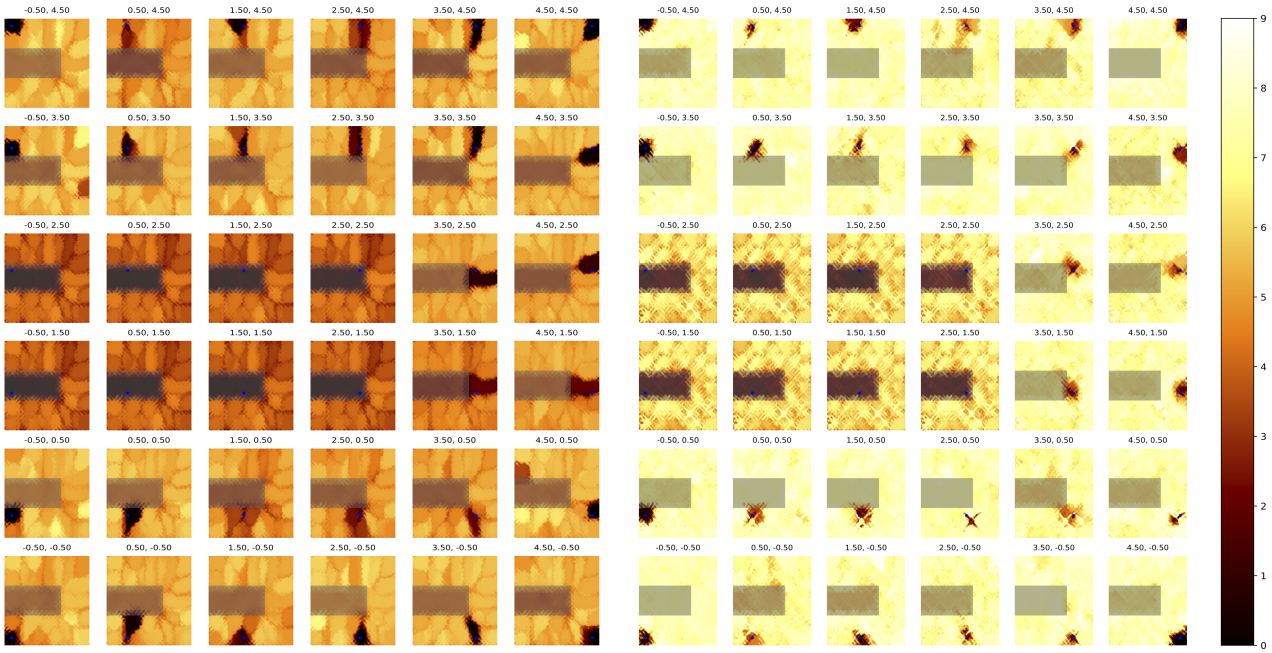


(a) $\beta = 0.5$ (b) $\beta = 1$

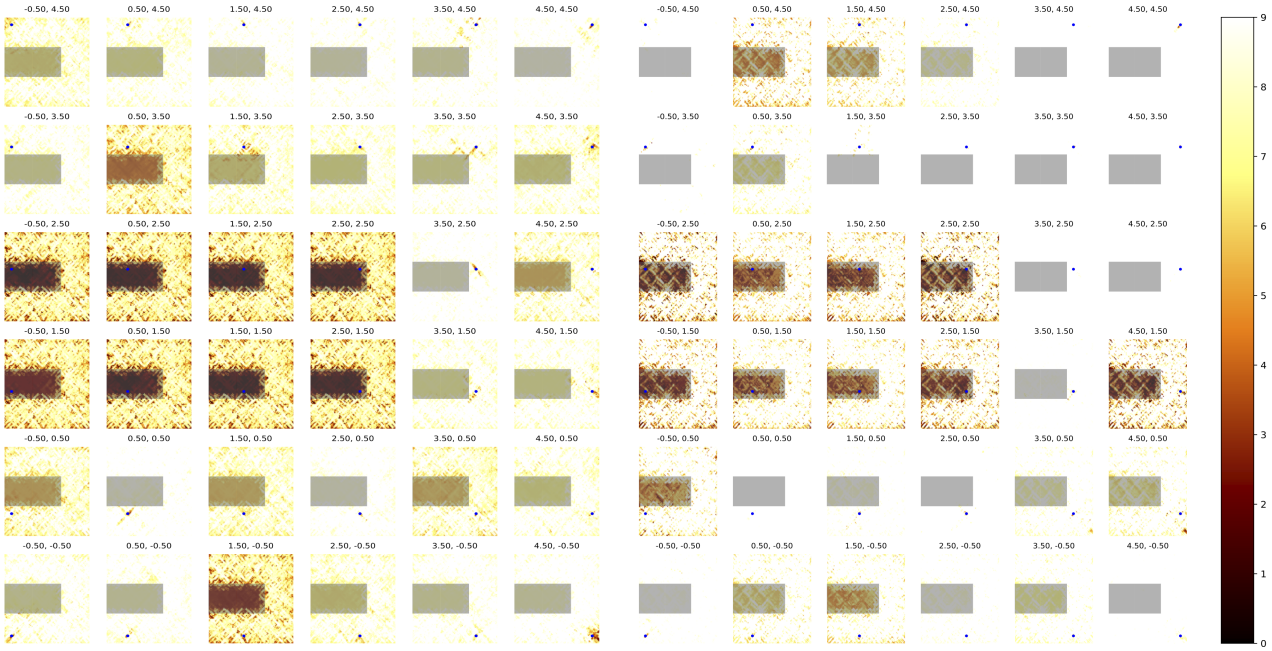


(c) $\beta = 2$ (d) $\beta = 6$

Figure 13: Laplacian in RL using pixel inputs. Higher β better approximates spectral graph drawing objective. $LR = 5e-5$

