UTILIZATION OF SUPERVISED AND REINFORCEMENT LEARNING IN THE AUTOMATION OF THE CLASSICAL ATARI GAME “PONG”

by

ANDREW J. WATERREUS, B.S.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

COMMITTEE MEMBERS:
Pranav Bhounsule, Ph.D., Chair
Yufei Huang, Ph.D.
Adel Alaeddini, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
August 2019

DEDICATION

This thesis is dedicated to my dear mother Pauline Waterreus. Thank you for all of your support and tolerating me for so long while I worked to fulfill one of my life goals.

ACKNOWLEDGEMENTS

First of all, I would like to thank my family for all the love, support, and encouragement that they have provided me as I worked to complete my postgraduate education and embark on my career, finally no longer a student, to the great excitement of my mother. I would also like to thank my advisor Dr. Pranav Bhounsule for all of the fabulous education and life advice that he has provided, committee members Dr. Yufei Huang and Dr. Adel Alaeddini, and all of my colleagues in the Robotics and Motion Laboratory for all of the support that they have provided me as I worked towards completion of this machine learning project and research. Lastly, I would like to thank the many UTSA staff members that have tolerated and encouraged me throughout the past five years, including Dr. Yesh P. Singh, Dr. Amir Karimi, Dr. James Johnson, and especially Mrs. Cayla Jimenez, who likely doesn’t get enough recognition for how much she does for the mechanical engineering students, and for always being so pretty.

August 2019

UTILIZATION OF SUPERVISED AND REINFORCEMENT LEARNING IN THE AUTOMATION OF THE CLASSICAL ATARI GAME "PONG"

ANDREW J. WATERREUS, M.S.
The University of Texas at San Antonio, 2019

Supervising Professor: Pranav Bhounsule, Ph.D.

The classical "Pong" game resembles 2-player table tennis and was developed by Atari in 1972. In this game, each player controls a small paddle to bounce a ball across the rectangular area to defend a small goal on either side of the arena. Pong became extremely popular and is generally considered to be the first commercially successful video game. It has even earned itself a spot in the Smithsonian Institution in Washington, D.C. because of its level of cultural impact.

This research project was an attempt to automate Pong through the use of two radically different machine learning methods, supervised learning and reinforcement learning. In supervised learning, an expert provides the training data, which consists of example input-output pairs to be used for the learning. In our case, a human controlled one paddle in response to the ball, while the other paddle moved up and down at a defined rate of 2.64 seconds per cycle. Then, an artificial neural network with four layers was trained from this dataset. In reinforcement learning, by using a reward system developed to encourage the paddle to defend the goal by bouncing the ball and receiving points, a controller agent was trained using Deep Q-Neural Networks (DQN). This method allows the computer to teach itself through trial and error.

The supervised learning method generated an automated paddle that was deemed unbeatable by several challengers; while the reinforcement learning method was only capable of producing a controller agent with an average of 3-5 ball bounces per episode.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE: INTRODUCTION

1.1 Thesis Contribution

This Thesis offers multiple benefits in its completion; with the automation of the classic Atari game Pong offering experience in the use of machine learning, and the product of the project being able to be used in future research projects. With the development of supervised and reinforcement learning Pong programs, by the utilization of both Matlab and Python IDEs, software development abilities were significantly improved for future utilization. Initially, in the procedure of completing this Thesis, supervised learning is used with labeled data of the state of the game system to train a fully connected Deep Neural Network (DNN) to make predictions on the movement of the paddle at each time step. This system was developed using MATLAB_R2018b and a version of the Pong game downloaded from the Internet. Both a computer vs. computer system, and a computer vs. human system were developed. This supervised learning method is considered to be a form of *pattern-based detection* because it is trained on data generated by volunteer human players, with random time steps having the current state and paddle movement label recorded. In order to effectively train the paddles, only data recorded from skilled players was saved to the data files.

Second, a reinforcement learning agent is trained using both Jupyter Notebook and PyCharm. It was decided to switch from Matlab to a Python IDE while training the reinforcement learning agent due to the availability of large machine learning libraries such as TensorFlow and Keras. For this system a Pong environment needed to be developed from scratch in order to allow the agent to effectively train and make accurate predictions of paddle movements based on the current state of the game (e.g. paddle location, ball location, and ball movement). This second approach is considered to be *approximate dynamic programming*, as the development of the DNN is performed through a trail and error method on the part of the program, and no labeled data needs to be supplied during training.

An additional contribution due to the project is that the resulting program files have been uploaded to GitHub.com, through the link, https://github.com/AJWater/Automated-Pong in order to make the programs freely available to anyone interested in attempting future research on this subject.

1.2 Thesis Organization

In the following chapters a thorough description of this Thesis project is given, with the structure being developed to provide clarity for each subsequent chapter. Chapter Two gives a quick review about the development of Pong and any modifications made in the versions utilized in this project. In this section the overall layout of the dataset and the game state utilized during training are thoroughly described. Chapter Two also reviews the preliminary knowledge in the field of machine learning as well as the subfield of deep learning, and how it is effective in performing the automation of video games such as Pong and many others. Chapter Three provides more thorough descriptions of the methods utilized throughout the course of this project, as well as the experiments used to evaluate the effectiveness of the learned playstyles developed during training sessions. Chapter Four presents the results acquired during the experiments. Chapter Five reviews the results and evaluates the data acquired through both machine learning techniques. This Thesis is concluded in Chapter Six with a description on the use of these machine learning methods in video game automation, a review of the insights gained throughout the completion of this project, and recommendations for future work to be performed in order to further improve the trained Pong playing agents.

CHAPTER TWO: BACKGROUND AND LITERATURE REVIEW

2.1 Introduction

This chapter quick reviews the history of the video game Pong developed by Atari, as well as how the versions used in the completion of this project have small modifications due to the programming being different from the original. Here, the overall layout of the datasets developed and utilized during training is also thoroughly described. In addition, preliminary knowledge of machine learning and the subfield of deep learning is introduced, as well as its effectiveness in video game automation.

2.2 Pong

Pong is a two-dimensional sports game that simulates table tennis developed by Atari engineer Allan Alcorn as a training exercise by Atari co-founder Nolan Bushnell in 1972. Surprised by the quality of his work, Bushnell and another Atari co-founder Ted Dabney decided to manufacture the game and sell it. Pong became the first commercially successful video game and is now even a permanent part of the collection at the Smithsonian Institution in Washington, D.C. because of its level of cultural impact.

Figure 1: Pong Arcade Video Game Console

2.2.1 Pong Layout

Pong was originally developed to simulate table tennis; therefore, it consists of two paddles and a ball, with the paddles bouncing the ball back and forth and trying to get it beyond the other paddle. The current score of each paddle is displayed on its side of the game screen. Evaluating the game layout, it can be determined that an effective description of the current state of the game came be made by just the (x,y) coordinates of the ball, current speed of the ball, x and y components of velocity of the ball, and the y-coordinate of each paddle. This is because these are the only components of the game that change while playing. This is commonly referred to as the, “State-space,” of the data, and the, “Label,” of the data is the decision of how the paddle should be moved based on each data set. The possible labels include up, down, or stay still. It was attempted to keep this constant throughout the project, even if the Pong game’s programming differed slightly from the original.

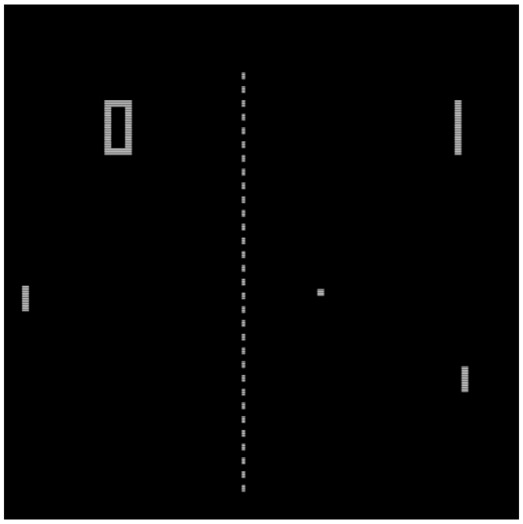


Figure 2: Original Pong Screen Layout

2.2.2 Matlab Supervised Learning Pong Layout

The version of Pong utilized for supervised learning was developed for Matlab and was downloaded from the Internet. The game is titled as, “DAVE’S MATLAB PONG v0.3,” and was originally developed by a person named David Buckingham. It states that it is, “a fast-paced two-player game inspired by Atari’s Pong,” although it does have some modifications, such as aesthetics and side walls added to make the goals smaller. Even though there are some modifications, the overall state space of the game at any time step can still effectively be described with the same information as the original Pong, due to only the same components changing during game play.

