TRANSITION SPACE DISTANCE LEARNING

by

Mark Nemecek

Department of Computer Science
Duke University

Date: ______________________________

Approved:

____
Ronald Parr, Supervisor

____
Rong Ge

____
Carlo Tomasi

____
Michael Zavlanos

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Department of Computer Science
in the Graduate School of
Duke University

2019

ABSTRACT

TRANSITION SPACE DISTANCE LEARNING

by

Mark Nemecek

Department of Computer Science
Duke University

Date: ______________________________

Approved:

____
Ronald Parr, Supervisor

____
Rong Ge

____
Carlo Tomasi

____
Michael Zavlanos

An abstract of a thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Department of Computer Science
in the Graduate School of
Duke University

2019

Copyright © 2019 by Mark Nemecek
All rights reserved

Abstract

The notion of distance plays and important role in many reinforcement learning (RL) techniques. This role may be explicit, as in some non-parametric approaches, or it may be implicit in the architecture of the feature space. The ability to learn distance functions tailored for RL tasks could, thus, benefit many different RL paradigms. While several approaches to learning distance functions from data do exist, they are frequently intended for use in clustering or classification tasks and typically do not take into account the inherent structure present in trajectories sampled from RL environments. For those that do, this structure is generally used to define a similarity between states rather than to represent the mechanics of the domain. Based on the idea that a good distance function in such a domain would reflect the number of transitions necessary to get to from one state to another, we detail an approach to learning distance functions which accounts for the nature of state transitions in a Markov decision process, including their inherent directionality. We then present the results of experiments performed in multiple RL environments in order to demonstrate the benefit of learning such distance functions.

Contents

Abstract **iv**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 Background **4**

3 Distance Functions in Approximate RL **6**

3.1 What is a Good Distance Function? 6

3.2 Discounted k-Nearest Neighbors 8

4 Manifold Learning **11**

5 Transition Space Distance Learning **14**

6 Experimental Results **18**

6.1 Cartpole 19

6.2 Catcher 21

6.3 Acrobot 21

6.4 Pendulum 23

6.5 Lunar Lander 26

7 Conclusions **28**

References **30**

List of Tables

6.1 Time to learn distance function, execution run time per episode, and number of remaining states 23

List of Figures

3.1 Problems with L_2 distance 7

4.1 Comparison of Manifold Learning Techniques 13

6.1 Cartpole Learning Curves - Mean and standard deviation over 10 training sequences 19

6.2 Catcher Learning Curves - Mean and standard deviation over 10 training sequences 20

6.3 Acrobot Learning Curves - Mean and standard deviation over 10 training sequences 22

6.4 Pendulum Learning Curves - Mean and standard deviation over 10 training sequences 24

6.5 Lunar Lander Learning Curves - Mean and standard deviation of the respective values over 10 training sequences 25

Chapter 1

Introduction

For many algorithms used in machine learning, some concept of distance is an integral component. Often, such a concept is used to define a level of similarity between data points in order to cluster or classify them for the purpose of analyzing the data set. Many notions of distance exist, but choosing which one to use for a particular problem may be yet another parameter to optimize.

Non-parametric and kernel-based RL methods often rely on some form of distance function, but it can be difficult to choose a useful one. A typical default is to use the L_2 norm in the ambient feature space, but this may not capture the peculiarities of the environment and, as described below, it may not capture the effect of state transitions. Exploration in both deep and shallow methods, while not a focus of this paper, can also be reliant on a notion of distance, whether explicitly or implicitly. Thus, there are many motivations for and possible applications of an approach to distance function learning that is oriented towards RL tasks.

Although a wide variety of methods for manifold learning and distance function learning exist, their application to RL can be non-trivial because they are targeted toward other types of tasks and do not account for the difference between proximity in transition space and proximity in ambient space. RL involves the collection of sample sets by acting in an environment and these sample sets consist of a number of *trajectories*, or sets of sequentially observed states, through the state space. In many domains, transitions can result in large jumps in the ambient state space, while states that are relatively close in ambient space could have significantly different values due to the presence of obstacles or rewards.

We detail a new method, Transition Space Distance Learning (TSDL), which is based on the idea that a good distance function for RL environments should reflect the mechanics of moving between states. To be more specific, TSDL learns distance functions which represent the distance in transition space rather than ambient space for such environments based on observed trajectories. This distance in transition space, which reflects the number of transitions necessary to move from one state to another, is a natural way to consider distance in RL environments.

Methods related to this new approach have been used with proto-value functions [Mah09], but there Laplacian eigenmaps were used to find basis functions for value functions rather than for learning a distance function and there was not a crisp distinction between transition space and other distances. Also, a variety of other methods for learning metrics have been presented. Other dimensionality reduction techniques could induce distance functions, such as locally linear embedding and ISOMAP, which are also unsupervised, nonlinear, embedding-based methods. However, these methods are largely intended for use in clustering and classification tasks rather than reinforcement learning.

HOLLER [TKS11] attempts to learn a distance metric specifically for reinforcement learning environments based upon the similarity between transition vectors where the same action is taken. However, this approach learns a Mahalanobis metric which is applied to the original feature space, which means that it is subject to the limitations of a linear transformation. Emigh et al. [EKB+16] present an approach which attempts to learn a weighting for the state features which can better predict state-action values by learning a metric which gives a lower distance between states whose state-actions values are more similar.

We present results on multiple control tasks, including two classic problems which are commonly used in RL literature, which compare the performance of the Dis-

counted k-Nearest Neighbor (DkNN) algorithm using our learned distance functions to that of Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) - both policy gradient methods - and to DkNN using the L_2 distance. These results demonstrate that our approach can extract more useful information from a given training sample set to achieve better performance in cases where few trajectories are available, and that the benefit of using a TSDL distance function over L_2 with DkNN becomes more significant in more complex environments.

Chapter 2

Background

A *Markov Decision Process* (MDP) can be represented as a 5-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ consisting of the state space $\mathcal{S}$, which may be discrete or continuous; the action space $\mathcal{A}$, which may also be discrete or continuous; the Markovian transition model P, where $P(s'|s,a)$ is the probability of transitioning to state s' after taking action a while in state s; the reward function R, where $R(s,a)$ is the reward for taking action a while in state s; and the discount factor $\gamma \in [0,1)$, which is applied to future rewards. A *policy* π maps states to actions where $\pi(s)$ denotes the action to take while in state s.

