Title page

dedications

# 1. Abstract

Large quantities of data are required for training and testing Machine Learning and Artificial Intelligence models. This can be difficult in some circumstances, such as where there is a large financial or time cost to collecting the data or where there are security and safety concerns. Video Games provide controllable and reproducible environments which can approximate real world conditions that can be used for testing and training models. Often these can be then transferred over to real world applications, such as using a virtual environment to train a robotic arm or driving a car through a virtual city for use in developing self-driving software. Video Games can also contain highly complex rulesets and action spaces which can test the robustness of a model.

A common approach is to use Imitation Learning whereby a model is trained on labelled data generated by an expert playing the game. In this case however the model will only ever be able to approximate the performance of the expert and follow closely their style of play. Another approach is reinforcement learning whereby a model is allowed to choose its own actions and receives a reward based on how good or bad that move is measured in relation to the game, such as staying alive vs being killed, keeping the car on the road vs crashing etc. Positive actions are encouraged while negative ones are discouraged and the model learns its own unique style of play, often breaking heuristics held to be the most optimal by humans and finding new, more efficient solutions. Reinforcement Learning however requires a vast amount of data and sometimes a game’s action space can be so large that learning even simple game rules, basic to human players, becomes almost impossible.

In this thesis a model will be trained to play a video pinball game using reinforcement learning. To overcome the initial training hurdle the model will start with a network pre-trained using data from a human expert. The expert will not utilise the full game action space, namely they will not tilt the table. It can then be observed whether the model is able to learn this behaviour itself to overcome poor play by the expert, in other words can it learn to tilt the table and avoid the ball draining, a move absent from the training data used for the imitation learning model.

Reinforcement learning is often used to train a model from scratch. This thesis will examine if reinforcement learning can be used to improve a pretrained network beyond human ability and learn novel behaviour not exhibited by the human expert. This could have interesting applications in areas such as robotics and finance which often respond poorly to pure reinforcement methods but may benefit from a hybrid approach. The thesis will also be examining the application of Imitation and Reinforcement learning to a more complex form of video pinball than has been used in past experiments. While these results are of great interest, if their findings can not scale up to more graphically complex simulations, approximating real world conditions, they may be of limited value.

Contents

[1. Abstract 1](#_Toc8162464)

[2. Artificial Intelligence 4](#_Toc8162465)

[2.1 Introduction 4](#_Toc8162466)

[2.2 Machine Learning 4](#_Toc8162467)

[2.3 Supervised Learning 5](#_Toc8162468)

[2.4 Reinforcement Learning 5](#_Toc8162469)

[2.5 Artificial Neural Networks 6](#_Toc8162470)

[2.5.1 Convolutional Neural Networks 8](#_Toc8162471)

[3. Artificial Intelligence in Games 8](#_Toc8162472)

[4. Pinball 11](#_Toc8162473)

[4.1 3D Pinball For Windows – Space Cadet 13](#_Toc8162474)

[4.2 Existing work in Artificial Intelligence and Pinball 15](#_Toc8162475)

[5. Methodology 18](#_Toc8162476)

[5.1 Research Undertaken 18](#_Toc8162477)

[5.2 Research Question 18](#_Toc8162478)

[5.3 Proposed Project Implementation 18](#_Toc8162479)

[5.4 Prototype 19](#_Toc8162480)

[6. Implementation 21](#_Toc8162481)

[6.1 Sprints 21](#_Toc8162482)

[6.1.1 Phase 1 21](#_Toc8162483)

[6.1.2 Phase 2 30](#_Toc8162484)

[7. Findings & Conclusions 35](#_Toc8162485)

[8. References 40](#_Toc8162486)

# 2. Artificial Intelligence

## 2.1 Introduction

There is no widely excepted definition of Artificial Intelligence as, by necessity, it first requires an agreed definition of intelligence. The human mind has been an area of research and study for centuries, beginning with the ancient Greek philosopher Aristotle who died in 322 BC (Russell & Norvig, 2010). Artificial Intelligence has built on research in areas such as neuroscience, philosophy, mathematics and others in an attempt to build computer systems which are capable of automating intelligent behaviour (Luger, 2002). This has involved studying how animal brains work and building systems that approximate them (see Artificial Neural Networks below) to questioning what intelligence is and whether it is even possible for a non-organic construct to possess it (Nilsson, 1998).

As Artificial Intelligence is the goal of researchers working in this area many of them have put forward definitions of what exactly intelligence is and what qualities an Artificially Intelligent machine should possess. Most famous is the test proposed by Alan Turing in which a human judge would converse through text with two other users, one of whom was a machine in an attempt to differentiate them (Turing, 1950). Some researchers have taken a “we’ll know it when we see it” approach to intelligence. One current work on the topic defines Artificial Intelligence as “The quest to build intelligent machines, for some definition of intelligence” (Togelius, 2018).

## 2.2 Machine Learning

Machine Learning is a branch of Artificial Intelligence in which data is used to train a model which can then be used to make predictions on new, unseen data. Machine Learning is one of the capabilities a computer would need to possess “to adapt to new circumstances and to detect and extrapolate patterns” so that it could pass a Turing test (Russell & Norvig, 2010). Artificial Intelligence can be seen as the high level considerations surrounding intelligence and computing whereas Machine Learning is a study of the algorithms involved in training a model or agent. The terms are frequently used interchangeably however and are often used to differentiate algorithms. Linear Regression would not be considered Artificial Intelligence and along with simpler algorithms such as decision trees and ensemble methods would be classed as Machine Learning. Artificial Intelligence is thus often reserved for the application of Artificial Neural Networks.

