2023

From Human Behavior to Machine Behavior

Zerong Xi
University of Central Florida

LEARNING EFFECTIVE REPRESENTATIONS FOR HUMAN BEHAVIOR MODELING

by

ZERONG XI
M.S. University of Southern California, 2017

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2023

Major Professor: Gita Sukthankar

ABSTRACT

A core pursuit of artificial intelligence is the comprehension of human behavior. Imbuing intelligent agents with a good human behavior model can help them understand how to behave intelligently and interactively in complex situations. Due to the increase in data availability and computational resources, the development of machine learning algorithms for duplicating human cognitive abilities has made rapid progress. To solve difficult scenarios, learning-based methods must search for solutions in a predefined but large space. Along with implementing a smart exploration strategy, the right representation for a task can help narrow the search process during learning.

This dissertation tackles three important aspects of machine intelligence: 1) *prediction*, 2) *exploration*, and 3) *representation*. More specifically we develop new algorithms for: 1) predicting the future maneuvers or outcomes in pilot training and computer architecture applications; 2) exploration strategies for reinforcement learning in game environments and 3) scene representations for autonomous driving agents capable of handling large numbers of dynamic entities.

This dissertation makes the following research contributions in the area of representation learning. First, we introduce a new time series representation for flight trajectories in intelligent pilot training simulations. Second, we demonstrate a method, Temporally Aware Embedding (TAE) for learning an embedding that leverages temporal information extracted from data retrieval series. Third, the dissertation introduces GRAD (Graph Representation for Autonomous Driving) that incorporates the future location of neighboring vehicles into the decision-making process. We demonstrate the usage of our models for pilot training, cache usage prediction, and autonomous driving; however believe that our new time series representations can be applied to many other types of modeling problems.

ACKNOWLEDGMENTS

My first and deepest thanks go to my advisor Dr. Gita Sukthankar for her invaluable guidance and continuous support during my entire Ph.D. journey. She has always encouraged me to explore my research interests and expand my knowledge base to wider areas. She devotes tremendous effort in all aspects to helping students grow toward good researchers. I would like to thank my intern hosts and colleagues Dr. Balakrishnan Varadarajan and Dr. Melody Guan from Waymo, who helped me to start my career in the research area of autonomous driving. I would also like to thank my friends, Zixiang Zhou and Shengnan Hu, for the inspiring time spent together, especially during the lockdown days since the pandemic. My appreciation also goes to my family for their encouragement and support to fight through the dark time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Benefiting from the rapid growth in computational resources, which has facilitated processing large datasets, artificial intelligence (AI) techniques, have made significant advances in the past decade and are now embedded within many types of software applications. In many usage scenarios, AI agents must interact and collaborate with humans. To be effective partners, AI agents should be capable of 1) modeling human behaviors, inferring intentions, and exhibiting good theory of mind (ToM) with human users and 2) behaving intelligently and interactively in complex multi-agent scenarios. This dissertation tackles challenges related to imbuing machines with the necessary cognitive machinery to understand human behavior; we focus on predicting human intentions, exploring complex environments, and representing multi-agent scenarios.

1.1 Intelligent Pilot Training Systems

Pilot training is very labor intensive, requiring instructors to provide both interactive feedback and after-action review; our aim is to create AI systems that can reduce instructor workload. A learning-based system can be useful by inferring the trainees' knowledge as well as predicting their task performance before failures occur.

In chapter 3, we investigate the problem of predicting student flight performance in a training simulation from multimodal features, including flight controls, visual attention, and knowledge acquisition tests. This is a key component of developing next generation training simulations that can optimize the usage of student training time. Two types of supervised machine learning classifiers (random forest and support vector machines) were trained to predict the performance of twenty-three students performing simple flight tasks in virtual reality. Our experiments reveal the following: 1) features derived from gaze tracking and knowledge acquisition tests can serve as an adequate substitute for flight control features; 2) data from the initial portion of the flight task is sufficient to predict the final outcome; 3) our classifiers perform slightly better at predicting student failure than success. These results indicate the feasibility of using machine learning for early prediction of student failures during flight training.

In chapter 4, we investigate the representation of pilot data. Many of the most popular intelligent training systems, including driving and flight simulators, generate user time series data. Instead, we present a comparison of representation options for two different student modeling problems: 1) early failure prediction and 2) classifying student activities. Data for this analysis was gathered from pilots executing simple tasks in a virtual reality flight simulator. We demonstrate that our proposed embedding which uses a combination of dynamic time warping (DTW) and multidimensional scaling (MDS) is valuable for both student modeling tasks. However, Euclidean distance + MDS was found to be a superior embedding for predicting student failure, since DTW can obscure important agility differences between successful and unsuccessful pilots.

1.2 Exploration in Reinforcement Learning

Efficient exploration is a core challenge of reinforcement learning, especially if the task has a long time horizon. In chapter 5, we introduce a method for estimating an upper bound for an exploration policy using either the weighted variance of return sequences or the weighted temporal difference (TD) error. We demonstrate that the variance of the return sequence for a specific state-action pair is an important information source that can be leveraged to guide exploration in reinforcement learning. The intuition is that fluctuation in the return sequence indicates greater uncertainty in the near future returns. This divergence occurs because of the cyclic nature of value-based reinforcement learning; improved estimates of the value function result in policy changes which in turn modify the value function. Although both variance and TD errors capture different aspects of this uncertainty, our analysis shows that both can be valuable to guide exploration. We propose a two-stream network architecture to estimate weighted variance/TD errors within DQN agents for our exploration method and show that it outperforms the baseline on a wide range of Atari games.

1.3 Predicting Data Access in Computer Architectures

Computer architectures require prediction to manage memory and storage operations such as prefetching, caching, and scheduling. Rather than relying on heuristics, a more powerful approach is to employ machine learning to learn general models for storage and retrieval. In chapter 6, we introduce a technique, Temporally Aware Embedding (TAE), for learning correlation functions directly from mixed data sequences. We use a force-based learning model in which the co-occurrence of data elements within a sliding temporal window creates attraction forces that increase the correlation magnitude, while repulsion forces act to distribute points more uniformly across the embedding space. Our experiments show that TAE outperforms both simple and state of the art strategies at predicting the next element of data access traces. The embeddings learned with our approach can be combined with other algorithms to perform prefetching and caching in architectures which require greater intelligence to keep pace with growing user storage demands.

1.4 Scene Representation for Autonomous Driving

In autonomous driving scenarios, behavior prediction and planning are important components of reacting to human drivers' behaviors and understanding the surrounding road environment. Improvements in machine perception have made it feasible for autonomous driving agents to work with semantic information rather than raw sensor data; however driving on roadways with human drivers remains challenging due to the large number and diversity of both static and dynamic objects on the road. This makes both representing and processing the scene data in real-time challenging.

For human drivers, an important aspect of learning to drive is knowing how to pay attention to areas of the roadway that are critical for decision-making while simultaneously ignoring distractions. Similarly, the choice of roadway representation is critical for good performance of an autonomous driving system. An effective representation should be compact and permutation-invariant, while still representing complex vehicle interactions that govern driving decisions. In chapter 7, we introduce the Graph Representation for Autonomous Driving (GRAD); GRAD generates a global scene representation using a space-time graph which incorporates the estimated future trajectories of other vehicles. We demonstrate that GRAD outperforms the best performing social attention representation on a simulated highway driving task in high traffic densities and also has a low computational complexity in both single and multi-agent settings.

CHAPTER 2
LITERATURE REVIEW

2.1 Predicting Flight Performance With Multimodal Features

There is a rich body of related work on predicting student performance [1, 2], pilot monitoring [3, 4], and leveraging visual attention features [5, 6]. Much of the research on predicting student performance has been conducted over the time horizon of a semester long course, using assignment grades as features [1]. These student coursework features can be supplemented with Learning Management System (LMS) data collected from platforms such as Moodle. An LMS system can collect detailed data about student engagement, including clicks, edits, page views, and total time spent online. Disappointingly, a sizable study conducted by Conijn et al. [2] revealed little benefit was gained by adding LMS features to in-between knowledge assessment tests for creating early intervention systems to detect students at risk for course failure. Our research aims to predict performance over a significantly shorter time horizon (minutes rather than months). Rather than using click data, student attention is measured using a gaze tracking system.

Visual attention features have been employed to track many aspects of student cognition, including workload, mind wandering [5], and problem solving progress [7]. Peysakhovich et al. [4] endorsed eye tracking integration as a general tool for enhancing cockpit safety and highlighted both pilot training and performance analysis as fruitful application areas. Here we include features to represent both the visual attention distribution and gaze entropy [6] across cockpit instruments.

Within the machine learning community, there has been previous work on training classifiers on small datasets. Commonly used strategies include transfer learning and supplementing the dataset with synthetic training examples. Our classifiers were trained using the SMOTE [8] technique to create synthetic minority class examples to supplement our small, unbalanced dataset.

2.2 Representing User Time Series Data in Intelligent Training Systems

Previous work on processing time series data includes financial analysis techniques such as autoregressive modeling, sequence learning for natural language processing, and sequence pattern mining for bioinformatics; however most of the techniques tend to be domain specific [9]. Our pilot flight analysis problem is somewhat akin to sketch recognition in that we are aligning spatial time series data. However unlike sketch recognition problems, there is more variance in our time series, and we lack a large master dataset of well executed flight examples. Esling and Agon divide time series methods according to representation techniques, distance measures, and indexing methods. Warping to improve temporal alignment has been shown to be valuable across many domains. There are many variants of it including non-uniform warping to handle local variations, fastDTW [10], and softDTW (which can serve as a loss function). However, rather than using DTW directly, we create an embedding from the DTW output using multidimensional scaling.

We use the following techniques in our work on time series data:

Principal Component Analysis (PCA) PCA [11] is a dimensionality reduction technique that computes a set of orthogonal basis to construct a subspace where the projected data retains most information. While the pairwise distance matrix is readily obtained with DTW, each row is a tuple of distances between the specific data point to a fixed set of referential data points. Therefore, it can be interpreted as the coordinates in a new feature space.

K Nearest Neighbors (KNN) The K Nearest Neighbors (KNN) algorithm is a simple instance based learning method that can be applied to classification problems [12]. In the KNN algorithm, an object is classified by a majority vote of its neighboring examples. Hence, the ’k’ in KNN is the parameter that refers to the number of the nearest neighbors of the target object that included in the majority voting process. The Euclidean distance metric is commonly used to determine the distance between training examples.

Logistic Regression The logistic regression algorithm is commonly applied to binary classification problems [13]. A logistic function is modeled in our work to fit the controller variables.

Naive Bayes Classifier (NB) The Naive Bayes algorithm is a practical classification method based on Bayes’ Theorem [14]. This algorithm operates on a strong independence assumption that each attribute of a class is unrelated to the others. In our work, we applied the Gaussian Naive Bayes algorithm to classify time series data, with the assumption that the classes are distributed according to a Gaussian distribution.

Random Forest (RF) The Random Forest algorithm is a meta classifier generated from a number of decision trees by utilizing random feature selections and bootstrapping various instances of the dataset. Specifically, each individual decision tree produces a classification output, and the RF model combines the prediction of each decision tree in order to make the final prediction [15].

Support Vector Machine (SVM) The SVM algorithm is a kernel-based supervised machine learning model used for classification. In the training phase, SVM constructs a model that maps the decision boundary for each class. Then by increasing the hyperplane margin between different classes, we can increase the distance between each class, in order to increase the classification accuracy [16].

Linear Discriminant Analysis (LDA) The LDA algorithm is one of the most commonly used models for classification. In the LDA algorithm, the most discriminant projection could be exploited by maximizing the distance between different classes while minimizing the distance between the same class [17].

2.3 Leveraging The Variance of Return Sequence for Exploration Policy

Several groups have proposed strategies for balancing exploration/exploitation in deep reinforcement learning including 1) extensions on count-based methods [18, 19, 20]; 2) noise injection techniques [21, 22]; 3) improving uncertainty estimation [23, 24]; 4) driving exploration with intrinsic motivation [25, 26] and 5) entropy-guided architectures [27, 28]. Our proposed weighted-variance guided exploration technique is a compatible addition to some of these other techniques (see Section 5.5 for further details).

Moving from tabular to deep reinforcement learning makes the problem of estimating quantities such as counts and variance more challenging. [19] showed how count-based techniques could be generalized to neural networks by learning a density model on state sequences; pseudocounts for states are then derived from the density model's recoding probability. In contrast our model learns the weighted variance over the return sequence rather than the state sequence.

[23, 29, 30] argue for the necessity of deep exploration and design two environments, a MDP chain and Deep Sea, to illustrate it. The immediate rewards direct the agent away from reaching the goal, and an uninformed search can expect to take $\mathscr{O}(|\mathscr{A}|^N)$ time to explore the goal, where $\mathscr{A}$ is the set of actions. However, the goals in those environments are designed to be achieved with consistent action preferences since action outcomes are deterministic. Therefore, the expected search time reduces to $\mathscr{O}(|\mathscr{A}|)$ among agents who have diverse but consistent preferences on actions. Bootstrapped DQN [23] and Randomized Value Function [29] achieve this effect by sampling a value function or an agent and applying it through a whole training episode. Though many other exploration methods, including ours, perform poorly in these artificial scenarios, they can be adjusted to achieve the goal by adding a set of randomly sampled but fixed noise $\varepsilon \in R^{|\mathscr{A}|}$ on top of the value function over a training episode. This setting gives the agents an action preference which remains consistent in a single training episode and diverse across episodes.

Bootstrapped DQN [23] uses the classic bootstrap principle to estimate uncertainty over Q-values. Agent actions are selected using a single Q function sampled from the posterior distribution. Rather than dithering like noise based models, bootstrapped DQN promotes deep exploration by maintaining policy continuity with regards to the single sampled Q-function. Uncertainty in our proposed architecture is quantified by the weighted standard deviation of the return sequences instead of through data partitions, but like Bootstrapped DQN, our architecture uses multiple (two) heads.

The use of randomness or noise to drive exploration is a common theme across many approaches. NoisyNets [21] directly injects noise into the weights of the neural networks; the parameters of the noise are learned in combination with the network weights. Like NoisyNet, our uncertainty is learned directly by the network, reducing the need for extra hyperparameters. Aleatoric uncertainty poses a common challenge for all exploration methods built on bonuses of uncertainty or curiosity. However these methods remain useful when aleatoric one doesn't dominate the overall uncertainty.

2.4 Learning Correlation Functions on Mixed Data Sequences for Computer Architecture Applications

Modern computer systems need to rapidly process large amounts of streamed and stored data. To reduce latency, most architectures make predictions about future data demands based on observed sequences in order to make prefetching, caching, and storage decisions [31]. Although there have been algorithms [32, 33] and heuristics developed to handle specific tasks, all of them share the same mathematical characteristic—the need to compute correlation functions from mixed data sequences. A correlation function provides a measure of statistical correlation between random variables, based on spatial or temporal distance [34]. Our aim is to rapidly learn a correlation function from limited data to model the temporal patterns that occur in data streams when related data is accessed.

Sequential data is commonly analyzed using two different types of approaches: pattern mining [35] and sequence learning [36]. Sequence pattern mining extracts frequently repeated subsequences that exceed a minimum support threshold in the dataset. This style of analysis has been employed to find block correlations in storage systems [32]; the C-Miner system extracts association rules based on subsequences found using Closed Sequential Pattern Mining. These rules were then integrated into correlation based prefetching and disk layout systems.

The prefetching problem is particularly well suited for sequence learning since the aim is simply to predict the next element that needs to be retrieved. This can either be structured as a classification or a regression problem and handled with the same type of neural network architectures that are commonly used in natural language processing [37]. [33] demonstrated the application of two LSTM models for prefetching: a single embedded LSTM (the benchmark in our paper) and a multi-task LSTM learned from clustered data. The key advantage of our method vs. the above approaches is that it both requires less data to train and less compute time to execute, making it highly suitable for integration into a computer architecture.

2.5 Graph Representation for Autonomous Driving

Schwarting et al. [38] provide a comprehensive related work overview on autonomous driving that includes research on the myriad systems required to deploy an intelligent driving robot including integrated perception and planning, vehicle dynamics and control, autonomy, and safety verification; however our work is mainly relevant for learning systems that require a bird's-eye roadway representation.

Learning to Drive. Neural network architectures for highway driving have improved substantially since the initial deployment of ALVINN (Autonomous Land Vehicle In a Neural Network) on CMU's Navlab vehicle [39]. Many of the systems have leveraged demonstrations from human drivers using techniques such as behavioral cloning [40] and inverse reinforcement learning [41]. We demonstrate the usage of our representation as a feature extractor for a discrete control system using Proximal Policy Optimization. Saxena et al. [42] learn a continuous controller with PPO for driving in dense traffic; they use an occupancy grid representation parameterized by longitudinal field of view. In flavor their representation is very similar to the grid occupancy representation used by Leurent and Mercat [43] as a benchmark for their social attention model. Leurent and Mercat demonstrate that their social attention representation outperforms both a vanilla PPO implementation and a spatial grid plus convolutional neural network.

Motion Prediction. A key innovation of GRAD is the usage of a space-time graph that incorporates future estimations of neighboring vehicle positions or trajectories into the roadway representation. There are several competitions that explicitly test the problem of driving trajectory prediction, which requires modeling both the intent of the drivers (what they intend to do) and control uncertainty (how they intend to do it). The MultiPath [44] technique does this by factoring intent uncertainty into a distribution over trajectory anchors with normally distributed control uncertainty. MultiPath++ [45] (the leader in the Waymo Open Dataset Motion Prediction Challenge) improves on MultiPath by modifying the state representation in several ways including incorporating context awareness to represent relationships between vehicles.

Graph Neural Networks. Graph neural networks have performed well at the motion prediction problem since they can be used to encode many types of interactions. Key differences between the models include the type of attention utilized [46, 47, 48], whether they represent homogeneous [49, 47] or heterogeneous roadway interactions [46, 50], or use a hierarchical [51] or non-hierarchical representation [49]. For instance, LaneGCN [50] uses a graph to represent vehicle interactions for motion prediction on the Argoverse motion prediction benchmark; however much of their graph is devoted to lane interactions which are not used in our model. SCALE-Net's [47] attention mechanism is induced by edge features on a homogeneous graph. HEAT [46] creates a heterogeneous graph with many different types of links for different roadway elements and also uses a specialized attention mechanism. Unlike these methods, GRAD creates a space-time graph which includes estimated future vehicle positions. Social-WaGDAT [48] uses pairs of graphs (both past and future), but has a very different network structure and attention mechanism. In this dissertation we evaluate our method vs. HEAT [46] which is a top performer at motion prediction for several scenarios in the INTERACTION [52] dataset.

CHAPTER 3
PREDICTING FLIGHT PERFORMANCE WITH MULTIMODAL FEATURES

3.1 Problem Statement

The aim of our research is to monitor and track complex knowledge and skill acquisition by applying machine learning algorithms to a mixture of physiological and performance-based indicators extracted from pilot training sessions. As the complexity of aircraft cockpit operations increases, so does the risk of human error. With training this risk can be reduced immensely. Automating the delivery of instruction through the use of intelligent training simulations and commercially available virtual reality headsets may offer a scalable and cost effective solution for increasing the amount of training pilots receive. If the data collected during training can be leveraged to predict task performance using machine learning, both student and instructor training time can be allocated wisely.

This chapter describes a set of experiments conducted to evaluate the suitability of different machine learning paradigms for predicting student performance on simple flight tasks executed in virtual reality. Supervised machine learning has achieved notable successes across multiple domains including image and speech recognition. However these successes were achieved through the use of complex deep learning models which can only be effectively trained with huge training sets, due to the large number of parameters. For instance, one of most popular computer vision systems, ResNet-50 has over 23 million parameters that need to be fit from training data. Hence our study only considers machine learning models that can be trained on data from a relatively small number of students. To compensate for the limited number of training examples, we assume that it is possible to capture data from multiple modalities, including knowledge tests, user interface controls, and visual attention, to create a small but rich dataset of student interactions.

Experiments were conducted in the Prepar3D flight simulator, which was designed to deliver immersive, experiential learning for both professional and academic pilot skills training. Subjects wore an HTC Vive Pro VR headset, and visual attention data was collected from a built-in Tobii eye tracker. During a two hour period, novice subjects were trained to perform simple flight tasks in Prepar3D. Each flight task tested their ability to achieve a target direction, airspeed, and altitude while monitoring the correct instruments. Students were graded as successful if they were able to achieve and maintain the target direction, airspeed, and/or altitude.

We evaluated the accuracy and F_1 score of two supervised learning classifiers, random forest (RFC) and support vector machines (SVM), at predicting student success using different multimodal feature sets. This chapter examines the following research questions:

- **RQ1**: are features derived from gaze tracking and knowledge tests an acceptable substitute for flight control features?
- **RQ2**: can data from the initial portion of the task be used to predict the final outcome?
- **RQ3**: is it possible to accurately predict student failure?

3.2 Experiments

Experiments were conducted on 23 subjects recruited from the University of Central Florida. The participant pool consisted of 15 male (64%) and 8 female aged 18-29 (M = 19.5, SD = 2.4). The subjects are considered novice pilots as none of them has received any prior flight training. Most (87%) of the subjects report little or no familiarity with flight simulators, and most (78%) of them rate fair or less on their video expertise and report an average weekly video game playing of 7.5 hours (SD = 8.4).

3.2.1 Procedure

The full timeline of the study is as follows:

- Informed consent process (5 minutes)
- Video game experience survey (5 minutes)
- Cognitive tasks (10 minutes)
- Training (15-20 minutes)
- Card sort I (5-10 minutes)
- Knowledge acquisition assessment (10-15 minutes)
- Flight simulator practice tasks (15 minutes)
- Card sort II (15-20 minutes)
- Flight simulator experimental tasks (15 minutes)

- Demographics survey (5 minutes)
- Study wrapup (5 minutes)

The training consisted of both text and images, and is organized into five sections: aircraft model; aircraft controls; flight maneuvers; flight instruments; simulator tasks. Participants are able to review the training materials at their own pace, and are free to move forward and backward through the materials but are instructed that they would not be able to return to any portion of the training once they began the testing session. Within the training, they receive conceptual information about the physics of flight, practical information for successful flight maneuver execution in the VR environment, and specific information about the flight simulator tasks used in the experiment.

