Learning a Distance Metric over Markov Decision Processes

A Thesis Presented in Partial Fulfillment of
the Honors Bachelor’s Degree

Kshitij Sachan

Abstract

A challenging aspect of reinforcement learning is learning in sparse reward tasks, in which the agent only receives reward when it reaches some goal. In practice, learning in these tasks is often sample-inefficient to the point of being infeasible. A smoother reward function that provides the agent with incremental feedback is much easier to learn from, but manually designing such a smooth reward function is time consuming, error-prone, and requires domain knowledge. Previous work has learned a distance metric conditional on a policy—a measure of the expected number of steps to transition from one state to another under a given policy—and used the negative predicted distance to the goal as a smooth reward. My thesis extends this work by learning the **minimum distance between two states over all policies**. I use the quantile loss function, an asymmetric loss function that is minimized when predicting a specific percentile of a distribution, to learn this minimal distance metric off-policy (i.e. I learn distances for the optimal policy given data from non-optimal policies). I demonstrate that predicting a small percentile such as 5th percentile of distances between two states given a non-optimal policy is a decent estimate of the minimum distance between those two states given the optimal policy. I then use this learned distance function for reward shaping to partially solve a high-dimensional MuJoCo AntMaze domain that is challenging with sparse rewards. Lastly, I demonstrate some failures of this approach to generalize to even higher dimensional Atari games and suggest possible research directions to make learning distance functions more robust to environments with extremely non-smooth transition dynamics.
1 Introduction

Reinforcement learning (RL) is a subfield of machine learning in which agents select actions to maximize their reward [6]. Just as a human slowly learns the right way to kick a soccer ball because they get a thrill from scoring a goal, or a dog learns the right way to beg to more frequently obtain a treat, RL agents modify their behavior to gain more reward. RL has demonstrated success in domains ranging from robot manipulation to video games [6].

A challenging aspect of RL is learning in sparse reward environments, where the reward is zero almost everywhere. In theory, sparse rewards are sufficient to learn a good policy, but in practice they can require exponentially more training time than dense reward functions, which provide the agent with more frequent reward. Handcrafting smooth reward functions is tedious and requires domain knowledge. Further, including an artificial smooth reward function may modify the resulting learned policy. Thus, it would be valuable if we could algorithmically generate a smooth reward. One approach to doing this is learning a distance metric between states (i.e. learning a notion of how far apart different states are) so that the reward function could be the negative learned distance to the goal. This reward function does not require human insight and preserves the optimal policy.

In this thesis, I explore this question of learning such a distance metric. Previous work has learned distances conditional on a policy [3]. I will focus on the harder problem of learning the minimum distance between two states over all policies without ever learning the optimal policy. The resulting distance metric is learned offline and can directly be used for reward shaping to learn the optimal policy. I do this by using a loss function that allows learning specific percentiles. An L1 loss function is minimized when predicting the median of a distribution. The quantile loss function is a generalization of L1 loss and allows the user to specify what percentile should be learned. I use quantile loss with percentile \( \approx 0 \) to learn the approximate minimum distance between states given a non-optimal policy and find that this is a decent estimate of the minimum distance between those states over all policies. I then describe how my learned distance metric can be used for RL applications such as goal-conditioned RL and exploration [5].

2 Background

In RL, we are interested in solving Markov Decision Processes (MDPs), which are defined by the 4-tuple \((S, A, P_a, R_a)\). \(S\) is the set of possible states, \(A\) is the actions the agent can take, \(P_a(s, a)\) is the transition function defining the probabilities with which actions transition between states, and \(R_a(s, a)\) is the reward received by the agent for executing action \(a\) from state \(s\). The agent does not have access to \(P_a\) and \(R_a\) and is trying to maximize the expected cumulative reward. The agent learns a policy, \(\pi : S \rightarrow A\) that is a probabilistic function from state to action taken in that state. In the sparse reward setting, the reward function is of the form \(R_a(s, a) = 0\) if \(s\) is the goal state and \(-1\) otherwise. The value function, \(V : S \rightarrow \mathbb{R}\), is the agent’s estimated future expected rewards from state \(s\) following the current policy.