## 2.3 Supervised Learning

Supervised Learning algorithms can be used when the “ground truth”, a variable which the model will predict, exists. Examples of Supervised Learning include fitting a model with labelled pictures of cats and dogs so that it can predict the label of unseen pictures, or providing a model with sentiment analysis from social media and fitting this to the current value of a stock asset so that it could predict a price at a future time. Regression algorithms are used where the ground truth is continuous, such as stock market prices, and classification algorithms are used where the ground truth is categorical.

## 2.4 Reinforcement Learning

Reinforcement Learning has been described as “learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The Learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.” (Sutton & Barto, 2017) Reinforcement Learning built on earlier work in the area of Temporal-Difference Learning which had been used by Arthur Samuel’s to develop a program that could play the board game Checkers (Sutton R. S., 1988) (Tesauro, 1995).

There are a number of machine learning paradigms and of recent Curiosity-driven Reinforcement learning has achieved strong results in video game environments such as VizDoom and Super Mario Bros (Pathak, Agrawal, Efros, & Darrell, 2017). For the purposes of this project only Q-learning and Deep Q-learning were examined.

Q-learning can be applied in cases where there is a small number of possible states and possible actions an agent can take in each state. Games such as Tic-tac-toe and Taxi from the OpenAI gym environment have limited board configurations and a small action space. A Q-table can be constructed where the rows represent all possible states and the columns represent all possible actions in each state.

An agent is coded to play the game by either executing a random move or by checking the current state of the game and selecting from the Q-table the action, for that state, with the highest Q-value or quality. At the beginning the Q-value for all actions is 0 and the agent executes moves randomly. The Q-value for each state action pair is updated using a formula such as the Bellman Equation. This formula updates the Q-value using a reward plus the discounted expected value of the next state. As the agent acts randomly it explores many different states, receiving positive rewards in some and negative rewards in others. The Q-value of state action pairs that lead to states with a positive reward will increase and the agent will prioritize these as it switches from mainly acting randomly to using the Q-values from the Q-table. This is achieved through the use of an exploration value which is set high at the beginning, encouraging the agent to act randomly, and decreases over multiple run-throughs of the environment to encourage the agent the use the Q-table.

In more complex environments, such as Super Mario Bros. or a game of pinball for example, it would be impossible to construct a table representing all possible states and actions. In this case a neural network can be used to approximate the Q-table. The state is provided to the network as input, in the form of pixel values or in some other form, and the network outputs a Q-value for each possible action that can be taken. Either the action with the highest value is executed by the agent or it chooses randomly. Regardless, the Q-value of the action taken is updated using the Bellman Equation or similar and the neural network is updated using backpropagation, as outlined below. This is process of using a neural network to approximate a Q-table is known as Deep Q-Learning.

## 2.5 Artificial Neural Networks

Artificial neural networks are systems modelled on the functioning of the animal brain and are currently one of the main areas of research in Artificial Intelligence. In their foundational paper Warren McCulloch, a psychiatrist and neuroanatomist, and Walter Pitts, a mathematician, described a logical calculus for constructing a model of a neuron. (McCulloch & Pitts, 1943). In animal biology neurons carry signals through the body’s nervous system. Signals are sent from one neuron to another if a threshold is overcome (the all or nothing principle) and McCulloch and Pitts outlined a process for representing this mathematically.

A number of important contributions have been made to expanding upon the original work of McCulloch and Pitts, most notably (Hebb, 1949) and (Rosenblatt, 1957). Neural Networks have fallen out of favour and suffered cuts to funding a number of times since due to fears over their limitations (Minsky & Papert, 1969) and technological limitations. Advances in computational power in recent years have led to a re-examination of papers and theories advanced in the 1970’s and 80’s which could not be tested or implemented at the time.

Neural networks are built from layers which each have multiple neurons. The simplest network will have an input layer, a single hidden layer, and an output layer. The input layer will have a number of neurons equal to the number of input features. The output layer will have a number of neurons equal to the expected output. For a regression or binary classification task there will be a single node. For multi class classification the output layer will contain a number of nodes equal to the number of classes under consideration (Haykin, 1999).

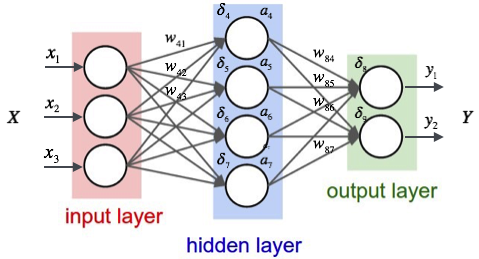
There are no clear guidelines on the construction of hidden layers and it is often left to trial and error. Networks containing more than one hidden layer are called Deep Neural Networks. Every neuron in the input layer has a single weighted connection with every neuron in the first hidden layer. Each neuron in the hidden layer takes the value of each of its connected neurons from the input layer, multiplies them by the associated weight and then sums them together. A single bias value is then added and this is then passed to an activation function which transforms the value. In the case of a Deep Neural Network, once the values of all neurons in the first hidden layer have been calculated it acts as the input layer for the second hidden layer, and so on. In this way the inputs are propagated through the neural network (Haykin, 1999).

Figure 2.1 - A fully connected neural network with 1 hidden layer.