During the testing session, subjects are tested on the concepts they learnt using a questionnaire consisting of 10 recall questions, 10 descriptive questions and 10 conceptual questions. An example question is “Attempting to ascend too quickly can result in the aircraft ___”. Data from the subject responses is encoded into knowledge mastery features to be used by the machine learning classifiers. Vectors $Q_i \in \{correct, incorrect\}^M, i = 1,2,3,...$ are reported where i is the subject's ID and M is the number of questions. The time the subjects spend on each question and their confidence are reported as well.

During the simulation session, subjects perform nine practice tasks and then nine experimental tasks in virtual reality using the Prepar3D flight simulator. Each task consists of flying the aircraft to a target direction, altitude, airspeed, or combination of two, within a limited time. Table 3.1 presents a summary of the experimental tasks. To keep the task easier, the aircraft is initialized in

Table 3.1: Scenario descriptions

Task ID	Time (seconds)	Difficulty	Initial Direction (degrees)	Target Direction (degrees)	Initial Airspeed (mph)	Target Airspeed (mph)	Initial Altitude (feet)	Target Altitude (feet)
1	60	Easy	180	270	130	NA	3000	NA
2	60	Easy	180	NA	130	80	3000	NA
3	60	Easy	180	NA	130	NA	3000	3500
4	80	Medium	180	90	130	NA	3000	3000
5	80	Medium	180	NA	130	80	3000	3000
6	80	Medium	180	NA	130	NA	3000	4000
7	100	Hard	180	270	130	NA	3000	1000
8	100	Hard	180	90	130	80	3000	NA
9	100	Hard	180	NA	130	80	3000	4000

the air, and subjects are not asked to perform takeoff or landings. During task execution, customized programs harvest aircraft status and gaze data as a time series.

3.2.2 Equipment

The following software and hardware are used in our experimental setup:

Qualtrics is a multimedia survey software platform that is used to administer experimental materials and collect participant responses in addition to interaction information (e.g., time to submit a response or time spent viewing materials). We use the platform to administer training, our experimental measures of knowledge acquisition and mental models, and demographics and video game experience surveys.

Prepar3D is a 3-D flight simulator developed by Lockheed Martin Co. to deliver immersive, experiential learning for professional and academic pilot training programs. Subjects interact with Prepar3D using a Logitech G X56 H.O.T.A.S. RGB Throttle and Stick Simulation Controller. Using the customized tools developed with the SDK, we are able to extract both the aircraft status data and controller data directly from Prepar3D.

HTC Vive Pro is a virtual reality headset with a built-in Tobii eye tracker. It has a resolution of 1440×1600 per eye, a refresh rate of 90Hz, and a field of view of 110 degrees. The built-in Tobii eye tracker has a gaze data output frequency of 120Hz, an accuracy of 0.5-1.1 degree, and a trackable field of view of 110 degrees.

Qualtrics is used during the training and testing sessions, while the other tools are used during the flight simulation sessions.

3.2.3 Data Collection

While the subjects perform the flight simulator experimental tasks, time series data is collected from three sources:

1. aircraft status data including the aircraft's *geodesic coordinates* (degree), *altitude* (feet), *3-D orientation* (degrees) and *3-D velocity* (feet per second);

2. flight control data including the *aileron* (percentage), *elevator* (percentage) and *throttle* (percentage);

3. subject gaze data (Figure 3.1), including *3-D gaze direction*, *pupil diameters* (mm), *eye openness* (percentage) and *area of interest*.

Additionally, we infer *outcome* $\in$ {success, fail} for each subject and task from aircraft status data by determining if the aircraft achieves the target status (within predefined error bounds) for a continuous period of greater than five seconds.

Data for training the machine learning classifiers is created by synchronizing the three sources and then only retaining the data from the time periods during which the subject is attempting to complete the task. The initial instruction phase during which the subject is still receiving verbal directions from the experimenter and the final success period (if any) are both removed from the time series. Obviously the aircraft status and flight control data captured during task success would be highly informative; however, our aim is to determine whether it is possible to predict task success before it occurs. All subsequent data processing and feature extraction procedures are performed on the truncated data.

Then we create a subset of the data from the first half of the time series to train the machine learning classifiers. Classifiers are either trained with data from the first half (*half*) or the entire time series minus the success period (*full*). By comparing the performance of these classifiers, we can explore the role of early student behaviors on overall flight performance. This is a key element of being able to create an early intervention system for preemptively detecting student failures.

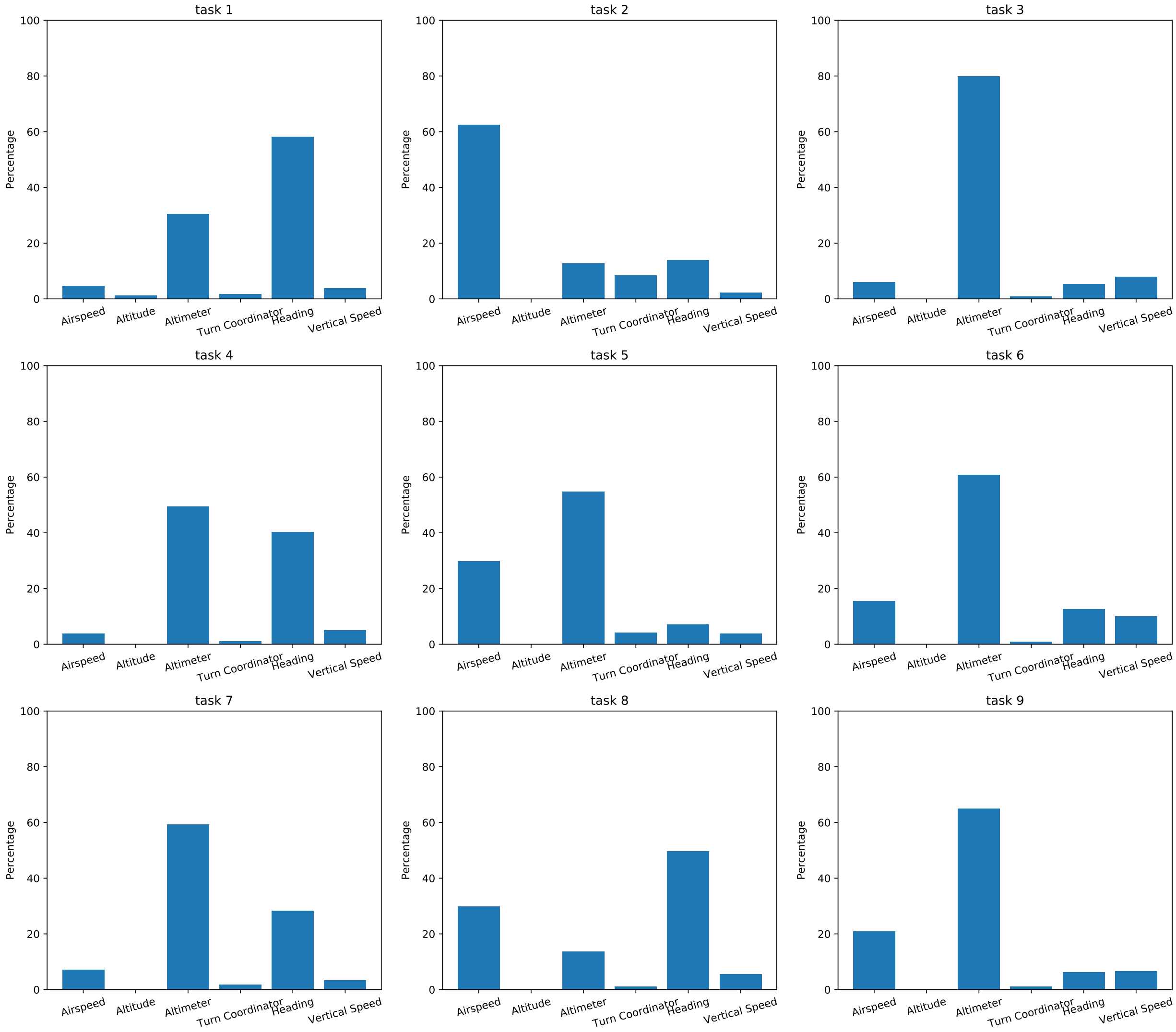


Figure 3.1: Average visual attention distribution on areas of interest across all subjects. We observe that 1) within the same task, the subjects tend to have similar visual focus resulting in a sharp skewness in the average distribution; 2) the focus varies drastically among different types of tasks; 3) visual attention is highly related to tasks in an expected way, e.g. the subjects pay significant attention to the *heading indicator* in turning-related tasks {1,4,7,8}, to the *airspeed indicator* in airspeed-related tasks {2,5,8,9}, to the *altimeter* in all altitude-related tasks excluding {1,2,8}.

3.3 Multimodal Features

We evaluate the performance of different combinations of multimodal features (gaze, flight control, and knowledge mastery) at predicting student flight performance.

3.3.1 Gaze Data

Since the tasks require the subjects to use the instrument panel to verify that the aircraft has achieved target status, the distribution of visual attention across the instruments is likely to be an informative feature. Six areas of interest (AOI) are designated as follows: *airspeed indicator*, *altitude indicator*, *altimeter*, *turn coordinator*, *heading indicator* and *vertical speed indicator*. Figure 3.2 shows the instrument panel marked with the AOIs. The gaze feature vector includes the proportional time distribution and stationary/transition entropy for the AOIs, both of which are explained below. How the subjects allocate their attention between the indicators reveals what information they consider most relevant to the ongoing task. Since our flight tasks require combinations of climbing, descending, slowing down and turning, each task is likely to be associated with a subset of the indicators. Therefore, we expect that a reasonable visual attention distribution is crucial to success. Hence the proportional time distribution of visual attention over the AOIs is used as a feature vector. The averages of these distributions across all the subjects are shown in Figure 3.1.

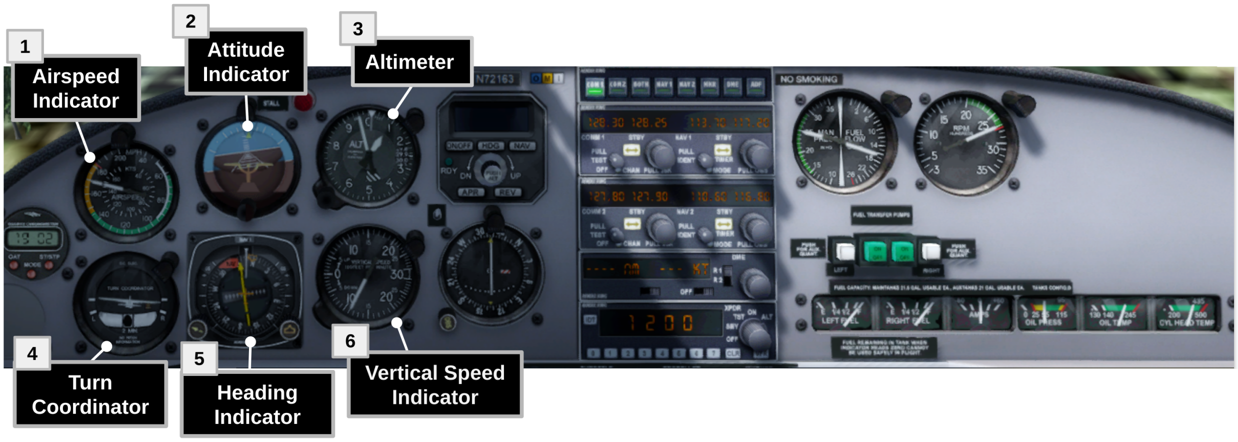


Figure 3.2: The instrument panel of the aircraft (Maule Orion) marked with our six AOIs.

According to [4], there exists an optimal visual scan path for a given visual problem. We propose that the theory is true for a given operational problem as well, since attention and operation are generally consistent. Instead of searching the visual scan path, we apply gaze transition entropy [6] here, which reflects the degree of the path's randomness.

Given a set $\mathscr{S}$ of AOIs and a gaze switching sequence across $\mathscr{S}$, the procedure of computing gaze transition entropy is as follows: firstly, a gaze transition matrix $C \in \mathbf{N}^{\|\mathscr{S}\| \times \|\mathscr{S}\|}$ is obtained by counting gaze transition from $i \in \mathscr{S}$ to $j \in \mathscr{S}$ as entry C_{ij}; secondly, stationary probabilities $\pi_i = \sum_{k \in \mathscr{S}} C_{ik} / \sum_{l,m \in \mathscr{S}} C_{lm}$ and transition probabilities $p_{ij} = C_{ij} / \sum_{k \in \mathscr{S}} C_{ik}$, where $i, j \in \mathscr{S}$, are calculated; finally, we obtain the entropy of transition distribution $Ht = -\sum_{i \in \mathscr{S}} \pi_i \sum_{j \in \mathscr{S}} p_{ij} \log_2 p_{ij}$ and the entropy of stationary distribution $Hs = -\sum_{i \in \mathscr{S}} \pi_i \log_2 \pi_i$.

3.3.2 Flight Control Data

All control data collected from Prepar3D, including *aileron* (percentage), *elevator* (percentage) and *throttle* (percentage), are utilized. Similar to [5], the features are the descriptive statistics of their distributions, including mean, median, standard deviation, skew and kurtosis. Range, maximum and minimum are excluded here because the time series automatically have a fixed range.

3.3.3 Knowledge Mastery

This feature vector is collected from the knowledge acquisition assessment test given in Qualtrics. Subjects are tested on the concepts they learnt using a questionnaire consisting of thirty questions, divided equally between recall, descriptive, and conceptual questions. Knowledge mastery is represented by a matrix $Q \in \{correct, incorrect\}^{N \times M}$, where N is the number of subjects and M is the number of questions. Many of the questions relate to the instruments required to complete the flight task and are thus likely to be a good indicator of performance.

3.4 Machine Learning Models

Since there are equal magnitudes of samples and features in our data collection, we only considered machine learning techniques that are resistant to overfitting. This chapter presents an evaluation of the random forest (RFC) vs. support vector machine (SVM) classifiers.

The random forest classifier is an ensemble method consisting of multiple decision trees, each of which is independently grown with a subset of features. The final classification is performed by weighting the voting based on the trees' performance on the training set. A decision tree partitions the feature space progressively to achieve an information gain in regions based on a measurement, such as Gini index or entropy, and assigns a class to each of them. Our RFC was constructed with 300 trees; a grid search was performed to select the best parameters for the maximum number of tree features and the best information measurement.

A support vector machine (SVM) is a discriminative classifier which employs a hyperplane to segregate the samples belonging to different classes. A kernel is generally applied to map the original feature space to a more separable space in which the hyperplane is placed. In our experiments, a grid search is performed to determine the best parameters for kernel, kernel coefficient, and misclassification penalty.

Similar to [5], a chance model is included as a baseline. It works by stochastically selecting the class for each testing sample with respect to the probability of the corresponding class in training set.

Due to the limited number of samples, models were evaluated with leave-one-out cross validation, which iteratively reserves one sample exclusively for testing purposes and includes all the other samples in the training set. Additionally, the high variance on the task success rate unbalances the dataset. Thus, for each individual experiment in cross validation, the training set is resampled with the oversampling method SMOTE [8] while leaving the testing sample unchanged.

3.5 Results and Discussion

Table 3.2 shows the accuracy and F_1 scores for predicting student flight task success and failure. Both RFC and SVM perform comparably well. The best performing model generally yields an improvement of more than 0.25 in all metrics over the chance model. Results are reported only for the best models based on the parameter grid search. Identifying where and how trainees fail is a key step towards engendering deliberate practice. Therefore, the F_1 score on predicting failure is of more importance than the other metrics listed in the table.

Among the models learned from a single data source, those learned on the control data achieve the best performance. This is unsurprising since the control data relates fairly directly to aircraft status, which in turn is used to judge task performance. Note that these results do not include features that relate to target achievement such as *airspeed*, *altitude* and *heading*. The models trained on gaze data are more predictive on failure cases than success. Improper allocation of visual attention can unilaterally result in task failure; however other factors such as proficiency of manipulation affect success. Combining control and gaze features results in an observable improvement. Like visual attention, the knowledge mastery features are more valuable for predicting failure. Theoretical knowledge appears to be a necessary but not sufficient condition to guarantee successful flight execution. Combining gaze and knowledge mastery features performs equivalently well to the control data alone at predicting task failure.

Finally, classifiers trained on the first half of the data yield similar performance to classifiers trained on the full dataset. It is possible that 1) behavioral observation over a short time window

Table 3.2: Performance metrics for the models using the best parameters.

Feature	**RFC** **Accuracy**	**RFC F$_1$** **Success**	**RFC F$_1$** **Failure**	**SVM** **Accuracy**	**SVM F$_1$** **Success**	**SVM F$_1$** **Failure**
(Chance)	.55	.35	.48	.55	.35	.48
Knowledge Mastery	.78	.47	.70	.73	.52	.67
Control (full)	.79	.59	.72	.77	.59	.73
Control (half)	.80	.55	.71	.81	.62	.77
Gaze (full)	.70	.46	.64	.75	.54	.69
Gaze (half)	.70	.48	.65	.71	.52	.68
Control+Gaze (full)	.78	.55	.71	.78	.60	.74
Control+Gaze (half)	.82	.62	.73	.79	.61	.77
Knowledge+Gaze (full)	.80	.48	.70	.79	.53	.74
Knowledge+Gaze (half)	.81	.47	.74	.79	.53	.74
All sources (full)	.80	.57	.70	.79	.56	.74
All sources (half)	.81	.58	.73	.82	.54	.78

is sufficient to determine student performance or 2) manipulations during early flight stages are crucial to the final outcome.

3.6 Chapter Summary

Our experiments show that it is feasible to accurately predict student failure on simple flight tasks from visual attention features gathered from the initial flight phase, combined with knowledge mastery features; these results affirmatively answer all our research questions. We also demonstrate that it is possible to train the machine learning classifiers on a very small dataset using a combination of techniques. These are important stepping stones towards the long-term vision of scalable, automated delivery of flight instruction using off the shelf virtual reality headsets.

CHAPTER 4
REPRESENTING USER TIME SERIES DATA IN INTELLIGENT TRAINING SYSTEMS

4.1 Problem Statement

The advent of cheap commercial virtual reality headsets has increased the feasibility of training motion tasks such as vehicle control in simulation. Although it is possible to utilize many data sources including gaze tracking, physiological response, and knowledge assessment questions, user control data remains an important predictor of task performance. When applying machine learning to student modeling problems, the question arises of how best to represent the data. In raw form, the data usually comprises a multi-dimensional time series of either position or control inputs. However, it can be preprocessed and vectorized in a variety of ways.

This dissertation presents a new technique for representing time series data that combines dynamic time scaling (DTW) and multidimensional scaling (MDS) to create a distance embedding for modeling the student performance of flight tasks in simulation. Our research questions are:

- **RQ1:** is it valuable to preserve the temporal ordering of the data in a time series or is it better to vectorize the data using the distribution features?
- **RQ2:** does the same representation perform well across multiple user modeling tasks?

We hypothesize that disparate student modeling problems are not likely to benefit from the same distance embedding; hence the data preprocessing pipeline must be configured for the research problem.

Data was gathered from 23 subjects in the Prepar3D flight simulator, which was designed to deliver immersive, experiential learning for both professional and academic pilot skills training. During a two hour period, novice subjects were trained to perform simple flight tasks in Prepar3D. Each flight task tested their ability to achieve a target direction, airspeed, and altitude while monitoring the correct instruments. Students were graded as successful if they were able to achieve and maintain the target direction, airspeed, and/or altitude. A time series was created from the user flight control data including the aileron (percentage), elevator (percentage) and throttle (percentage).

Figure 4.1 shows our proposed embedding method. First dynamic time warping (DTW) is applied to the time series to crate a distance matrix between all the flight controller time series. Multidimensional scaling is then applied to embed the data as points in an abstract Cartesian space while preserving abstract distance relations. This output is then used for our student modeling problems: early failure prediction and classifying student activity.

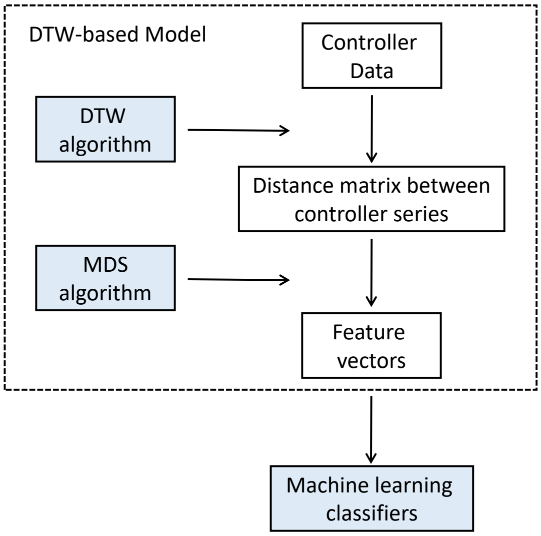


Figure 4.1: DTW-MDS embedding technique

4.2 Method

Our aim is to perform two student modeling tasks: 1) predict whether students will succeed or fail at a flight task and 2) identify the flight maneuver that the pilot is executing. Both of these are useful in the context of an adaptive training simulator.

First we create a distance matrix between the flight controller time series using Dynamic Time Warping (DTW). DTW is an time series analysis algorithm for measuring similarity between temporal sequences. It calculates an optimal match of minimal accumulated distance by warping the timeline while preserving the chronological order. In its simplest form, given two sequences s and t and a distance measurement $d(\cdot,\cdot)$, a table $\mathscr{D}[i,j]$, where i is the index from s sequence and j is that from t, is incrementally updated:

$$\mathscr{D}[i,j] = d(s[i],t[i]) + \min(\mathscr{D}[i,j-1],\mathscr{D}[i-1,j],\mathscr{D}[i-1,j-1]).$$

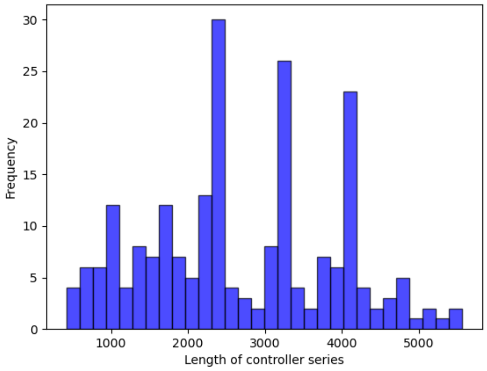


Figure 4.2: Distribution of controller series length

Given a specific flying task, generally the pilots will execute flight control commands in a similar order but not at an identical pace. Aligning the sequences with DTW removes these pace differences. For our experiments, we use fastDTW [10], an improved variant of DTW in terms of space and time.