A *value function* V can be defined such that $V^\pi(s)$ is the expected total discounted reward for starting in state s and performing actions according policy π. We can also define an *action-value function* $Q^\pi(s,a)$, which gives the value of taking action a from state s and then following π afterwards. In general, the goal of a learning agent is to learn the optimal policy π^* which maximizes the expected future reward from each state. We can express the optimal value function V^* and action-value function Q^* via the Bellman equation as:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s,a,s') V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} Q^*(s',a')$$

In reinforcement learning, the learning agent does not have access to the transition model P or the reward function R of an MDP. However, experience is collected by acting in the environment and usually takes the form of a set of tuples (s,a,r,s').

The agent must then use these samples to learn a policy for the environment by using either a model-based approach where it attempts to learn the underlying MDP or a model-free approach where it learns a value or action-value function directly and extracts a policy from that function.

Chapter 3

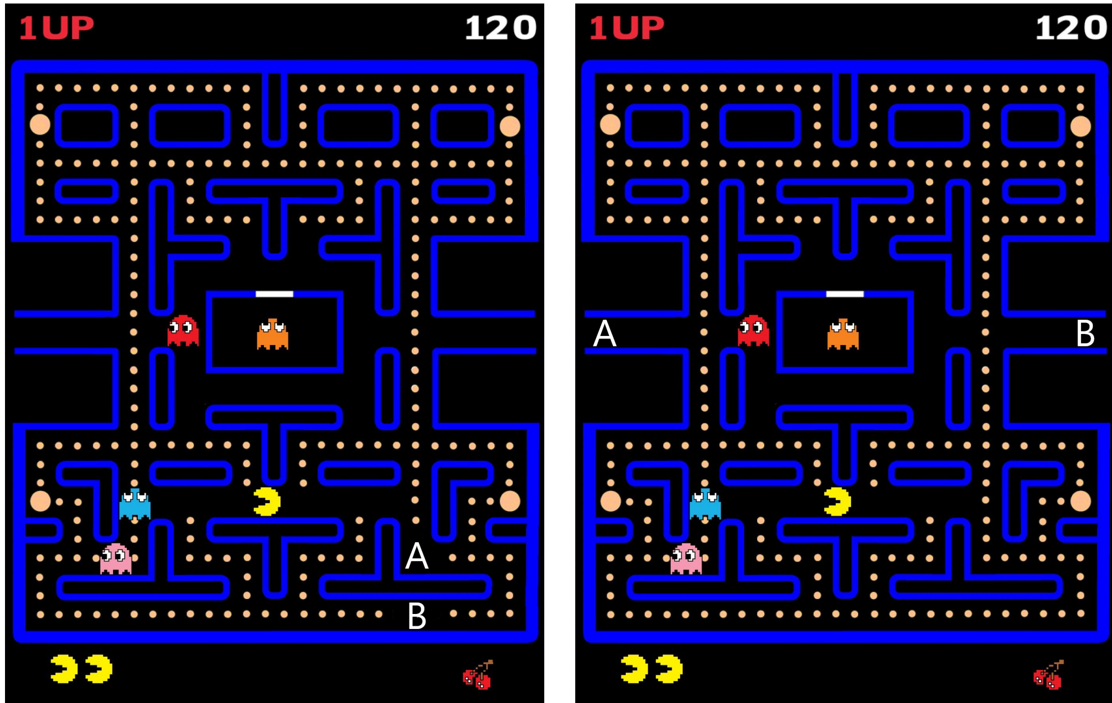
Distance Functions in Approximate RL

Distance functions can often play an explicit or implicit role in value function approximation methods. Non-parametric methods for RL often use a distance function to determine how to average across states that are "close" to each other. This was done in Kernel-based RL [OS02], which was one of the first provably convergent RL algorithms in continuous space. As one might expect, the performance of such algorithms can be highly dependent upon the choice of distance function. This issue persists in RKHS versions of RL algorithms [TP09]. Methods that make assumptions about the smoothness of the value function [PP11] also need a reasonable notion of distance to operate efficiently. Even methods that incorporate some sort of manifold learning [Mah09] into the value function approximation still need a notion of distance that is at least locally reasonable. One of the challenges in getting such methods to work empirically is how to construct a graph for the graph Laplacian that combines proximity in ambient space with proximity in transition space. Our work here helps give some insight into that question.

3.1 What is a Good Distance Function?

When using a distance function explicitly, there are many such functions to choose from, such as the Manhattan distance or some Mahalanobis metric. A naive, but common, choice would be to simply use the L_2 distance in the ambient space, i.e., the feature space used to represent the state. However, these choices frequently do not capture the dynamics of the environment in which an agent is operating.

For example, in the game Pac-man, an agent must move Pac-man around the



(a) Obstacles between the start and goal

(b) Teleportation from start to goal

Figure 3.1: Problems with L_2 distance

maze to collect dots and avoid or eat the ghosts, depending on his current status. We consider a case where Pac-man's position is represented as part of the state space by his x and y coordinates in the maze. In Figure 3.1(a), if we wanted to move from position A to position B, we would need to move all the way around the obstacle between those points, but the L_2 distance between them in the state space would measure the length of a line segment which passes through the obstacle, thus underestimating the travel distance required. In a related case, if we wanted to move between points A and B in Figure 3.1(b), we could move to the left edge of the board and be teleported to the right edge, immediately adjacent to B. Here, the L_2 distance would drastically overestimate how far we needed to move Pac-man to reach B, as it would measure the line segment from A to B, which passes through the center of the maze.

In these examples, we see that the L_2 distance can be a poor indicator of how far an agent must travel to move between two states. In order for a distance function to account for the dynamics of the environment, it should reflect actual paths through state space which an agent can follow. As any such path would be made up of transitions between states, a good measure of distance would therefore be proportional to the total number of transitions required for an agent to move between the states in question. It is worth noting here that this idea of distance is inherently directional in many domains – frequently, it is not possible to return to a previously-visited state, but even in cases where it is possible, it will often require a different number of transitions to do so.