Goal-conditioned RL is a generalization of standard RL where there is now more than one goal [5]. For example, the agent could be breaking down a large problem into a series of subgoals. Value functions can be extended to goal-conditioned RL—they are now of the form \(V : S \times S \rightarrow \mathbb{R}\), where the second input is the goal state being targeted.

[3] define the distance conditioned on a policy between two states \(s_i\) and \(s_j\) as the number of steps it takes following \(\pi\) to transition from \(s_i\) to \(s_j\), given that they were visited in the same trajectory (a trajectory is a sequence of states from the start state to some terminal state). Note that the distance is a distribution, not a single value, because both the policy and environment can be stochastic. Therefore, in practice, we are interested in the simpler task of learning the expected value of the distance between two states:

\[
D^\pi (s_i, s_j) = \mathbb{E}_{\tau \sim \rho_\pi | s_i, s_j} \left[ \sum_{k=0}^{j-i} \gamma^k \right],
\]

where \(\rho_\pi\) is the distribution of trajectories induced by \(\pi\), \(\tau\) is a trajectory sampled from that distribution, and \(\gamma\) is a discount factor to deal with infinite trajectories. In most of my experiments, \(\gamma = 1\). We might also be interested in the minimum distance, which is the minimum expected distance between \(s_i\) and \(s_j\) over
all policies:

\[ d^*(s_i, s_j) = \min_\pi d^\pi(s_i, s_j). \] (2)

To avoid confusion, note that \( d^* \) and \( d^\pi \) are not metrics by the math definition. Both \( d^\pi \) and \( d^* \) are not symmetrical and \( d^\pi \) does not necessarily satisfy the triangle inequality.

3 Related Work

Now we turn to the question of learning equation (1). The lowest fidelity approach is using the Euclidean distance; i.e. \( d^*(s_i, s_j) \propto \|s_i - s_j\|_2 \). This fails on toy examples and on images (Figure 1). On the other end of the spectrum, dynamic programming methods can be used to learn perfectly accurate distance functions [5]. To understand this approach, we can reframe the distance learning problem as solving an MDP. Suppose the original task is learning \( d^* \) for some MDP \( M = (S, A, P, R_M) \). Consider a corresponding MDP \( M' = (S, A, P, R_{M'}) \) where \( R_{M'} \equiv -1 \). Then, \( d^* \) in \( M \) is the same as the the goal-conditioned value-function for \( M' \) starting from \( s_i \) and targeting goal \( s_j \): \( d^*(s_i, s_j) = -V(s_i, s_j) \). Any temporal difference algorithm such as Q learning can be used to learn \( d^* \).

This bootstrapping approach is too heavy duty for our purposes. We do not want to solve an MDP to learn our distance function, and we also do not need a perfectly accurate reward function for reward shaping. In fact, if we had such a function, we would not need to do any learning; at each step, the agent would pick the action that minimized distance to the goal. Instead, we only need a reward shaping function that is accurate enough to reduce learning time and not disrupt the optimal policy.

A simpler approach to learning a distance metric is supervised learning. [8] learned a local distance metric to determine if two states were within a k-step threshold by learning an embedding such that the Euclidean distance between two states in embedding space is proportional to the distance between the two states in problem space. They were only interested in learning an extremely local distance metric (which is not sufficient for reward shaping), so we can not use the same contrastive-like loss function they used.

[3] also used supervised learning to directly train a distance metric (as opposed to an embedding network). They learned a global metric that they used for reward shaping on image-based observations, exactly the task we are trying to solve. I will explain the details of their algorithm, which they call Dynamical Distance Learning (DDL) because my work builds upon theirs. To solve a sparse reward MDP, DDL alternates between two steps: learning a distance function and policy improvement. First, they learn a policy-conditional distance function, \( d^\pi \), by 1) performing rollouts of a fixed policy to collect trajectories and 2) training a distance network on every pair of states in those trajectories with the labels as the number of steps to transition between those states. This amounts to minimizing the loss function:

\[ L(\phi) = \mathbb{E}_{\tau \sim \rho_{\pi}} \left[ \left( d^\pi_\phi(s_i, s_j) - (j - i) \right)^2 \right], \]
where $\phi$ is a network parameterizing $d^\pi$. Once the distance function is learned, DDL uses the negative distance to a subgoal state as the reward function to train a new policy. This new policy is used to train a distance function, and the cycle repeats. In this way, they learn a series of distance functions and policies, $\pi_1, \ldots, \pi_n$, that converge to the optimal policy $\pi^*$.