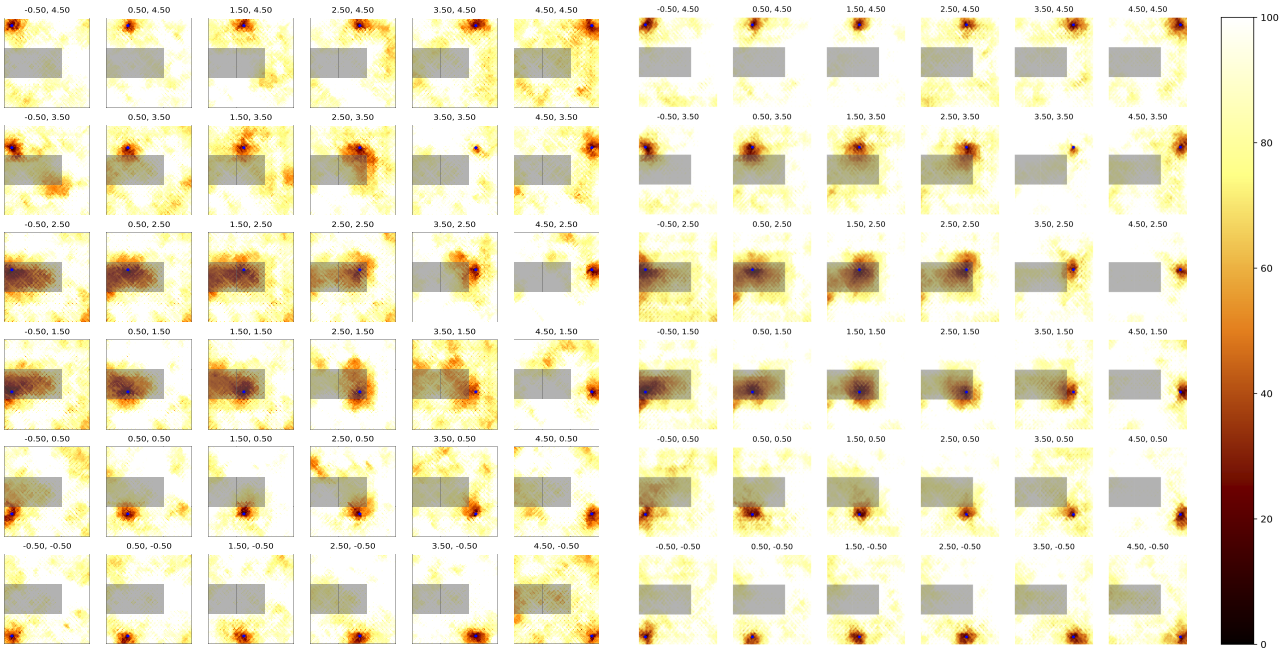


(a) 16 (b) 32

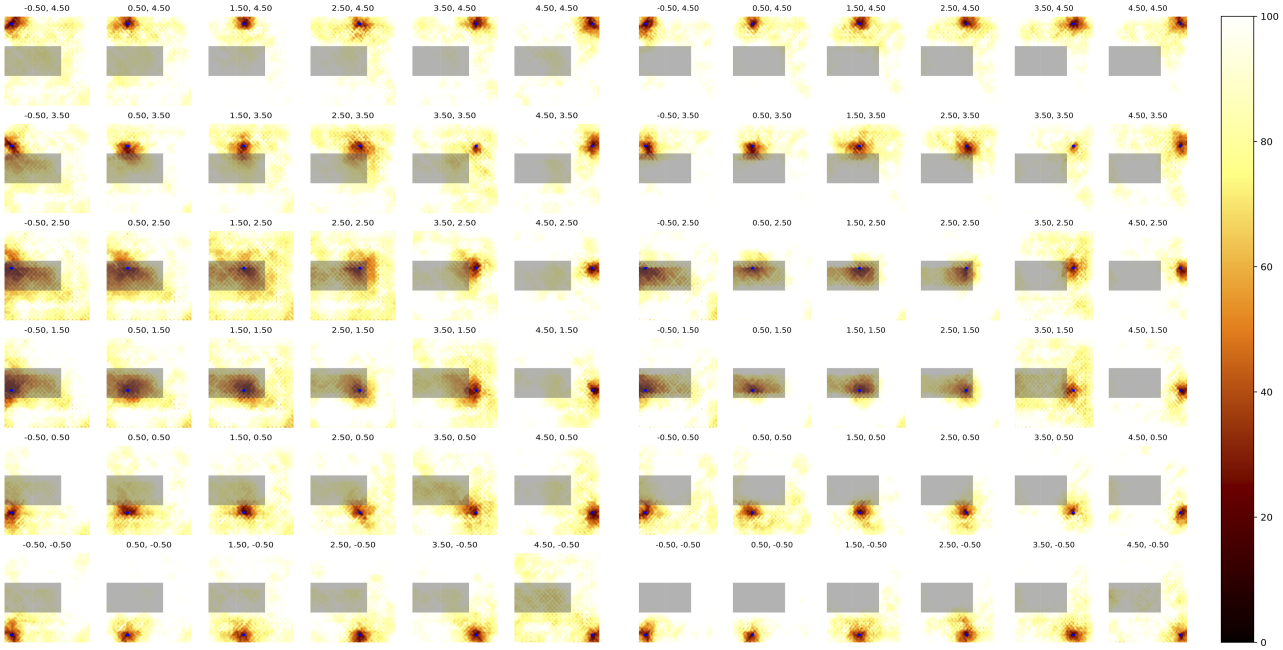


(c) 64 (d) 128

Figure 14: 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 15: Increasing the embedding