3.2 Discounted k-Nearest Neighbors

In this paper, we focus on what is arguably the weakest form of function approximation that uses a distance function: a variation on k-Nearest Neighbor (kNN) approximation. The reason for this is to focus attention on benefit of a learned distance function in a manner that doesn't convolve the effect of distance function learning with function approximation, though we acknowledge that this limits the scalability to higher dimensions. This weak function approximator can also be thought of as building an approximate, non-parametric model in which transitions from the k nearest neighbors serve as a Monte Carlo approximation of the transition model in the Bellman equation.

The version of kNN approximation we use, which we call Discounted k-Nearest Neighbors (DkNN), applies a penalty to the values from neighbors based on their distance from the query point. While originally inspired by the C-PACE [PP13a] algorithm's formula for action-value functions, DkNN differs by applying a multiplicative penalty based on the discount factor which is exponential in the distance

rather than a subtractive, linear term. Intuitively, if the distance function represents the number of transitions needed to reach the neighbor, then this discount factor accounts for the accumulated discount that would be applied to that neighbor's value as a successor of the query point if we unrolled the Bellman equation.

If $\tilde{Q}$ is the estimated action-value function and (s_i, a_i, r_i, s'_i) is a sample, we let

$$\tilde{Q}(s_i, a_i) = \frac{1}{k} \sum_{j=1}^{k} \gamma^{d_{ij}} [r_j + \gamma \max_{a'} \tilde{Q}(s'_j, a')] \tag{3.1}$$

where γ is the discount factor, $d_{ij} = d(s_i, a_i, s_j, a_j)$ is the distance between the state-action pairs, and $j = 1$ to k are the k nearest sampled neighbors to (s_i, a_i), possibly including itself. We can define an approximate Bellman operator, $\tilde{B}$, based on equation 3.1 as follows:

$$\tilde{B}\tilde{Q}(s_i, a_i) = \frac{1}{k} \sum_{j=1}^{k} \gamma^{d_{ij}} [r_j + \gamma \max_{a'} \tilde{Q}(s'_j, a')] \tag{3.2}$$

Theorem 1. *$\tilde{B}$ has a unique fixed point.*

Proof. Suppose we have action-value functions Q_1 and Q_2 such that $||Q_1 - Q_2||_\infty = \epsilon$,

then $\forall(s,a) \in \mathcal{S} \times \mathcal{A}$,

$$
\begin{aligned}
\tilde{B}Q_1(s,a) &= \frac{1}{k}\sum_{j=1}^{k} \gamma^{d_{ij}} [r_j + \gamma \max_{a'} Q_1(s'_j, a')] \\
&\leq \frac{1}{k}\sum_{j=1}^{k} \gamma^{d_{ij}} \left[r_j + \gamma \max_{a'} \left[Q_2(s'_j, a') + \epsilon\right]\right] \\
&= \frac{1}{k}\sum_{j=1}^{k} \left[\gamma^{d_{ij}} \left[r_j + \gamma \max_{a'} Q_2(s'_j, a')\right] + \gamma^{d_{ij}}\gamma\epsilon\right] \\
&\leq \frac{1}{k}\sum_{j=1}^{k} \gamma^{d_{ij}} \left[r_j + \gamma \max_{a'} Q_2(s'_j, a')\right] + \gamma\epsilon \\
&= \tilde{B}Q_2(s,a) + \gamma\epsilon \\
\implies \tilde{B}Q_1(s,a) &\leq \tilde{B}Q_2(s,a) + \gamma\epsilon
\end{aligned}
$$

Thus, $\tilde{B}Q_1(s,a) \leq \tilde{B}Q_2(s,a) + \gamma\epsilon$. As we can perform the same derivation after swapping Q_1 and Q_2, we can further conclude that

$$||\tilde{B}Q_1(s,a) - \tilde{B}Q_2(s,a)||_\infty \leq \gamma\epsilon$$

and thus that $\tilde{B}$ is a contraction in max-norm and has a unique fixed point. □

Theorem 1 tells us that repeated application of $\tilde{B}$ will converge asymptotically to a fixed point regardless of the initial action-value function with which we start. Therefore, we can create value iteration or policy iteration algorithms around this operator to learn the action-value function at this fixed point.

Chapter 4

Manifold Learning

Manifold learning is a general approach to nonlinear dimensionality reduction which assumes that the data are not only inherently low-dimensional, but that they lie on a *manifold* – a topological space which resembles Euclidean space locally around any given point – that is embedded in the high-dimensional ambient space. Using this approach, there are many techniques which attempt to recover the manifold. Once the manifold is known, the data can compared in the new space. Often, this is used for clustering and classification tasks.

Some of these methods include locally linear embedding (LLE) [RS00], which represents each data point as a linear combination of its neighbors and attempts to find a low-dimensional embedding which retains these combinations via eigendecomposition, and its variants, Modified LLE [ZW06] and Hessian LLE [DG03], which address limitations of LLE when the number of neighbors is larger than the number of input features. Other techniques take different approaches, such as Local Tangent Space Alignment (LTSA) [ZZ02], which finds a manifold where the tangent spaces of the neighborhoods of data points are aligned, Laplacian Eigenmaps [BN03], which constructs a graph and uses the eigenvectors of its Laplacian as an embedding, classical multidimension scaling (MDS) [KW78], which chooses an embedding which maintains pairwise distances between the data points, and t-distributed Stochastic Neighbor Embedding (t-SNE) [LH08], which calculates the pairwise probabilities that the data points are related and finds an embedding in which these probabilities are retained. The approach we detail in Chapter 5 is inspired by, and most closely related to, ISOMAP [TSL00].

ISOMAP is a method of manifold learning that starts with constructing a graph from the data by using each point as a node and adding an edge between each pair nodes if the nodes are "close". Two known techniques for generating these edges are

- ϵ-neighborhoods (or ϵ-balls): the nodes for two points i and j are connected if $||i - j||_2 < \epsilon$
- k-nearest neighbors: nodes i and j are connected if i is one of the k-nearest neighbors of j or j is one of the k-nearest neighbors of i

Once it is determined between which nodes edges should exist, the weights assigned to those edges must be selected. In contrast to an approach like Laplacian eigenmaps[BN03], weights for any edges between nodes are typically set to the Euclidean distance between the corresponding data points, rather than using binary weights or a heat kernel. ISOMAP then calculates the pairwise distances between all of the data points as the shortest path distances in the graph and attempts to recover an embedding which preserves these distances using classical multidimensional scaling (MDS) [KW78].