4 Learning A Distance Metric

I seek to extend DDL by directly learning $d^*$ without learning intermediate $d^{\pi_1} \ldots d^{\pi_n}$. Once we know $d^*$, we can use the negative distance to the goal to learn the optimal policy. My algorithm is based off the intuition that every transition between states $s_i$ and $s_j$ upper bounds the number of steps required to travel between them with the optimal policy. Rather than using mean squared error to learn a distance metric online for a single policy and then iterate that process to reach an optimal policy (as with DDL), I learn a distance metric offline, using the quantile loss function to estimate the minimum transition time between states (corresponding to $d^*$). Quantile loss is an asymmetric loss function that is minimized when predicting a certain percentile of the distribution (e.g. 5th percentile):

$$L_\alpha(y, \hat{y}) = \begin{cases} \alpha(\hat{y} - y) & \hat{y} - y \geq 0 \\ (\alpha - 1)(\hat{y} - y) & \hat{y} - y < 0 \end{cases},$$

where $y$ is the true value, $\hat{y}$ is the predicted value, and $\alpha$ is the percentile of the distribution we are interested in estimating. There is an L2 version of this loss function that replaces the $(\hat{y} - y)$ terms with $(\hat{y} - y)^2$. To better understand the properties of the quantile loss function, look at Figure (2).

My algorithm replicates [3]'s training scheme, except I learn a distance metric offline. Each episode:

1. Select a subgoal (over time, these subgoals will approach the true goal state)
2. Use the learned distance metric for reward shaping (minimize negative distance to the subgoal)
3. Update the distance metric using the new observed trajectories, minimizing the loss function:

$$\mathcal{L}(\phi) = \mathbb{E}_{\tau \sim \rho_{\pi_i}, i \sim [0, T], j \sim [i, T]} \left[ L_\alpha \left( d_{\phi}^\pi(s_i, s_j), j - i \right) \right],$$

where $\phi$ is the network parameterizing $d$, $T$ is the max episode length, and $L_\alpha$ is the quantile loss.

[3] used human preferences to select subgoals. To make the training procedure simpler, I assumed access to an oracle that select the subgoal in the replay buffer that is closest to the true goal state (this avoids the need to sample from the state space).

5 Results

The three questions I tried to answer with my experiments were:

1. What are the properties of a quantile distance metric (i.e. a distance metric learned with a quantile loss function)?
2. Does quantile learning help us learn $d^*$ directly from non-optimal policies?
3. Can the learned $d^*$ be used for reward shaping to solve sparse reward MDPs?

5.1 Properties of a quantile distance metric

To better understand the quantile loss function, I ran some experiments independent of distance learning. I trained several neural networks with L1 and L2 quantile loss functions on noisy data (Figure 2). As expected, the nth percentile L1 quantile loss function is minimized when predicting the nth percentile of the data. This is an important property for applications that will be explored in the discussion section.
Figure 2: Each curve corresponds to a neural network learned with a different quantile loss function (color indicates quantile). (a) uses $L^1$ quantile loss and (b) uses $L^2$ quantile loss. The networks were trained on data (in blue) sampled from the distribution $f(x) = \sin x + Y + \epsilon$ where $P(Y = 0) = 0.9$, $P(Y = 10) = 0.1$, and $\epsilon$ is some small Gaussian noise. This produces two sine curves in the dataset with the lower one having 90% of the data. The $L^1$ $n$th quantile loss is minimized when predicting the $n$th percentile of the distribution, so the 0.1, 0.5, and 0.85 quantile curves all lie along the bottom sine curve. All networks minimizing $L^1$ quantile losses for quantile larger than 0.9 will predict values along the top sine curve distribution. $L^2$ loss is smoother, so the different quantiles interpolate between the two sine curves.

Figure 3: Point maze experiment
(a) Ant Umaze task

(b) Distance to goal state (star) observed in train data

(c) Learned distance functions for different quantiles. Top row: Predicted distance to goal state (star). Bottom row: Predicted vs actual distances between state pairs (each point corresponds to a single state pair). The black line is $y = x$. Note that the x-axis has a different scale on each plot.