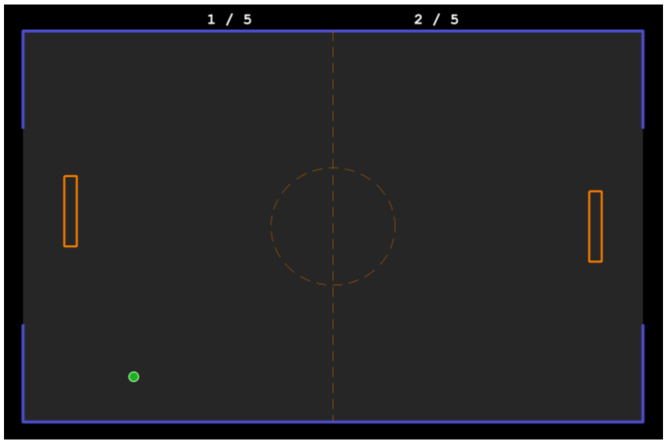


Figure 3: Matlab Supervised Learning Pong Screen Layout

2.2.3 Python Reinforcement Learning Pong Layout

For a period of time, the Matlab version of Pong was attempted to be utilized for the reinforcement learning section of this Thesis project, as the game worked effectively and the program was well understood by this point, but for reinforcement learning, large changes would need to be made in order to save the current game state and then determine the appropriate reward to give to the agent based on that state. Essentially a new game program would need to be made, but luckily in a course, EE 5453 Engr Programming II, several semesters earlier, a Pong game had been developed from scratch that utilized Python as the programming language. Because the game program was entirely developed by me, it was completely understood how it worked and using Python opened the availability of using TensorFlow and Keras to further simplify machine learning endeavors. The Python Pong program was largely developed to operate like the Matlab program in order to keep the results from each of the machine learning techniques comparable. The state space is the same, with the same number of components being the only things that change during game play.

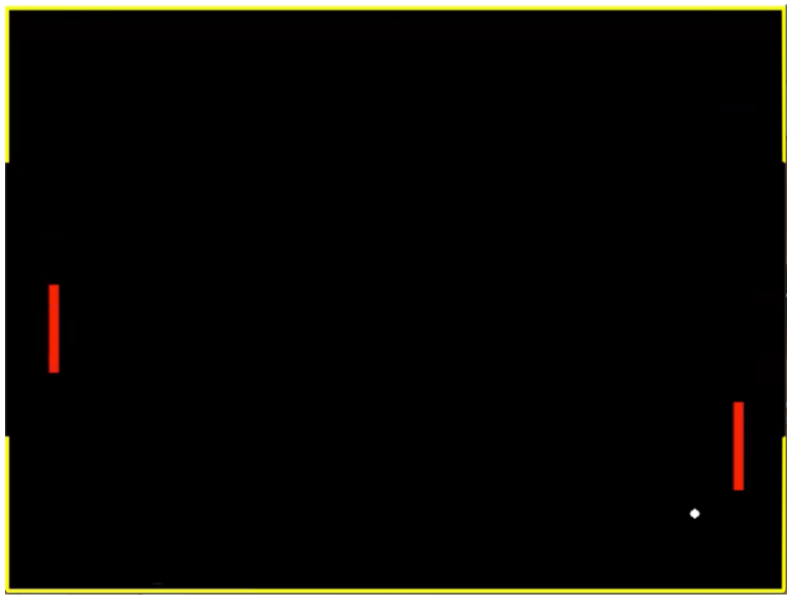


Figure 4: Python Reinforcement Learning Pong Screen Layout

2.3 Machine Learning Preliminaries

Both methods utilized in this project, supervised learning and reinforcement learning, were performed through the use of deep neural networks, which are commonly classified as deep learning, a subfield of machine learning. Therefore, a background on artificial intelligence and machine learning concepts will be reviewed, and a framework for each method of deep learning utilized will be discussed.

2.3.1 Artificial Intelligence and Machine Learning

The field of Artificial Intelligence (AI) was born in the 1950s when computer scientists set out to determine if computers could “think” like humans. Alan Turing, a young British polymath devised a mathematical description of how humans use information in order to solve problems and make decisions, and thus set out to allow computers or machines to do the same. The challenge was the computers of the 1950s not being able to store commands, and the cost of using one being very expensive. In 1956, what is generally considered to be the first AI was developed by the Research and Development (RAND) Corporation and presented at the Dartmouth Summer Research Project on Artificial Intelligence (DSRPAI) conference. Although the conference fell short of initial expectations, this event is generally considered to have catalyzed the next twenty years of AI research. For the next two decades, AI thrived due to improvements with computers and additional funding from the government. The Defense Advanced Research Projects Agency (DARPA) was particularly interested in a machine that could transcribe and translate spoken language. It was soon after this point where the obstacles of AI were truly revealed, primarily in the lack of computational power for performing substantial tasks. In order to develop a computer to understand language and its complexity, it was determined that computers were still millions of times too weak. Funding from most governments dwindled at this time, and it would be until the 1990s and 2000s when the landmark goals of AI would finally be achieved. In 1997 the revolutionary match between IBM's Deep Blue and chess grand master Gary Kasparov occurred, with Deep Blue defeating the chess world champion. In the same year, speech recognition software was developed and implemented on Windows. These advancements are generally considered to have been achievable due to the fundamental limit of computer storage had caught up to humanity's requirements. We now live in the age of "big data," in which we have levels of stored data too cumbersome for humans to process. Now AI has grown to the level of being everywhere, with many industries finding it quite fruitful, such as technology, banking, and entertainment. Soon (generally believed to be twenty years), we can expect to see driverless vehicles on the roads, machines that surpass human intelligence, and even the possibility of sentient robots requiring us to discuss the topic of machine policies and ethics.

2.3.2 Machine Learning and Deep Learning

The birth of machine learning came about in 1958 with the development of what is known as the Perceptron, a mathematical model of how the neurons in our brains operate. The Perceptron would receive a group of signals like the dendrites of a neuron, and after processing the inputs, the Perceptron would output a 0 or a 1, representing the neuron either firing or not out through the axon. This allowed for binary classification, and with groups of Perceptrons being operated together, they could learn to become more accurate with more training examples. This will be more clearly defined in the following chapter. An image of the first trainable Perceptron built with hardware can be seen below in Figure 5. A significant amount of progress was made with grouping multiple Perceptrons in a layer, as seen below in Figure 6, as this offered additional outputs, but a short-coming of the Perceptron was identified when additional layers were attempted to be added in order to make an artificial neural network. The method by which

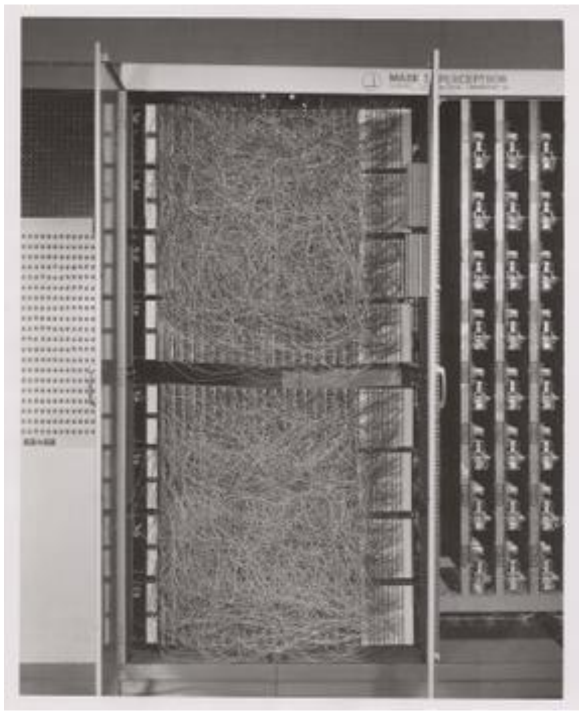


Figure 5: Mark I Perceptron at the Cornell Aeronautical Laboratory

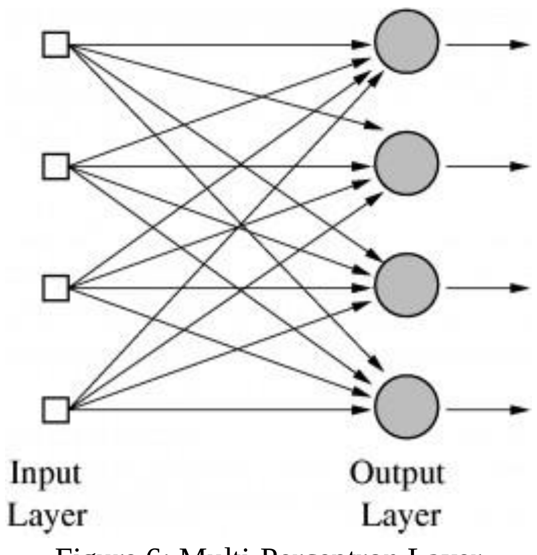


Figure 6: Multi-Perceptron Layer

a Perceptron is trained only allows for an input and an output layer due to there being no method to determine by how much the hidden layer weights impacted the outputs. It was not until 1986 where the solution, known as, “Learning representations by back-propagating errors,” by David Rumelhart, Geoffrey Hinton, and Ronald Williams became popularized. Now multiple hidden layers could be added networks and extremely powerful “deep” networks could perform amazingly powerful achievements.