A toy problem sometimes used to illustrate the effectiveness of manifold learning techniques is the Swiss Roll, as shown in Figure 4.1. The data are drawn from a shape which resembles a sheet of paper which has been loosely rolled up in a direction parallel to one edge such that the cross section looks like a spiral with gaps between the layers which are large proportional to the thickness of the layers. As such, the data can be thought of as lying (approximately) on a 2D manifold which has been embedded in 3D space. In Figure 4.1, we show results for applying a number of different manifold learning algorithms implemented Scikit-learn [PVG+11] to a Swiss Roll dataset. As we can see, the different approaches vary in their effectiveness at "unrolling" the Swiss Roll to recover a rectangular shape in two dimensions.

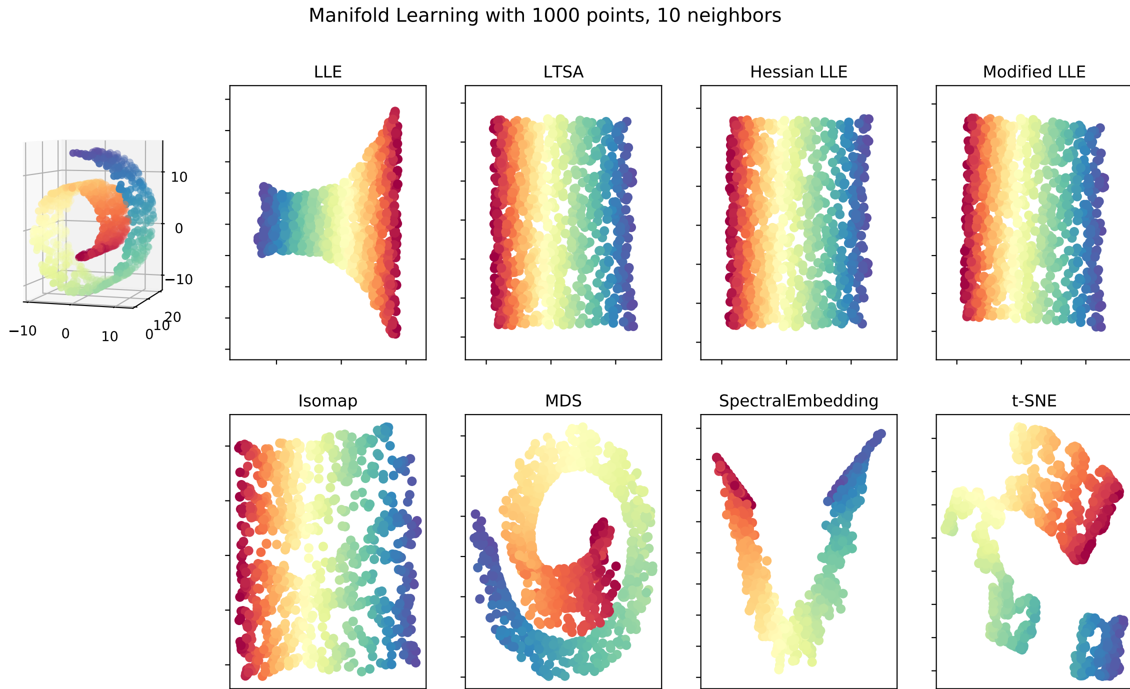


Figure 4.1: Comparison of Manifold Learning Techniques

While these manifold-based techniques can be effective for some problems and one could calculate the distance between two data points as their L_2 distance in the learned embedding, the Euclidean nature of that embedding precludes the use of directed distances. As discussed previously, the mechanics of RL environments are often inherently directional, thus frequently making the techniques presented in this chapter unsuitable for learning a distance function which reflects the idea of a "good" distance function detailed in Chapter 3.

Chapter 5

Transition Space Distance Learning

In this new approach, which we call Transition Space Distance Learning (TSDL), we learn distance functions using a method that exploits the inherent structure of the states and transitions observed in a reinforcement learning environment to more accurately reflect the mechanics of the environment. More specifically, this method learns from observed trajectories to represent the distance in transition space rather than ambient space. In essence, TSDL learns distances which represent the number of transitions necessary to move from one state to another. This is a very intuitive notion of distance as it accounts for dynamics of the environment where something like ambient L_2 distance does not accurately reflect the mechanics of moving between the states, e.g., when blocked by an obstacle and forced to go around or when something akin to teleportation through state space occurs. While our method is inspired by ISOMAP, the need for directed distances precludes the use of techniques based on Euclidean embeddings like ISOMAP itself and the other methods described in Chapter 4.

We construct a graph in a fashion that is similar to ISOMAP, but while we retain the idea of creating edges between a given point and the points in its ϵ-neighborhood, we make use of directed edges based on the observed transitions, as shown in Algorithm 1. We consider each transition in the sample set and add a directed edge from the start state to the end state of the transition, which we call a *transition edge*, as well as from the start state to each state in its ϵ-neighborhood, which we call ϵ-*edges*.

The edges added to the graph are weighted according to a scheme that diverges significantly from standard ISOMAP, and which we term *proportional weighting*. As

each transition (s, s') is considered, the transition edge is given a weight of 1, while the weight for an ϵ-edge between s and $s'' \in N_s$ (where N_s is the ϵ-neighborhood of s), $w(s, s'')$ is set based on the ambient L2 distance between those states normalized to the ambient L2 distance of the transition. More specifically, where $d_2(\cdot, \cdot)$ is the L2 distance in ambient space, the weight is calculated as follows:

$$w(s, s'') = \frac{d_2(s, s'')}{d_2(s, s')}$$

The purpose of this weighting scheme is that we want the transition distance to drive the scale of how neighbors are weighted.[1]

After constructing this graph, the distance between observed states s_0, s_1 under our learned distance function is simply the shortest path distance from s_0 to s_1 in the graph. The pairwise distance matrix can be calculated efficiently in advance and the distance between any two states then retrieved as needed. However, as novel states are by definition not included in the data used to construct the graph, there are no corresponding nodes for them and we cannot compute the shortest path distance from a novel state to an observed state exactly. We can modify the approach from Landmark ISOMAP [ST03] to approximate this distance.