Our final embedding is created using multidimensional scaling (MDS) [53] which embeds data as points in an abstract Cartesian space while preserving pairwise distances. This has the additional benefit of dimensionality reduction. While DTW can be interpreted as a distance measurement on an infinite dimensional space, MDS is applied here to translate the obtained pairwise distance matrix into the continuous features that are preferred by machine learning algorithms.

4.3 Results

This section presents the experimental results from applying our proposed DTW-MDS embedding technique to two student modeling problems: 1) predicting task failure and 2) identifying the flight maneuver. We also present an ablative analysis where we compare the performance of different vectorization methods, distance functions, dimensionality reduction techniques, and machine learning classifiers.

Both user modeling tasks can be represented as classification problems. For our experiments, we tested all the standard classifiers that have been shown to work well with small datasets: 1) Support Vector Machines (SVM), 2) Random Forest Classifiers (RF), 3) KNN, 4) Logistic Regression, 5) Gaussian Naive Bayes, and 6) Linear Discriminant Analysis. Stratified five-fold cross-validation process was applied, and the average accuracy score over five folds is reported.

The dataset includes time series from students executing nine different flight tasks in which they were asked to bring the aircraft to a target direction, altitude, airspeed, or combination of two, within a limited time. The dataset is balanced across maneuvers and reasonably well balanced between successful and failed student performances. Figure 4.2 shows the distribution of time series lengths across the data. Note that it is not simply possible to predict either student failure or task type by the length of the time series alone.

4.3.1 Early Student Failure Prediction

To predict student failure, we analyzed data from the first half of the time period and and attempted to predict whether they would succeed at maneuvering the aircraft to the desired position by the time limit.

First we experimented with omitting the temporal information and simply using the descriptive statistics to represent the time series. The data was vectorized using the mean, median, standard deviation, skew, and kurtosis from the aileron, elevator and throttle percentages [54].

Table 4.1 compares this method to our proposed DTW-MDS method. Our proposed method leverages the time series information effectively and is just as successful at predicting student failure with half the sequence as if it had the full sequence. Neither using the raw time series nor DTW alone performs well as a vectorization strategy (results not shown). Figure 4.3 shows the same trend across multiple classifiers. SVM is the best performer on the early failure prediction problem which is unsurprising since it often does very well on binary classification tasks.

4.3.2 Identifying Flight Maneuver

Since we only need to identify the flight maneuver after the training simulation is complete, these experiments were conducted with the full control data sequence. Table 4.2 shows the performance of the proposed DTW-MDS embedding across different classification models. We conduct two separate evaluations. In the first we trained and tested models with all the controller data, both

Table 4.1: Comparison of vectorization strategies

Models	Controller length	
	Full sequence	Half sequence
SVM (Distribution Statistics)	0.77	0.62
RF (Distribution Statistics)	0.79	0.80
DTW+MDS SVM (Time Series)	**0.926**	**0.926**
DTW+MDS RF (Time Series)	0.878	0.890

from students who succeeded and failed at the maneuver. However since including the poor examples weakens the quality of the flight maneuver dataset, we conduct another training/testing process with only the successful flight examples. We believe that this condition approximates the performance we would achieve with more expert pilots. The Random Forest model achieves the highest classification accuracy, since it is more resilient at handling data with varying scales.

4.3.3 Distance Metric Comparison

To further validate the effectiveness of our embedding strategy, we compare the usage of DTW vs. Euclidean distance as an input to MDS. Specifically, given two controller series S and T, we compute the Euclidean distance between them as:

$$d_e = \sqrt{(s_1 - t_1)^2 + (s_2 - t_2)^2 + \ldots + (s_n - t_n)^2},$$

Table 4.2: Performance of DTW-MDS applied to flight maneuver identification

Models	Training strategies	
	Training/Testing with all examples	Training/Testing with successful only
SVM	0.649	0.875
KNN	0.582	0.762
Logistic Reg	0.618	0.850
Gaussian NB	0.609	0.863
Random Forest	**0.676**	**0.906**
LDA	0.622	0.875

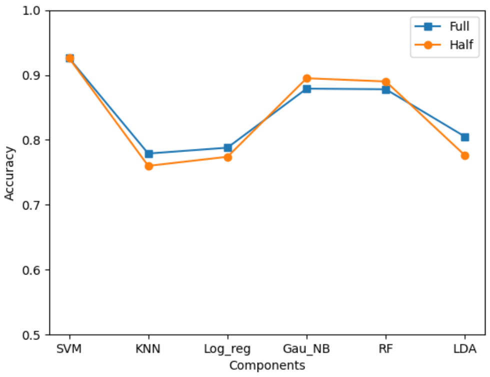
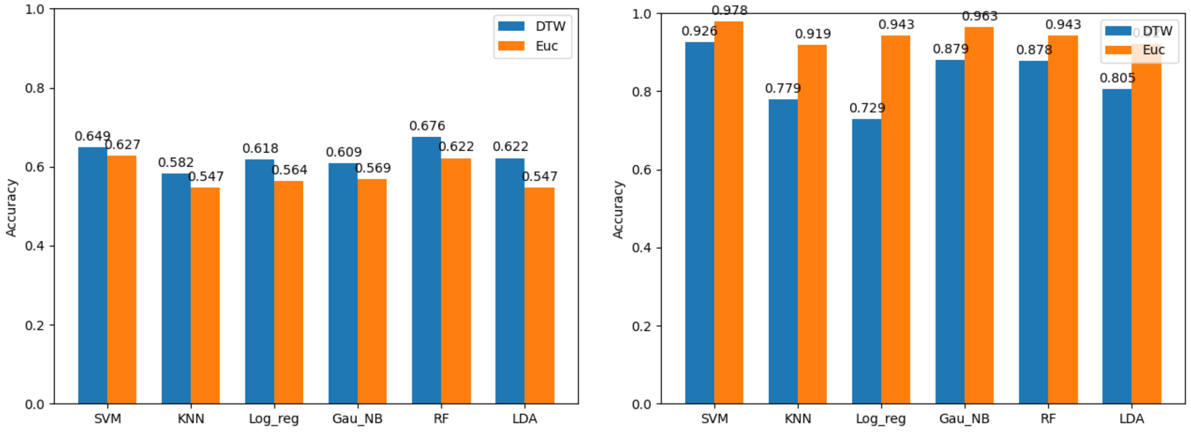


Figure 4.3: DTW-MDS embedding across different classifiers

where n is the length of series S and T. Note that the length of the two series may vary widely. To tackle this problem and align the sequences for comparison, we select the shorter sequence and pad it with zeros until the lengths of the two sequences are equal. After obtaining the distance matrix using this procedure, we pass it through MDS to create the embedding.

Figure 4.4 shows the comparison between the use of DTW vs. Euclidean distance for both user modeling tasks. For both training scenarios and all the machine learning models, DTW outperforms the Euclidean distance method at flight task prediction. We believe that DTW is very good at extracting the shape of the different flight trajectories for each of the maneuvers and is more robust to variations across pilots.

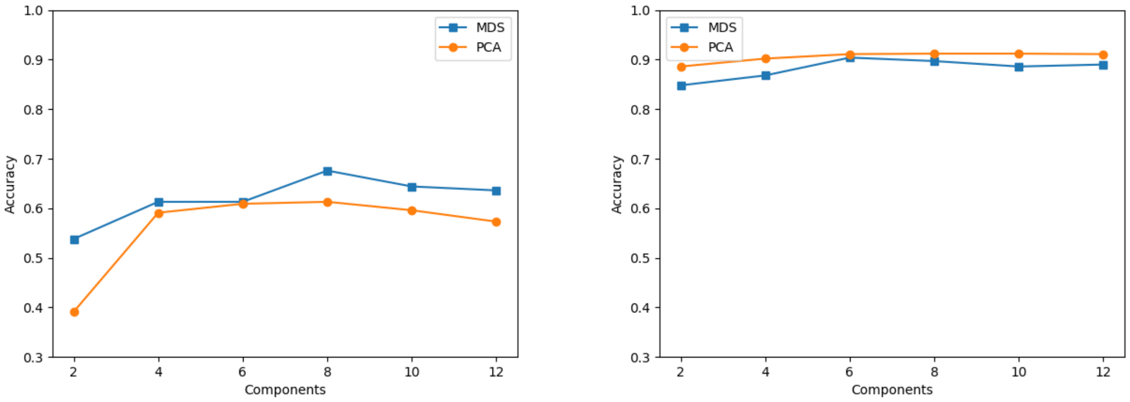
However for failure prediction we observe that the DTW embedding lags behind the use of Euclidean distance. This verifies our hypothesis that separate embeddings should be used for different modeling problems. DTW artificially obscures the pacing differences between the successful



(a) Maneuver identification

(b) Failure prediction

Figure 4.4: DTW-MDS vs. Euclidean-MDS



(a) Maneuver identification

(b) Failure prediction

Figure 4.5: DTW-MDS vs. DTW-PCA

and unsuccessful pilots. Previous work on time series [9] has shown that despite its shortcomings Euclidean distance is remarkably successful at many time series problems.

4.3.4 Comparison of MDS with PCA

Here we compare the second part of our embedding technique (MDS) vs. another commonly used dimensionality reduction technqiue (PCA). Dimensionality reduction not only helps preserve discriminative information, but also filters out noisy features. For these experiments, we used the Random Forest Classifier since it was consistently a top performer.

The comparison of MDS vs. PCA is shown in Figure 4.5. The MDS algorithm slightly outperforms the PCA algorithm at maneuver identification (as shown in Figure 4.5 (a)), while the performance of both two algorithms on failure prediction is comparable (as shown in Figure 4.5 (b)). This validates our choice of MDS as an embedding technique.

We also evaluated the performance of both algorithms with different levels of dimensionality reduction. The best classification performance occurs with about eight components, but failure prediction appears to be less sensitive to this hyperparameter than maneuver identification.

4.4 Chapter Summary

This chapter introduces an embedding technique, DTW-MDS, for representing time series data in intelligent training systems. We demonstrate its utility at two different pilot training problems in a virtual reality flight simulator: 1) early failure prediction and 2) maneuver identification. DTW-MDS enables us to leverage temporal information from the flight controller time series more effectively than vectorizing it using distribution statistics (**RQ1**). DTW-MDS performs fairly well across all conditions (**RQ2**) but it is outperformed by an alternate embedding (Euclidean-MDS) at failure prediction.

This verifies our belief that disparate student modeling problems are not likely to benefit from the same distance embedding; hence the data preprocessing pipeline must be configured for the specific research problem. In future work, we are continuing to expand our training system to include additional user modeling tasks such as problem diagnosis and new embedding techniques for knowledge assessment questions.

CHAPTER 5
LEVERAGING THE VARIANCE OF RETURN SEQUENCES FOR EXPLORATION POLICY

5.1 Problem Statement

Having a good exploration policy is an essential component of achieving sample efficient reinforcement learning. Most RL applications use two heuristics, visitation counts and time, to guide exploration. *Count-based exploration* [55] assumes that it is worth allocating the exploration budget towards previously unexplored actions by awarding exploration bonuses based on action counts. *Time-based exploration* [56] is usually implemented using a Boltzmann distribution that reduces exploration during later stages of the learning process. This dissertation presents an analysis of the benefits and drawbacks of weighted sequence variance for guiding exploration; we contrast the performance of weighted variance with the more familiar weighted temporal difference (TD) error.

Our intuition about the merits of weighted variance as a heuristic to guide exploration is as follows. Imagine that the returns are being summed in a potentially infinite series. Weighted variance can be computed online in order to estimate the convergence speed of the series for a specific state-action pair. We estimate the upper bound using uncertainty, modeled as the weighted standard deviation, as an exploration bonus to guide action selection.

Fluctuation in the return sequence may foretell greater uncertainty in the near future returns that should be rectified through allocation of the exploration budget. Value-based RL algorithms are particularly susceptible to divergence, since improvements in the value function result in rapid policy changes which in turn affect the value estimation. Unlike event counts, weighted variance is more sensitive to the dynamics of the return sequence; if multiple visitations yield consistent reward, weighted variance will quickly prioritize a different state-action sequence even if the total event counts are smaller. We present an empirical analysis showing how weighted variance reacts to the dynamics of raw, smoothed, and residual return sequences. Variation in the return sequence can be modeled as a slower convergence trend layered with transient fluctuations. Our method builds on this intuition, and we present an empirical analysis showing how weighted variance reacts to the dynamics of raw, smoothed, and residual return sequences.

[57] illustrates how policy and value functions change throughout the learning process during off policy temporal difference control. Though the monotonic property is not guaranteed when the model is updated with Monte Carlo methods, generally the return sequences on state-action pairs converge to the value of optimal policy. An incremental updating scheme in which the target Q-value of a state is based on estimation of successive states flattens the sequences. We believe that the variation of the sequence is predictive of the uncertainty of the near future returns.

Computing weighted variance within a deep reinforcement learning framework is a challenging problem, due to the instability of deep neural networks. Simply computing the variance directly from output of DQN risks overestimating the error. The second contribution of the dissertation is introducing two-stream network architecture to estimate either weighted variance or TD errors within DQN agents. Our new architecture (Variance Deep Q Networks) uses a separate σ stream to estimate a weighted standard deviation of the outputs from the original stream.

5.2 Background: Deep Reinforcement Learning

Our aim is to learn an action policy for a stochastic world modeled by a Markov Decision Process by balancing the exploration of new actions and the exploitation of actions known to have a high reward. This is done by learning a value function ($Q(s,a)$) using the discounted return information ($G(s,a)$) and learning rate (α):

$$Q(s,a) = Q(s,a) + \alpha \cdot (G(s,a) - Q(s,a)) \tag{5.1}$$

Actions are selected using the learned value function. This chapter illustrates how our weighted variance exploration approach can be integrated into agents using deep Q-learning.

Deep Q Networks [58] utilize deep neural networks as approximators for action-value functions in Q learning algorithms. The updating procedure of the function is formulated as an optimization problem on a loss function:

$$L_{\mathrm{DQN}}(\zeta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \cdot \max_{b\in\mathscr{A}} Q(s',b;\zeta^{-}) - Q(s,a;\zeta)\right)^2\right] \tag{5.2}$$

where ζ are the parameters of the network, $\mathscr{A}$ is a set of valid actions and D is a distribution over a replay buffer of previously observed transitions. A target network with parameters ζ^{-} is regularly synchronized with ζ and used to estimate the action values of the successor states; the use of a target network promotes estimation stability. Since the original introduction of DQN, several improvements to the updating procedure and network architecture have been proposed.

Double DQN [59] updates the network according to a different rule in which the action for the successor state is selected based on the target network rather than the updating network. This change alleviates the overestimation problem by disentangling the estimation and selection of action during optimization steps. The loss function for Double DQN is:

$$\begin{aligned} L_{\text{DDQN}} = & \mathbb{E}_{(s,a,r,s')\sim D} \\ & \left[\left(r + \gamma \cdot Q(s', \text{argmax}_{b \in \mathscr{A}} Q(s', b; \zeta); \zeta^{-}) - Q(s, a; \zeta) \right)^2 \right]. \end{aligned} \tag{5.3}$$

Our proposed technique guides exploration and can be coupled with other improvements to the reinforcement learning process. Although we construct our models using the above extensions, our technique is solely for guiding exploration and can benefit from other improvements, such as Bootstrapped DQN [23], Distributional DQN [60, 61], and Multi-Step Learning [57]. Moreover, it can be combined with the other exploration methods, such as count-based methods [19] or Noisy DQN [21], to guide exploration during different training stages.

5.3 Measuring Variance for Exploration

During Monte Carlo policy evaluation, the value function $Q(s,a)$ for a particular state-action pair is updated using a sequence of returns $\mathscr{G}_n(s,a) = G_1(s,a), G_2(s,a), ..., G_n(s,a)$. This series can start diverging due to the cyclic nature of value-based approaches; the changing value function results in policy improvements which in turn modify the value function. We believe that the agent should leverage information from these variations to quantify uncertainty in order to explore non-optimal, but still promising, actions. Specifically, agents can follow a greedy exploration policy based on an upper bound:

$$\pi(s) = \text{argmax}_{a \in \mathscr{A}} Q(s,a) + \sigma(s,a) \cdot c, \tag{5.4}$$

where σ is a measurement of uncertainty and c is a fixed hyper-parameter which adjusts the extent of exploration.

To measure the uncertainty of returns for a specific state-action pair, we propose 1) a weighted variance estimation method for general RL, 2) a neural network architecture, and 3) novel optimizing strategy which explicitly estimates either weighted variance or weighted TD error in the DRL configuration.

5.3.1 Reinforcement Learning with Variance Estimation

Although the vanilla form of sequence variance doesn't reflect the higher importance of the recent returns, we define the uncertainty as an exponentially decaying weighted standard deviation

$$\sigma_n(s,a) = \sqrt{\frac{\sum_{i=1}^{n}(1-\alpha)^{n-i}(G_i(s,a) - Q_n(s,a))^2}{\sum_{i=1}^{n}(1-\alpha)^{n-i}}}, \tag{5.5}$$

where $Q_n(s,a)$ is the value function which is updated using $\mathscr{G}_n(s,a)$ and α (the update step size) in Eq. 5.1.

The update formula for σ is as follows

$$\sigma_{n+1}(s,a) = \sqrt{\begin{array}{l} (1-\alpha)\cdot\left[\sigma_n^2(s,a) + (Q_{n+1}(s,a) - Q_n(s,a))^2\right] \\ +\alpha\cdot(G_{n+1}(s,a) - Q_{n+1}(s,a))^2 \end{array}}. \tag{5.6}$$

The first term inside the square root represents the adjusted estimation of variance on $\mathscr{G}_n(s,a)$ with the updated $Q_{n+1}(s,a)$, and the second term is the estimation from the incoming $G_{n+1}(s,a)$. Instead of estimating the variance from TD-errors [62], we deduce it from the mathematical definition of weighted variance. That results in an additional adjustment on the estimation of the previous variance with the updated Q value.

When updates are performed using the above formula, $\sigma(s,a)$ is biased during the early stage, due to the undecidable prior $\sigma_0(s,a)$ as well as the bias incipient to the usage of a small n. We propose two strategies for initializing the σ function: 1) warming up with an ε-greedy method with ε decayed to 0 to ensure a gradual transition to our exploration policy, which effectively starts with a larger n; 2) initializing $\sigma_0(s,a)$ as a large positive value to encourage uniform exploration during early stages, which is theoretically sound since the variance of the value of a state-action pair is infinitely large if it has never been visited.

5.3.1.1 Update Formula

Here we show that the weighted standard deviation in equation 5.5 can be obtained by updating σ with formula 5.6 in Variance Estimation. For simplicity, the inputs to all the following functions are hidden as they are the same state-action pair (s,a). While updating with Eq. 5.1, we have

$$
\begin{aligned}
Q_n &= (1-\alpha)^n Q_0 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} G_i && (5.7)\\
&= \alpha \sum_{i=1}^{n} (1-\alpha)^{n-i} G_i && (5.8)
\end{aligned}
$$

The equal sign in Eq. 5.8 is established by setting $Q_0 = 0$. If we expand $1/\alpha$ as a power series:

$$
\begin{aligned}
\frac{1}{\alpha} &= \frac{1-(1-\alpha)^n}{1-(1-\alpha)} \cdot \frac{1}{1-(1-\alpha)^n} && (5.9)\\
&= \sum_{i=0}^{n-1} (1-\alpha)^i \cdot \frac{1}{1-(1-\alpha)^n} && (5.10)\\
&\approx \sum_{i=0}^{n-1} (1-\alpha)^i && (5.11)\\
&= \sum_{i=1}^{n} (1-\alpha)^{n-i} && (5.12)
\end{aligned}
$$

The approximate sign is used here to reflect that the second term of the multiplication approaches 1 as $n \to \infty$ because $\alpha \in (0,1]$. The value function can be re-written as an exponentially weighted sum of the return sequence by applying the above expansion of power series:

$$
Q_n \approx \frac{\sum_{i=1}^{n} (1-\alpha)^{n-i} G_i}{\sum_{i=1}^{n} (1-\alpha)^{n-i}} \tag{5.13}
$$

We denote $A_n = \sum_{i=1}^{n} (1-\alpha)^{n-i}$, then

$$\sigma_{n+1}^2 = \frac{\sum_{i=1}^{n+1}(1-\alpha)^{n+1-i}(G_i - Q_{n+1})^2}{A_{n+1}} \tag{5.14}$$

$$= \frac{\sum_{i=1}^{n}(1-\alpha)^{n+1-i}(G_i - Q_{n+1})^2}{A_{n+1}} + \frac{(G_{n+1} - Q_{n+1})^2}{A_{n+1}} \tag{5.15}$$

$$= (1-\alpha) \cdot \frac{\sum_{i=1}^{n}(1-\alpha)^{n-i}(G_i - Q_{n+1})^2}{A_{n+1}} + \frac{(G_{n+1} - Q_{n+1})^2}{A_{n+1}} \tag{5.16}$$

$$\begin{aligned} &= (1-\alpha) \cdot \frac{\sum_{i=1}^{n}(1-\alpha)^{n-i}(G_i - Q_n + Q_n - Q_{n+1})^2}{A_{n+1}} \\ &\quad + \frac{(G_{n+1} - Q_{n+1})^2}{A_{n+1}} \end{aligned} \tag{5.17}$$

$$\begin{aligned} &= (1-\alpha) \cdot \Bigg[\frac{\sum_{i=1}^{n}(1-\alpha)^{n-i}(G_i - Q_n)^2}{A_{n+1}} \\ &\quad + \frac{2\sum_{i=1}^{n}(1-\alpha)^{n-i}(G_i - Q_n)(Q_n - Q_{n+1})}{A_{n+1}} \\ &\quad + \frac{\sum_{i=1}^{n}(1-\alpha)^{n-i}(Q_n - Q_{n+1})^2}{A_{n+1}} \Bigg] + \frac{(G_{n+1} - Q_{n+1})^2}{A_{n+1}} \end{aligned} \tag{5.18}$$

$$\begin{aligned} &\approx (1-\alpha) \cdot \frac{A_n}{A_{n+1}} \cdot \left[\sigma_n^2 + 0 + (Q_{n+1} - Q_n)^2\right] \\ &\quad + \frac{(G_{n+1} - Q_{n+1})^2}{A_{n+1}} \end{aligned} \tag{5.19}$$

$$\approx (1-\alpha)\left[\sigma_n^2 + (Q_{n+1} - Q_n)^2\right] + \alpha(G_{n+1} - Q_{n+1})^2 \tag{5.20}$$

where the first approximate sign comes from Eq. 5.13 and the second one comes from Eq. 5.11.