CHAPTER THREE: METHODS

3.1 Introduction

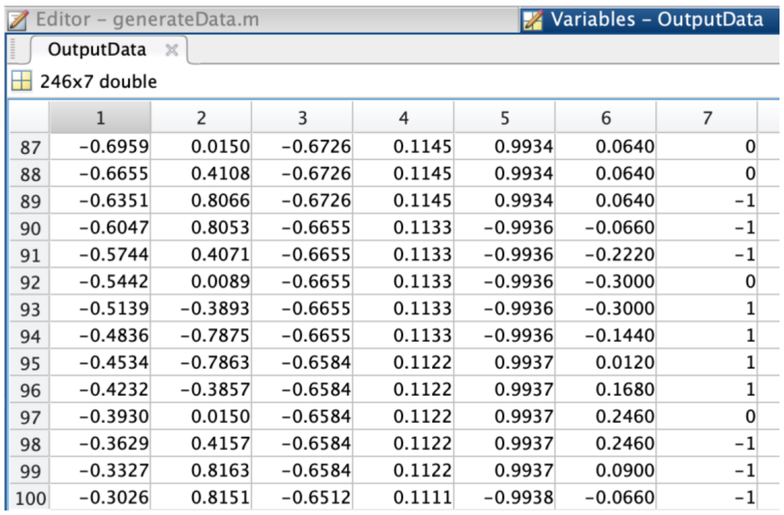
This chapter looks at the two techniques used in this project as approaches for automating Pong. First, the method of utilizing supervised learning will be reviewed, and then it will be evaluated how well the method was followed. Second, the method of reinforcement learning will be analyzed, and its effectiveness in automating Pong will be determined.

3.2 Supervised Learning

The majority of practical machine learning consists of supervised learning; so, it should not be a surprise that one method utilized in this project consists of its use. The “Machine Learning Stanford University” course by Andrew Ng completed on coursera.org was a significant assistant at this point in the project, as it had a section that focused on supervised learning and neural networks being utilized in its fulfillment. The course was completed using Matlab, and with it being the programming language best known at the time, it was decided to utilize Matlab in this portion of the project. A working Matlab program of the Pong video game developed by David Buckingham was downloaded from the Internet and then modified so that the state space could be acquired at each time step, as well as that any data inputs and outputs could be recorded to be utilized by machine learning program files. After evaluation, it was determined that each paddle would need to be trained separately, and that each paddle did not need to record the current location of the other paddle at each time step, as it did not play a significant role in an effective play style. By doing this, the state space could be reduced to only six input variables: ball x-coordinate, ball y-coordinate, ball speed, ball velocity x-component, ball velocity y-component, and the y-coordinate of the paddle being trained. Due to a lack in processing power on the computer being used, it was considered to reduce the inputs further by combining the ball velocity components into a Theta variable to represent the direction that the ball was traveling in, but this was determined unwise, as significant further modifications would need to be made to the game file in order to change it to utilize a Theta value, and having the two velocity components would actually be useful in the development of the classifier during training.

3.2.1 Supervised Learning Data Generation

One of the characteristics of supervised learning is that the dataset provides the labels for each state space sample. Due to this, a dataset needed to be generated, prior to training, that labeled each state sample as a game state where the human player either moved up, stayed still, or moved down. A sample of a normalized dataset can be seen below in Figure 7. Columns 1 through 6 represent samples of the state space, and the 7th column represents the label based on the current state. Each row represents a separate sample to be utilized in training. A dataset for



Editor – generateData.m | Variables – OutputData

OutputData

246x7 double

	1	2	3	4	5	6	7
87	-0.6959	0.0150	-0.6726	0.1145	0.9934	0.0640	0
88	-0.6655	0.4108	-0.6726	0.1145	0.9934	0.0640	0
89	-0.6351	0.8066	-0.6726	0.1145	0.9934	0.0640	-1
90	-0.6047	0.8053	-0.6655	0.1133	-0.9936	-0.0660	-1
91	-0.5744	0.4071	-0.6655	0.1133	-0.9936	-0.2220	-1
92	-0.5442	0.0089	-0.6655	0.1133	-0.9936	-0.3000	0
93	-0.5139	-0.3893	-0.6655	0.1133	-0.9936	-0.3000	1
94	-0.4836	-0.7875	-0.6655	0.1133	-0.9936	-0.1440	1
95	-0.4534	-0.7863	-0.6584	0.1122	0.9937	0.0120	1
96	-0.4232	-0.3857	-0.6584	0.1122	0.9937	0.1680	1
97	-0.3930	0.0150	-0.6584	0.1122	0.9937	0.2460	0
98	-0.3629	0.4157	-0.6584	0.1122	0.9937	0.2460	-1
99	-0.3327	0.8163	-0.6584	0.1122	0.9937	0.0900	-1
100	-0.3026	0.8151	-0.6512	0.1111	-0.9938	-0.0660	-1

Figure 7: Dataset Sample for Supervised Learning

training each paddle would need to be generated, as each paddle would be utilizing a different trained agent for a controller.

In generating the data, a human controlled one paddle while the other paddle was automated to move up and down at a rate of 2.64 seconds per cycle, which allowed the automated paddle to move over the majority of its side of the game screen. This would provide additional randomness in the data collected, compared to if the paddle were to just stay still. A barrier was then placed over the goal for the automated paddle. With this added feature, the only way that the data collection episode could end would be if the ball were to get past the human controlled paddle. This would allow for longer episodes with more data being collected per game, and the paddle would be trained to play defensively. During the data generation phase, the five features defining the ball, the vertical position of the human controlled paddle, and the action chosen by the human were recorded. In order to prevent the training data from being composed of sequential time step data, the data was recorded only from each fifth time step. It required approximately 60-70 games to be played in order to develop the desired 40 thousand training data examples, with an added feature of being able to exclude training data from games where the human was determined to have played poorly. Next a separate test dataset, for the same paddle, consisting of approximately 3 thousand data examples, was generated to be used to evaluate the effectiveness of the trained neural network. Once completed, this entire process was then repeated, with the human controlling the opposite paddle.

3.2.2 Neural Network Development

After the training and test datasets were generated for both paddles, the next step is to develop an effective Neural Network to be trained to match our input data to the labeled output data. A simple example of a neural network can be seen below in Figure 8. In this example, the x's represent the input signals, the w's represent the weights, the δ's represent the summation of all of the signals being input to the next node, the a's being the δ's after the activation function, and the y's represent the output predicted by the network.