To do so, we consider the k-nearest neighbors of our novel state s in the training data, N_s. Let $d_{max} = \max_{s' \in N_s} d_2(s, s')$, then $\forall s_o \in \tilde{S}$ where $\tilde{S}$ is the set of observed training states,

$$d(s, s_o) = \min_{s' \in N_s} \left[\frac{d_2(s, s')}{d_{max}} + d(s', s_o) \right]$$

Essentially, this approximates $d(s, s_o)$ by considering paths which go through each of the neighbors of s and choosing the shortest one. The normalization with d_{max} is done so as to retain the ranking of the lengths of the paths through all of the

[1] Edge weights denote proximity, not probability. Incorporating both proximity and probability, which would add robustness to transition noise, is deferred for future work.

neighbors while rescaling to be closer to the transition space normalization done when constructing the graph.

Algorithm 1 Build Proportional Weight Matrix

1: //U: Set of unique states in the training set
2: //T: Set of transitions in the training set
3: **function** BuildPropWeightMatrix(U,T)
4: $W \leftarrow \infty$ //Weight matrix initialized with no edges
5: **for all** $s, s' \in T$ **do**
6: $W_{s,s'} \leftarrow 1$
7: $d_t \leftarrow d_2(s, s')$
8: **for all** $s'' \in N(s)$ **do**
9: **if** $W_{s,s''} \neq 1$ **then**
10: $d_n \leftarrow \frac{d_2(s,s'')}{d_t}$
11: $W_{s,s''} \leftarrow \min\{W_{s,s''}, d_n\}$
12: **end if**
13: **end for**
14: **end for**
15: **return** W
16: **end function**

As the size of the pairwise distance matrix grows quadratically with the number of distinct observed states, increasing the size of the training set can quickly lead to the need for large amounts of memory and disk space to process and store the matrix, as well as significantly increased computation time to work with it.

Whereas ISOMAP applies MDS to the distance matrix to find a lower-dimensional Euclidean embedding in which to work, we make use of directed distances which cannot be represented in such an embedding. Instead, to combat resource requirement growth, we employ a sparsification procedure (see Algorithm 2) which chooses a set of representative states and aggregates their weight vectors with their η-neighbors - those within a radius of η under L_2 in the ambient space. These representative states are the only ones retained in the distance matrix.

Algorithm 2 Sparsify Adjacency Matrix

```
1: //W: Unsparsified weight matrix
2: //U: Set of unique states in the training set
3: //η: Radius to use for aggregation
4: //N_η: η-neighbor map (excludes self)
5: function Sparsify(W, U, η, N)
6:     W' ← ∞ //Initialized with no edges
7:     R ← RepresentativeStates(U, η)
8:     for all r ∈ R do
9:         W'_{r,r} ← 0
10:        for all r' ∈ R ∩ {r} do
11:            W'_{r,r'} ← min_{n∈N_η(r)} W_{n,r'}
12:        end for
13:    end for
14:    return W', R
15: end function
16:
17: function RepresentativeStates(U, η)
18:    r ← Uniform(U) //Choose the first state at random
19:    R ← {r} //Set of representative states
20:    d_f ← max_{s'∈U} d_2(r, s') //L_2 Distance to furthest state
21:    while d_f > η do
22:        r ← argmax_{s'∈U} min_{s∈R} d_2(s, s')
23:        R ← R ∪ {r}
24:        d_f ← max_{s'∈U} min_{s∈R} d_2(s, s')
25:    end while
26:    return R
27: end function
```

Chapter 6

Experimental Results

We test TSDL as a component of the Discounted k-Nearest Neighbor method (TSDL-DkNN) in four domains implemented as benchmarks in the OpenAI Gym toolkit [BCP+16] and one from the PyGame Learning Environment [Tas16], and compare performance results with Trust Region Policy Optimization (TRPO) [SLA+15] as implemented in the rllab library [DCH+16] and Proximal Policy Optimization (PPO) [SWD+17] as implemented in the OpenAI baselines library [DHK+17]. The training data for TSDL were collected with a uniform random policy as a batch prior to training while TRPO and PPO were allowed to explore as part of their standard training method. For TRPO and PPO, it was found that the initial, untrained networks (which used Xavier initialization [GB10]) were biased towards distributions over the action indices such that performance varied greatly depending which index was assigned to which action. To account for this bias, we aggregate their performance over all permutations of the actions in each environment. As no such bias was observed with TSDL-DkNN, it's results are based only on the canonical action indices. In all cases, the best-performing parameter set found via a parameter sweep is used for comparison, and the policy gradient methods used a learning rate of 0.01 as this was found to be the most consistently good value. Each policy was tested by executing it for 100 episodes and then aggregating the cumulative rewards for those episodes. The domains are presented in order of increasing difficulty.

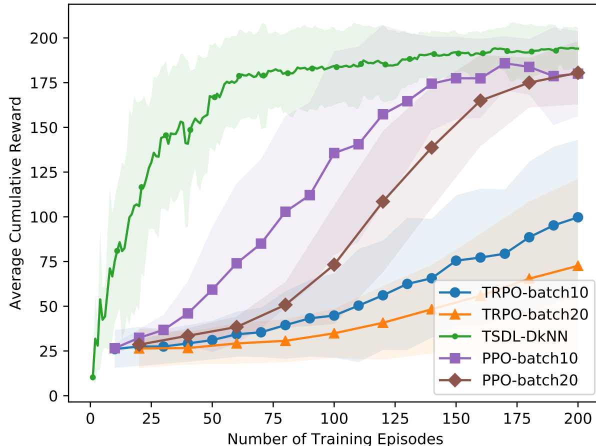


Figure 6.1: Cartpole Learning Curves - Mean and standard deviation over 10 training sequences

6.1 Cartpole

In the Cartpole domain (also know as the Inverted Pendulum problem [WTG96]), an agent must balance a pendulum attached to a cart by only applying forces to the cart, which can only move in one dimension. This is a classic control problem which is frequently used in the literature when testing reinforcement learning algorithms. In the Gym implementation, it has a continuous, four-dimensional state space, actions are deterministic, and each episode can run for a maximum of 200 steps while receiving a reward of +1 for each step. Although nominally four-dimensional, Cartpole is effectively a two-dimensional domain because the position of the cart has little impact on performance in most cases.