The weights of a neural network are generally initialised using random values. By modifying these weights the neural network can be used to approximate any continuous mathematical function. This is achieved by training the network using inputs and their ground truth. Initially the inputs will be propagated forward using the random weights. The output is compared to the ground truth with the difference between them acting as a measure of the current weights error or the cost of using the current weights values. This error is then propagated backwards through the network with adjustments being made by gradient descent to the weights so as to most efficiently decrease the cost (Haykin, 1999).

Backpropagation was developed in the 1960’s but is today most associated with the work of Geoffrey Hinton (Rumelhart, Hinton, & Williams, 1986).

### 2.5.1 Convolutional Neural Networks

Convolutional neural networks are a specific type of neural network used mainly, but not exclusively, for image processing tasks. The input layer of a convolutional neural network takes the pixel values of an image. Multiple channels can be provided representing each pixels Red, Green and Blue (RGB) value or a single greyscale channel. Greyscale is common where colour may be unimportant or to save on computing resources.

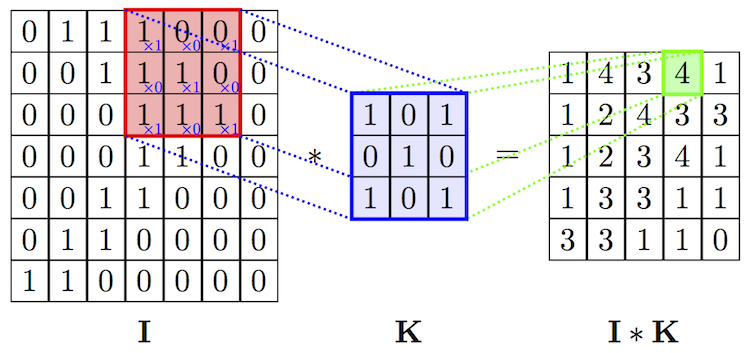
Special convolutional layers are used to extract features from the image. A kernel passes over a section of the image and calculates a value which represents high-level features such as edges or shapes. For example, in a greyscale image a 3x3 would pass over the image from left to right, top to bottom, calculating a value to represent each 3x3 block in the image.

Figure 2.2 – A 3x3 kernel pass over an image and creates new features

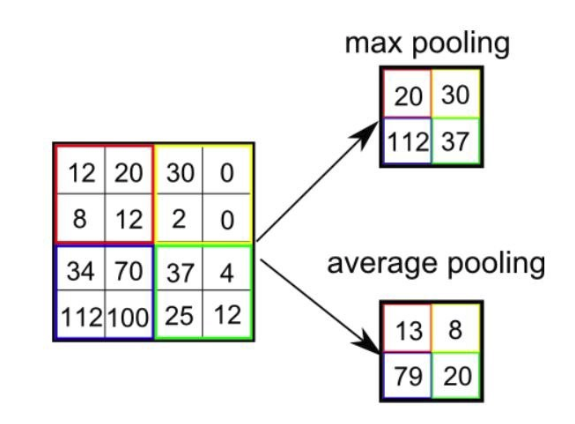
Often the convolutional layer will create a large amount of features and result in slow processing. To speed this up or in cases where resources are limited a pooling layer can be used to reduce the size of the convolved features and can also extract the most dominant features. Similar to how a kernel works pooling looks at a specific segment of the convolved features and calculates a value to represent that segment. Two popular methods are Max Pooling which takes the maximum value in that region and Average Pooling which takes the average of the values in that region.

Figure 2.3 – Examples of max and average pooling

Multiple Convolution-Pooling pairs can be chained together to learn features. They are not fully connected as in the case of a neural network; each layer is connected to only a part of the next layer. This allows specific parts of the network to specialise in recognising specific features. The final convolution or pooling layer is followed generally by a single fully connected layer which is in turn connected to the output layer. At each step an activation function is used to determine which nodes activate and send their values forward to the next layer. As with a Simple Neural Network the error can be backpropagated to improve the Convolutional Neural Networks performance.

# 3. Artificial Intelligence in Games

Games have long been used as a tool to build and test artificial intelligence algorithms. Early research focused on board games but recent achievements of the Deepmind team with Go have led many to view classic board games as no longer providing an avenue for further advancement (Togelius, 2018). Attention is now turning more towards video games with massive action spaces, such as StarCraft II (Arulkumaran, Cully, & Togelius, 2019).

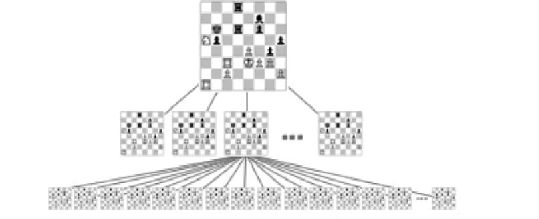
In the earliest days of research into Artificial Intelligence games provided a complex and reproducible environment in which to test algorithms, clear guidelines that could be used to measure the success or otherwise of the algorithm and an interesting challenge. Alan Turing reinvented the MinMax algorithm to play chess. In 1952 A.S. Douglas developed software that could master the game of Tic-Tac-Toe as part of his doctoral thesis and the early basis of Reinforcement Learning was put in place by Arthur Samuels in developing an algorithm that could learn to play Checkers (Yannakakis & Togelius, 2018).

Figure 3.1 – An example of MinMax applied to chess.