Though the updating formula is biased as n doesn't approach infinity in practice, the bias is negligible. Because all the above approximations are rooted in Eq. 5.11 and $(1-\alpha)^n$ converges to 0 rapidly as n grows.

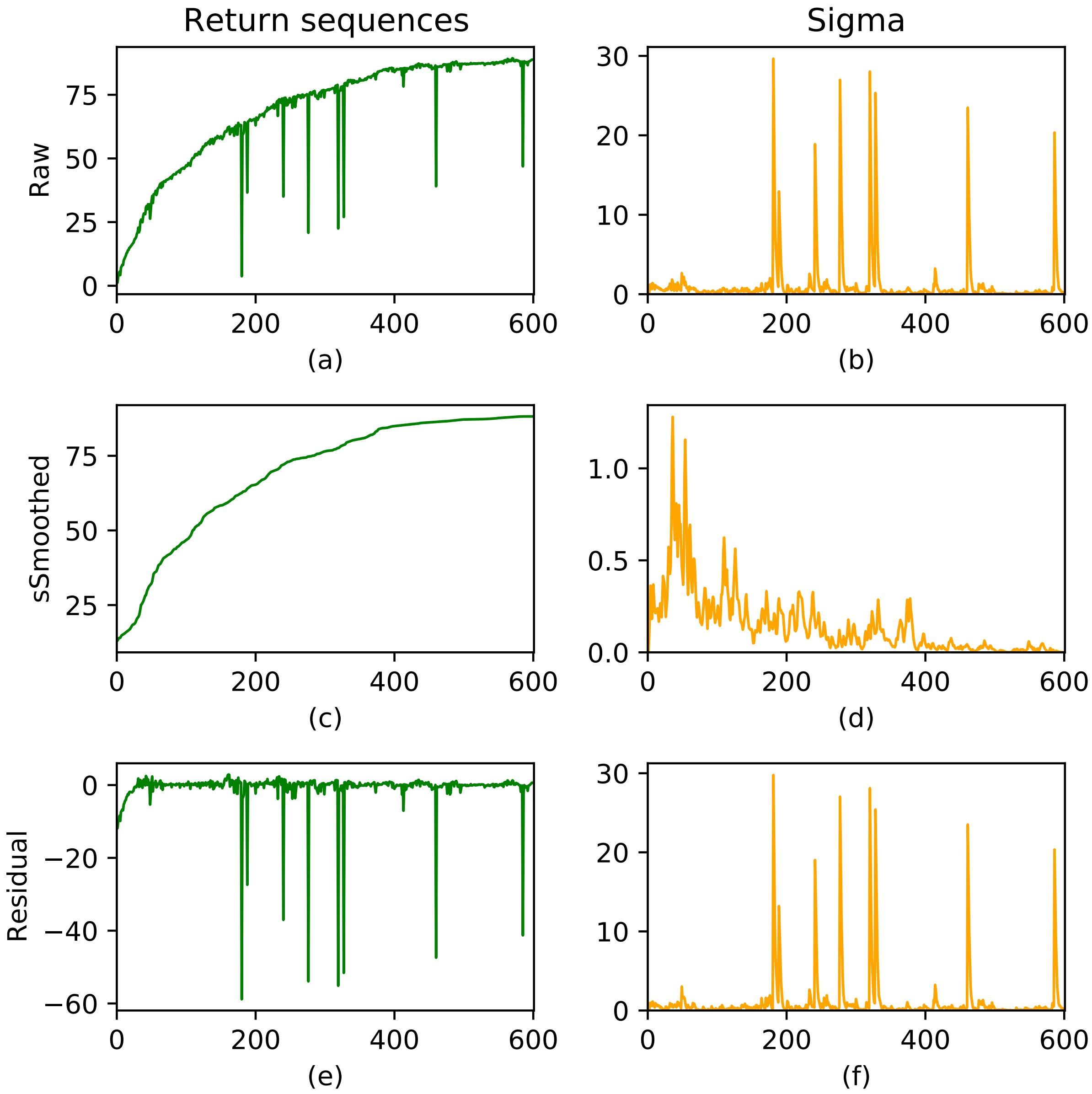


Figure 5.1: Return sequences and corresponding σ values over visitations on a particular state-action pair.

5.3.1.2 Empirical Analysis

[57] prove that the policy and value functions monotonically improve over sweeps in policy iteration. In this simple form, return sequences for individual state-action pairs are monotonically increasing and usually converge fast. However, the same statement is not true when the policy is updated with Monte Carlo settings where the returns are random samples. In this situation, fluctuations in returns are inevitable due to the stochastic property; an incremental updating scheme is adopted to stabilize learning as well as avoid the excessive impact of malicious samples.

Here we illustrate the characteristics of the return sequences and analyze how variance guides exploration. The raw return sequence shown in Figure 5.1(a) is randomly sampled from frequently visited state-action pairs in the Cartpole balancing problem and truncated to 600 visitations. To disentangle the impact of the convergence trend from transient fluctuations, smoothed version is extracted from the raw sequence (shown in Figure 5.1(c)), while the residual is shown in Figure 5.1(e). Then we apply our Variance Estimation method on each of the sequences independently and show the σ values over visitations.

Since the σ values are directly integrated into the Q values used by our exploration policy (shown in Equation 5.4), a greater σ value usually results in an increase in exploration budget. Meanwhile, the nature of weighted sum balances the importance of learned Q value and the history of its change which is captured by σ.

In Figure 5.1(d), we observe that the σ value is greater when the return sequence changes at a faster rate. As the sequence converges, the σ value approaches zero. An interesting but unobvious observation is that the σ value spikes when there is change in the convergence rate of the return sequence. Since variance is essentially a Euclidean distance metric, it is capable of capturing second-order information.

The sequence shown in Figure 5.1(f) isolates the impacts of transient fluctuations from the overall trend. We observe that whenever there is an excessive fluctuation, the σ value spikes to a high magnitude to demand immediate exploration. Once the return goes back to its normal range, the σ value decreases simultaneously. Those quick responses are useful since excessive fluctuations are harmful to the estimation of Q values. A timely investigation of exploration budget eliminates this negative impact before it propagates to more states. Meanwhile, frequent fluctuations beget an increase in σ value and result in more exploration to determine its value.

In conclusion, the exploration policy on our constructed upper bound effectively allocates exploration budget to accelerate convergence in important states as well as alleviate the impact of fluctuations.

5.3.2 Variance Deep Q Networks (V-DQN)

Our new algorithm, V-DQN, incorporates weighted variance into the exploration process of training DQNs. Due to the known instability of deep neural networks during the training process, it is risky to calculate the weighted variance from composing multiple estimations (e.g., the state-action values before and after the optimization step).

Algorithm 1 Variance DQN (V-DQN)

```
Input: exploration parameter $c$; minibatch $k$; target network update step $\tau$;
Input: initial network parameters $\zeta$; initial target network parameter $\zeta^-$;
Input: Boolean DOUBLE
Initialize replay memory $\mathscr{H} = \emptyset$
Observe $s_0$
for $t \in \{1, ..., T\}$ do
    Select an action $a \leftarrow \text{argmax}_{b \in \mathscr{A}} Q(s, b; \zeta) + |\sigma(s, b; \zeta)| \cdot c$
    Sample next state $s \sim P(\cdot|s, a)$ and receive reward $r \leftarrow R(s, a)$
    Store transition $(s_{t-1}, a, r, s_t)$ in $\mathscr{H}$
    for $j \in \{1, ..., k\}$ do
        Sample a transition $(s_j, a_j, r_j, s'_j) \sim D(\mathscr{H})$    ▷ $D$ can be uniform or prioritised replay
        if $s'_j$ is a terminal state then
            $G \leftarrow r_j$
        else if DOUBLE then
            $b^*(s'_j) = \text{argmax}_{b \in \mathscr{A}} Q(s'_j, b; \zeta)$
            $G \leftarrow r_j + \gamma Q(s'_j, b^*(s'_j); \zeta^-)$
        else
            $G \leftarrow r_j + \gamma \max_{b \in \mathscr{A}} Q(s'_j, b; \zeta^-)$
        end if
        $\hat{\sigma} \leftarrow G - Q(s_j, a; \zeta)$
        Do a gradient step with loss $(G - Q(s_j, a; \zeta))^2 + (\hat{\sigma}^2 - \sigma^2(s_j, a; \zeta))^2$
    end for
    if $t \equiv 0 \pmod{\tau}$ then
        Update the target network $\zeta^- \leftarrow \zeta$
    end if
end for
```

Instead of computing the target variance as a byproduct, we propose a two-stream neural network architecture along with an novel loss function to allow end-to-end training while estimating the weighted standard deviation.

It simplifies the optimization for variance by ignoring the adjustment on the previous variance, which is closer to the form in [62]. Since the deep neural networks with gradient descent cannot strictly follow the above updating formula, we believe it's an acceptable compromise. Our empirical results demonstrate the effectiveness of the simplification.

Variance DQN uses neural networks with a separate σ-stream to estimate a weighted standard deviation of the outputs from the original stream on moving targets, which is common in the context of deep reinforcement learning where the value function improves as the policy evolves. In practice, the σ-stream shares lower layers, e.g. convolutional layers, with the original stream to reduce computational demands.

The loss function for Variance DQN is a sum of mean square error losses on the original stream, which is identical to formula 5.2 (for DQN) or formula 5.3 (for Double DQN), and the square of the σ-stream:

$$L_{\text{V-DQN}} = \mathbb{E}_{(s,a,r,s')\sim D} \left[(G - Q(s,a;\zeta))^2 \left((G - Q(s,a;\zeta))^2 - \sigma^2(s,a;\zeta) \right)^2 \right] \tag{5.21}$$

$$s.t.\ G = \begin{cases} r + \gamma \cdot \max_{b \in \mathscr{A}} Q(s', b; \zeta^-) & \text{for DQN} \\ r + \gamma \cdot Q(s', \operatorname{argmax}_{b \in \mathscr{A}} Q(s', b; \zeta); \zeta^-) & \text{for DDQN} \end{cases} \tag{5.22}$$

It is worth noting that the Q function used in the second part of the loss function on the σ-stream doesn't contribute to gradients directly. Therefore, the optimization steps are in effect unchanged for the original stream on the Q value, except for the shared lower layers. While the sign of the σ-stream's output is eliminated in the loss function, we need to do the same during the exploration process. The modified exploration policy is

$$\pi(s) = \text{argmax}_{a \in \mathscr{A}} Q(s,a) + |\sigma(s,a)| \cdot c \tag{5.23}$$

The full procedure is shown in Algorithm 1. We also propose a variant of our method (TD-DQN) which updates σ-stream with absolute temporal difference error. The loss function for TD-DQN is

$$\begin{aligned} L_{\text{TD-DQN}} =& \mathbb{E}_{(s,a,r,s')\sim D} \\ & \left[(G - Q(s,a;\zeta))^2 + (|G - Q(s,a;\zeta)| - \sigma(s,a;\zeta))^2 \right] \end{aligned} \tag{5.24}$$

where G is the same as Equation 5.22.

Both networks measure the uncertainty of the Q value based on the return history in order to construct an upper bound for exploration policy. The difference between the approaches can be interpreted based on their implicit usage of different distance metrics: Euclidean (Variance DQN) vs. Manhattan (TD-DQN). Generally, V-DQN is more sensitive to fluctuations in the return sequence, investing a greater amount of the exploration budget to damp variations.

There may be applications in which it is is valuable to estimate still higher order statistics, such as the skew or kurtosis of the return sequence. However, this sensitivity can also sabotage exploration by directing resources away from promising areas of the state space that are slowly trending towards convergence; overemphasizing the elimination of small variations could ultimately result in longer training times. While the c hyper-parameter in our exploration policy adjusts the trade-off between exploration and exploitation, the choice of the distance metrics used to measure sequence variation determines the distribution of exploration time.

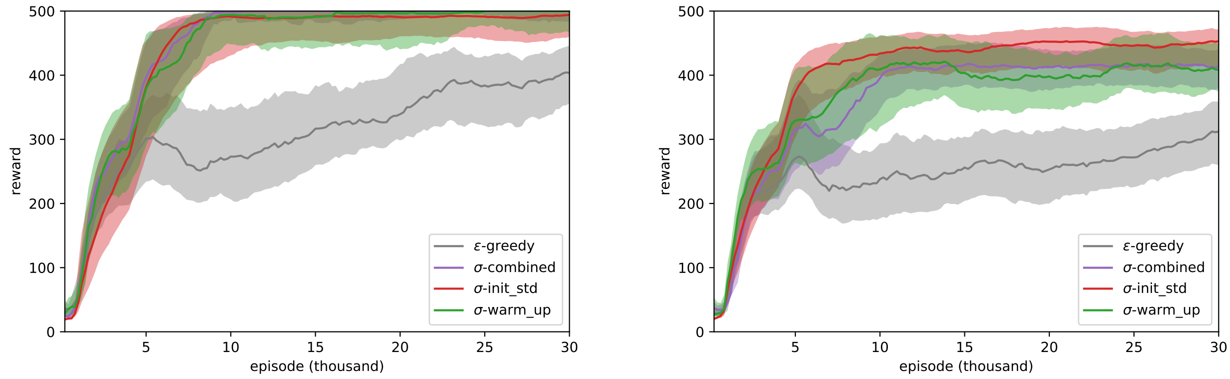
5.4 Results

To illustrate how weighted variance improves exploration, this chapter first presents results on its usage in tabular Q-learning for the Cartpole inverted pendulum problem. Then we report the performance of our proposed techniques (V-DQN and TD-DQN) on the Atari game benchmark. The results show that our two stream architecture for guiding exploration with weighted variance or weighted temporal difference outperforms the standard DDQN benchmark.

5.4.1 Cartpole

To demonstrate the effectiveness of our Variance Estimation (VE) method, we compare it with ε-greedy on Cartpole balancing problem. For this experiment, we use the classic Q-learning algorithm [63] with a tabular look-up Q-table.

The Cartpole environment has a 4-dimensional continuous state space $\mathscr{S} = \mathbb{R}^4$ and a discrete action space $\mathscr{A} =$\{*Push Left, Push Right, Do Nothing*\}. It provides a reward of 1 in every time step. The episode terminates if any of the following conditions is met: 1) the episode length is greater than 500, 2) the pole falls below a threshold angle, 3) the cart drifts from the center beyond



(a) Episode reward ($\varepsilon_{\text{eval}} = 0$)

(b) Episode reward ($\varepsilon_{\text{eval}} = 0.05$)

Figure 5.2: Average episode rewards in the Cartpole balancing problem. The curves and the shadowed areas represent the means and the quartiles over 9 independent runs. The models are evaluated for 10 evaluation episodes every 200 training episodes.

a threshold distance. With this setting, the maximum accumulated reward any policy can achieve is 500. To apply the tabular Q-learning configuration, the continuous state space is discretized into 18,432 discrete states by dividing $\{$*Cart Position, Cart Velocity, Pole Angle, Pole Velocity*$\}$ into $\{12, 8, 16, 12\}$ intervals.

With grid search, ε-greedy achieves the best performance when the discounting factor $\gamma = 1.0$ and the exploration rate ε decays from 1.0 to 0.01 in 5000 episodes.

For Variance Estimation, we experimented on both initialization methods as well as a combination of them. A similar configuration is applied on warming up with the ε-greedy method in which ε decays from 1.0 to 0.0 during 5000 episodes. The initial standard deviation is set to 5000 for initializing the σ_0 method to ensure sufficient early visits on states. The combination method

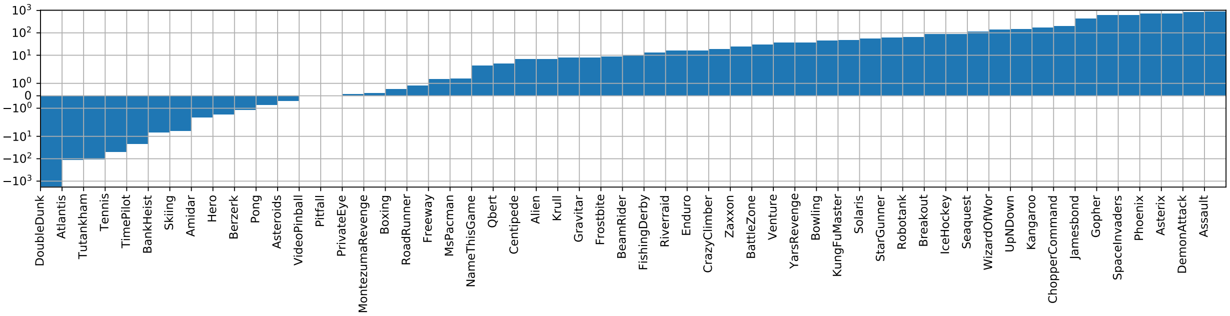


Figure 5.3: Improvement in normalized scores of V-DQN over DDQN in 200M frames

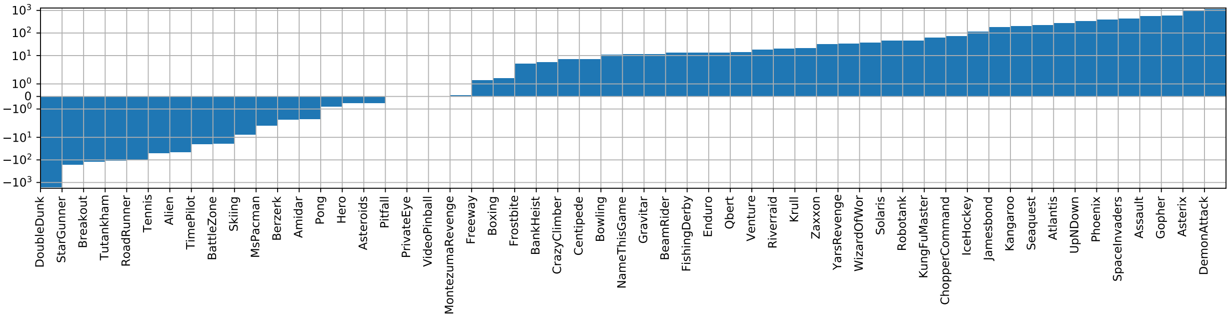
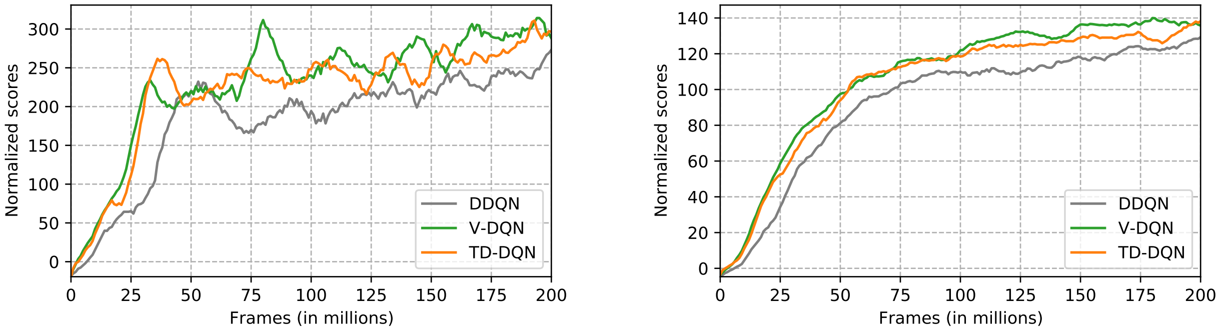


Figure 5.4: Improvement in normalized scores of TD-DQN over DDQN in 200M frames

warms up with ε-greedy while retaining the large initial standard deviation; it uses the same hyperparameters. The value of c is set to 1.5 for initializing the σ_0 method and 0.5 for the other two.

To reduce the possibility of over-fitting, we evaluate the models with an additional environment in which the agent has a probability $\varepsilon_{\text{eval}} = 0.05$ of acting randomly. All of our methods outperform ε-greedy consistently for both evaluation settings. When the training time is prolonged, the baseline method generally achieves similar scores to our methods, but requires approximately 10 times the training episodes.



(a) Mean normalized scores of all Atari games over 200M frames.

(b) Median normalized scores of all Atari games over 200M frames.

Figure 5.5: The mean and median of the normalized training curve over all 55 Atari games

5.4.2 Atari Games

We evaluate our DQN-based algorithms on 55 Atari games from the Arcade Learning Environment (ALE) [64], simulated via the OpenAI Gym platform [65]. Defender and Surround are excluded because they are unavailable in this platform. The baseline method (denoted as DDQN) is DQN [58] with all the standard improvements including Double DQN [59], Dueling DQN [66] and Prioritized Replay [67].

Our network architecture has a similar structure to Dueling DQN [66], but with an additional σ-stream among fully connected layers. The 3 convolutional layers have 32 8×8 filters with stride 4, 64 4×4 filters with stride 2, 64 3×3 filters with stride 1 respectively. Then the network splits into three streams of fully connected layers, which are value, advantage and σ streams. Each of the streams has a hidden fully connected layer with 512 units. For the output layers, the value stream has a single output while both advantage and σ streams have the same number of outputs as the valid actions.

The random start no-op scheme in [58] is used here in both training and evaluation episodes. The agent repeats no-op actions for a randomly selected number of times between 1 to 30 in the beginning to provide diverse starting conditions to alleviate over-fitting. Evaluation takes place after freezing the network every 250K training steps (1M frames). The scores are the averages of episode rewards over 125K steps (500K frames) where episodes are truncated at 27K steps (108K frames or 30 minutes of simulated play).