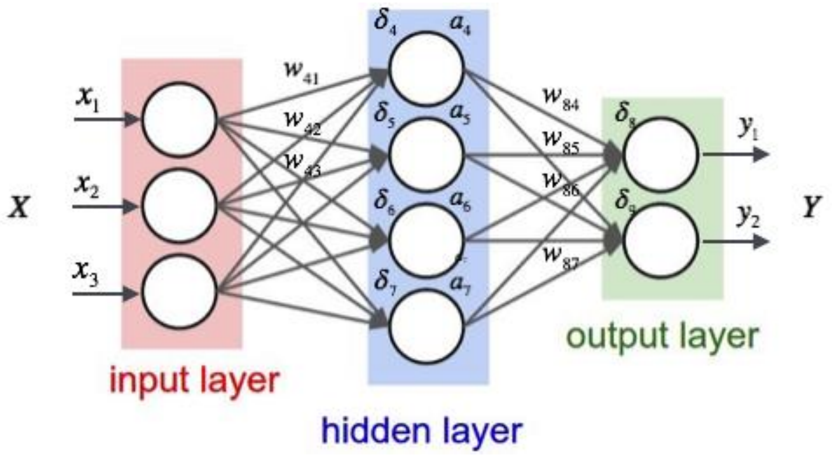


Figure 8: Example Neural Network

Again, the machine learning course from coursera.org was extremely helpful, as it provided an example of developing a neural network, as well as the extensive Matlab programs required to train it. The course's original exercise utilized a network consisting of three layers, but for predicting the actions for Pong, an additional hidden layer was added, making our network one of four layers. A significant amount of modifications needed to be made in order to have the program utilize this new layer correctly, as it had originally been developed to have one hidden layer. Our network had 6 inputs values, 6 neurons in the first hidden layer, 5 neurons in the second hidden layer, and then 3 actions in the output layer.

3.2.2 Neural Network Training

After the network model is developed, the weights need to be initialized and then the technique known as gradient decent is used to optimize the weights to provide the network that most accurately predicts the correct output for a given input.

For gradient decent in optimizing a neural network, a cost function needs to be chosen in order to define the network as a function with respect to the error between the predictions and the target value. For Pong, the Mean Squared Error (MSE) function was chosen.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{3.1}$$

Where

- **n** is the total number of training examples.
- **i** is the current training example being evaluated.
- **y**$_i$ is the predicted output value from the network.
- $\hat{\mathbf{y}}_\mathbf{i}$ is the target value as defined by the training data.

The cost function is often indicated by $J(\theta_{(i)})$ with $\theta_{(i)}$ indicating the value of the "i"th weight. This can be seen below in Figure 9. Next, for gradient decent, the technique known as forward propagation is performed. Forward propagation will take the input values for each example and determine the output that the neural network predicts for each. This will indicate our current location on the chart as shown by Figure 9. While performing this action, the values of the signals in each neuron are recorded, both before and after the activation function, which can be seen in Figure 10. Next, back propagation is performed on the network in order to calculate and record the partial derivatives for the cost function with respect to each weight in the network. An example of this can be seen in Figure 11 below. In calculating the partial derivatives, the chain rule is important in order to work backwards from the cost function to the weights. Figure 11 has an example that shows how to calculate the partial derivative of the cost function with respect to w$_1$.

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1} \qquad (3.2)$$

Calculating the partial derivatives will allow us to understand how each weight effects the final cost function and to know how the weights should be modified in order to reduce the cost function to a minimum.

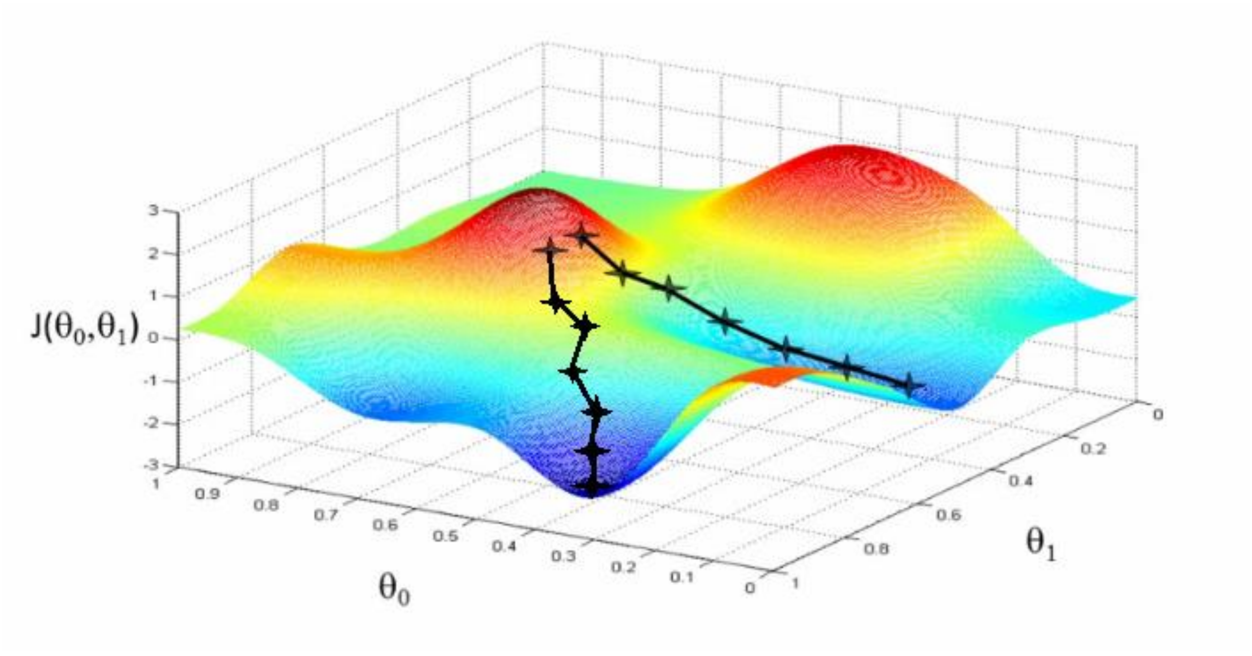


Figure 9: Gradient Decent Example

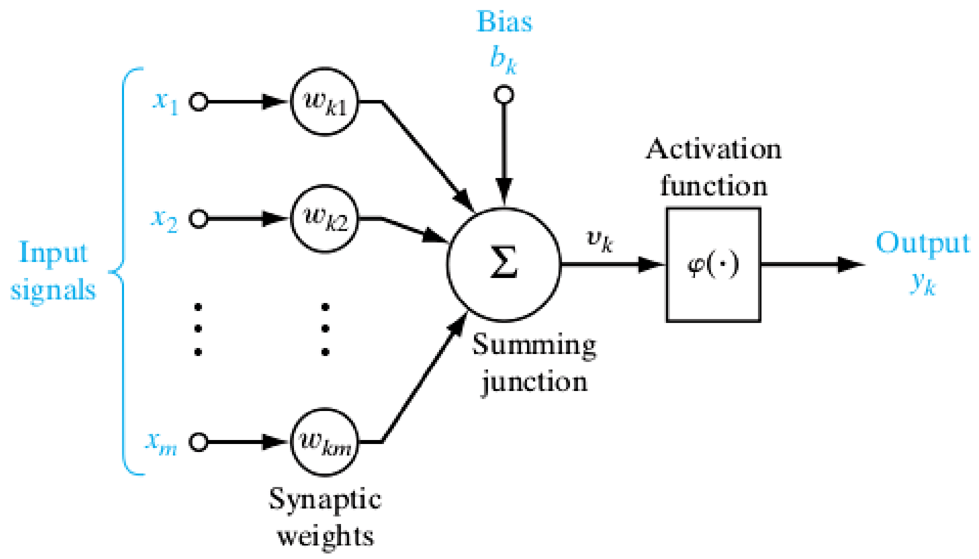


Figure 10: Forward Propagation Example

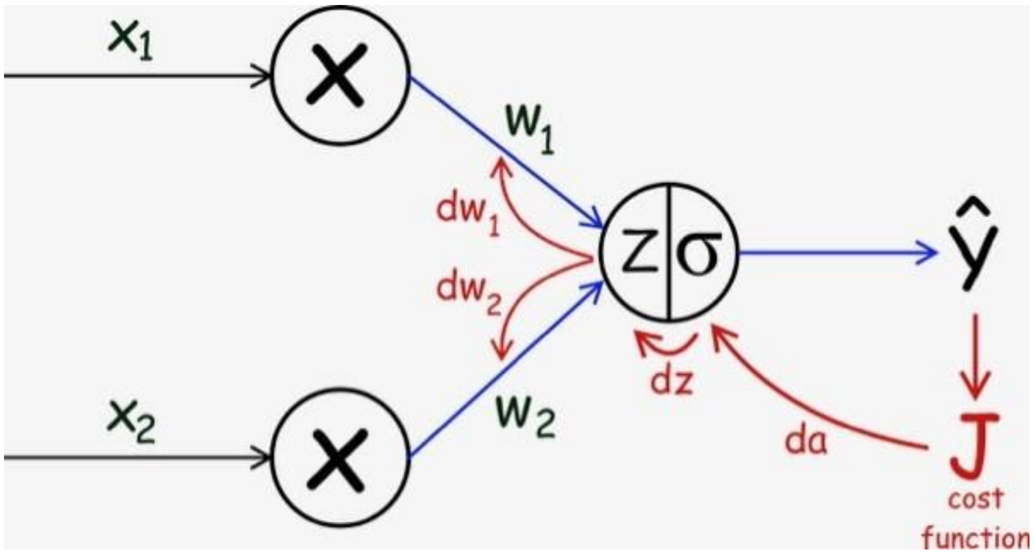


Figure 11: Back Propagation Example

3.2.3 Neural Network Testing

After the neural network has been trained by using the training dataset, the trained network is evaluated by using the test dataset. This would be performed by recording the predicted output values from the network after each test example is run through it. These recorded values are then compared to the test examples' real-world labels, with an accuracy then being calculated and recorded. This helps to determine if during the training phase, the gradient decent converged to a local minimum, or if in a more ideal minimum. By performing multiple trials of the training phase and testing each, the most optimal trained network can be utilized.