Figure 4: Ant Umaze experiments
Next, I wanted to verify that the training process of supervised learning over the replay buffer produced a reasonable distance metric, $d^\pi$ in simple domains. This is the same technique used by DDL to learn a distance metric on policy. I performed this experiment on the Point maze domains in the MuJoCo physics simulator [7, 1]. The agent is moving a 2D force-actuated ball (along the X and Y axis) to reach a goal state in a 2D maze. The observations are the x and y locations and velocities of the point, and the agent can provide a force in the x and y directions.

I trained a distance network using mean-squared error on 100 episodes of expert data from the D4RL dataset (this data was collected using a controller rather than a policy, so we know that the policy is optimal) [2]. After optimizing several hyperparameters such as the batch size, learning rate, neural network architecture, and which features to train on, the learned distance metric was accurate to within 5 steps of the true distances. The learned distance plot is in Figure 3. Visual inspection shows that the predicted distance to the star (goal state) is reasonable. It is clear, for example, that the distance metric did not simply learn Euclidean distance because it predicts the bottom right corner is far away from the goal state in MDP space even though they are nearby in problem space.

After verifying that the supervised learning approach worked on a simple task, I ran new experiments to: 1) try a higher-dimensional problem and 2) use quantile loss instead of mean squared error. These experiments were run using D4RL data for the Ant agent, which has a 30 dimension observation space and an 8-dimensional action space [2]. Ant has highly unstable and nonlinear dynamics, so even the expert controller data had noisy distance labels, meaning the observed distance between the same pair of states varied a lot between different trajectories (Figure 4b).

The learned distance metrics are plotted in Figure 4c. For all quantiles, the learned distance metrics have the right gradient—the predicted distance increases as the agent gets further from the goal. The predicted distances are less smooth than in the point maze example, likely because the ant agent is much higher dimensional than point. The high dimensionality means that simply plotting distances as a function of 2D position does not capture all of the variance in the data, but it also means that we should expect our learned distance metrics to be less accurate.

As the quantile value decreases, the predicted distance to the goal from all states decreases and the distance gradient is preserved, which is what we would expect to see. The second row of Figure 4c shows the predicted vs true distances for each state pair the model was trained on. In an ideal world, the predictions would lie along the black line, which is $y = x$ (i.e. the predicted value would always be the actual value). In reality, there is irreducible error because the training data is noisy and reducible error because state space is too high dimensional to learn a perfect distance metric. When quantile = 0.5, the model overpredicts and underpredicts evenly, and there are a few extreme overpredicted outliers (there are no such underpredicted outliers because the model cannot predict negative distances). As the quantile value decreases, the model overpredicts less and less frequently. By the time we look at the quantile = 0.01 model, the model is predicting less than the observed value for almost all points. In theory, a model that is learning $d^\pi$ from a non-optimal policy $\pi$ should have this property: the observed distance between a state pair only upper bounds the true distance so it should never predict more than that observed distance. However, the quantile = 0.01 plot predicts that the distance from the start state to the goal is around 200 steps, which is not possible even with an optimal policy, suggesting that we must be careful to not select a quantile value that is too small or else the model will underpredict heavily.

One issue with these learned distance metrics is that there are certain state pairs for which the model is especially inaccurate. For example, the predicted distance for the quantile = 0.3 model from $(10, -0.5)$ to the goal is extremely low, around 200 steps. That is clearly incorrect because states that are closer to the goal have larger predictions. This inaccuracy is an artifact of the supervised learning training procedure—we do not use bootstrapping to enforce the triangle inequality, so there is no guarantee that the predicted distances of very nearby states to the goal will be consistent. In this case, there were few trajectories that passed through the point $(10, -0.5)$ and reached the goal state because the policy typically found a shorter path to the goal. As a result, the distance learner did not have much training data for this state pair and had a bad prediction. This kind of out-of-distribution error comes up in later experiments as well.
5.2 Learning $d^*$ from non-optimal policies