Research into Artificial Intelligence in games was limited by memory restrictions and processing capability. Some of these were overcome in the 1990’s; in 1994 the Chinook Checkers Player defeated the World Checkers Champion Marion Tinsley and in 1997 the World Chess Champion Gary Kasparov was defeated by the IBM developed Deep Blue. In 2007 the game of Checkers was solved (Yannakakis & Togelius, 2018). Prior to this, in 1992, TD-Gammon had learned to play the game of Backgammon at the level of a human player using Temporal Difference Learning (Tesauro, 1995) and following Deep Blue IBM developed Watson, a system capable of answering questions using Natural Language Processing. In 2011 it won $1 million in a special game against former Jeopardy! champions.

Further advances in processing power in the early 2000’s allowed for artificial intelligence to be applied to classic arcade style video games. In particular the 1980’s arcade games Pac-Man and Ms. Pac-Man became testbeds for research into computational intelligence as well as robotics, biology, sociology and psychology (Rohlfshagen, Liu, Perez-Liebana, & Lucas, 2018). OpenAI, founded by Elon Musk, created the Gym library for python which allowed programmers to build and test algorithms on video games from the Atari 2600. The most notable work built on this environment was the DeepMind’s paper on achieving better than human performance on a number of these games using only the raw pixel data (Mnih, et al., Playing Atari with Deep Reinforcement Learning, 2013). Though the Gym environment is no longer maintained by OpenAI it is still an important tool in learning Artificial Intelligence and is the basis for many tutorials introducing new users to this field of study.

In 2015 the DeepMind developed program AlphaGo was became the first to defeat a human professional Go player and the following March it defeated Lee Sedol, an 18 times world champion widely considered the greatest of all time. In 2017 AlphaGo further defeated the world No. 1 Ke Jie after which AlphaGo was retired. Two new versions were developed, the most recent of which, AlphaZero, is capable to defeating the world’s best Go and chess players. (Silver, et al., 2017) Following these overwhelming achievements in classic board games attention has shifted to sophisticated modern video games Defence of the Ancients (DOTA) and StarCraft II. Both have massive action spaces, complex economic systems and build trees. This complexity renders many traditional Artificial Intelligences approaches unusable. DeepMind built a program which in early 2019 defeated two of the top StarCraft II players and OpenAI developed an Artificial Intelligence which defeated a number of the top professional DOTA teams. However, both played in heavily restricted environments; DeepMind’s AI is only able to play as and against one of the different races existing in the StarCraft II game and both it and OpenAI 5 receive detailed information relating to their current position and the position of other players and entities, instead of the raw pixel data on the screen.

Going forward there are a number of reasons why video games in particular will have an important role to play in advancing Artificial Intelligence research. They provide a complex and easily reproducible environment that can be used to test and challenge the robustness of algorithms. Simulations can be sped up to provide vastly more data than could be applied in real world applications; AlphaGo played millions of games against itself, far more than a human player could hope to play in their lifetime. Video games are highly changeable; a researcher can alter parts of the game to test the algorithm under specific conditions or generate data under a specific game state which a human may observe rarely, if ever.

Figure 3.2 – An early implementation of Deepminds StarCraft II AI showing how it represented the game state.

It is at times difficult to understand the actions of human players in complex games like DOTA and StarCraft II; building algorithms that can replicate their behaviour may give insights into complex human thought and processing. Video games also provide a novelty factor; a human can relate to a story about a program playing a video game that they know really well in a way they could not relate to other types of AI research. This can be leveraged by companies or researchers to gain media attention and funding.

# 4. Pinball

A number of games were created in the 15th and 16th centuries which can be seen as predecessors of pinball. These games all share a common theme “in which players competed to move balls to specific places in exchange for points” (Ruben, 2018).

An early form of pinball called bagatelle, based on billiards, was created in France in the late 1700’s and spread to the United States around the time of the American Revolution. In 1871 Montague Redgrave patented a bagatelle using a spring loaded plunger to launch the ball onto the scoring area as well as further increasing the incline of the table (Ruben, 2018). Around this time also the table became smaller and began to use marbles or metal balls of billiard balls. In 1880 the first coin operated bagatelle game was created and following this tables were created to be placed in public venues such as bars for public use. As the game evolved it was also described as the “pin game” after the pins which made up much of the play area. This would later become “pinball”.

In 1933 Harry Williams developed an early version of the tilt-switch which in modern games will lock the table and prevent any additional points being scored, causing the ball to drain. This was made possible due to his earlier developments the same year in adding electricity to the table. This meant that sounds, music and moving parts could be added. The first was a solenoid which could force a ball out of a hole and up the table, against gravity. In 1937 Bally developed bumpers which would drive the ball around the table and in 1938 illuminated scoreboards were added. Williams would go on to establish Williams Manufacturing Company, later WMS Industries, Inc, one of the main pinball machine manufactures until they ceased producing them in 1999. (Ruben, 2018)

At this stage the scoring area was made up of a series of holes obstructed by metal pins sticking out of the table. These holes had points values and players would launch anywhere up to 10 balls onto the scoring area with the hope of landing them in these holes. Points would be manually tallied at the end of each game. The addition of electricity allowed for these to be tallied within the game. In 1934 Fiorello La Guardia, the Mayor of New York, began a clamp down on gambling which was controlled by elements of organised crime. Two thousand slot machines belonging to the gangster Frank Costello were seized and pinball moved in to fill the vacuum. The previous year manufacturers Bally created a pinball machine called Rocket which was capable to paying out money for high scores.