3.3 Reinforcement Learning

Reinforcement learning is a very powerful form of machine learning, with the automation of games being one area where it excels at. Google's DeepMind AI has developed an artificial neural network that can play multiple Atari games, has defeated the world champions for both Chess and Go, and has even learned how to walk in simulated environments without even being shown what walking looks like. All of this has been done by using reinforcement learning, and through trial and error training. Reinforcement learning is generally achieved through the training of an agent to choose actions to perform while operating in an environment, in order to earn the greatest reward. The agent will perform an action and observe the effect it had on its environment, as well as the reward it obtained by choosing that action at its original state. This is performed in a continuous loop, or for a predefined number of episodes, in order for the agent to improve from its previous experiences and rewards. This can be seen in Figure 12 below.

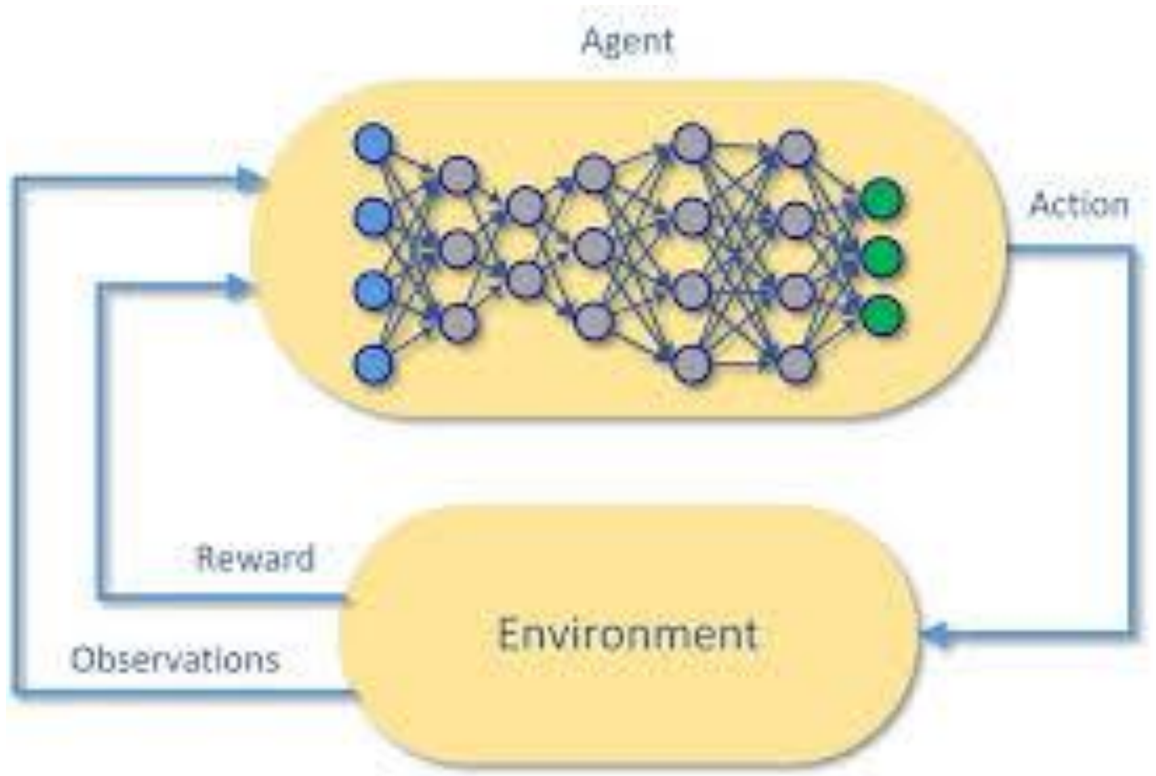
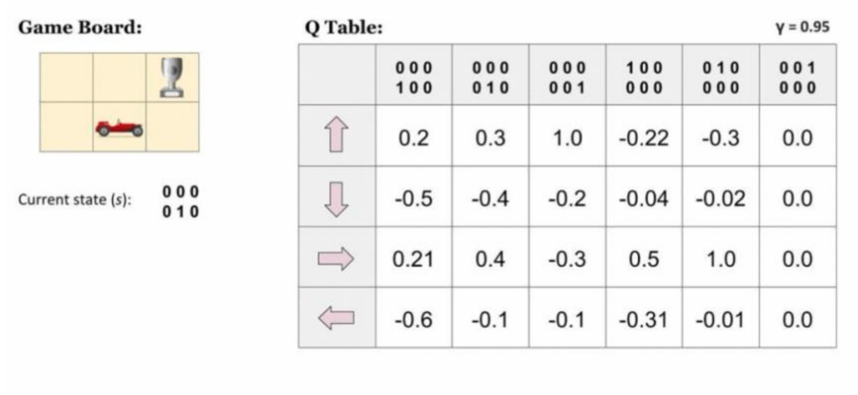


Figure 12: Reinforcement Learning Model

3.3.1 Standard Q-Learning

One algorithm of reinforcement learning is known as Q-learning and utilizes what is known as a Q-table to represent the agent and choose which actions to play while in its current state. An example of a Q-table is visualized for clarity in Figure 13 below, with the Q-table being on the right, and the game environment being on the left.


Game Board:
Current state (s): 000 010
Q Table:
γ = 0.95

	000 100	000 010	000 001	100 000	010 000	001 000
⇧	0.2	0.3	1.0	-0.22	-0.3	0.0
⇩	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
⇨	0.21	0.4	-0.3	0.5	1.0	0.0
⇦	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Figure 13: Q-table and Game Environment Example

The rows indicate the possible actions (a), while the columns represent all possible states (s) that the game board can be in. Because the states only indicate where the race car is, there is only six possible states. This means that there are only 24 possible (s,a) pairs, and 24 Q(s,a) values which are indicated by the numerical values. For Q-learning, the Q(s,a) values are initialized, and the agent acts in one of two ways, either the action depicted by the Q-table's highest Q(s,a) value for the current state, or a random action is chosen in order to encourage exploration of all (s,a) pairs over time for ideal training. The action that was chosen is taken, and the next state (s') and the reward (r) are observed. In order to update the Q-table, the previous state's Q(s,a) value is updated by using the Bellman equation:

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q(s',a') - Q(s,a)] \tag{3.3}$$

Where

- **Q(s,a)** is the current state-action pair's Q-value.
- $\boldsymbol{\alpha}$ is the learning rate, or how much you accept the new value versus the old value.
- **R(s,a)** is the reward obtained after completing a certain action at a certain state.
- $\boldsymbol{\gamma}$ is the discount factor, or how the immediate and future rewards are balanced.
- **maxQ(s',a')** is the maximum Q(s,a) value in the state that the agent moves to after the action chosen in the original state.

The Bellman equation will update one Q(s,a) value per reinforcement learning loop; so this method can be computationally expensive for large Q-tables. With enough training, the Q-table will reach the optimal Q-values, or what is known as Q*(s,a). For Pong, there are three actions per state, but due to the number of possible states when evaluating six input features, the number of possible (s,a) pairs is enormous, particularly due to most of the features not being restricted to integer values.

3.3.2 Deep Q Neural Networks (DQN)

With the scale of our Q-table being too large, a technique that can be utilized is known as Deep Q Neural Networks (DQN) which functions very similarly to Q-learning. The primary difference is that the Q-table is replaced with a Deep Neural Network (DNN) which will attempt to approximate the Q-table through, "Non-linear function approximation." For this method, as seen in Figure 14 below, the DQN now represents the agent, it only evaluates the current state features as inputs, and the outputs are the approximations for the Q(s,a) values for each action available at the original state.