We use the Adam optimizer [68] with a learning rate of 6.25×10^{-5} and a value of 1.5×10^{-4} for Adam's ε hyper-parameter over all experiments. The network is optimized on a mini-batch of 32 samples over prioritized replay buffer every 4 training steps. The target network is updated every 30K steps.

The exploration rate of DDQN decays from 1.0 to 0.01 in 250K steps (1M frames) and retains that value until the training ends. Our methods do not rely on ε-greedy so that is simply set to 0 for all the steps. Instead, the value of c impacts the actual exploration rates of our methods, which are defined here to be the proportion of actions different from the optimal ones based on the current Q value function. Empirically, the performance on Atari games does not vary significantly over a wide range of c values, which is an unusual finding. A possible explanation is that most of the actions in those games are not critical. To keep the exploration policy from drifting too far from the exploitation policy, we set c to be 0.1 for both V-DQN and TD-DQN over all experiments to keep the average exploration rates of the majority of the Atari games to reside roughly between 0.01 and 0.1.

A summary of the results over all 55 Atari games is reported in Table 5.1. The raw and the normalized scores for individual games are compiled in Table 5.4 and Table 5.3. To compare the performance of agents over different games, the scores are normalized with human scores

$$\text{Score}_{\text{Normalized}} = 100 \times \frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Random}}}{\text{Score}_{\text{Human}} - \text{Score}_{\text{Random}}} \tag{5.25}$$

where both the random and the human scores were taken from [67].

The results clearly show that our proposed methods for guiding exploration, V-DQN and TD-DQN, both improve on the standard DDQN benchmark. Although there are small differences in the ranking, both versions perform well in the same games, and underperform the benchmark in a small set of games. The mean and median statistics do not reveal significant differences between V-DQN and TD-DQN. Our intuition remains that V-DQN is likely to more sensitive to fluctuations and will allocate more exploration budget to damp them out.

Table 5.1: Summary of normalized scores. See Table 5.3 for full results.

	DDQN	V-DQN	TD-DQN
Median	151%	164%	164%
Mean	468%	547%	533%

5.5 Chapter Summary

This chapter presents an analysis of the benefits and limitations of weighted variance for guiding exploration. Both weighted convergence and its close cousin, weighted temporal difference, can be used to quantify the rate of convergence of the return series for specific state-action pairs. The return dynamics of value-based reinforcement learning is particularly susceptible to diverging as value improvements beget policy ones. This dissertation introduces a new two-stream network architecture to estimate both weighted variance/TD errors; both our techniques (V-DQN and TD-DQN) outperform DDQN on the Atari game benchmark. We have identified two promising directions for future work in this area.

Addressing Cold Start with Unified Exploration While our methods capture the divergence of return sequences, they suffer from the "cold start" problem. It is unlikely that they will perform well for either empty or short sequences. To address this, we propose two simple initialization methods for tabular configurations in this chapter. This issue is somewhat alleviated by the generalization capacity inherent to function approximators like deep neural networks. However, larger state space where most of the states will never be visited still pose a problem. Our method can further benefit from unification with the other exploration methods. Count-based upper confidence bound (UCB) methods [19] have a stronger effect on balancing the visits among states in the early stage. This effect decays gradually as visits increase. This characteristic makes it a natural complement for our sequence-based methods. Noisy DQN [21] is another option that assigns greater randomness to less visited states. Our method focuses on promising actions whereas Noisy DQN chooses actions more randomly in those areas of the state space. We ran some experiments on a rudimentary design in which the linear layers of Q-value stream were replaced with noisy ones; our preliminary results (not reported) show an improvement by hybridizing the two architectures.

Beyond Q Learning The key intuition behind our methods is that exploration should be guided with with a measure of historical variation. Generally, the returns based on the estimated Q values of successive states are more consistent than those built on purely episodic experience. Therefore, it is natural to extend the idea of using return sequences to algorithms where state or action values are available, such as actor-critic methods.

Table 5.2: Atari DQN Hyperparameters

Hyperparameter	Value	Description
$c_{\text{V-DQN}}$	0.1	Weighting factor of σ-stream in exploration policy in V-DQN
$c_{\text{TD-DQN}}$	0.1	Weighting factor of σ-stream in exploration policy in TD-DQN
mini-batch size	32	Size of mini-batch sample for gradient step
replay buffer size	1M	Maximum number of transitions stored in the replay buffer
initial replay buffer size	50K	Number of transitions stored in the replay buffer before optimization starts
optimization frequency	4	Number of actions the agent takes between successive network optimization steps
update frequency	30000	Number of steps between consecutive target updates
$\varepsilon_{\text{init}}$	1.00	Initial exploration rate of ε-greedy method
$\varepsilon_{\text{final}}$	0.01	Final exploration rate of ε-greedy method
N_ε	1M	Number of actions that the exploration rate of ε-greedy method decays from initial value to final value
α	0.0000625	Adam optimizer learning rate
$\varepsilon_{\text{ADAM}}$	0.00015	Adam optimizer parameter
evaluation frequency	250K	Number of actions between successive evaluation runs
evaluation length	125K	Number of actions per evaluation run
evaluation episode length	27K	Maximum number of action in an episode in evaluation runs
max no-op	30	Maximum number of no-op actions before the episode starts

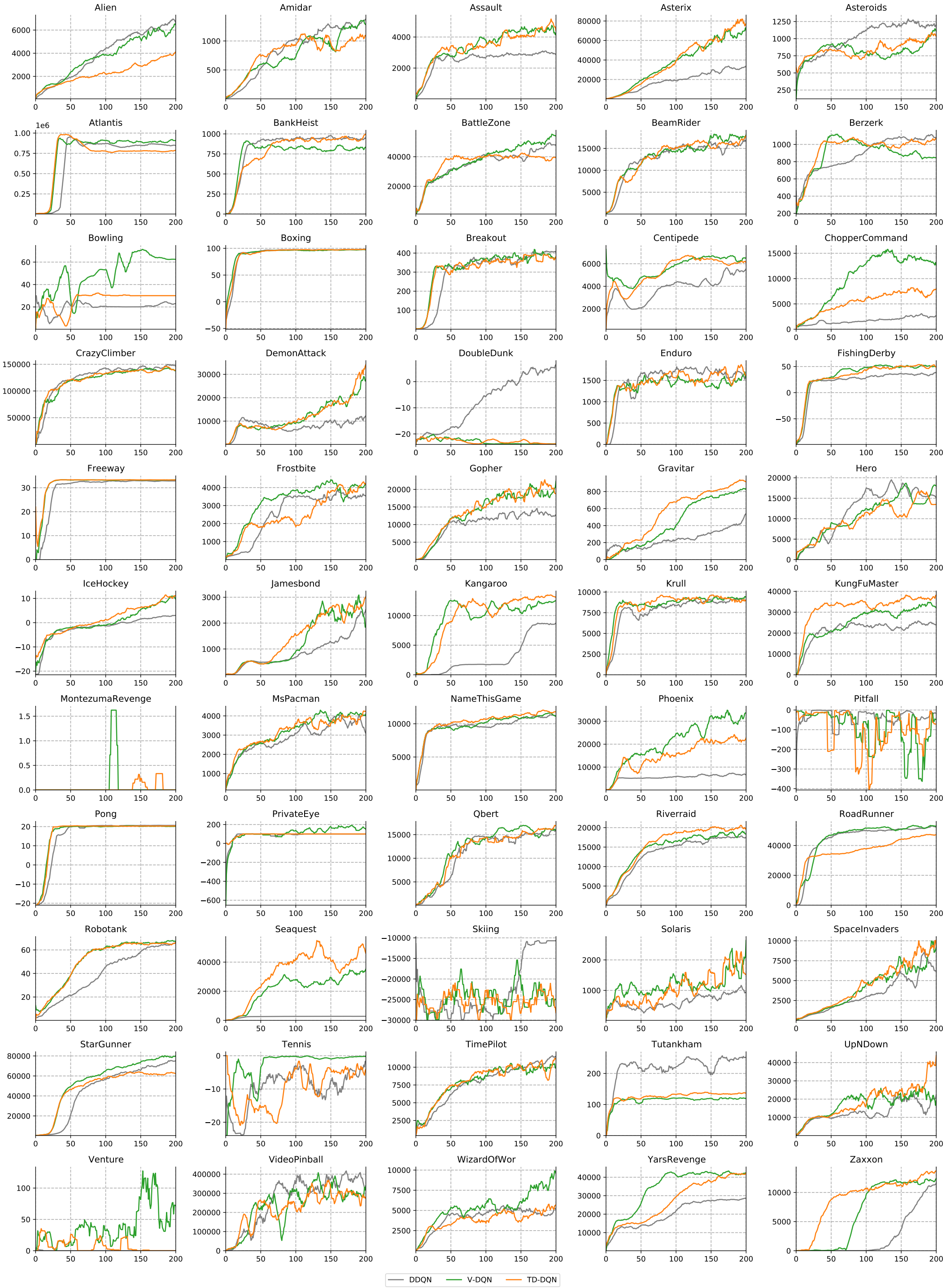


Figure 5.6: Training curve on Atari games from a single training run each. Episodes start with up to 30 no-op actions. Each data point is an average of episode rewards from 500K frames of evaluation runs, and smoothed over 10 data points.

Table 5.3: Normalized scores.

Game	DDQN	V-DQN	TD-DQN
Alien	123%	**130%**	76%
Amidar	**99%**	97%	97%
Assault	764%	**1659%**	1324%
Asterix	537%	1280%	**1490%**
Asteroids	**2%**	1%	1%
Atlantis	7445%	7334%	**7714%**
BankHeist	165%	158%	**170%**
BattleZone	168%	**199%**	149%
BeamRider	143%	153%	**156%**
Berzerk	**51%**	49%	49%
Bowling	7%	**52%**	18%
Boxing	902%	903%	**904%**
Breakout	1764%	**1851%**	1644%
Centipede	57%	64%	**64%**
ChopperCommand	39%	**237%**	111%
CrazyClimber	639%	**658%**	646%
DemonAttack	563%	1415%	**1700%**
DoubleDunk	**1568%**	-145%	-90%
Enduro	267%	**284%**	281%
FishingDerby	151%	**164%**	164%
Freeway	130%	**131%**	131%
Frostbite	108%	**116%**	112%
Gopher	1071%	**1689%**	1665%
Gravitar	14%	22%	**25%**
Hero	**83%**	81%	82%
IceHockey	139%	227%	**253%**
Jamesbond	890%	**1315%**	1071%
Kangaroo	354%	529%	**552%**
Krull	909%	917%	**931%**
KungFuMaster	150%	199%	**214%**
MontezumaRevenge	-1%	**-0%**	-1%
MsPacman	32%	**33%**	29%
NameThisGame	207%	210%	**218%**
Phoenix	134%	**852%**	531%
Pitfall	5%	**5%**	5%
Pong	**116%**	115%	115%
PrivateEye	-1%	**-1%**	-1%
Qbert	146%	150%	**160%**
Riverraid	136%	152%	**156%**
RoadRunner	826%	**827%**	724%
Robotank	1006%	**1071%**	1051%
Seaquest	6%	122%	**226%**
Skiing	**43%**	37%	35%
Solaris	-5%	**50%**	40%
SpaceInvaders	889%	**1527%**	1318%
StarGunner	922%	**984%**	760%
Tennis	**197%**	147%	146%
TimePilot	**387%**	365%	366%
Tutankham	**215%**	108%	109%
UpNDown	366%	516%	**707%**
Venture	-1%	**36%**	18%
VideoPinball	425%	**425%**	425%
WizardOfWor	168%	**309%**	206%
YarsRevenge	61%	**98%**	95%
Zaxxon	151%	175%	**183%**

Table 5.4: Raw Scores.

Game	Random	Human	DDQN	V-DQN	TD-DQN
Alien	128.3	6371.3	7807.3	**8236.9**	4895.4
Amidar	11.8	1540.4	**1521.6**	1495.0	1494.0
Assault	166.9	628.9	3697.5	**7829.8**	6284.9
Asterix	164.5	7536.0	39782.0	94509.1	**110013.6**
Asteroids	871.3	36517.3	**1464.3**	1317.6	1278.0
Atlantis	13463.0	26575.0	989675.0	975100.0	**1024975.0**
BankHeist	21.7	644.5	1050.1	1007.7	**1082.3**
BattleZone	3560.0	33030.0	53153.8	**62133.3**	47460.0
BeamRider	254.6	14961.0	21296.0	22765.7	**23234.0**
Berzerk	196.1	2237.5	**1228.1**	1205.0	1190.7
Bowling	35.2	146.5	42.7	**93.6**	54.7
Boxing	-1.5	9.6	98.6	98.7	**98.8**
Breakout	1.6	27.9	465.5	**488.3**	434.1
Centipede	1925.5	10321.9	6695.1	7271.9	**7282.8**
ChopperCommand	644.0	8930.0	3900.0	**20289.7**	9835.0
CrazyClimber	9337.0	32667.0	158346.2	**162828.0**	159952.0
DemonAttack	208.3	3442.8	18418.2	45977.5	**55200.6**
DoubleDunk	-16.0	-14.4	**9.1**	-18.3	-17.4
Enduro	-81.8	740.2	2113.5	**2250.2**	2225.0
FishingDerby	-77.1	5.1	46.6	**57.7**	57.7
Freeway	0.1	25.6	33.2	**33.6**	33.6
Frostbite	66.4	4202.8	4516.2	**4883.8**	4702.4
Gopher	250.0	2311.0	22331.4	**35061.4**	34555.7
Gravitar	245.5	3116.0	637.5	869.5	**976.1**
Hero	1580.3	25839.4	**21606.6**	21244.0	21476.5
IceHockey	-9.7	0.5	4.5	13.5	**16.1**
Jamesbond	33.5	368.5	3014.2	**4438.8**	3622.5
Kangaroo	100.0	2739.0	9450.0	14057.6	**14679.4**
Krull	1151.9	2109.1	9855.5	9930.0	**10065.0**
KungFuMaster	304.0	20786.8	31070.7	40984.4	**44177.4**
MontezumaRevenge	25.0	4182.0	0.0	**9.1**	3.3
MsPacman	197.8	15375.0	5038.5	**5246.7**	4585.7
NameThisGame	1747.8	6796.0	12181.1	12359.4	**12774.7**
Phoenix	1134.4	6686.2	8568.6	**48424.7**	30623.9
Pitfall	-348.8	5998.9	0.0	**0.0**	0.0
Pong	-18.0	15.5	**20.8**	20.6	20.6
PrivateEye	662.8	64169.1	100.0	**200.0**	100.0
Qbert	183.0	12085.0	17551.4	18051.7	**19250.0**
Riverraid	588.3	14382.2	19322.9	21545.0	**22073.8**
RoadRunner	200.0	6878.0	55381.7	**55437.1**	48526.7
Robotank	2.4	8.9	67.8	**72.0**	70.7
Seaquest	215.5	40425.8	2789.8	49230.8	**91277.1**
Skiing	-15287.4	-3686.6	**-10314.1**	-10990.9	-11215.2
Solaris	2047.2	11032.6	1572.0	**6497.6**	5615.0
SpaceInvaders	182.6	1464.9	11580.8	**19757.7**	17086.2
StarGunner	697.0	9528.0	82076.7	**87577.8**	67768.6
Tennis	-21.4	-6.7	**7.5**	0.2	0.0
TimePilot	3273.0	5650.0	**12460.7**	11941.4	11975.0
Tutankham	12.7	138.3	**283.0**	147.8	150.1
UpNDown	707.2	9896.1	34346.4	48118.3	**65673.9**
Venture	18.0	1039.0	9.6	**382.9**	197.1
VideoPinball	20452.0	15641.1	584388.2	**632013.8**	631348.0
WizardOfWor	804.0	4556.0	7115.1	**12388.1**	8551.7
YarsRevenge	1476.9	47135.2	29332.9	**46319.6**	44894.4
Zaxxon	475.0	8443.0	12488.4	14409.8	**15028.3**

CHAPTER 6
LEARNING CORRELATION FUNCTIONS ON MIXED DATA SEQUENCES FOR COMPUTER ARCHITECTURE APPLICATIONS

6.1 Introduction

This dissertation introduces a method, Temporally Aware Embedding (TAE), for learning an embedding that leverages the temporal information extracted from data retrieval sequences. Since our algorithm solely relies on sampled sequences, it can be used even in complex architectures where the data semantics are not known to the observer. Our assumptions are: 1) several retrieval processes occur concurrently resulting in a single mixed sequence and 2) the correlation is roughly proportional to the first order Markov statistics of the random process.

TAE's correlation embedding is calculated by applying a kernel function to a sliding window over the temporal sequence. Data co-occurrences within the sliding window result in attractive forces that increase the magnitude of correlation while repulsive forces operate to distribute the data more uniformly.

Our method yields an informed embedding which can be utilized in combination with other heuristics and machine learning techniques in order to intelligently prefetch, cache, or store data. Our experiments demonstrate that it outperforms both simple heuristics and state of the art deep learning strategies at predicting the next element of real database sequences as well as synthetically generated datasets.

6.2 Method

This section provides a description of our algorithm, Temporally Aware Embedding (TAE), for learning embeddings from mixed sequential data.

6.2.1 Problem Statement

Assume that there is a set $\mathbb{S}$ such that every $a,b \in \mathbb{S}$ has an underlying correlation $cor(a,b) \in \mathbb{R}$ and that $\mathscr{D} = (x_1,...,x_T) \in \mathbb{S}^{\mathbb{N}}$ is a sequence sampled on $\mathbb{S}$ subject to $P(x_{t+1}|x_t) \propto \frac{cor(x_t,x_{t+1})}{\sum_{x\in\mathbb{S}} cor(x_t,x)}$. TAE embeds every $a \in \mathbb{S}$ based solely on whether a appears in $\mathscr{D}$ into a vector space $\mathbb{V}$ in which $d(v_a, v_b)$ is negatively correlated to $cor(a,b)$. Our desiderata are to create an algorithm that is 1) highly robust to noise and 2) able to deal with a random mixture of multiple sequences $\mathscr{D}_{mix} \in \mathbb{S}^{\mathbb{N}}$.

6.2.2 Learning the Embedding

First we sample one point $v_x \in \mathbb{V}$ for each $x \in \mathbb{S}'$ from an initial distribution (Gaussian or uniform) where $\mathbb{S}' = \{a \in \mathbb{S}|a \in \mathscr{D}\}$. Then we apply a physics based model of attraction and repulsion forces

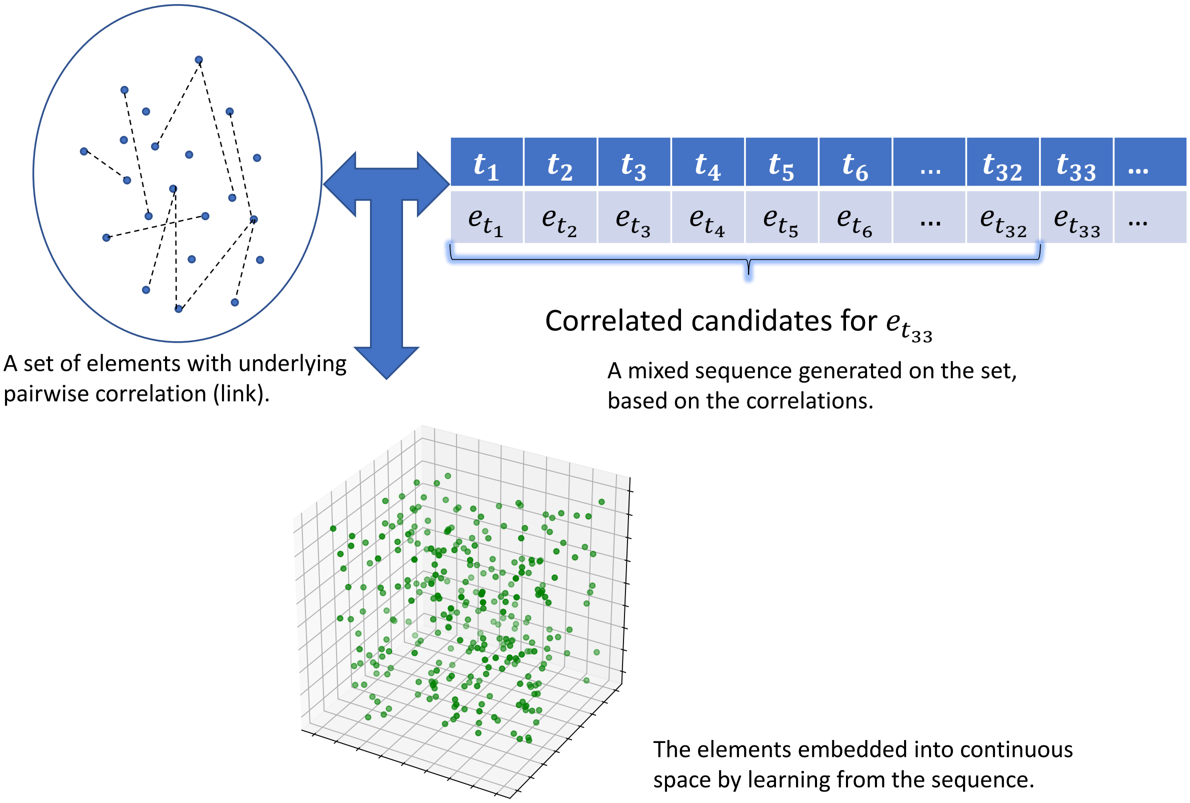

t_1 t_2 t_3 t_4 t_5 t_6 … t_{32} t_{33} …
e_{t_1} e_{t_2} e_{t_3} e_{t_4} e_{t_5} e_{t_6} … $e_{t_{32}}$ $e_{t_{33}}$ …
Correlated candidates for $e_{t_{33}}$
A set of elements with underlying pairwise correlation (link).
A mixed sequence generated on the set, based on the correlations.
The elements embedded into continuous space by learning from the sequence.

Figure 6.1: The mixed data sequence (upper right) is generated by multiple processes simultaneously making data accesses on a set (upper left) where an underlying pairwise correlation exists between related data. Our method embeds the elements into a N-dimensional continuous space (lower) based on the dependency shown in the sequence.

to modify the position of the points based on their co-occurrence in the observed data sequence. The final embedding is obtained after the model converges to a stable state.