Figure 6.1 shows the average cumulative reward over 100 test episodes as a function of the number of training episodes. Here TSDL used $\epsilon = 0.25$, while TRPO and PPO used networks with two hidden layers of eight neurons. As we can see, the policies learned by TRPO increase in performance slowly with the number of training episodes with little difference between a batch size of 10 episodes and a batch size of

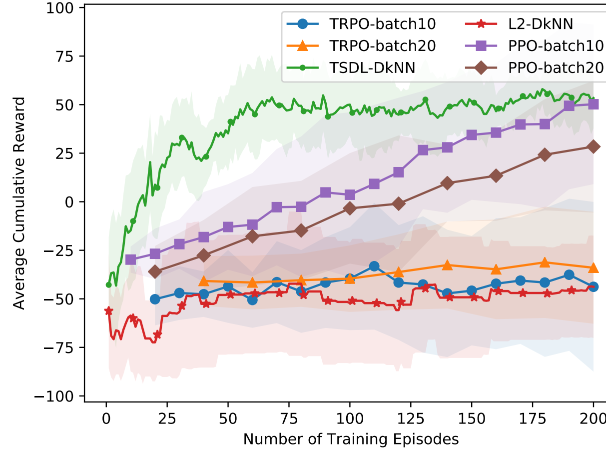


Figure 6.2: Catcher Learning Curves - Mean and standard deviation over 10 training sequences

20 episodes while PPO learns considerably faster - especially with the smaller batch size. In comparison, DkNN using the TSDL distance function begins to approach the maximum possible performance - a cumulative reward of 200 - with fewer than 200 episodes of training data. While TRPO and PPO will reach a similar performance level given enough data, there is some information present in the smaller data sets which DkNN takes advantage of but TRPO and PPO do not.

It is worth noting that in Cartpole, the superior performance of the TSDL-DkNN curve is largely due to DkNN. We don't show the graph for L2-DkNN because it is so close to the TSDL-DkNN curve for both mean and standard deviation. Since Cartpole is effectively a two-dimensional problem, it shouldn't be surprising that the ambient distance function is sufficient. As shown in subsequent sections, the benefit of a learned distance function increases as problems get more difficult.

6.2 Catcher

In the Catcher domain, an agent must move a paddle left or right to catch pieces of fruit which drop from above it. We use an implementation from the PyGame Learning Environment [Tas16], which presents a four-dimensional state space with deterministic actions. The environment has been modified so that an episode consists of a single fruit drop with the agent receiving a reward of +1 for catching the fruit and -1 for missing it. This change was made in order to provide more consistency in the number of samples per episode to allow for more direct comparisons between methods. It is justified because it is possible for an agent to catch the fruit dropped from any location regardless of the position of the paddle when the fruit is dropped, so the optimal policies will not change due to this modification.

In this domain, TRPO and L2-DkNN have similar performance and see little improvement across the range of sample sizes. In comparison, PPO shows significant improvement, but TSDL-DkNN with $\epsilon = 0.19$ shows a large advantage for all but the largest sample sets. Moreover, even though TSDL-DkNN and PPO eventually reach the same level of performance, TSDL-DkNN reaches this level with far fewer episodes of training data – about 75 episodes – where PPO requires around 200.

6.3 Acrobot

In the Acrobot domain [DS94], an agent attempts to "swing up" a double pendulum where the joint on the end is fixed and the agent can only apply torque to the joint between the links. The agent's goal is to swing the tip of the second pendulum up above a certain height, which requires swinging back and forth to build up momentum. This is a problem frequently used to study agents' performance in control tasks. We use the reference implementation from OpenAI Gym [BCP+16], which has

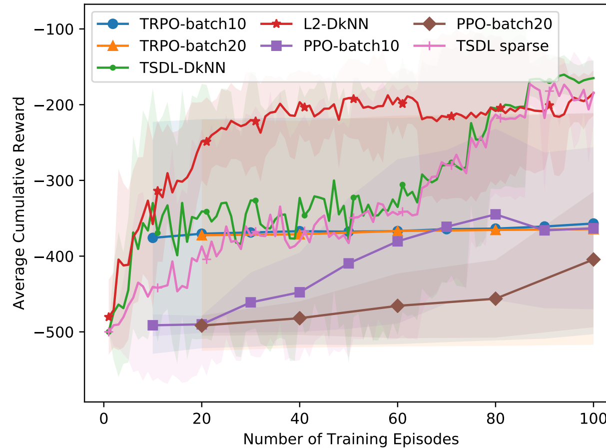


Figure 6.3: Acrobot Learning Curves - Mean and standard deviation over 10 training sequences

a six-dimensional state space with discrete, deterministic actions. An agent receives a reward of -1 per timestep unless a goal state is reached, in which case it receives a reward of 0.

In Figure 6.3, we compare the performance of TSDL-DkNN to L2-DkNN, TRPO, and PPO. TSDL used $\epsilon = 0.13$ while TRPO and PPO used two-layer networks with eight nodes per layer. Due to the reward structure and 500 step limit, a uniform random agent may take many episodes to reach a goal state for the first time. Given this, it is not surprising that it takes more episodes to see significant performance improvement than in some other environments. While TSDL-DkNN and TRPO starts out with similar performance, TRPO's curve remains largely flat while TSDL-DkNN sees much improvement with larger training set sizes. That said, the results here are a bit different than the previous domains as L2-DkNN shows better performance than TSDL-DkNN for a large number of training sets, but TSDL-PkNN does eventually surpass L2-DkNN and reaches its peak sooner.

It must be noted that the "stair step" shape of parts of the curve for the TSDL

Table 6.1: Time to learn distance function, execution run time per episode, and number of remaining states

	unsparsified	sparsified
Learning Time (s)	4074.53	684.57
Run Time (s)	36.70 ± 14.59	6.27 ± 2.58
Number of States	49620	13860

distance function is a result of the nature of the reward structure for this environment. An agent given these data will only have observed -1 rewards for all states in training sets without a successful episode. As the values for the curve are averaged over 10 training sequences, the steps correspond to reaching a number of episodes such that an additional training sequence includes a terminal episode and thus resulting in better performance from policies learned from that sequence. It seems that TSDL-PkNN may be more sensitive to this issue than L2-DkNN in this environment. Although it was not a focus of our work, this suggests that combining exploration with distance function learning could be fruitful, as discussed further in Section 7.