In 1935 Jacob Mirowsky, the operator of a candy store, was brought before court on charges of operating an illegal gambling room. He argued that pinball was a game of skill, not chance like slots. To prove this he argued that in a contest between three skilled players and an amateur the skilled players would consistently outscore the amateur. This challenge was accepted by the court. Mirowsky provided three skilled players who played before the court but none were able to score enough to meet even the lowest payout threshold. They were also unable to make shots called by the judge as a test of control. A detective who had never played before was able to achieve a score on his first attempt close to that of the skilled players. On January 21 1942 La Guardia signed an order allowing police to smash pinball machines and publicity shots where taken of the Mayor destroying machines with a sledge hammer. Some 11,000 were destroyed before he left office in 1945. (Ruben, 2018)

One of the most important games in the development of pinball is Humpty Dumpty, released in 1947 by David Gottlieb. This was the first game to possess solenoid driven flippers, activated by buttons on the side of the machine. Up until this point in time there was no way to stop balls which were heading towards the bottom of the table from entering the drain and not counting towards points. The player had little to no control over the ball once they had used the plunger and the amount of force applied to the ball was their only way of manipulating it. This development meant that the player could now exert control over the ball and direct it around the table. This in turn led to the development of more complex tables and rulesets. Pins and holes to trap the ball began to disappear to be replaced, in the 60’s and 70’s, with a wide range of targets and the introduction of toys and gimmicks. In 1960 Gottlieb’s son Alvin introduced the extra ball as a reward for skill.

On the 13th of May 1976 Sharpe played a game of pinball before the New York City Council to demonstrate that it was a game of skill, similar to the Mirowsky case forty years beforehand. He was able to call and make shots, explaining how the various targets effected his score. Following this demonstration the ban on pinball was lifted in New York; some states had already rescinded their anti-pinball legislation and others would follow suit after more demonstrations. (Ruben, 2018)

This easing of the laws around pinball machines happened at a time of great innovation in how the machines were manufactured. Up to this point they had been electromechanical but in 1976 switched to using solid-state electronics, such as circuit boards and digital displays. Previous to this score was kept by a mechanical scoring reel in the machine which could occasionally fail to add points if the ball was moving fast enough around the table. Scores could now be recorded accurately and saved to memory and designers could create more interesting playfield configurations and more complex rules as well as incorporate digital sound effects.

This history of the major innovations in pinball has been presented to show its evolution from a game of chance to a game of skill. In the documentary Wizard Mode Robert Sharpe estimated that pinball was 80% skill. (Drillot & Petry, 2016) While most games of skill also involve some degree of luck a large part of the game of pinball is dependent on the laws of physics and the quirks of kicker bumpers. Machine Learning methods cannot be applied successfully to games of pure chance like the lottery or rock paper scissors. The historical outline of pinball’s development following the addition of flippers shows how it became a game of skill which could be mastered by talented, dedicated players.

These innovations prompted Adam Ruben to state that the top players in the world possess three key skills: “knowledge of rule sets, ability to aim shots with the flippers, and ability to shake the machine just enough to save the ball but not enough to tilt.” (Ruben, 2018)

## 4.1 3D Pinball For Windows – Space Cadet

3D Pinball for Windows – Space Cadet is a demo version of a game titled Full Tilt! Pinball. The demo was bundled with a number of different iterations of Microsoft Windows. While it was removed from Windows before the launch of Vista, the demo’s popularity means that it is still possible to download and play today on newer machines.

Space Cadet is a space themed pinball video game. It shares a number of components and general similarities with other pinball games but is also relatively simplistic and lacks any major innovation. On starting the game the user is presented with a window showing the pinball table. The user can launch the ball from it’s starting position on the bottom right, activate the left and right flipper located at the bottom centre of the table and tilt or nudge the table. This represents the real world practise of physically moving the table to influence the movement of the ball.

Figure 4.1 – 3D Pinball for Windows – Space Cadet

The games physics are set to emulate the functioning of a real, physical pinball table. These tables are tilted so that the ball moves by gravity towards a drain at the bottom of the table. In front of the drain are two flippers (called the left and right flipper) which the user can control to apply force to the ball and drive it up the table. Skilled players can use the flipper to “trap” the ball and hold it in place so as to line up specific shots, such as hitting a certain target.

The aim of pinball is to accumulate as high a score as possible. This is done by having the ball interact with specific parts of the table. There are a variety of physical targets which increase the score when hit by the ball. Some of these targets interact with each other to form “missions” which result in higher score rewards. For example, on the midway point on the left hand side of the table is the “Launch Ramp”. By hitting the targets on the body of the ramp the user can select a mission and lock this mission in by driving the ball up the Launch Ramp. One mission involves successfully sending the ball up the Launch Ramp three times. A skilled player will be able to trap the ball with the left flipper so as to send it back up the ramp immediately after. Doing so rewards the player with a “Reflex Shot”, a score increase for replicating the same shot in quick succession.

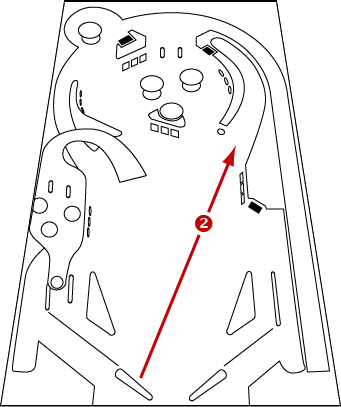
In general, once the ball ends up in the drain three times the game ends. However, there are certain conditions which allow the player to replay a ball or receive an extra ball. A skilled player would have an understanding of the reward structure and the rules which allow them to receive extra attempts. While, as with any pinball table, the basics of the game are extremely simple, there are a wide range of complex rules and strategies required to secure a high score (Irons, 2005).