The only information source that we use to aggregate correlated points is the data sequence, $\mathscr{D}$. Intuitively, every pair of consecutive points v_{x_t}, $v_{x_{t+1}}$ should be considered for aggregation. However, due to the existence of noise and interleaved sequences, consecutive points may be separated by irrelevant ones. Instead, we assume that two points v_{x_t}, $v_{x_{t+i}}$ are correlated if $i \leq l$ for some preselected $l \in \mathbb{N}$. Even with our mixed data sequences, if two points appear together within a window of length l in $\mathscr{D}$ multiple times they are likely to be the result of the same data access process.

Specifically, at each $t = 1, ..., T$, we take a weighted sum of the points $v_{x_{t-l}}, ..., v_{x_{t-1}}$, based on their temporal distances to v_{x_t} in $\mathscr{D}$, as the source of an attractive force. Then v_{x_t} moves a distance in $\mathbb{V}$ scaled by the factor $\alpha \in \mathbb{R}$, which may decay over time, as the result of applying the force. A kernel $\mathscr{K} = (k_1, ..., k_l)$, which is either linear or exponential in our experimental settings, serves as the weighting function. Separate kernels can be applied to different dimensions of $\mathbb{V}$ to explore multiple types of correlations. The attraction formula is shown in Equation 6.1.

$$v_{x_t} := (1 - \alpha) v_{x_t} + \alpha \sum_{i=1:l} k_i v_{x_{t-i}} \tag{6.1}$$

A space created by attractive forces would collapse resulting in lost volume as well as damaging the smoothness of functions. To prevent this, we incorporate repulsive forces to distribute points more uniformly. A simple idea is to linearly expand $\mathbb{V}$ to maintain a standard such as normalizing each dimension by the standard deviation. This can work well but often embeds most points into a smaller area, while drifting a minority too far. Thus we introduce a repulsive force which is inversely proportional to a power γ of distance and scaled by a factor $\beta \in \mathbb{R}$, as shown in Equation 6.2. In practice, we only consider the repulsion force between neighboring points to reduce the computation time.

$$v_{x_t} := v_{x_t} + \beta \sum_{x \in \mathbb{S}'} (v_{x_t} - v_x) \cdot d^{-\gamma}(v_{x_t}, v_x) \tag{6.2}$$

Finally, we obtain an embedding table in which every $a \in \mathbb{S}'$ has a corresponding point $v_a \in \mathbb{V}$. Note the embedding will not encompass elements that were not observed during the data access sequence.

6.2.3 Correlation-based Prediction

We believe that Temporally Aware Embedding produces correlation functions that can be used by a variety of data access operations in combination with other algorithms. For prefetching, an obvious implementation is to retrieve the N most correlated elements given $u(a)$ where $a \in \mathbb{S}$. A naive approach is to calculate $d(v_a, v_b)$ for every $b \in \mathbb{S}'$ based on the embedding table that we obtain from TAE and prefetch the element with the least distance to v_a.

Algorithm 2 Temporally Aware Embedding

```
Given mixed sequence $\mathscr{D}$, the desired number of mapping dimensions $n$, kernel $\mathscr{K}$, attraction factor $\alpha$, repulsion factor $\beta$, repulsion power $\gamma$, stopping criterion $\tau$
$\mathbb{S} \leftarrow$ UniqueElements($\mathscr{D}$)
**for** $s \leftarrow \mathbb{S}$ **do** $\mathscr{M}(s) \leftarrow \mathscr{N}(d; \mathbf{0}, \mathbf{I})$
**end for**
**while true do**
    $\mathscr{M}' \leftarrow \mathscr{M}$
    $\mathscr{M}'' \leftarrow \textsc{Attractive}(\mathscr{D}, \mathbb{S}, \mathscr{K}, \mathscr{M}', n)$
    **for** $s \leftarrow \mathbb{S}$ **do** $\mathscr{M}(s) \leftarrow (1-\alpha) \cdot \mathscr{M}'(s) + \alpha \cdot \mathscr{M}''(s)$
    **end for**
    $\mathscr{M}'' \leftarrow \textsc{Repulsive}(\mathscr{M}', \mathbb{S}, \gamma)$
    **for** $s \leftarrow \mathbb{S}$ **do** $\mathscr{M}(s) \leftarrow \mathscr{M}(s) + \beta \cdot \mathscr{M}''(s)$
    **end for**
    **if** $\sum_{s \in \mathbb{S}} d(\mathscr{M}(s), \mathscr{M}'(s)) < \tau$ **then break**
    **end if**
**end while**
**return** $\mathscr{M}$
```

Algorithm 3 Attraction Force

```
function ATTRACTIVE($\mathscr{D}, \mathbb{S}, \mathscr{K}, \mathscr{M}, n$)
    $l \leftarrow$ Length($\mathscr{K}$)
    $c \leftarrow 0^{\mathbb{S}}$
    $\mathscr{M}' \leftarrow 0^{\mathbb{S} \times n}$
    for $i \leftarrow l \ldots$ do
        $\mathscr{M}'(\mathscr{D}_i) \leftarrow \mathscr{M}'(\mathscr{D}_i) + \sum_{j=i-l:i} \mathscr{K}_j \mathscr{M}(\mathscr{D}_{i-j})$
        $c_s \leftarrow c_s + 1$
    end for
    for $s \leftarrow \mathbb{S}$ do $\mathscr{M}'(s) \leftarrow \mathscr{M}'(s)/c_s$
    end for
    return $\mathscr{M}'$
end function
```

Algorithm 4 Repulsion Force

```
function REPULSIVE($\mathscr{M}$, $\mathbb{S}$, $\gamma$)
    $\mathscr{M}' \leftarrow 0^{\mathbb{S} \times n}$
    for $s \leftarrow \mathbb{S}$ do
        $\mathscr{M}'(s) \leftarrow \sum_{s' \in \mathbb{S}} (\mathscr{M}(s') - \mathscr{M}(s)) \cdot d^{-\gamma}(\mathscr{M}(s), \mathscr{M}(s'))$
    end for
    return $\mathscr{M}'$
end function
```

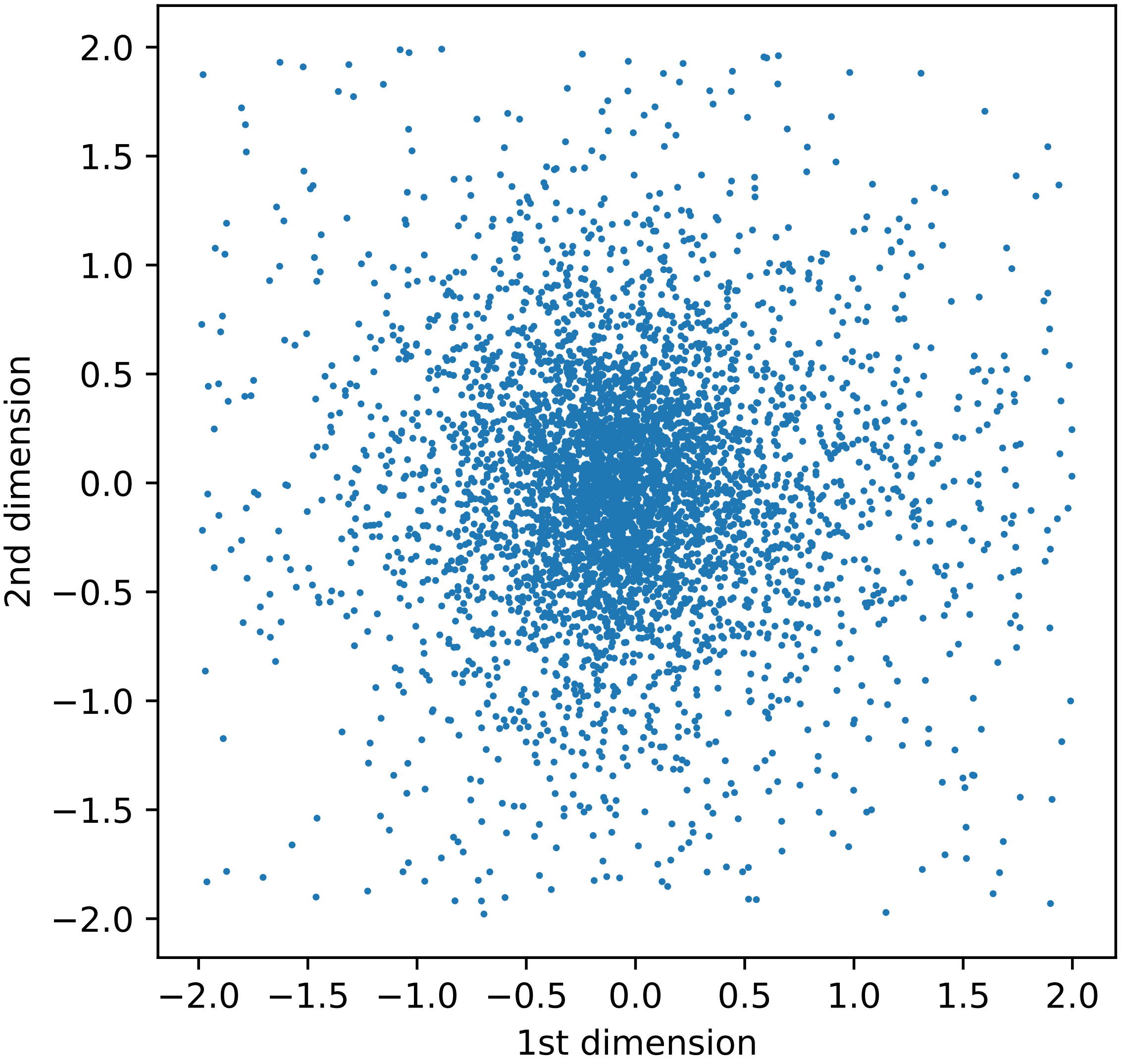


Figure 6.2: A 2D visualization of the embedding discovered by TAE on synthetic data, Attraction forces concentrate some of the data in the center, while the repulsion forces spread the rest of the data across the space.

6.3 Experiments

To evaluate the utility of the correlation function learned by TAE, we compare it to other approaches on a simplified prefetching scenario. To do this, we simply retrieve the N most correlated elements for each location on a sequence, and then evaluate the frequency of their occurrence in the near future. The experiments are performed both on simulated and real computer storage access traces.

6.3.1 Synthetic Data

To generate synthetic data, we simulate a computer storage scenario. Initially, a dataset $\mathbb{S}$ of N elements is generated, in which each of the elements is assigned an index. A correlation function $cor(a,b), a,b \in \mathbb{S}$ is imposed by assigning a uniformly sampled value between a number of randomly selected pairs and among consecutively indexed elements. Then a list of processes independently sample $\mathbb{S}$ based on the correlation function to produce a mixed sequence $\mathscr{D}$. The procedure for generating synthetic data is shown in Algorithm 5.

For our experiments, we generated four sets of N=5K, 10K, 20K, 50K elements respectively, in each of which neighboring elements and 10N pairs are set correlated. A mixed sequence of length 100N is sampled from each set.

Algorithm 5 Mixed Sequence Generator

```
Given set $\mathbb{S}$, correlation function $cor(\cdot,\cdot)$, number of processes $M$, length of sequence $T$, process switch probability $p_{\text{switch}}$, process reinitialize probability $p_{\text{reinit}}$
$\mathscr{A} \leftarrow$ Uniform($M; \mathbb{S}$)
$a \leftarrow$ Uniform(1; $M$)
for $t \leftarrow 1, T$ do
    if Uniform() $\leq p_{\text{switch}}$ then
        $a \leftarrow$ Uniform(1; $M$)
    end if
    if Uniform() $\leq p_{\text{reinit}}$ then
        $\mathscr{A}_a \leftarrow$ Uniform(1; $\mathbb{S}$)
    else
        $\mathscr{A}_a \leftarrow$ Sample(1; $\mathbb{S}, cor(\mathscr{A}_a, \cdot)$)
    end if
    $\mathscr{D}_t \leftarrow \mathscr{A}_a$
end for
return $\mathscr{D}$
```

6.3.2 Computer Storage Traces

We also apply TAE to computer storage I/O trace datasets, Financial1 and Financial2, from the UMass Trace Repository. These datasets were gathered from I/O traces of Online Transactional Processing systems running at two large financial institutions. The logical block address (LBA) information is used as the index, and the addressed storage unit is treated as the element. Because of the spatial locality and the distributed characteristics of computer storage, we assume that the correlation function on storage units is similar to that generated by our mixed sequence generator. Moreover, contemporary computers and servers are usually running a large number of threads simultaneously. Therefore, the real-world data access traces are both mixed and noisy, which is substantially different from many natural language processing sequence learning tasks.

Financial1 is a mixed sequence of length 5334987, which contains 710908 distinct units. Financial2 is a mixed sequence of length 3699195, which contains 296072 distinct units. As no higher-level information, e.g. file distribution or OS thread id, is provided with those datasets, we do not know the ground truth correlation function information for these traces.

6.3.3 Baseline Approaches

We compare TAE to the following techniques:

1. **Informed Guess**. This heuristic simply makes random guesses informed by the prior probabilities of the elements, thus leveraging the degree of data concentration.

2. **Local Neighbors**. Prediction is made by by selecting a neighbor based on a Gaussian distribution, thus leveraging the degree of data locality.

3. **Embedding LSTM** [33]: This approach learns the differences in computer storage addresses by assuming that the patterns are spatially invariant. It encodes the most frequent deltas (differences) into one-hot bins and then learns the sequence using a LSTM. They also propose an advanced approach combining clustering and LSTM. The advanced approach is designed to handle the long thread-switching interval of computer memory, which does not exist in computer storage. For this comparison, we use the same configuration as [33], which includes a 128×2 LSTM, 500k training steps, a sentence length of 64, embedding size of 128, and is restricted to the 50,000 most frequently appearing deltas.

4. **Transition Matrix**. This technique predicts the most likely next element based on transition probabilities. Rather than using consecutive elements, we augment the transition probability of any two elements appearing within a short window. Conceptually, this approach is the most similar to TAE.

Table 6.1 shows the configuration of TAE used for our experiments. Based on our initial pilot studies, we concluded that its performance is not very sensitive to modifications in the hyperparameters.

Table 6.1: TAE Hyperparameters

Kernel	linear
Window Size	64
Alpha	0.5
Beta	5e-7
Gamma	2
Number of Dimensions	8
Stopping Criterion	5e-3

Table 6.2: The probability that top-1 prediction appears in the following 128 time steps.

Approach\Dataset	Synthetic (5K)	Synthetic (10K)	Synthetic (20K)	Synthetic (50K)	Financial1	Financial2
Informed Guess	0.0215	0.0109	0.0053	0.0021	0.0479	0.0328
Local Neighbors	0.0441	0.0330	0.0287	0.0256	0.0612	0.0901
Embedding LSTM	0.0267	0.0219	0.0132	0.0103	0.1368	**0.2417**
Transition Matrix	0.2036	0.1979	0.1955	0.1961	0.2476	0.1801
TAE	**0.2708**	**0.2621**	**0.2586**	**0.2580**	**0.3239**	0.1947

Table 6.3: The average occurrences of top-1 prediction in the following 128 time steps.

Approach\{ }Dataset	Synthetic (5K)	Synthetic (10K)	Synthetic (20K)	Synthetic (50K)	Financial1	Financial2
Informed Guess	0.0264	0.0134	0.0065	0.0027	0.0980	0.0532
Local Neighbors	0.0526	0.0390	0.0335	0.0296	0.0916	0.1215
Embedding LSTM	0.0360	0.0300	0.0181	0.0150	0.2500	0.3469
Transition Matrix	0.2855	0.2800	0.2745	0.2764	0.5298	**0.4322**
TAE	**0.3827**	**0.3677**	**0.3610**	**0.3610**	**0.6711**	0.4249

6.3.4 Results

In each experiment, the first 70% of the sequence is used for training, and the last 30% is reserved for testing. The experimental results are reported both on the probability (Table 6.2) and the occurrence (Table 6.3) of the top-1 prediction in a window of the following 128 time steps. Because of the mixing, consecutive elements in a sequence are unlikely to be generated by the same process. Therefore, we count those as successful predictions provided they appear within a short future window.

Our results show that the mixed synthetic sequences are very challenging and defy the commonly used concentration and locality heuristics. Those heuristics (Informed Guess and Local Neighbors) perform better with the financial traces, which exhibit a higher skewness of the distribution and a stronger locality. As we expected, the embedding LSTM [33], which performs well in computer memory scenarios, is not suitable for mixed sequences that contain large numbers of elements. The LSTM does outperform the other methods on the Financial2 dataset. However the training of a LSTM network costs dozens of GPU hours, compared to a few minutes for TAE.

Surprisingly, our approach significantly outperforms transition matrix, which uses the same underlying idea of strengthening the correlation between elements that appear within a short time window. We believe that the transition matrix fails to model the indirect correlation between elements which is captured by our technique. Since TAE learns an embedding, it is easier to combine with other machine learning methods. It is also worth noting that our approach has a similar prediction accuracy across synthetic datasets of varying sizes, which demonstrates its scalability.

6.4 Discussion

This section discusses possible extensions of Temporally Aware Embedding to handle unseen data elements along with a new correlation-based sampling technique.

6.4.1 Mutual Mapping

Only the embedding of $a \in \mathbb{S}'$ is meaningful with Temporally Aware Embedding. However, if there exists another meaningful encoding, e.g. a representation for a localization property in $\mathbb{S}$, the embedding can be extended to any $a \in \mathbb{S}$ through a mutual mapping between those two representations using continuous function approximators. Such an encoding exists for certain problems. For example, the addresses in computer systems are commonly encoded as tuples of bits, subsets of which may contain segment or page information.

Assume $u(a) \in \mathbb{U}$ is a proper encoding for any $a \in \mathbb{S}$. Two learning models, specifically deep neural networks, $\mathscr{F}(v_a) = u(a)$ and $\mathscr{G}(u(a)) = v_a$ can be trained respectively with the encoding and embedding pairs of $x \in \mathbb{S}'$. Based on continuity restriction, $\mathscr{F}$ is able to map every point v_a to $\mathbb{U}$ as a sample that is both temporally and locally correlated to nearby $b \in \mathbb{S}'$. Meanwhile, $\mathscr{G}$ allows us to discard the space-consuming embedding table. More importantly, it can embed unknown $a \in \mathbb{S}$ as long as $u(a)$ can be acquired.

6.4.2 Correlation-based Sampling

Since mutual mapping discards the table and extends the embedding, we propose a sampling method that can be extended to situations where $a \notin \mathbb{S}'$ or when some elements that are highly correlated to a do not appear in $\mathbb{S}'$.

Our sampling method starts by acquiring the embedding for a, which is $v_a = \mathscr{G}(u(a))$. Afterwards, a set of candidates $\{v_{b_1}, ..., v_{b_N}\}$ is sampled in $\mathbb{V}$ from a Gaussian distribution with $\mu = v_a$ and pre-selected small $\sigma \in \mathbb{R}$. We can obtain $u(b_i) = \mathscr{F}(v_{b_i})$ for $i = 1, ..., N$, and find the corresponding elements $b_1, ..., b_N$ with them if u is invertible. The procedure is as follows:

$$u(b_i) = \mathscr{F}(\mathscr{N}(\mathscr{G}(u(a)), \sigma^2)) \text{ for } i = 1, ..., N \tag{6.3}$$

According to mutual mapping, the sampled elements are neighbors to a both in $\mathbb{V}$ and $\mathbb{U}$. Thus they are highly correlated to a.

6.5 Chapter Summary

This chapter introduces a technique, Temporally Aware Embedding (TAE), for learning correlation functions from mixed data sequences. The intuition behind TAE is that an embedding can be learned from element co-occurrences within a sliding time window. Our technique is simple and fast, which makes it highly suitable for computer architecture applications such as prefetching. It is robust to noisy and mixed data sequences, and also scales to datasets with large numbers of unique elements.

We evaluated TAE vs. a set of popular heuristics and deep learning methods. Even though the TAE learning rule is based largely on first order Markov statistics, it significantly outperforms a transition model created from a direct estimate of those statistics. We believe that the TAE embedding is more effective at capturing indirect relationships that are missed by the transition matrix. For instance, two elements that co-occur with another shared element, yet do not appear together, are not considered correlated by the transition matrix but are likely to appear in close proximity in our embedding. In future work, we plan to extend our embedding and sampling techniques to handle unseen data elements.

CHAPTER 7
A GRAPH REPRESENTATION FOR AUTONOMOUS DRIVING

7.1 Problem Statement

Attention is governed by both endogenous (top-down) and exogenous (stimulus-driven) cognitive processes that empower human drivers to selectively scrutinize the regions of the roadway most relevant to their planned trajectory [69]. In higher density traffic, stimulus-driven control becomes harder since there are more potential distractions on the road; thus the internal representation used by the agent must scale to larger numbers of objects by selectively focusing on the objects most likely to affect the driver's decision-making. One way that this can be achieved is by employing an egocentric foveal representation; however, this does not scale well to cooperative multi-agent driving systems. This chapter introduces a new representation, GRAD (Graph Representation for Autonomous Driving) that possesses several key desiderata: 1) it is global, enabling a single representation to be shared across multiple agents; 2) it incorporates the future location of neighboring vehicles into the decision-making process; 3) it is more computationally efficient than attention mechanisms; 4) the same representation is applicable to both driving and trajectory prediction tasks.

Leurent and Mercat [43] identify roadway representation as a key problem for autonomous driving and note that most behavioral planning systems either employ a list of features for every vehicle or a spatial occupancy grid to represent traffic on the road. They propose a social attention model [43] which represents egocentric dependencies between the driver and surrounding vehicles; it handles varying length inputs and is permutation invariant. Their social attention technique uses a multi-headed attention model that focuses on vehicles that can potentially interfere with the planned route.

Graph neural network models have been utilized as a popular representation for vehicle trajectory prediction problems [49, 51, 47, 48, 46, 50]. Unlike previous work, GRAD incorporates short-term estimated future vehicle trajectories into its graph representation. This provides GRAD with better lookahead properties than other graph neural network representations. This chapter compares our proposed GRAD feature representation vs. both social attention [43] and HEAT (Heterogeneous Edge-enhanced graph ATtention network) [46], a top performing highway motion prediction technique. We compare the usage of different models as an input to an RL agent that learns a highway driving policy for dense traffic scenarios using Proximal Policy Optimization (PPO) [70].