Table 6.1 shows the run time to learn the distance function, the episode run time during policy execution, and the number of states retained in the distance matrix for an unsparsified and a sparsified distance function learned from a 100-episode training set. This indicates that a significant reduction in computation and storage with little to no change in performance, as seen in Figure 6.3, is possible with the sparsification routine. Here the sparsification process was applied with $\eta = 0.06$ as going above that threshold caused performance to begin degrading more significantly.

6.4 Pendulum

The Pendulum domain is another classic control problem which bears similarities to both the Acrobot and Cartpole environments. It involves a single-link pendulum

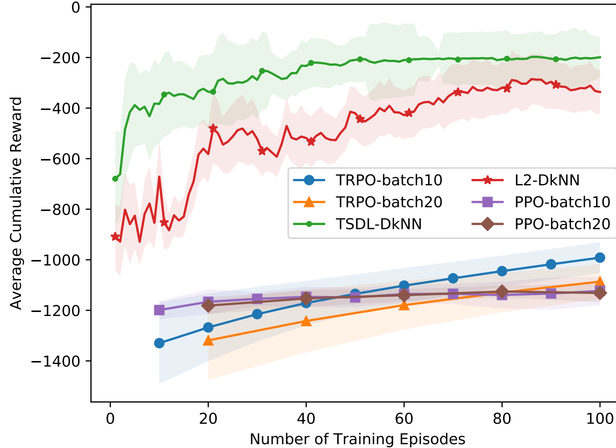
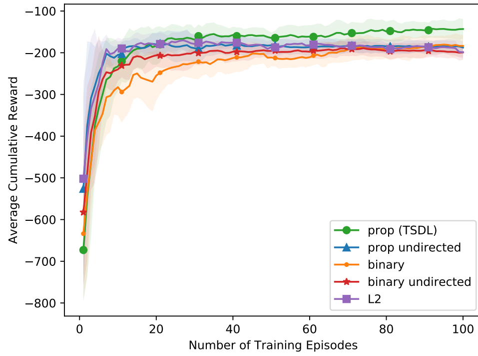


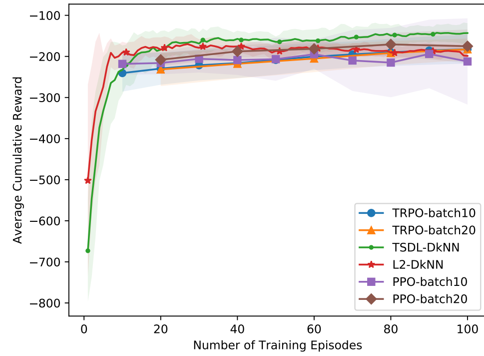
Figure 6.4: Pendulum Learning Curves - Mean and standard deviation over 10 training sequences

which is attached to a fixed point about which torque is applied in order to swing the pendulum up from a downward starting position and then keep it balanced as close to vertical as possible. Thus, it is both a swing-up task like Acrobot and a balancing task like Cartpole. A large difference from those two tasks, however, is that it uses a dense reward function where the reward given is based on the pendulum's angle from vertical, its angular velocity, and the magnitude of the torque applied. We again use an OpenAI Gym implementation, which has a three-dimensional continuous state space, but we discretize the continuous action space into five actions evenly spread in the allowed torque range.

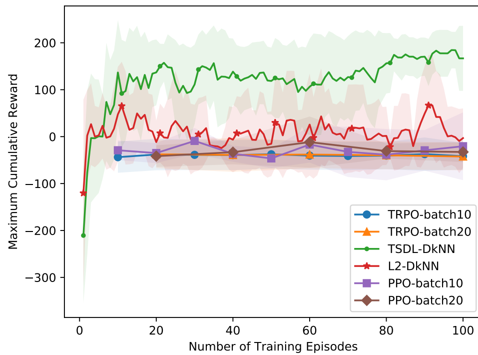
The results for this environment are shown in Figure 6.4, where we can see that there is a large performance gap between DkNN with either distance function and the policy gradient methods. Additionally, TSDL-DkNN (using $\epsilon = 0.06$) shows a significant performance benefit over L2-DkNN for all sample set sizes and plateaus around 60 episodes with an average cumulative reward around -200, which may be a practical limit for this environment.



(a) Ablation study comparing adjacency/weighting schemes



(b) Average cumulative reward



(c) Maximum cumulative reward

Figure 6.5: Lunar Lander Learning Curves - Mean and standard deviation of the respective values over 10 training sequences

While this environment has a lower dimensional state space than those previously presented, it is in some ways more complicated because it requires an agent to learn two distinct behaviors: the swing-up, which requires building up angular velocity, and the regulation of the pendulum to keep it balanced in an upright position with minimal torque applied and minimal angular velocity.

6.5 Lunar Lander

OpenAI Gym's Lunar Lander environment is a simulation of the process of landing a vehicle on the lunar surface that is inspired by a number of video games with similar descriptions. An agent's goal is to land the vehicle on the surface without the body making contact with the ground. The eight-dimensional state space and its discontinuities make this a more difficult problem, but reward shaping is used to provide a reward function more conducive to learning. We use a modified version of the reference environment that provides deterministic actions.

To understand why each aspect of TSDL is helpful, we look to an ablation study. We compare the performance using DkNN with an L_2 distance function and graph distance functions learned using four schemes for constructing the weight matrix: proportional weighting with directed edges (TSDL itself), proportional weighting with undirected edges, binary weighting with directed edges, and binary weighting with undirected edges. The results of the study can be seen in Figure 6.5(a) and show us that while proportional weighting results in better policies than binary weighting for both the directed or undirected case, using directed edges will result in better policies when using those proportional weights. This is consistent with the intuition that many transitions between states are inherently one directional, so an accurate distance function must account for this. This is also a reason why we cannot simply use ISOMAP itself or perform MDS on our distance matrix - the resulting embedding is presumed to be a Euclidean space, which does not allow for different distances between two states depending on the direction of travel. Performance with the L_2 distance is comparable to the proportional undirected case. While these two have a slight advantage over TSDL below 20 episodes, TSDL's performance after 30 episodes is never matched by the other distance functions.