Figure 4.2 – A guide on how to launch the ball up the Hyperspace Chute (Irons, 2005).

A bug was identified in the game which was never fixed whereby the flippers appeared to change in power. It was noticeable that at times they would impart less force to the ball as they would at other times. This is still present in the downloadable versions of the game today. An anecdotal fix for this issue is to run another game in the background; a player previously found that starting up Command & Conquer: Red Alert 2 and minimizing that window resulted in better performance of Space Cadet (neoseeker, 2003). During this project it was found that opening the launch window for Overwatch fixed this issue.

## 4.2 Existing work in Artificial Intelligence and Pinball

To date pinball has seen only limited use for research in machine learning. The most noted example was the application of Q-Learning by Google Deepmind to 49 games from the Atari 2600. The model was given only the pixel display as input and had no prior understanding of how any of the games were played, learning from the rewards provided to it as it played. The highest results of all were achieved on Video Pinball (1980) with the model showing an improvement over a human benchmark of 2539% (Mnih, et al., 2015). This built on the groups earlier research into the application of reinforcement learning to just seven games, of which Video Pinball was not included (Mnih, et al., 2013). The Video Pinball game is quite rudimentary and visually unappealing and in demonstrating the results of the 2015 findings Deepmind would instead use the 3rd best performing game; Breakout.

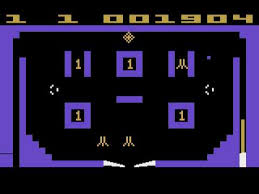
As part of their course students at the University of Virginia followed the same approach set by Deepmind, using the Gym environment developed by OpenAI to train a model to play Video Pinball from pixel inputs using reinforcement learning. In their results they found that “the performance of the model was not comparable to a human, because it made some mistakes while predicting the location of the ball. But we think that over time, it can surpass the high score in pinball given enough training time.” (Rosenberg & Somappa, 2017)

Figure 4.3 – Video Pinball for the Atari 2600

Also in 2017 smallplanet, a mobile app development company based in New York developed, as a skunkworks project, an app that could play pinball on a physical machine. A mobile phone running the app could be held in place above the table and an Omega2 board was connected to the physical machine so that it could receive commands from the phone and drive the flippers. In phase 1 the app was trained on expert play, in phase 2 the app would be trained using reinforcement learning however the project has been on hiatus since March of 2018 (smallplanet, 2018).

Outside of these few examples there are a few references to other projects applying machine learning to pinball. The algorithm described above in (Mnih, et al., 2013) and (Mnih, et al., 2015) was implemented in Lasagne/Theano (Sprague, 2017) and then used by a blogger alias That’s So Deep Dude to train a model to play Video Pinball (sodeepdude, DQN learns video pinball, 2017). This blogger called into question Deepminds findings, suggesting that the human level score used as a benchmark for Video Pinball was far below actual average human level ability (sodeepdude, Comparison of human scores and “human scores” in Atari, 2017).

A number of limited experiments in the use of machine learning in pinball have been carried out with only sparse documentation on the video sharing site YouTube. In 2011 a video was posted showing pinball being played on a physical The Lord of the Rings table (Metcalf, 2011). Attached to this was a laptop displaying a top down view of the table, showing this to possibly be an earlier version of (smallplanet, 2018). From what little is shown on the laptop it appears however that the camera is detecting the current position of the ball and activating the flippers if it is in a certain section. Considering that many of the major machine learning paradigms had not been implemented at the time of the recording of this video it is more likely that it is using a set of hard coded rules.

Evolutionary algorithms were used on a video depicting an AI playing Pinball (1983) for the NES (Parodi, 2015). In the comment section it is explained that the game was played in an emulator which called a script to read the games RAM status in each frame. This then had to read to understand which values corresponded to the ball position and other game variables. However no further details are given as to the algorithms implementation.

Using reinforcement learning and the ML Agents environment within Unity an AI was trained to play pinball with only the ball position and velocity as input (LipowitzFilms, 2018). While the agent receives a high score it is continuously activating the flippers, even when the ball is at the other end of the table though it does appear to exhibit an ability to hit specific high scoring shots indicating it is not hitting the ball randomly. This agent is trained by having the ball spawn randomly on the table in the event of it draining. The ability to build a table in Unity and control every aspect of it has interesting implications for future research into games and pinball in particular. In pinball certain conditions need to be met for certain areas of the table to be accessible or for high scoring opportunities to occur. Even under expert human play these conditions might be met very rarely making it difficult, if not impossible, to secure sufficient training data for these cases. By programming a table in Unity it would be possible to recreate any conditions needed for training the model and also create specific tests to check the models performance.

Pinball has shown itself to be an interesting arena in which to apply and test machine learning principals. It is sufficiently complex to present a challenge in finding an optimal approach to applying machine learning but is not so complex as to require large scale investment in hardware or cloud compute to solve. Pinball has also been underutilised as an application area for machine learning showing that it can continue to be an area of interest in future research.

# 5. Methodology

## 5.1 Research Undertaken

The main areas of research considered were those relating to capturing state and input data from gameplay, building a Convolutional Neural Network for image processing and Reinforcement Learning. While it is possible it is today impractical to build Neural Networks by hand. There are a number of libraries which do this instead and greatly speed up the process. Keras is one of the most popular libraries at the moment but others were examined in brief.

Previous attempts at similar projects were also examined. The Space Cadet pinball game was chosen because it had never been used in Artificial Intelligence testing before and because pinball has seen only limited application in the field. There were however a number of projects which offered a guildline on how to progress, most notably the simplicity of the model implemented by smallplanet (smallplanet, 2018).