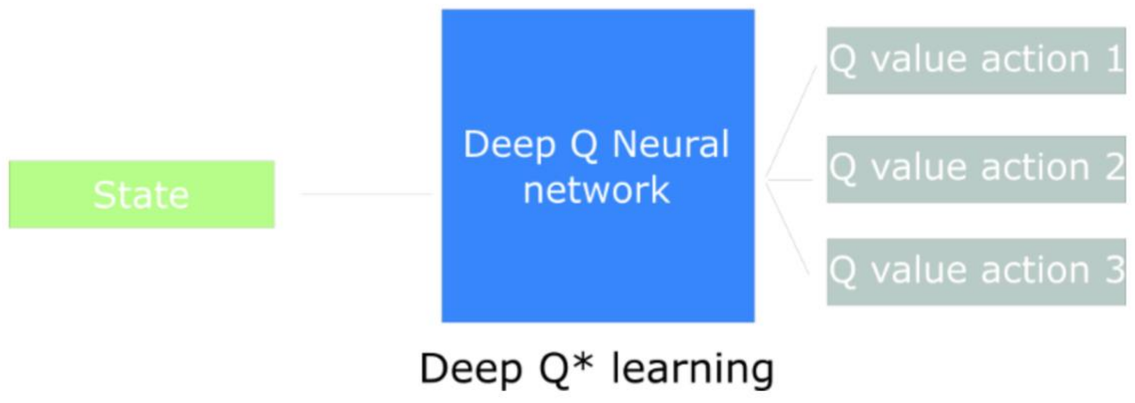


Figure 14: Deep Q Neural Network Diagram

In Q-learning we want to update Q(s,a) values during each iteration of the reinforcement learning loop, but for DQNs we want to update the weights (w) of the deep neural network. This is done with a modification to the Bellman equation (3.3).

$$\Delta w = \alpha[\left(R + \gamma maxQ(s', a, w)\right) - Q(s, a, w)]\nabla_w Q(s, a, w) \tag{3.4}$$

Where

- $\boldsymbol{\Delta w}$ is the change in weights
- $\boldsymbol{\alpha}$ is the learning rate
- **R** is the reward obtained after completing a certain action at a certain state.
- $\boldsymbol{\gamma}$ is the discount factor, or how the immediate and future rewards are balanced.
- $\boldsymbol{maxQ(s', a, w)}$ is the maximum possible value from the next state with the current weights.
- $\boldsymbol{Q(s, a, w)}$ is the current prediction of the Q-value with the current weights.
- $\boldsymbol{\nabla_w Q(s, a, w)}$ is the gradient of our current predicted Q-value.
- $\boldsymbol{\left(R + \gamma maxQ(s', a, w)\right)}$ represents the maximum possible Q-value for the next state

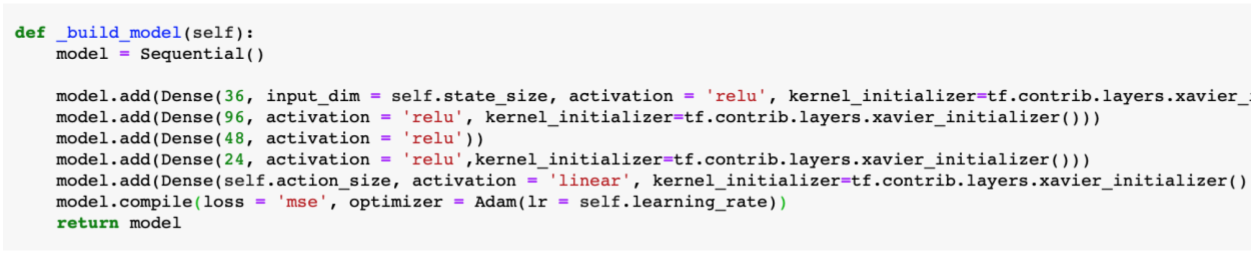
This will allow for the calculation of how to modify the weights in the neural network in order to get the current prediction of the Q-value to have a smaller difference between the maximum Q-value of the next state. With enough training, the DQN will reach a state of very closely approximating the Q*(s,a) values of a Q-table.

3.3.3 DQN Training

The training of a DQN is similar to the optimization of a Q-table in that first it needs to be developed and initialized, and then through trial and error it will learn the optimal method to obtain higher rewards. In developing the DQN, a deep neural network is modeled for the proper number of inputs and outputs, and the weights are given initial values as in the development of a neural network. Next, the environment needs to have a reward function or system developed for it that will assist the training agent in learning the proper actions to perform for specific states.

3.3.3.1 DQN Modeling

For modeling the Pong DQN, it was very useful to have developed a Pong environment using Python IDE which was similar to the Matlab version, so that the machine learning and neural network libraries known as TensorFlow and Keras could be utilized. These libraries are developed for fast experimentation with neural networks and focus on being user friendly. Unlike how much modification it took to introduce just one additional hidden layer to the network in the Matlab code from the supervised learning section of the machine learning coursera.org online course; to introduce any additional layers when using these python libraries only takes one line of code as seen in Figure 15 below, which represents the modeling of an entire deep neural network.



```
def _build_model(self):
    model = Sequential()

    model.add(Dense(36, input_dim = self.state_size, activation = 'relu', kernel_initializer=tf.contrib.layers.xavier_
    model.add(Dense(96, activation = 'relu', kernel_initializer=tf.contrib.layers.xavier_initializer()))
    model.add(Dense(48, activation = 'relu'))
    model.add(Dense(24, activation = 'relu',kernel_initializer=tf.contrib.layers.xavier_initializer()))
    model.add(Dense(self.action_size, activation = 'linear', kernel_initializer=tf.contrib.layers.xavier_initializer()
    model.compile(loss = 'mse', optimizer = Adam(lr = self.learning_rate))
    return model
```

Figure 15: TensorFlow and Keras Neural Network Modeling

These libraries even allow for the easy modification of layer size, activation function, initializer, cost function, and much more.

3.3.3.2 DQN Reward Function

Developing a reward function to assist the training agent in learning the proper actions to perform for specific states was one of the most challenging parts of this section of the project, as it needed to model the environment and the positive and negative actions that could occur for the training agent. Due to the multiple features being evaluated, and the multitude of state-action pairs that were available, the reward function quickly became very complex. Although the goal was primarily to train the paddles to defend the goal from the ball, events other than bouncing the ball needed to be accounted for during training. Through multiple trials, the reward system that achieved the best results was:

- +.1 points per time step (for playing longer) unless a different event occurred
- -2 points for bumping against a wall
- +.5 points per time step if in front of its goal
- +50 points for bouncing the ball
- -(0-25) points if it did not defend its goal from the ball. This was a function based on where the defending paddle was located compared to the ball when the goal was made. Developed to encourage the paddle to defend the goal from the ball.
- +(0-10) points if the paddle scored a goal. This was a function based on where the scoring paddle was, relative to its goal, when the goal was scored. Developed to encourage the goal to return to the center when the ball is on the other side of the game screen.

This reward function encourages the paddle to play defensively, reward it for staying alive longer, and to give reward points for scoring goals if it returns to the optimal location of the middle when the ball is on the other side of the game screen.

3.3.3 DQN Testing

The trained DQN and the over-all reinforcement learning method can be evaluated by plotting the total reward point number obtained by the agent per training episode. This will indicate if the agent is improving with respect to time spent training, as the plotted line will increase and should eventually converge to a higher value if it has enough time, data, and computer processing power. The DQN weights are also recorded for each 100 episodes, so that if a more ideal set of weights is obtained at an earlier episode, they can be used, rather than only the last episodes set.

CHAPTER FOUR: RESULTS

4.1 Introduction

This chapter looks at the results obtained in the completion of the two methods described in the previous chapter and displays them in an easily understandable fashion. In the completion of both methods, interesting results were obtained for both the supervised and the reinforcement learning portions of the project. The results from each method are significantly different in how they indicate the capabilities of the automated Pong game paddles, and the differences in the game program files make the comparison of the methods challenging

4.2 Supervised Learning Results

In supervised learning, the accuracy of the trained agent in making the same choice of action for a given input state as the provider of the test dataset, is how the effectiveness of the agent is generally evaluated. The results for the accuracy of each paddle, when tested on both the training and the test datasets can be seen below in Table 1.