Our dissertation makes the following contributions:

1. We introduce GRAD (Graph Representation for Autonomous Driving): a global, general representation that can be shared across multiple agents and used for both driving and prediction tasks.

2. We demonstrate that GRAD outperforms the social attention model which is a top performer at driving in the *highway-env* simulation environment and possesses a comparable number of trainable parameters.
3. For the autonomous driving task, we show that GRAD outperforms HEAT [46], which is a top performer at pure behavior prediction problems across multiple types of roadways.
4. We present an analysis on how modifying the space-time graph used by GRAD affects driving performance.

7.2 Graph Representation for Autonomous Driving

7.2.1 Graph Transformer

As graphs are a flexible and powerful representation for numerous real-world datasets, including social networks, citation data, and biological structures, there have been significant research efforts dedicated to incorporating graph representations into neural networks [71, 72, 73, 74]. This chapter is an attempt to apply graph neural networks to the autonomous driving domain. Although graph neural networks have been very successful at driving trajectory prediction [50, 46, 47, 48], they have not be evaluated as a representation for learning policies for autonomous driving.

In our proposed architecture, edge attributes play an important role in encoding the locality, which requires the graph neural network to be capable of exploiting edge attributes. Shi et al. [74] introduce a graph transformer operator where the edge attributes are utilized for both attention

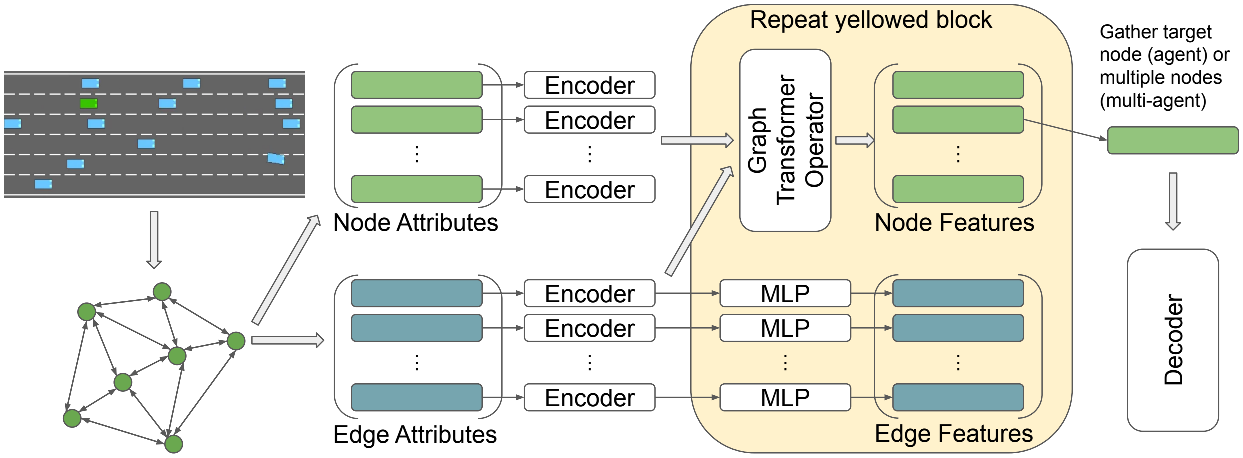


Figure 7.1: Global Graph Representation

computation and feature aggregation; Fey and Lenssen [75] provide a neat implementation of the operator. Given a graph $(X, \mathscr{E})$ where X contains node features and $\mathscr{E}$ denotes edge features, the i-th node is updated as follows [75]

$$\mathbf{x}'_i = \mathbf{W}_1\mathbf{x}_i + \sum_{j \in \mathscr{N}(i)} \alpha_{i,j}(\mathbf{W}_2\mathbf{x}_j + \mathbf{W}_3\mathbf{e}_{i,j}) \tag{7.1}$$

where the attention coefficient $\alpha_{i,j}$ is

$$\alpha_{i,j} = \text{softmax}\left(\frac{(\mathbf{W}_4\mathbf{x}_i)(\mathbf{W}_5\mathbf{x}_j + \mathbf{W}_6\mathbf{e}_{i,j})}{\sqrt{d}}\right) \tag{7.2}$$

d (feature dimension) is used to rescale attention appropriately.

7.2.2 Global Graph Representation

The architecture of GRAD is depicted in Figure 7.1. A graph representation is well suited for selectively limiting attention since each agent perceives the interacting objects accurately via agent-centric attributes through direct links and the farther ones via the information flow over graph convolutional operators.

Given a road scene, an observation is a collection of road objects (i.e. vehicles if modeling vehicle interactions only), $X \in \mathbb{R}^{N \times d_x}$ where N is the number of objects and d_x is the dimensions of features. It contains the localization information for each of the objects, including coordinates, velocity, and acceleration. By applying a distance metric (as described in Section 7.2.3) on the localization information, a directed graph $(X, \mathscr{L})$ is constructed with objects as nodes, and links limited by either a maximum number of nearest neighbors or a fixed distance to connecting nodes. For each link connecting node i and node j, the edge attributes $\mathscr{E}_{i,j}$ are populated with relative information between the pair, including relative position, velocity, and acceleration. Note that objects are linked by a pair of directional links with different edge attributes. Therefore, from the viewpoint of object i, $\mathscr{E}_{i,j \in \mathscr{L}_i}$ contains the agent-centric information of a small surrounding sub-region.

Depending on whether the history is used, either MLP or sequence-based neural networks are applied to each node X_i and each edge $\mathscr{E}_{i,j}$ respectively. Then the encoded graph $(X', \mathscr{L}, \mathscr{E}')$ is passed through multiple blocks of graph transformer operators [74] which are capable of utilizing the edge attributes as shown in Section 7.2.1. While the operators commonly update only the nodes, the edge features are updated using a separate MLP within each block. The details of the block are shown in the yellow box in Figure 7.1. The output retains the original graph structure.

It worth noting that the architecture in Figure 7.1 is an "overkill" version which can be plugged into multi-agent settings or used directly as global scene representation. For the single agent configuration described in this chapter, the computation can be reduced to be within the scope of the single target node (driving agent), which can be visualized as a pyramid over multiple graph convolutional layers. This only requires updating the target node representing the driving agent by making queries on the nodes directly linked to it in the last graph convolutional layer.

7.2.3 Space-Time Graph

Algorithm 6 Space-Time Graph

Require: coordinates $\mathbf{x}$, velocities $\mathbf{v}$, accelerations $\mathbf{a}$, horizon T, discount γ

```
function WAYPOINT($\mathbf{x_1}, \mathbf{v_1}, \mathbf{a_1}, \mathbf{x_2}, \mathbf{v_2}, \mathbf{a_2}, T, \gamma$)
    for $t = 0 ... T$ do
        $\mathbf{x_1}^t = \mathbf{x_1} + \mathbf{v_1} \cdot t + \mathbf{a_1} \cdot t^2/2$
        $\mathbf{x_2}^t = \mathbf{x_2} + \mathbf{v_2} \cdot t + \mathbf{a_2} \cdot t^2/2$
        $d^t = \|\mathbf{x_1} - \mathbf{x_2}\| \cdot \gamma^t$
    end for
    $d = \min_{1 \leq t \leq T} d^t$
    return $d$
end function
```

```
function TRAJECTORY($\mathbf{x_1}, \mathbf{v_1}, \mathbf{a_1}, \mathbf{x_2}, \mathbf{v_2}, \mathbf{a_2}, T, \gamma$)
    for $t = 0...T$ do
        $\mathbf{x_1}^t = \mathbf{x_1} + \mathbf{v_1} \cdot t + \mathbf{a_1} \cdot t^2/2$
        $\mathbf{x_2}^t = \mathbf{x_2} + \mathbf{v_2} \cdot t + \mathbf{a_2} \cdot t^2/2$
    end for
    for $t = 1...T$ do
        $\mathbf{l_1}^t = (\mathbf{x_1}^{t-1}, \mathbf{x_1}^t)$
        $\mathbf{l_2}^t = (\mathbf{x_2}^{t-1}, \mathbf{x_2}^t)$
    end for
    for $s = 1...T$ do
        for $t = 1...T$ do
            ▷ Euclidean distance between line segments
            $d^{s,t} = ||\mathbf{l_1}^s, \mathbf{l_2}^t||_l$
        end for
    end for
    $d = \min_{1 \leq s,t \leq T} d^{s,t} \cdot \gamma^{|s-t|}$
    return $d$
end function
```

In the first step of vanilla GRAD, a graph is built by linking the neighboring nodes based on their distance measurements, i.e. Euclidean distance on synchronous positions in the past and current time steps. To further concentrate direct attention on the objects that are highly likely to be involved in future interactions, we propose a Space-Time Graph which builds the graph not merely on the current localization but also the estimated future positions or trajectories.

In the simplest form, given the current localization $(\mathbf{x}_i, \mathbf{v}_i, \mathbf{a}_i)$ of node i, the future position at time step t is estimated with the motion equation

$$\mathbf{x}_i^t = \mathbf{x}_i + \mathbf{v}_i \cdot t + \mathbf{a}_i \cdot t^2/2 \tag{7.3}$$

and the estimated trajectory $\mathscr{T} = (\mathbf{x}_i^0, \mathbf{x}_i^1, ..., \mathbf{x}_i^T)$ where T is the time horizon.

When working with positions, as shown as WAYPOINT function in Algorithm 6, the distance is the minimum measurement between a pair of coordinates sampled synchronously, that is

$$d_{i,j}^t = \|\mathbf{x}_i^t - \mathbf{x}_j^t\| \tag{7.4}$$

$$d_{i,j} = \min_{0 \leq t \leq T} d_{i,j}^t \cdot \gamma^t \tag{7.5}$$

where $\gamma \geq 1$ is a discount factor to reflect the higher importance of the nearer future. While being vulnerable to the change of motion status, it performs reasonably well with a limited time horizon. The position estimations can be further improved with more sophisticated prediction methods; for instance, the graph can be constructed cyclically and incrementally along with predictions for behavior prediction tasks.

Even without more accurate estimations, the robustness can be improved by measuring the distance between trajectories instead of positions, shown as the TRAJECTORY function in Algorithm 6. Given the trajectory $\mathscr{T} = (\mathbf{x}_i^0, \mathbf{x}_i^1, ..., \mathbf{x}_i^T)$, its representation is translated to line segments

$\mathscr{T} = (\mathbf{l}_i^1, \mathbf{l}_i^2, ..., \mathbf{l}_i^T)$ where $\mathbf{l}_i^t = (\mathbf{x}_i^{t-1}, \mathbf{x}_i^t)$. The distance is the minimum measurement among pairwise line segments between the trajectories of a connecting pair of nodes, that is

$$d_{i,j}^{s,t} = \|\mathbf{l_i}^s, \mathbf{l_j}^t\|_l, s = 1...T, t = 1...T \tag{7.6}$$

$$d_{i,j} = \min_{0 \leq s,t \leq T} d_{i,j}^{s,t} \cdot \gamma^{|s-t|} \tag{7.7}$$

where $\gamma \geq 1$ is a discount factor to penalize the pairs of line segments which are expected to occur during more distant time frames. The code for our system can be found at: `https://github.com/zerongxi/graph-sdc`, and the hyperparameters used with GRAD are shown in Table 7.2.

7.3 Experiments

7.3.1 Task

We validate the proposed graph representation within a reinforcement learning framework performing a simulated highway driving task. GRAD is plugged into Proximal Policy Optimization (PPO) [70] as the feature extractor shared by both policy and value networks. Because of the plug-and-play property of GRAD, it can easily be paired up with other RL algorithms and other tasks such as behavior prediction. We have tested with DQN as well and observed consistent results to those shown in Section 7.3.3 for PPO. Our PPO implementation is described in Table 7.3.

Highway-env [76] is a simulated environment of highway driving scenarios for behavioral decision-making tasks. We use it to create new highway driving scenarios, but it is also possible to replay data from existing datasets such as NGSIM [77]. In sparse scenes with few vehicles, many algorithms perform comparably well, and both multi-headed attention and the GRAD representation degenerate to simply focusing attention on the few cars. Therefore, we conduct the experiments on a highway scenario where the agent drives forward on an one-way multi-lane road with heavy traffic density. The parameters for our *highway-env* experiments are given in Table 7.1, and Figure 7.2 shows the simulator.

The environment is configured to receive inputs at 2 frames per second within a maximum of 30 seconds, unless the agent crashes, which terminates the episode immediately. Therefore, the episode length is 60 at maximum. The scene is configured to contain 6 lanes of the same heading direction and 100 vehicles on road.

Observation The observations are the kinematics status of the agent and the surrounding vehicles, i.e. {coordinates, velocity, heading}. The maximum number of observed surrounding vehicles is 32, and their speed ranges from {15, 25}.

Action The action space is a discrete set {OneLaneLeft, Idle, OneLaneRight, SpeedLevelUp, SpeedLevelDown}, in which the speed levels are {20, 25, 30, 35, 40}. The actions are executed by a layer of speed and steering controller which is included in *highway-env* on top of the continuous low-level control.

Reward The reward consists of 1) a reward of 0.2 for surviving each time step, 2) a reward linearly mapped from the driving speed of (20, 40) to (0, 0.8). Therefore, the maximum reward is 1.0 per action step and is 60.0 per episode given that the maximum episode length is 60.

Table 7.1: Highway-env

Parameter	Value
Number of visible vehicles	32
See vehicles behind	False
Action type	Discrete meta action
Speed levels	{20, 25, 30, 35, 40}
Episode length	60 = 2 fps × 30s
Number of road lanes	6
Number of vehicles	100
Traffic density	1.8
Rewarded speed range	[20, 40]
Reward for surviving	0.2
Maximum reward for speed	0.8

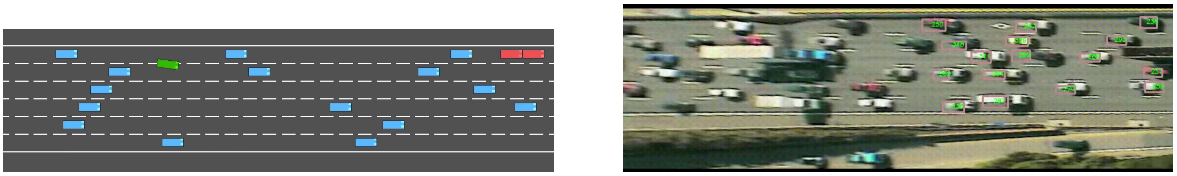
Each model is trained for 500k action steps, and evaluated for 24 episodes with deterministic policies every 10k training steps.

7.3.2 Baseline

The social attention (SA) technique [43] models agent interactions using a multi-headed attention mechanism. By default it uses the agent-centric coordinate system for surrounding vehicles

Table 7.2: Graph Representation for Autonomous Driving

Parameter	Value
Maximum number of links per node	8
Maximum distance of linked nodes	40
WAYPOINT-time horizon	5 seconds
WAYPOINT-sampling frequency	4 per second
WAYPOINT-γ	2.0
TRAJECTORY-time horizon	1 second
TRAJECTORY-γ	1.0



(a) Simulated driving scene in highway-env [76]

(b) Realistic driving scene in NGSIM dataset [78]

Figure 7.2: Comparison of simulated and realistic driving scene on highway.

Table 7.3: Proximal Policy Optimization

Parameter	Value
Number of training steps	500k
Number of environments	12
Number of steps per update	512
Buffer size	6144
Number of epochs	10
Batch size	256
Learning rate	1e-3 $\rightarrow$ 5e-4 in 40% steps
Dimensions of policy networks	[256, 256]
Dimensions of value networks	[256, 256]
Discount factor	0.9

while representing the agent itself in global coordinates. Therefore, it has separate encoders for the agent and the other vehicles, and applies cross attention to allow the agent to make queries on the surrounding vehicles.

As our method works directly in the global coordinate system, which is easier to extend to multi-agent tasks without excessive additional computations, we compare GRAD to both the original SA (SA-agent-centric) and SA with the global coordinate system (SA-global).

HEAT [46] models agent interactions with a specialized heterogeneous edge-enhanced graph attention network. It is capable of modeling the interactions among different agent types by incorporating an interaction type into the edge attributes. Aside from the difference in the graph attention network, HEAT constructs the graph on static status while GRAD takes the future status into consideration using a Space-Time Graph.

To make a fair comparison, the neural networks in both our method and the baseline have the same number of layers and a similar number of parameters. The characteristics of the models are shown in Table 7.4.

7.3.3 Results

A multi-head attention mechanism can be viewed as equivalent to a version of the full graph GRAD without edge attributes where each of the nodes is allowed to make queries on all other nodes. However, we show that GRAD can achieve comparable performance with a sparser graph. Our experiments demonstrate that it outperforms SA consistently with sufficient links. GRAD has two

Table 7.4: Model Characteristics

Architecture	Social Attention	GRAD / HEAT
Layer sizes	Encoder: [192, 192, 192] Attention: [192] Heads: 2 Decoder: [256, 256]	Encoder: [128, 128] Attention: [128, 128] Heads: 2 Decoder: [256, 256]
Number of parameters	$\sim$770k	$\sim$720k
Queries per node	All nodes	Linked nodes
Permutation invariant	Yes	Yes
Coordinate system	Agent-centric	Global

advantages: 1) it utilizes a mixture of global and agent-centric coordinate systems, 2) it has two attention layers while SA has only one. The experiments also demonstrates that it outperforms HEAT particularly when the links are sparse, which validates the effectiveness of the Space-Time Graph.

Sparse graph To demonstrate that the accurate agent-centric information of farther objects is unnecessary, we experiment over different levels of graph sparsity by limiting either the maximum number of links per node or the maximum distance of connecting nodes. As shown in Figure 7.3, GRAD achieves considerable performance with merely 8 links per node at maximum, or within a radius of 30 which is approximately the distance the agent travels in one second. Furthermore, GRAD with Space-Time Graph suffers much less from the harsher limitation on links as it con-

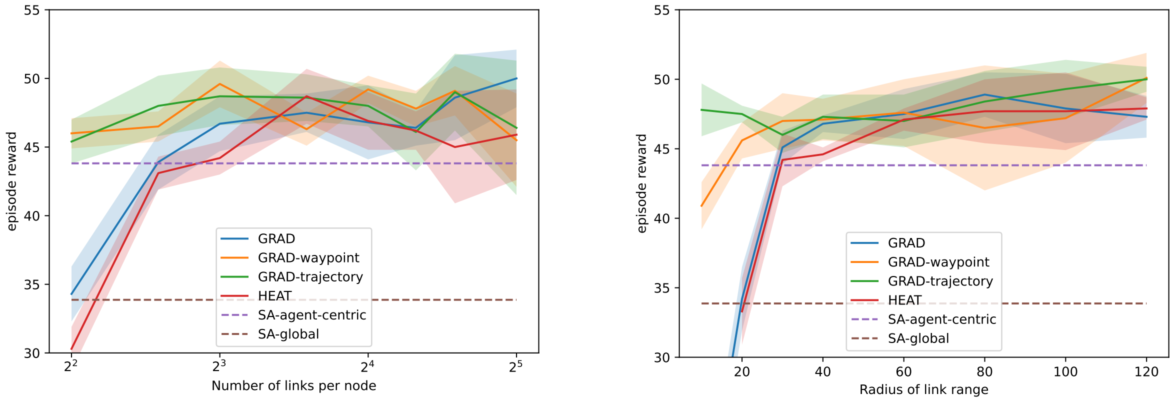
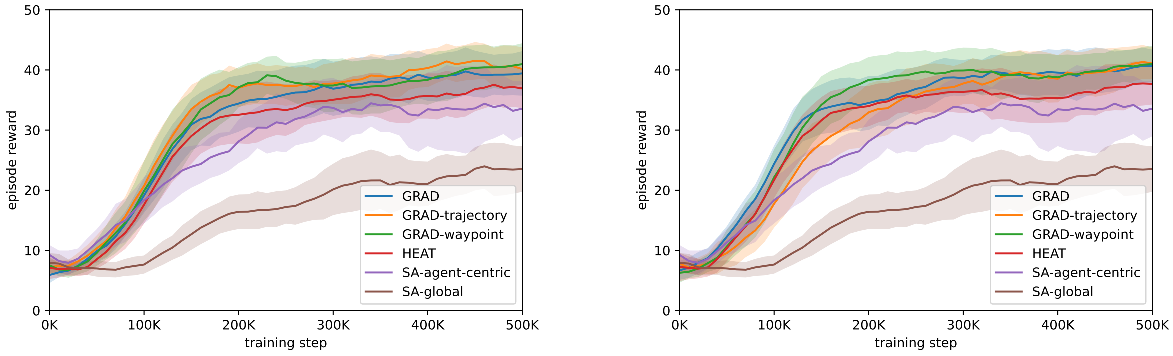


Figure 7.3: Episode rewards of best policy with limited number of links per node (left) or limited link range (right). Each data point is an average of 5 independent runs, and the shadows show the standard deviations. All versions of GRAD with a sufficient number of links outperform both types of social attention (SA), marked by the dotted lines.

centrates the direct attention on most important objects. This difference in performance converges as this limitation is mitigated.

Convergence speed A good representation needs to provide necessary information in a form that the model can easily use. As shown in Figure 7.4, GRAD and its variants generally converge in fewer than 60% training steps compared to SA. We believe that these differences in convergence speed and sample efficiency might be even more accentuated in prediction tasks.