## 5.2 Research Question

Can networks pretrained on expert human gameplay be used to accelerate the training of a model through reinforcement learning. An investigation into whether a network can develop behaviours not exhibited in the human gameplay examples. An examination of a more complex environment than has been used in previous research into pinball and Artificial Intelligence to see if commonly used algorithms are still robust enough to perform well.

## 5.3 Proposed Project Implementation

The project needed to be faced in two phases. The first was to build a Convolutional Neural Network which could talk in the states of a pinball game as an image and output the prediction of the next button to press. Training data would then be provided to train the model.

In the second phase the model generated in the first phase would be used to allow an agent to play a game of pinball itself and use changes to the games score to act as rewards for implementing Reinforcement Learning. The second phase required the construction of an additional Convolutional Neural Network to convert the image of the game score to digits which could be used for determining when it had changed.

## 5.4 Prototype

|  |  |  |
| --- | --- | --- |
| **Prototype Number** | **Start Date** | **Finish Date** |
| 1 | 02/10/2018 | 03/10/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Install CUDA development tools | Complete |
| 2 | Install and verify Tensorflow-GPU | Complete |
| 3 | Install and verify Keras | Complete |

The CUDA toolkit, provided by NVIDIA, allows Tensorflow to train using GPU instead of CPU. This is particularly useful for convolutional neural networks. Tensorflow has its own implementation of Keras but the GPU version contains a number of bugs which, in testing, were found to throw errors when attempting to import a previously trained model. It was decided to install and Keras and use Tensorflow-GPU as the backend.

|  |  |  |
| --- | --- | --- |
| **Prototype Number** | **Start Date** | **Finish Date** |
| 1 | 05/10/2018 | 11/10/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Write script to capture screen and player input. | Complete |
| 2 | Write script to take input from a model and execute this in game. | Complete |

Two scripts, written in Python, were needed to provide the basic pipeline for the project. The first script, when ran from the command line, would capture a section of the screen and save this alongside an array indicating which, if any, buttons were pressed. The 3D Pinball for Windows – Space Cadet game has a default size and position when it opens and the script was written to take and save screenshots of a part of the game window. This was achieved by using the PIL library for screen capture and the python-keyboard library to check if specific keys were being pressed at each frame. A game of pinball was played to test this and samples of the data were verified visually.

This data would allow a model to be trained but another script would be needed to send the models predicted output to the keyboard so it could be executed by the game. A second script was written as a test to simply take in the button presses from the game played above and execute these. The script needed to take in the array representing the input and determine which keys should be pressed or released based on this. By opening a game window and running this script it was proven that the game could respond to input sent to it from a model.

# 6. Implementation

## 6.1 Sprints

The completed prototype was able to capture human gameplay and execute command in a format identical to the format a trained model would produce. From this stage the first phase, finding the best parameters for a Convolutional Neural Network and generating data to train a model, could begin.

### 6.1.1 Phase 1

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 1 | 12/10/2018 | 18/10/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Write script to build dataset | Complete |
| 2 | Play pinball to generate data | Complete |
| 3 | Build basic model | Complete |
| 4 | Expand script from task 2 of previous sprint to use this model | Complete |
| 5 | Train model and test | Complete |

A script was written that would take a sample of the frames and inputs from a game to build a dataset that could be used in training a model. A number of games of pinball were played to collect data and datasets were generated from this, one for each game. As a test a basic model consisting of two convolutional layers and a dense layer was created and trained on these datasets.

The script from task 2 of the previous sprint was expanded to load this model and send screen shots of the game screen to it. The model would return a prediction which could be then be sent to the keyboard and executed by the game.

Tasks 2 and 5 were repeated numerous times to examine how the network responded to training data. In doing so a number of problems were highlighted. The model exhibited “holding” behaviour, for example holding the left flipper to hit the ball but never releasing it after. It became apparent that the network was fitting on the position of the plunger and flippers as opposed to position of the ball on the table.

Another problem was that the keyboard could not react to all of the output of the model. If, over 3 frames, the network predicted nothing, left flipper, nothing, the single input of left flipper for a frame would not be enough to cause the keyboard to respond. A button would only be pressed if the input was “on” for more than 2-3 frames. While the network eventually learned to execute sequences of button presses this was a concern at the beginning.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 2 | 19/10/2018 | 26/10/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Play pinball to generate data | Complete |
| 2 | Build “stacked” datasets | Complete |
| 3 | Build and train new model, test | Complete |

Based on the problems presented in the previous sprint a new approch was taken whereby every four frames were “stacked” to create a new dataset. As such the input for the neural network at t0 (time zero, the current frame) was the frame at t0, t-1, t-2 and t-3. It was hoped this would present context to the network as to the direction, velocity and trajectory of the ball and allow it to build features indicating if the ball was moving towards/away from the flippers etc. The user input was also updated to 7 inputs representing press left, press right, press plunger, do nothing, release left, release right, release plunger.

The network quickly began to “do nothing” after holding the plunger, launching the ball on 66-75% of times the ball was in the launch tube. It continued however to hold the flippers held after hitting the ball once and was not fitting any of the new “release” actions. These made up only a tiny minority of the balanced datasets and so further measures were taken, adding the class\_weight parameter to the .fit() call. This allowed instances which appeared less in the dataset, such as plunger held down, to be given a greater weight when being fit to the network.