Table 1: Supervised Learning Training and Test Accuracy

Paddle	**Training Data Accuracy (%)**	**Test Data Accuracy (%)**
Left Paddle	68.41%	61.84%
Right Paddle	82.16%	72.82%

4.3 Reinforcement Learning Results

In deep Q neural network reinforcement learning, the reward points per training episode plots are generally used to indicate the improvement of the training agent from its initial state. If a significant number of episodes are played, it is convenient to use a running average function in order to generate a smoother plot line, compared to indicating the reward point value achieved for many thousands of episodes. Due to the significant probability of randomness in the training phase of reinforcement learning, with several random number generators being used, the plots may take many thousand episodes to converge on a close approximation of Q*(s,a), or an optimal operation technique for the agent in its environment. Below, you can see the resulting plots for the left paddle in figure 16. The left plot represents the initial training of the left paddle's DQN, where the weights at episode 3500 are used. In the right plot, additional training is performed, with the weights from 1500 being used afterwards. The large spike values at the beginning of the plots is due to the running average function not having the required number of data points yet, and this causing errors in the values that are output to the plots. So, the peaks at the beginning of each plot are ignored for each case.

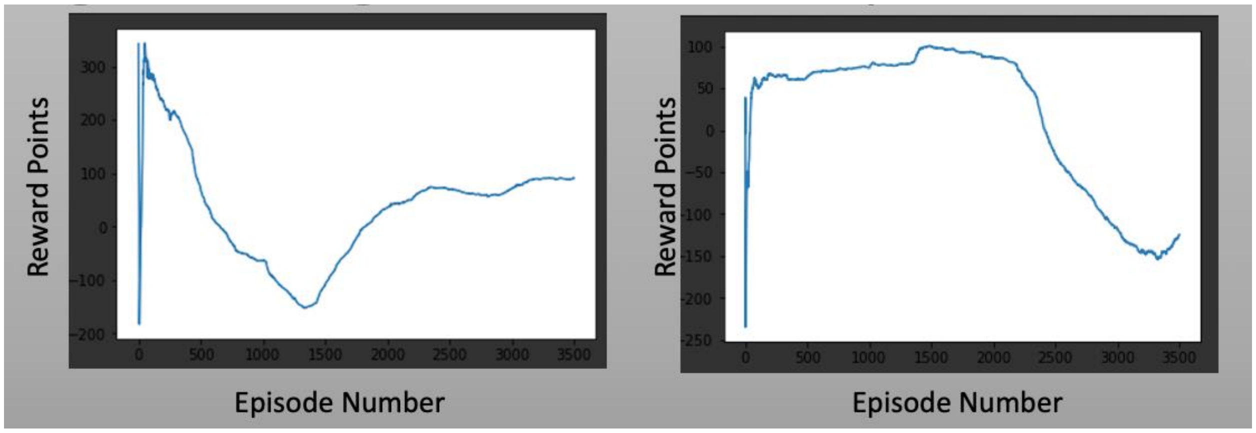


Figure 16: Left Paddle DQN Running Average Training Plots

The plots for the right paddle can be seen in Figure 17 below, with the left plot representing the initial training, and the right plot representing the subsequent training. For the left plot, the weights from episode 1700 were used for the subsequent training, and the weights from episode 4000 of the right plot were utilized in the DQN to control the right paddle.

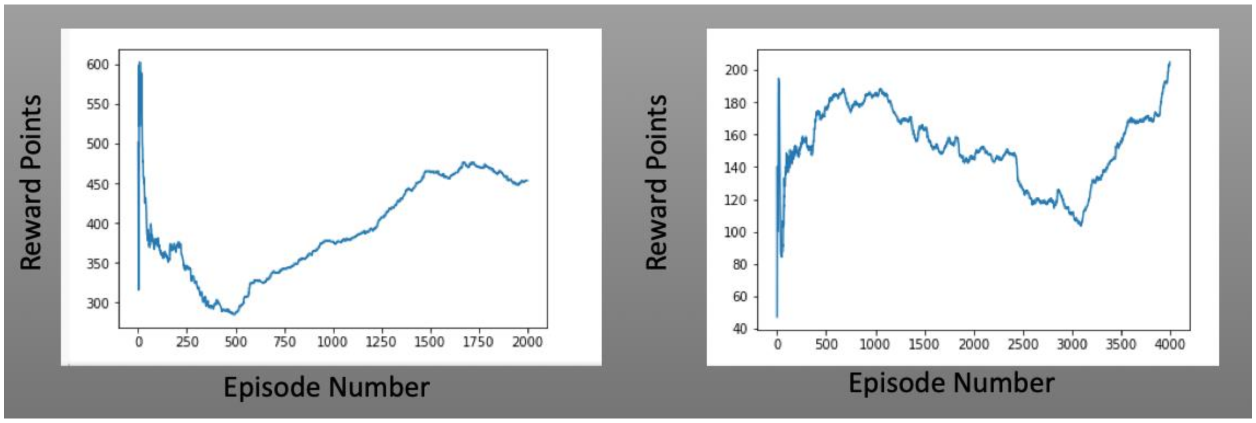


Figure 17: Right Paddle DQN Running Average Training Plots

CHAPTER FIVE: DISCUSSION

5.1 Introduction

This chapter discusses the results displayed in chapter four and any unique details that were found in their evaluation. The results obtained in both methods of this project had interesting features that quickly attract the attention of most observers and experienced machine learning specialists. These features could be the subject of future work or projects, as both methods could potentially be improved.

5.2 Features of Supervised Learning Results

The first noticeable interesting feature is the significant difference in the accuracies between the left and right paddles during the supervised learning portion of the project. Even though both paddles had approximately equal training and test datasets, the neural networks converged to a minimum with a significant difference in accuracy and playstyle. Although the left paddle had 61.84% accuracy on the test dataset, and the right paddle had 72.82% accuracy, the left paddle was considered significantly more challenging to defeat in a human vs. computer match, with some challengers deeming it, “unbeatable,”. The left paddle would play completely defensively and only move away from the middle of its goal for short periods of time. It was also very skilled at predicting where it needed to be in order to intercept the ball from a significant distance away. The right paddle on the other hand would move in short bursts towards the ball when the ball was on its half of the game screen, and then stay stationary for extended periods of time. The largest difference between the two paddles was that the left paddle would return to the middle before staying stationary, and the right paddle would stay stationary as soon as the ball began moving towards the other side of the screen. A potential cause of this uniqueness and large difference in the accuracies, is that the right paddle’s training dataset may have a higher percentage of samples where the training label was, “0,” or the action was to stay still. For a human player, an effective Pong playstyle is to return to the center after bouncing the ball and wait for the ball to start to come back, as this provides the shortest distance needed to move in order to defend the entire goal. In doing this though, the player will not move for extended periods of time compared to a player that does not return to the middle, who will need to attempt to predict the ball’s future location much sooner and begin to move at an earlier time. If the samples where the left paddle stayed still are primarily when it is in the middle, it could be trained to associate those to features with each other, but if the right paddle has a large number of points being away from the middle and stationary, it may not learn the powerful playstyle characteristic of returning to the middle. Because only every fifth time step was recorded when generating the datasets, the datasets have some randomness in the data collected, and they may not collect data examples which would help emphasize ideal playstyle features. A potential solution to this problem for the right paddle would be to filter the dataset from a percentage of the samples when the paddle is staying still away from the goal. If this reduces the size of the dataset by a significant amount, additional new data could be generated, with emphasis being put into playing with a more ideal playstyle.

5.3 Features of Reinforcement Learning Results

The results for the reinforcement learning quickly attract attention due to the unusualness of the plots obtained when compared to the plot that is generally shown to be expected from a deep Q neural network, which can be seen below in Figure 18. This plot is shown to gradually increase until the average reward score converges to what could be considered the approximation of the Q*(s,a) of Q-learning. The most noticeable difference between the plot in Figure 18 and the DQN resulting plots in Figure 16 and Figure 17 is the large spike in the plots for the DQN results.