size in our approach improves the distance estimates.

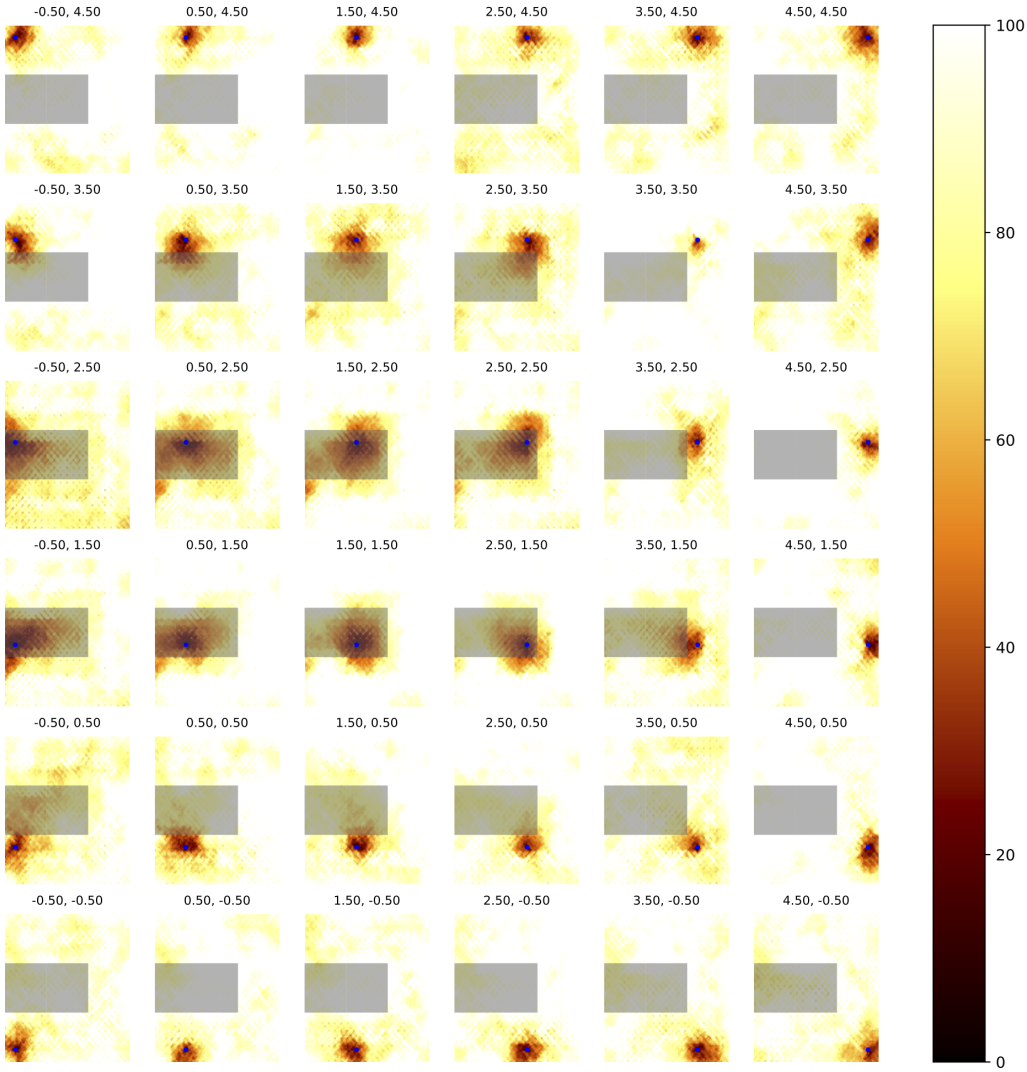


Figure 16: Full heatmap of our approach using pixel inputs. Our approach does not require negative sampling.