In the experiments so far, we have learned a distance metric on-policy. Now, we turn to the question of learning $d^*$ off-policy (using the trajectories of policies $\pi_1, \ldots, \pi_n$ to learn a $d^\pi$ for some other $\pi'$). To start with, we ran the simplest possible form of this experiment: collected trajectories from a non-optimal policy $\pi$ and used that to train a distance network $d^\pi'$ that ideally is similar to $d^*$. These experiments were not performed in the maze domains. Instead, the MuJoCo agent is in an open grid and receives reward for moving in a single direction. The episode terminates early if the agent falls over. This setup was a better candidate for the experiment because there are not multiple paths to the end state (there are obviously multiple joint torques to get to any given state but the agent must pass through the same x-position because it only moves along a line), so the information from a non-optimal policy might be more useful. The distance metric was trained on the replay buffer from training a “medium” level agent, which is all of the transitions until the agent received 1/3rd the reward of the expert agent per episode. Then, the distance metric was evaluated on how well it predicted distances for the optimal policy. Initially, I ran these experiments with the Hopper and Walker agents, but MSE learned an effective distance metric so I switched to the harder halfcheetah domain. It is likely that the Walker and Hopper tasks were low-dimensional enough that there was not a big difference between the medium replay buffer and expert trajectories. In the halfcheetah domain, on the other hand, quantile learning is demonstrably better than MSE at learning a distance metric (Figure 5).

In the halfcheetah domain, the x position of the agent is sufficient to reasonably predict the distance from the start state—it is mostly linear with a small nonlinearity at the start as the agent gets up to max speed. The MSE loss predictions for $x < 2.5$ are very wide and overestimate the true minimum observed distances. The quantile = 0.01 predictions are much closer to the observed minimum. The largest deviation is seen when $x < 2.5$ because this is where the original policy was the worst. In order for the agent to get to large values of $x$, the policy must be decent, which means that there will be less of an improvement from quantile learning over MSE. If we allowed for infinitely long episodes, this effect would be slightly mitigated because we would see somewhat poor policies also reach large $x$ values, but it would not be fully gone because in really bad policies the agent will fall over before it gets to large $x$ values. This hints at one of the challenges with assessing the effect of quantile learning: sampling from the replay buffer is not an unbiased estimate of random samples over all trajectories because episodes are capped to a max length during training. The estimate is naturally biased towards zero, even without quantile loss. In practice, this is not a major issue because the bias becomes smaller as states are closer together.

Qualitatively, these results demonstrate that quantile learning is useful for learning $d^*$ off-policy. However, I have skirted around collecting quantitative results for several reasons. When learning a distance metric on-policy using quantile loss, we learn a point estimate of the nth percentile of the distribution. At test time, it is simple to compare the predicted value with the true nth percentile of that distribution (although we do not have direct access to the distribution $d^*$, we could sample from it by learning an optimal goal-conditioned
policy and repeatedly performing rollouts between state pairs of interest).

The theoretical analogue to learning a distance metric off-policy is that as the quantile value approaches 0 we learn the \textit{minimum} of $d^*$. This is challenging to assess at test time for two reasons: 1) as observed previously in the Ant umaze experiments, quantile learning can become unstable when the predicted quantile is very small and 2) approximating the minimum of a normal-ish distribution with sampling requires many samples for reasonable confidence. Further, for reward shaping purposes we are interested in minimizing the expected distance to the goal, not minimizing the minimum distance to the goal, so learning $\min d^*$ is not that useful (there are other applications of quantile distance metrics described in the Discussion for which this might be more useful).

The quantitative experiment set was as follows: learn $d^*$ from a non-optimal policy and use it to predict the observed distances for an optimal policy. Selecting a few state pairs, repeatedly sampling a policy between those states, and minimizing the mean squared error between the $n$th percentile of the observed distance and the model’s predictions would produce very little test data. Instead, the optimal policy was rolled out repeatedly, and, similar to the training procedure, I calculated the quantile L2 loss between the observed distance of every pair of states and the predicted distance. Since the results already demonstrated that $d^*$ can be learned effectively for the one-dimensional movement halfcheetah domains, this experiment were run in the original maze domains. Intuitively, learning $d^*$ off-policy in a 2D grid such as a maze should be harder than in the 1D movement task because there are many paths to get from one state to another. I was unable to learn an optimal policy to collect training trajectories for the ant umaze domain, so I instead switched back to using the point agent. The non-optimal policy that the training data was collected from was an agent that took random actions half the time and optimal actions the other half of the time.