When comparing to the policy gradient methods for this environment, we also

examine the results in terms of the average cumulative reward over 100 episodes, as before, as well as the maximum cumulative reward. TSDL performed best with $\epsilon = 0.05$ and the policy gradient methods used two-layer networks with 32 nodes per layer. In Figure 6.5(b), we can see that TRPO and PPO have very similar average performance and that TSDL achieves better results given the same number of training episodes. DkNN with the L2 distance starts off very close to TSDL, but at about 20 episodes, it's performance largely plateaus at a level in line with TRPO and PPO while TSDL continues to improve at a faster rate.

We must note that we are only considering only a small number of episodes relative to the number expected to be necessary to "solve" the environment and reliably land the vehicle in the target zone. Although TSDL has only a small advantage in comparison based on the average after 100 training episodes, a closer examination shows that TSDL is the only method to discover a policy that ever lands in the target zone. Figure 6.5(c) shows the maximum performance achieved for each learned policy over 100 test runs. This maximum is then averaged over the 10 training runs used for each method. The positive average for TSDL shows that TSDL is consistently finding policies than can land in the target zone in some test cases, while the averages close to zero for the other methods indicate that these methods rarely or never find policies that land in the target zone in any of their 100 test episodes.

Chapter 7

Conclusions

We presented a method for learning distance functions which respect the observed transition mechanics of an environment in order to enhance reinforcement learning algorithms that use a distance function. In conjunction with just a simple variant a kNN approximator, the DkNN algorithm, this method learns distance functions which improve the quality of policies relative to using the L_2 distance with the gap widening in more complex domains, and offers performance advantages over TRPO and PPO in these deterministic environments with small sample set sizes. It is noteworthy that DkNN did so well in comparison to to the policy gradient methods as they performed exploration where DkNN did not, as it used sample sets collected in batch with a uniform random policy. A natural extension would be to combine our approach with something like C-PACE [PP13a], which relies heavily upon a distance function to explore a continuous MDP in a sample-efficient manner. Although we expect our approach would gracefully handle small amounts of transition noise, e.g., noise concentrated in a region around a nominal target state, further work is required to extend our approach to handle noisy actions that could allow for transitions to states that are further apart in ambient space.

Our approach learned distance functions which worked well with DkNN, but k-nearest neighbor-based approximators are relatively weak and would not scale well to higher dimensions. This method could be used with any of the more powerful techniques which relies on a distance function, such as the various other non-parametric and kernel-based approaches. As the method is batch-based, it is not efficient to use in an online learning setting, but some approaches to incremental ISOMAP could

potentially be adjusted to work here as well. The sparsification process should be further explored as it appears promising to allow for use of this method with larger data sets. Alternatively, a parametric representation such as a neural network could be used, which would also have the benefit of making these distance functions easier to integrate with NN-based learning techniques. Even in the presence of strong function approximation, it may be useful to have an explicit notion of distance that can be used to assess coverage of the state space during learning and to guide exploration. Recent work has investigated generic representations like hash functions [THF+17] for this purpose, but the incorporation of a distance function that respects that dynamics of the environment might be beneficial.

References

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003.

[Chu05] Fan Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

[DCH+16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[DG03] David L. Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[DHK+17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[DS94] Gerald DeJong and Mark W. Spong. Swinging up the acrobot: an example of intelligent control. *Proceedings of the American Control Conference*, 2:2158–2162, 1994.

[EKB+16] M. S. Emigh, E. G. Kriminger, A. J. Brockmeier, J. C. Prncipe, and P. M. Pardalos. Reinforcement learning in video games using nearest neighbor interpolation and metric learning. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(1):56–66, March 2016.

[FGMS09] Amir M Farahmand, Mohammad Ghavamzadeh, Shie Mannor, and Csaba Szepesvári. Regularized policy iteration. In *Advances in Neural Information Processing Systems*, pages 441–448, 2009.

[GB10] Xavier Glorot and Y Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[KB09] G.D. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1015–1023, 2009.

[KT12] Branislav Kveton and Georgios Theocharous. Kernel-based reinforcement learning on representative states. In *AAAI*, 2012.

[KW78] J.B. Kruskal and M. Wish. *Multidimensional Scaling*. Number no. 11 in 07. SAGE Publications, 1978.

[LH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[Mah09] Sridhar Mahadevan. Learning Representation and Control in Markov Decision Processes: New Frontiers. *Found. Trends Mach. Learn.*, 1(4):403–565, 2009.

[MM07] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[NHR99] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[OS02] Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161–178, Nov 2002.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[PP11] Jason Pazis and Ronald Parr. Non-Parametric Approximate Linear Programming for MDPs. In *AAAI*, 2011.

[PP13a] Jason Pazis and Ronald Parr. PAC-optimal exploration in continuous space markov decision processes. In *AAAI Conference on Artificial Intelligence*, 2013.

[PP13b] Jason Pazis and Ronald Parr. Sample complexity and performance bounds for Non-Parametric Approximate Linear Programming. In *AAAI*, 2013.

[PP13c] Jason Pazis and Ronald Parr. Sample complexity and performance bounds for non-parametric approximate linear programming. *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, pages 782–788, 2013.

[PTPZ10] M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for Markov decision processes. In *Proceedings of the 27th International Conference on Machine Learning*, pages 871–878, 2010.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[SLA+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[Spo95] M. W. Spong. The swing up control problem for the acrobot. *IEEE Control Systems*, 15(1):49–55, Feb 1995.

[ST03] Vin De Silva and Joshua B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15*, pages 705–712. MIT Press, 2003.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Tas16] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[THF+17] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2750–2759, 2017.

[TKS11] Matthew Taylor, Brian Kulis, and Fei Sha. Metric learning for reinforcement learning agents. *Agents and Multiagent Systems*, pages 2–6, 2011.

[TP09] Gavin Taylor and Ronald Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1017–1024, 2009.

[TSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[WTG96] H. O. Wang, K. Tanaka, and M. F. Griffin. An approach to fuzzy control of nonlinear systems: stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1):14–23, Feb 1996.

[ZW06] Zhenyue Zhang and Jing Wang. Mlle: Modified locally linear embedding using multiple weights. In *Advances in Neural Information Processing Systems*, volume 19, pages 1593–1600, 01 2006.

[ZZ02] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *SIAM Journal of Scientific Computing*, 26:313–338, 2002.