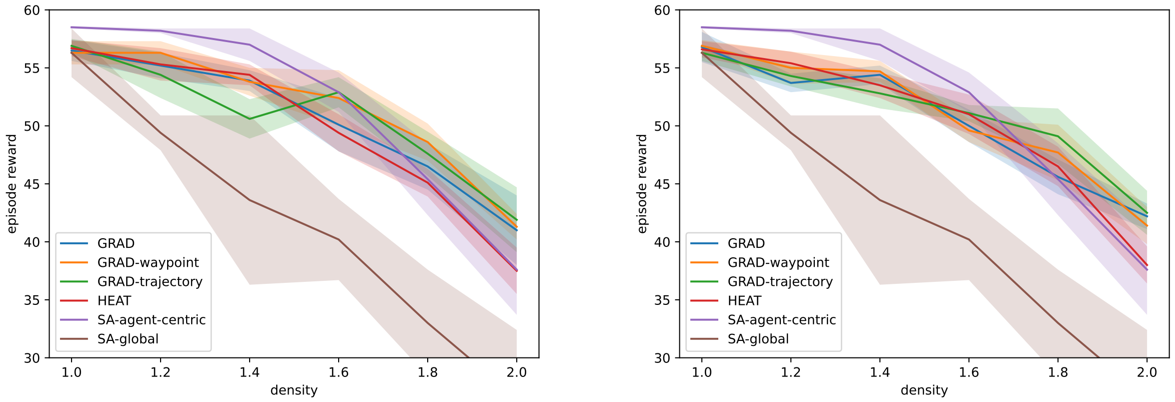
Traffic density To study the expressive power of crowded scenes, we evaluate our method and the baseline in various simulated densities of traffic. Higher density traffic both makes the task more difficult and reduces the amount of episode rewards. For example, the agent has to slow down to wait for space to overtake in some states, which makes it impossible to reap the full speed



(a) Number of neighbors = 8

(b) Radius = 40

Figure 7.4: Episode rewards in evaluation over training steps. Each data point is an average of 5 independent runs and smoothed within a window size of 9. The shadows are the standard deviations of the runs.



(a) Number of neighbors = 8

(b) Radius = 40

Figure 7.5: Episode rewards of best policy trained and evaluated with various traffic densities. Each data point is an average of 5 independent runs, and the shadow shows the standard deviations. GRAD is less affected by denser traffic as shown by its gentle slope compared to SA.

reward. In Figure 7.5, we observe that GRAD shows greater robustness when faced with denser traffic; it can be seen that the slopes are much gentler than those of the baselines. This is critical for driving in the real-world where traffic density can be quite high.

7.4 Complexity Analysis

As the GRAD architecture operates directly in the global coordinate system, the representation remains symmetrical for all nodes regardless of the agent or the surrounding vehicles. Thus it's straightforward to extend to cooperative multi-agent tasks, such as multi-agent planning or behavior prediction, without excessive additional computations. GRAD has the exact same complexity for either single or multi-agent settings. As a comparison, agent-centric based methods [43, 45] must centralize the coordinates of all objects and execute the neural networks for each agent of interest individually.

Attention mechanisms [79] update each node using information aggregated across all nodes, which intrinsically has a computational complexity of $\mathcal{O}(n^2)$ for an input size of n, unless we merely allow the target node to make one-time queries in a single layer setting like social attention [43]. The empirical results show that this can only work well in agent-centric coordinate systems, which increases the complexity to $\mathcal{O}(n^2 \cdot m)$ for m agents in multi-agent settings.

As GRAD is capable of generating a global scene representation using efficient implementations such as k-d [80] or ball trees [81], the graph can be built within $\mathcal{O}(k \cdot n \log n)$ time where k is the maximum number of links per node. The Space-Time Graph does not necessarily easily fit into this framework, but it is still able to retain the same computational complexity by refinements on top of the Euclidean distance. The graph transformer operations require $\mathscr{O}(k)$ per node adding up to $\mathscr{O}(k \cdot n)$ for all nodes. Therefore, GRAD exhibits a computational complexity of $\mathscr{O}(k \cdot n \log n)$ for either single or multi-agent settings.

7.5 Limitations

This work is focused on modeling the interactions among vehicles. *Highway-env* [76] provides a relatively simple environment for concept validation that has the following limitations: 1) it includes no agent types other than vehicles and 2) road information is implicitly embedded in the coordinate system. However, it's important to have a comprehensive perception of the environment in the real world, including more complex roadway structures, traffic signals, and diverse agent types such as cyclists and pedestrians. It is straightforward to plugin GRAD into a larger architecture like Multipath++ [45] to model the agent interaction component. However, this obviously does not utilize the full potential of our graph representation.

Instead, there are two ways to integrate every object on the road into GRAD: 1) mark each category of object as a type of node and apply heterogeneous graph convolutional operators like [82]; 2) map various types of objects into the same embedding space.

To further compress the computation to handle a larger number of objects, GRAD can benefit from dimensionality reduction operators such as pooling. One possible solution is to gradually reduce the dimensions over graph convolutional layers, and to have area-level representations instead of node-level ones for the agents to make queries on. Assuming that only the agents are actively in motion, another possible solution is to apply graph convolutional operators on agents only, which effectively permits only the agent types to perform queries on all types of objects.

7.6 Chapter Summary

This chapter introduces the GRAD representation for autonomous driving systems that rely on bird's-eye roadway information. GRAD is a global, permutation-invariant representation that retains the same computation complexity in single or multi-agent settings. A key innovation is GRAD's usage of a space-time graph that incorporates estimated future trajectories of neighboring vehicles. Each output node of GRAD functions as a high-level aggregation of local scene information. Our empirical results on using GRAD to learn highway driving policies with PPO show that GRAD is both efficient and robust to crowded traffic conditions.

CHAPTER 8
CONCLUSION

8.1 Overall Summary

Possessing the ability to interact with human-centered environments is an important desideratum for an intelligent agent. Therefore, understanding and modeling human behaviors is a longstanding research topic in artificial intelligence. In this thesis, we explore three aspects of this area: 1) predicting the pilot behaviors and outcomes in flight tasks with multi-modal features including flight controls, visual attention, knowledge acquisition tests, and time series data (chapter 3, 4); 2) investigating an environment using a novel variance-based exploration policy for reinforcement learning (chapter 5); 3) representing and encoding roadways scenes for autonomous driving with a computationally efficient and decentralized graph transformer architecture (chapter 7).

8.2 Future Work

8.2.1 Efficient Representation for Complex Environments

From the perspective of the No Free Lunch Theorem, an appropriate representation effectively reduces the search space by incorporating prior knowledge. Therefore, the representation of an environment is a primary and essential step toward understanding and solving the problem of interest. The complexity of an environment can be quantified using various factors, including number of entities, heterogeneity, and dynamics. For example, an autonomous driving scenario can easily involve hundreds of thousands of objects, which are of various types including vehicles, pedestrians, cyclists, traffic lights, road graphs, obstacles, etc. Meanwhile, a large number of them are dynamically changing, such as the traffic light's status, and the motion of moving objects.

A popular approach is to represent such scenes as 2-D or 3-D grids and process them with convolutional neural networks (CNN). However, areas of a typical scene can differ drastically in terms of information density. CNN-based approaches are less stable at dealing with these differences and computationally inefficient in sparse areas. Advances in the transformer architecture cast a fresh light on this problem: 1) transformers do not require a fixed input structure like CNN, which makes it suitable for processing object-wise representation; 2) transformers are capable of dealing with long-range dependencies and thus a large number of objects. Wayformer [83] validates the effectiveness of the transformer on this problem. However it introduces more challenges, such as the non-scalable quadratic computational complexity, and the loss of positional information.

Our work on introducing graph transformers to encode driving scenes (GRAD) serves as a starting point to tackle those challenges. While the transformer can be viewed as a fully connected graph, GRAD sparsifies the graph with prior knowledge to enhance computational efficiency. Also it incorporates the relative information into link attributes to bridge the positional information. However it still lacks the ability to deal with the heterogeneous nature of the driving scenes. Also future work can be done on improving the effectiveness of the links and thus further sparsifying the graph.

8.2.2 Learning from Human Behavior and Beyond

The complexity of a problem can be easily of exponential growth along with the horizon of the solution. Therefore, an agent exploring without prior knowledge can severely limit its application range to toy problems in simple environments, while the real world is generally overwhelming and complicated. It usually requires wisdom and effort to incorporate prior knowledge into a policy, and sometimes it can be exceedingly difficult, especially for complicated problems.

A more feasible direction is to let the agents learn from demonstrations, i.e. human behaviors in most cases. Basic LfD algorithms simply clone the behaviors with prediction algorithms. However, this solution does not learn the underlying structure of the problems, and so results in poor generalization as well as being limited to the quality of the demonstrations. The poor generalization also makes it less scalable to higher complexity, as the requirement of demonstrations grows fast.

Beyond imitation, inverse reinforcement learning reduces the problem to learning a reward function instead of imitating the behaviors directly, and it is a step toward understanding the problem. On top of the learned reward function, the agent can seamlessly explore further into the search space. It is also easier to combine the reward function with other prior knowledge, which is usually represented as a reward function as well. Furthermore, it allows the possibility of incorporating a wider range of algorithms such as planning, and building more interpretable models, both of which are particularly meaningful for safety-critical problems, e.g. autonomous driving. Our work on efficient exploration policy [84] is a starting point towards building more intelligent agents capable of navigating complex environments.

APPENDIX A
IRB APPROVAL

UCF

Dr. Gita Sukthankar
Department of Computer Science
University of Central Florida
Orlando, FL 32816-2362
Tel: 407-823-4305
Email: gitars@eecs.ucf.edu

Mar 10, 2023

To Whom It May Concern:

Zerong Xi completed all of his human studies research under **IRB MOD00000390 Enhanced Learning Effectiveness of Virtual Reality Flight Trainers** (PI: Steve Fiore and Co-PI Gita Sukthankar). The IRB will not be closed until Aug 2023, since the research work is ongoing.

Sincerely,

Dr. Gita Sukthankar
Professor
Department of Computer Science
University of Central Florida

UCF

UNIVERSITY OF CENTRAL FLORIDA

Institutional Review Board
FWA00000351
IRB00001138Office of Research
12201 Research Parkway
Orlando, FL 32826-3246

APPROVAL

September 12, 2019

Dear Stephen Fiore:

On 9/12/2019, the IRB reviewed the following submission:

Type of Review:	Modification / Update
Title:	VRI: Enhanced Learning Effectiveness of Virtual Reality Flight Trainers
Investigator:	Stephen Fiore
IRB ID:	MOD00000390
Funding:	Name: Lockheed Martin Corporation
Grant ID:	None
IND, IDE, or HDE:	None
Documents Reviewed:	• informed consent 9.11.pdf, Category: Consent Form; • Joggle Research App CogTest Battery for Lockheed.docx, Category: Test Instruments; • Protocol updated 8.1 , Category: IRB Protocol;

The IRB approved the modifications on 9/12/2019.

In conducting this protocol, you are required to follow the requirements listed in the Investigator Manual (HRP-103), which can be found by navigating to the IRB Library within the IRB system.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Kamille Chap

Kamille Chaparro
Designated Reviewer

LIST OF REFERENCES

[1] N. Thai-Nghe, L. Drumond, A. Krohn-Grimberghe, and L. Schmidt-Thieme, “Recommender system for predicting student performance,” *Procedia Computer Science*, vol. 1, no. 2, pp. 2811–2819, Jan. 2010.

[2] R. Conijn, C. Snijders, A. Kleingeld, and U. Matzat, “Predicting Student Performance from LMS Data: A Comparison of 17 Blended Courses Using Moodle LMS,” *IEEE Transactions on Learning Technologies*, vol. 10, no. 1, pp. 17–29, Jan. 2017.

[3] F. Dehais, J. Behrend, V. Peysakhovich, M. Causse, and C. D. Wickens, “Pilot Flying and Pilot Monitoring’s Aircraft State Awareness During Go-Around Execution in Aviation: A Behavioral and Eye Tracking Study,” *The International Journal of Aerospace Psychology*, vol. 27, no. 1-2, pp. 15–28, Apr. 2017.

[4] V. Peysakhovich, O. Lefrançois, F. Dehais, and M. Causse, “The Neuroergonomics of Aircraft Cockpits: The Four Stages of Eye-Tracking Integration to Enhance Flight Safety,” *Safety*, vol. 4, no. 1, p. 8, Mar. 2018.

[5] C. Mills, R. Bixler, X. Wang, and S. K. D’Mello, *Automatic Gaze-Based Detection of Mind Wandering during Narrative Film Comprehension*. International Educational Data Mining Society, 2016.

[6] K. Krejtz, A. Duchowski, T. Szmidt, I. Krejtz, F. González Perilli, A. Pires, A. Vilaro, and N. Villalobos, “Gaze Transition Entropy,” *ACM Transactions on Applied Perception*, vol. 13, no. 1, pp. 4:1–4:20, Dec. 2015.

[7] ANA. SUSAC, ANDREJA. BUBIC, JURICA. KAPONJA, MAJA. PLANINIC, and MARIJAN. PALMOVIC, “EYE MOVEMENTS REVEAL STUDENTS’ STRATEGIES IN SIMPLE EQUATION SOLVING,” *International Journal of Science and Mathematics Education*, vol. 12, no. 3, pp. 555–577, Jun. 2014.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002.

[9] P. Esling and C. Agon, “Time-series data mining,” *ACM Computing Surveys*, vol. 45, no. 1, pp. 12:1–12:34, Dec. 2012.

[10] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, Jan. 2007.

[11] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1, pp. 37–52, Aug. 1987.

[12] G. Batista and M.-C. Monard, *A Study of K-Nearest Neighbour as an Imputation Method.*, Jan. 2002, vol. 30.

[13] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: A methodology review," *Journal of Biomedical Informatics*, vol. 35, no. 5, pp. 352–359, Oct. 2002.

[14] I. Rish, "An empirical study of the naive Bayes classifier," p. 6, 2001.

[15] H. Dureja, S. Gupta, and A. K. Madan, "Topological Models for Prediction of Pharmacokinetic Parameters of Cephalosporins using Random Forest, Decision Tree and Moving Average Analysis," *Scientia Pharmaceutica*, vol. 76, no. 3, pp. 377–394, Sep. 2008.

[16] S. Tong and E. Chang, "Support vector machine active learning for image retrieval," in *Proceedings of the Ninth ACM International Conference on Multimedia*, ser. MULTIMEDIA '01. New York, NY, USA: Association for Computing Machinery, Oct. 2001, pp. 107–118.

[17] W. Książek, M. Abdar, U. R. Acharya, and P. Pławiak, "A novel machine learning approach for early detection of hepatocellular carcinoma patients," *Cognitive Systems Research*, vol. 54, pp. 116–127, May 2019.

[18] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying Count-Based Exploration and Intrinsic Motivation," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016.

[19] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. Sydney, NSW, Australia: JMLR.org, Aug. 2017, pp. 2721–2730.

[20] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[21] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy Networks For Exploration," in *International Conference on Learning Representations*, 2018.

[22] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter Space Noise for Exploration," in *International Conference on Learning Representations*, 2018.

[23] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep Exploration via Bootstrapped DQN," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016.

[24] N. Nikolov, J. Kirschner, F. Berkenkamp, and A. Krause, "Information-Directed Exploration for Deep Reinforcement Learning," in *International Conference on Learning Representations*, 2018.

[25] N. Chentanez, A. Barto, and S. Singh, "Intrinsically Motivated Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 17. MIT Press, 2004.

[26] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 2778–2787.

[27] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement Learning with Deep Energy-Based Policies," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 1352–1361.

[28] E. Hazan, S. Kakade, K. Singh, and A. V. Soest, "Provably Efficient Maximum Entropy Exploration," in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 2681–2691.

[29] I. Osband, B. V. Roy, D. J. Russo, and Z. Wen, "Deep Exploration via Randomized Value Functions," *Journal of Machine Learning Research*, vol. 20, no. 124, pp. 1–62, 2019.

[30] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh, B. V. Roy, R. Sutton, D. Silver, and H. V. Hasselt, "Behaviour Suite for Reinforcement Learning," in *International Conference on Learning Representations*, Mar. 2020.

[31] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Elsevier, Oct. 2011.

[32] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-Miner: Mining block correlations in storage systems," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, ser. FAST'04. USA: USENIX Association, Mar. 2004, p. 13.

[33] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning Memory Access Patterns," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, Jul. 2018, pp. 1919–1928.

[34] J. A. Gubner, *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, Jun. 2006.

[35] P. Fournier-Viger and J. C.-W. Lin, “A Survey of Sequential Pattern Mining,” *Data Science and Pattern Recognition*, 2017.

[36] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[37] M. Sundermeyer, R. Schlüter, and H. Ney, “LSTM neural networks for language modeling,” in *Interspeech 2012*. ISCA, Sep. 2012, pp. 194–197.

[38] W. Schwarting, J. Alonso-Mora, and D. Rus, “Planning and Decision-Making for Autonomous Vehicles,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018.

[39] D. A. Pomerleau, “ALVINN: An Autonomous Land Vehicle in a Neural Network,” in *Advances in Neural Information Processing Systems*, vol. 1. Morgan-Kaufmann, 1988.

[40] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, “End-to-End Driving Via Conditional Imitation Learning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 4693–4700.

[41] D. Silver, J. A. Bagnell, and A. Stentz, “Learning Autonomous Driving Styles and Maneuvers from Expert Demonstration,” in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, J. P. Desai, G. Dudek, O. Khatib, and V. Kumar, Eds. Heidelberg: Springer International Publishing, 2013, pp. 371–386.

[42] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, “Driving in Dense Traffic with Model-Free Reinforcement Learning,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 5385–5392.

[43] E. Leurent and J. Mercat, “Social Attention for Autonomous Decision-Making in Dense Traffic,” Nov. 2019.

[44] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, “MultiPath: Multiple Probabilistic Anchor Trajectory Hypotheses for Behavior Prediction,” in *Proceedings of the Conference on Robot Learning*. PMLR, May 2020, pp. 86–99.

[45] B. Varadarajan, A. Hefny, A. Srivastava, K. S. Refaat, N. Nayakanti, A. Cornman, K. Chen, B. Douillard, C. P. Lam, D. Anguelov, and B. Sapp, “MultiPath++: Efficient Information Fusion and Trajectory Aggregation for Behavior Prediction,” in *2022 International Conference on Robotics and Automation (ICRA)*, May 2022, pp. 7814–7821.

[46] X. Mo, Z. Huang, Y. Xing, and C. Lv, "Multi-Agent Trajectory Prediction With Heterogeneous Edge-Enhanced Graph Attention Network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 9554–9567, Jul. 2022.

[47] H. Jeon, J. Choi, and D. Kum, "SCALE-Net: Scalable Vehicle Trajectory Prediction Network under Random Number of Interacting Vehicles via Edge-enhanced Graph Convolutional Neural Network," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2095–2102.

[48] J. Li, H. Ma, Z. Zhang, and M. Tomizuka, "Social-WaGDAT: Interaction-aware Trajectory Prediction via Wasserstein Graph Double-Attention Network," Feb. 2020.

[49] X. Li, X. Ying, and M. C. Chuah, "GRIP: Graph-based Interaction-aware Trajectory Prediction," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Oct. 2019, pp. 3960–3966.

[50] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning Lane Graph Representations for Motion Forecasting," in *Computer Vision – ECCV 2020*, ser. Lecture Notes in Computer Science, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 541–556.

[51] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "VectorNet: Encoding HD Maps and Agent Dynamics From Vectorized Representation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, Jun. 2020, pp. 11 522–11 530.

[52] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kummerle, H. Konigshof, C. Stiller, A. de La Fortelle, and M. Tomizuka, "INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps," Sep. 2019.

[53] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, Mar. 1964.

[54] Z. Xi, O. Newton, G. McGowin, G. Sukthankar, S. Fiore, and K. Oden, "Predicting Student Flight Performance with Multimodal Features," in *Social, Cultural, and Behavioral Modeling*, ser. Lecture Notes in Computer Science, R. Thomson, H. Bisgin, C. Dancy, A. Hyder, and M. Hussain, Eds. Cham: Springer International Publishing, 2020, pp. 277–287.

[55] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Heidelberg: Springer, 2006, pp. 282–293.

[56] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, May 1996.

[57] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[59] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016.

[60] M. G. Bellemare, W. Dabney, and R. Munos, "A Distributional Perspective on Reinforcement Learning," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 449–458.

[61] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional Reinforcement Learning With Quantile Regression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018.

[62] C. Sherstan, D. R. Ashley, B. Bennett, K. Young, A. White, M. White, and R. S. Sutton, "Comparing Direct and Indirect Temporal-Difference Methods for Estimating the Variance of the Return," *Proceedings of Uncertainty in Artificial Intelligence*, 2018.

[63] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[64] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013.

[65] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," Jun. 2016.

[66] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. New York, NY, USA: JMLR.org, Jun. 2016, pp. 1995–2003.

[67] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *ICLR (Poster)*, 2015.

[68] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR (Poster)*, 2015.

[69] Y.-C. Lee, J. D. Lee, and L. Ng Boyle, "Visual Attention in Driving: The Effects of Cognitive Load and Visual Disruption," *Human Factors*, vol. 49, no. 4, pp. 721–733, Aug. 2007.

[70] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017.

[71] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[72] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *International Conference on Learning Representations*, Feb. 2018.

[73] S. Brody, U. Alon, and E. Yahav, "How Attentive are Graph Attention Networks?" in *International Conference on Learning Representations*, Sep. 2021.

[74] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. Montreal, Canada: International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 1548–1554.

[75] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," Apr. 2019.

[76] E. Leurent, "An environment for autonomous driving decision-making," 2018.

[77] B. Coifman and L. Li, "A critical evaluation of the Next Generation Simulation (NGSIM) vehicle trajectory dataset," *Transportation Research Part B: Methodological*, vol. 105, pp. 362–377, Nov. 2017.

[78] V. Alexiadis, J. Colyar, J. Halkias, R. Hranac, and G. McHale, "The Next Generation Simulation Program," *Institute of Transportation Engineers. ITE Journal*, vol. 74, no. 8, pp. 22–26, Aug. 2004.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[80] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[81] S. M. Omohundro, *Five Balltree Construction Algorithms*. International Computer Science Institute Berkeley, 1989.

[82] Z. Hu, Y. Dong, K. Wang, and Y. Sun, “Heterogeneous Graph Transformer,” in *Proceedings of The Web Conference 2020*, ser. WWW ’20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 2704–2710.

[83] N. Nayakanti, R. Al-Rfou, A. Zhou, K. Goel, K. S. Refaat, and B. Sapp, “Wayformer: Motion Forecasting via Simple & Efficient Attention Networks,” Jul. 2022.

[84] Z. Xi and G. Sukthankar, “Estimating the Variance of Return Sequences for Exploration,” in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2021, pp. 429–434.