The results of this experiment are shown in Figure 6. Each line corresponds to a network trained with a different quantile. Different lines are not comparable because the loss from different quantile functions have different magnitudes. Instead, the important trend is the minimum along each line (the black point). As the training quantile value decreases, so does the test quantile with minimum loss. This suggests that if we are interested in reward shaping, MSE might be more useful than quantile loss. However, this is with the qualification that the 2D maze task is more challenging for learning $d^*$ off-policy. In the 1D movement task, lower quantile values were better than MSE.

Figure 6: Ant Umaze Value Function
5.3 Using $d^*$ for reward shaping

For the first experiment, I learned $d^*$ offline using expert data and used $d^*$ for reward shaping. For this experiment, we expect MSE to perform better than quantile learning because we are learning the distance metric on-policy. This experiment is not novel - I was simply attempting to a result from the DDL paper as a precursor to the next experiment of learning a distance metric online. This experiment was run in the Ant maze experiments using the previously learned distance functions (Figure 4c). Unfortunately, these distance functions fail to produce an optimal policy when the agent starts at $(0,0)$. This is because the distance metric was learned from an optimal policy without random restarts, so it has poor state space coverage. For example, the distance metric was never trained on trajectories where the agent went left from the start state, but when using the distance metric for reward shaping the agent went left and found a local maxima. To further support this theory, the agent quickly learned ($< 50$ episodes) to reach the goal when it began in the top row of the maze (Figure 7).

Since the agent was able to learn an optimal policy starting from regions where it had good state space coverage, I continued to the hardest task of learning a distance metric online while doing RL. I was unable to get the agent to learn an optimal policy while learning a distance metric online, likely because this is much harder than the task from the DDL paper. In the DDL paper, the agent only had to reach successive subgoals, which gave a natural gradient to the learning task. Instead in my version of the experiment, the agent received reward based on the negative distance to the goal (not negative distance to the subgoal)—if the agent did not see any trajectories involving the goal state, then the distance to the goal is not learned. Unfortunately, I ran out of time to set up the corresponding experiment to the DDL paper with quantile loss.

5.4 Extending to Montezuma’s Revenge

The Atari game Montezuma’s Revenge is a holy grail problem in RL. It is challenging because is a long-horizon sparse reward problem. I did not expect to solve it using reward shaping because of how large the game is. However, there are other applications of distance metrics I have not previously described for which it is a useful test bed.

One such example is goal-conditioned RL (described in the Background section). Identifying when an agent has reached a subgoal in large state spaces is nontrivial. In a continuous state space, the agent might never reach the exact goal. Instead, we want to know if they are within some epsilon ball of the goal, which we can determine by checking if $d^*(s,g) \leq \epsilon$ where $g$ is the goal, $s$ is the current state, and $\epsilon$ is a user-defined threshold.

Previous work has used distance learning for goal-conditioned RL [8]. My work extends this because quantile distance metrics allow us to define quantile epsilon balls. Rather than requiring the expected value of the distance between the current state and the goal to be less than $\epsilon$, we can now require any percentile of the distribution be less than some $\epsilon$. This is important for MDPs that do not have smooth distance functions (smooth is an informal terminology). All the MuJoCo agents in this research have smooth distance functions because $d(s_1,s_2) \approx d(s_1,s_3)$ if $s_2 \approx s_3$ (where two states are approximately equal if they have similar x-
 Atari games such as Montezuma’s Revenge do not have such smooth distance functions. In Montezuma’s Revenge, the player has multiple lives, and after a player loses a life they are teleported to a reset state. This means that the distance distribution to a goal state can be the extremely non-continuous distribution of 10 steps if the player makes a jump without dying and ≥ 100 steps if the player dies and has to reset. In this case, we might want to define an epsilon ball where the player makes it to the goal state within 10 steps with very high probability, which is what an L1 quantile loss function allows us to do (see Figure 2a). The agent would only be within 10 steps of the goal if it successfully completed whatever obstacle was reducing its probability to reach the goal.