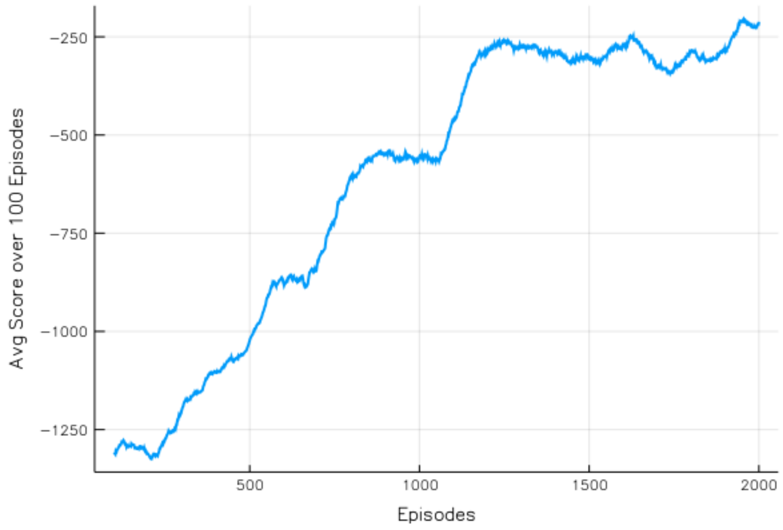


Figure 18: Standard DQN Running Average Training Plot from Example

A solution for this is that a new or improved function needs to be utilized in the developed DQN program, as it does not operate correctly until the first full set of the number of episodes that it is intended to display the average of, have been calculated. This effects the initial section of the plot, and ruins a portion of the data being displayed. Another feature that is noticeable is that the result plots do not appear to converge to a value, as compared to the Figure 18 example DQN plot. Because the example DQN plot appears to converge, it likely will not significantly improve if more training is performed with the weights obtained at the end of its training. The resulting DQN plots however, do not appear to converge, and this is why additional training was performed during the training phase. The reward function was a significant challenge in the development of the DQN, as it needed to represent the environment and the potential events that could occur, and yet not become too complicated. The complexity of the reward function likely is a significant cause of the resulting DQN plots not converging smoothly to any Q*(s,a). Although the resulting plots do not appear to converge to any particular value, the use of additional training was capable of boosting the performance of the paddles to a level of being able to noticeably track the ball and attempt to bounce it back to the other paddle. On average, the paddles could achieve between three and five bounces before either paddle would miss the ball. Neither paddle achieved a performance level where it was deemed capable of challenging a human opponent, so the Pong environment was never developed to accept inputs from anything but the DQN agents as of this time.

5.4 Computational Processing Power and Time

Another challenge of this project was the computational processing power and time available to me, especially in the reinforcement learning phase. Originally, a lab computer that had been used to perform deep learning experiments on in the past was deemed ideal for this project, as it was believed to have a significant amount of RAM and a powerful GPU, but this was not immediately confirmed. It was only after several months that it was investigated and discovered that all of the hardware upgrades had been the previous user’s personal hardware and had thus been removed when their project was completed. This was troublesome for performing the training for the deep Q learning, as some training attempts could take up to 15 hours to complete, and thus had to run all night long. This meant that if a new reward function or network model was to be tested, I would have to wait until the following day before I could view the results.

CHAPTER SIX: CONCLUSION AND FUTURE WORK

6.1 Introduction

Machine learning is an extremely powerful and versatile development that is having, and will have, a significant impact on society, with one area of prowess being that of automation. Supervised and reinforcement learning are particularly effective in the area of automation of video games, with major software developers like Google's DeepMind AI and IBM even using them in the automation of multiple complex games and activities. Throughout this project, a significant amount of experience has been acquired in the areas of software development and machine learning; all while developing functional programs of automated versions of Pong while using two different programming languages. The results obtained contained interesting features that present the opportunity of future research projects to attempt to develop and improve performance on.

6.2 Machine Learning and Automation

The impact of machine learning has been quickly growing in society, with the power of computers playing a significant role in that growth. With the development of powerful new neural network techniques and deep learning, the automation of more and more complex things has become available to humanity. Initially, simple Atari games were automated, but now, games like Doom, Go, and even StarCraft II have been automated to a super-human level. This level of automation of these games may seem irrelevant to many at this time, but by automating them, it allows for the development of more powerful AI techniques and ideas. An example of this being Google's AlphaFold, an AI that can correctly predict the folding of proteins at a much higher accuracy than any human, even with it being one of the most challenging problems in all of science. This could have significant effects on research in associated areas, due to the ability to predict future protein structures.

6.3 Insights Obtained

Throughout the completion of this project, a large number of insights have been obtained, with the most significant being those in software development and knowledge in how to perform machine learning for unique environments or conditions. During the completion of this project, a working, basic AI, was developed to play Pong with both a supervised learning technique, and a reinforcement learning technique. The supervised learning trained agent was even deemed, “unbeatable,” by some, and has yet to be defeated since its training. The reinforcement learning portion of the project was more interesting on a personal level, but the ability of getting the agent to operate at a human challenging level has not yet been achieved.

The challenges of developing supervised and reinforcement learning systems have become much clearer; with the area of reinforcement learning appearing to require a more significant amount of effort in order to develop an effective controlling agent. Throughout this project I have obtained a significant amount of knowledge for programing with both Matlab and with Python, and greatly appreciate the machine learning and neural network libraries available to us, such as “TensorFlow” and “Keras”.

6.4 Future Work

There are several potential areas of future research opportunities offered by this project due to the unique results obtained during testing of both machine learning methods. The supervised learning portion has unusual results in that there is a large difference between the test accuracies for both paddles, and yet the left paddle with its lower accuracy result is much more challenging to defeat. Future work could be done to improve the right paddle which spends a very large portion of the time staying stationary, even though it shows a noticeably higher test accuracy than the left paddle.

The reinforcement learning has several areas that could be improved. A potentially easier task would be to improve the display of the data after training through the improvement of the running average function being used. A more challenging task would be to optimize the reward function in order to improve the training results and cause the rewards per episode plots to noticeably converge to a value after increasing from an initial point.

REFERENCES

Libguides.utsa.edu. (2019). *LibGuides: Dissertations and Theses: Find Dissertations & Theses*. [online] Available at: https://libguides.utsa.edu/dt/findDT [Accessed 9 Aug. 2019].

Science in the News. (2019). *The History of Artificial Intelligence - Science in the News*. [online] Available at: http://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/ [Accessed 9 Aug. 2019].

Coursera. (2019). *Coursera | Online Courses & Credentials From Top Educators. Join for Free*. [online] Available at: https://www.coursera.org/?skipBrowseRedirect=true&skipRecommendationsRedirect=true&tab=completed [Accessed 9 Aug. 2019].

Brownlee, J. (2019). *Supervised and Unsupervised Machine Learning Algorithms*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ [Accessed 9 Aug. 2019].

En.wikipedia.org. (2019). *Artificial neural network*. [online] Available at: https://en.wikipedia.org/wiki/Artificial_neural_network#Deep_stacking_networks [Accessed 9 Aug. 2019].

Vink, R. (2019). *Programming a neural network from scratch - Ritchie Vink*. [online] Ritchievink.com. Available at: https://www.ritchievink.com/blog/2017/07/10/programming-a-neural-network-from-scratch/ [Accessed 9 Aug. 2019].

dzone.com. (2019). *An Introduction to the Artificial Neural Network - DZone AI*. [online] Available at: https://dzone.com/articles/an-introduction-to-the-artificial-neural-network [Accessed 9 Aug. 2019].

Developer News. (2019). *An introduction to Deep Q-Learning: let's play Doom*. [online] Available at: https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/ [Accessed 9 Aug. 2019].

Medium. (2019). *5 Regression Loss Functions All Machine Learners Should Know*. [online] Available at: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0 [Accessed 9 Aug. 2019].

Medium. (2019). *Qrash Course: Reinforcement Learning 101 & Deep Q Networks in 10 Minutes*. [online] Available at: https://towardsdatascience.com/qrash-course-deep-q-networks-from-the-ground-up-1bbda41d3677 [Accessed 9 Aug. 2019].

Medium. (2019). *Simple Reinforcement Learning: Q-learning*. [online] Available at: https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56 [Accessed 9 Aug. 2019].

Zokaie, T., Osterkamp, T.A., and Imbsen, R.A. 1991. “Distribution of Wheel Loads on Highway Bridges,” A report prepared for N.C.H.R.P., Transportation Research Board.

VITA

Andrew Waterreus is from Bulverde, TX, just north of San Antonio. He earned both a Bachelor's and Master's degree in Mechanical Engineering, with a concentration in Robotic Systems and Controls at The University of Texas at San Antonio. His research focus is on robotic systems and machine learning. His future plans include becoming a PE (Professional Engineer) in mechanical engineering, and work with a research and development company, to be able to help push the boundaries of technology to new levels.