Quantile distance metrics also enable exploration [3]. The goal of exploration algorithms is to get the agent to its frontier of knowledge (the edges of the states it can reach) and then do exploration from there. Using the distance function, an agent can navigate to the state in its replay buffer that is furthest from the start state. Rather than navigating to states that are far away in expected value, the agent might want to navigate to states that are far away even in the best case (e.g., furthest with quantile = 0.01). An obvious example of this is the opposite of the goal-conditioned example: a state that is within 10 steps with very high probability but 1000 steps if the agent dies and has to restart might be far away in expected value but is not a state that the agent needs to explore.

Along this vein, I spent about three months of this research attempting to learn distance metrics for Montezuma’s Revenge with very little success. This was not unexpected as learning from image-based observations is challenging. However, I was unable to learn a decent distance metric even after I extracted the relevant state features (the distance learner only saw the number of lives remaining, x-y position, and whether or not the agent has the key). My hypothesis for why distance learning in Montezuma’s Revenge is so challenging is because: 1) discontinuous the discontinuity of the distance functions (Figure 8) and 2) episodes are extremely long (> 4000 steps), which makes training on every pair of states computationally infeasible. Two strategies I tried that had promising preliminary results to improve distance learning in discontinuous domains were discounting and predicting variance.

Setting \( \gamma \) from Equation 1 to a value less than 1 discounts distances for states that are far apart (this is akin to the concept of discounting future rewards). Intuitively, this is a reasonable thing to do because we are less certain that our policy is optimal for states that are far apart so we should do additional contraction on those states. More practically, discounting is useful because most of the loss comes from states that are far apart so this allows our model to rebalance which state predictions are furthest apart.

Uncertainty prediction entails building a model that predicts a distance as well as its uncertainty about that distance. Once discounting is added, the observed variance in distances for states that is far away goes to approximately 0 because the discounted distance approaches the infinite geometric sum of \( \gamma \) (Figure 9). For example, the discounted distance between two states that are 100 vs 500 steps apart with \( \gamma = 0.97 \) is 31.75 and 33.33, respectively. As a result, the model is confident in its predictions about states that are far away and less certain about predictions of states that are nearby. There is a large literature on uncertainty prediction of regression models [4]. I tried several approaches. The most promising approach I found was turning the regression problem of predicting the expected distance between two states into a classification problem: is the expected discounted distance between 0-10 steps, 11-20, 21-30, and so on. Then, the model’s uncertainty was the entropy of the class probabilities for each distance bucket. While this approach was theoretically elegant and had some promising preliminary results, I was unable to use it to learn a robust uncertainty classifier. Intuitively, this makes sense because predicting the variance of the distribution should be more challenging than predicting the mean and the model was already struggling to do that.

6 Conclusion

In this thesis, I have demonstrated that quantile loss can be used to learn \( d^* \) from a non-optimal policy, and this can in turn be used to learn an optimal policy via reward shaping. I was unsuccessful at replicating some results from the DDL paper and therefore somewhat unsuccessful at extending their work by learning a distance metric online. I was also unable to successfully learn distance metrics for Montezuma’s Revenge. Preliminary results suggest that discounting and uncertainty prediction could be a method to improve accuracy for states with discontinuous distance functions. Further research is needed to theoretically ground what it means for a distance function to be smooth and how that affects reward shaping.
Figure 8: Histograms of observed distance between state pairs in Montezuma’s Revenge. Left column: Number of steps between state pairs that start and end on the middle platform. Most of these state pairs are very close together because the states are close together, but there are sometimes that the player walks back onto the platform > 200 steps later. This is an example of a state pair that is bad for exploration because it may have a large expected value distance but is nearby most of the times. Right column: Number of steps between state pairs where the player starts on the right rope and ends on the bottom left ladder. This distribution is bimodal with a peak around 350 steps if the player makes it directly without dying and then another peak around 550 steps if the player fails to jump over the skull and has to restart. Note that both histograms are far from a normal distribution, which makes them harder to learn.
Figure 9: The state space for Montezuma’s Revenge was discretized into buckets based on x-y position, lives left, and whether or not the agent has the key. Each point corresponds to all transitions between two state buckets. The x-axis is the minimum observed distance for any transition in that bucket, and the y-axis is the variance of distances in that bucket. When there is no discounting ($\gamma = 1$), the variances are not a useful predictor of minimum distance. However, as discounting is added ($\gamma$ gets smaller), nearby states have high observed variance and far away states have lower variance.

References