An extra layer was added to the start of the network in an attempt to create more features. The first two layers were given larger filter sizes which seemed to have a negative effect on the new model.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 3 | 28/10/2018 | 4/11/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Play pinball to generate data | Complete |
| 2 | Handle data normalisation | Complete |
| 3 | Experiment with number of inputs | Complete |
| 4 | Build and train new model, test | Complete |

The building of a series of stacked frames in the last sprint presented a new problem in relation to normalisation. Up to this point in time it had been possible when building a dataset to normalise the frames, save the dataset and read it in to the python package Pandas without trouble. Normalisation involves reducing all values to a decimal between 0 and 1. In memory terms a decimal requires much more space to store than the whole number RGB values, up to a factor of 1000 times more memory. Previously each data point had consisted of just one screenshot but it now consisted of 4 screenshots. Saving these normalised stacks was time consuming and loading them would often cause the application to crash on my home PC.

These stacks were no longer normalised when the dataset was built, instead a batchnormalization layer was added to the network. This took care of the memory problem but was not optimal as normalisation was being carried out on each individual batch of inputs, meaning the decimals arrived at may not have been consistent from one batch to another. This was because it was using the standard deviation of the values across the batch and there was no way of having it divide by 255, the maximum pixel value, which would provide consistency across all batches.

The three extra inputs created in the last sprint appeared to have made the model perform worse. It’s possible that with significantly more training data the network could have fitted well to them but the choice was taken to drop them. Part of the inspiration for this was that features like these would not have been available in the Deepminds Atari paper (Mnih, et al., Playing Atari with Deep Reinforcement Learning, 2013) or in the code of the smallplanet.pinball experiment (smallplanet, 2018), indicating it should be possible to achieve good results without these.

Figure 6.1 – A screenshot of the table converted to grayscale

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 4 | 4/10/2018 | 16/11/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Play pinball to generate data | Complete |
| 2 | Create custom layer for data normalisation | Complete |
| 3 | Experiment with network architecture | Complete |
| 4 | Build and train new model, test | Complete |

The batch normalisation solution in the previous sprint did not seem to be adversely effecting the network however it was a subpar compromise which left open the possibility that it could have negative ramifications further down the line. A satisfactory and permanent solution was achieved by creating a custom keras layer and setting this as the first layer in the network. It would take in the batch of inputs, divide each one by 255 and feed this on to the next layer. This problem was the last major structural challenge in this stage of the project. Due to the changing nature of the amount of outputs and the normalisation problem the previously collected data often had to be discarded. With this challenge solved it would now be possible to ramp up data collection and experiment primarily with the actual network architecture and hyperparameters.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 5 | 21/11/2018 | 30/11/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Play pinball to generate data | Complete |
| 2 | Experiment with network architecture | Complete |
| 3 | Build and train new model, test | Complete |

The 100th game of expert play pinball was played in this sprint providing sufficient data that allowed for diverse behaviour to be seen with changes to the network. A wide array of networks could be trained to high training and test accuracy but would then perform poorly when tested by giving it control of a pinball game. It became apparent that the network, when it was in a “state” (left flipper pressed, left flipper/all buttons released) would stay in that position, as outlined in sprint 4. The network was having difficulty at “transition points” when it should go from button released to button pressed and vice versa. These transitions points, which were represented by the 3 additional actions added in sprint 4 and removed in sprint 5, occurred rarely. It was these few transition points that the network was having trouble identifying, leading to high accuracy but poor game performance.

This showed the limitations of using accuracy or any other metric as a measure of the model’s performance. High accuracy was essential but a low training and validation loss became the new benchmark alongside this.

During this sprint networks with eight hidden layers achieved the best performance. This consisted of a convolutional layer, a maxpooling layer and a dropout layer. These three layers were repeated and fed into a flatten layer, then a dense layer and a dropout layer. The output of this was passed to the output layer.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 6 | 3/12/2018 | 14/12/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Play pinball to generate data | Complete |
| 2 | Experiment with hyperparameters | Complete |
| 3 | Build and train new model, test | Complete |

During this sprint a number of networks were tested using different node and hyperparameter configurations using the network architecture from the previous sprint. Trained on 150 games for 2 epochs the network was able to launch the ball well when tested and would also hit the ball on most occasions but would often keep the flipper extended, causing the ball to become trapped or crash out. By manually pressing a button on the keyboard to return the flipper to its starting position scores of between 100,000-400,000 points could be achieved.

On the 3rd of December a network containing 32 nodes, 3x3 filters and elu activation in each of the convolutional layers, 3x3 maxpooling and a drop out of 0.2 being flattened into a 128 node dense layer with 0.5 dropout was tested. Iterations of this network would be experimented with further but this would prove to be the final network architecture selected.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 7 | 10/1/2019 | 25/1/2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Play pinball to generate data | Complete |
| 2 | Build and train new model, test | Complete |

Games 200 and 250 were played in this sprint. When fitted these led to much better performance by the network and much more consistent ball launches. The network had previously displayed a tendency to trap the ball if it was going down either the left or right outer shoot by extending the flipper early and keeping the ball trapped. These could be undone by manually pressing a button to reset the flipper but often it would activate again and keep the ball trapped, sometimes necessitating a restart. After 250 games the flipper tended not to activate at all in this situations. While this appeared to be a step back the model could now play a full game without human intervention. At this point it was decided to stop generating data from expert play and move onto the next stage of applying reinforcement learning to a network pretrained on these 250 games.

It would be interesting to return at a future point in time and see how far the network could be taken with just expert play, however this would probably take another 250 – 750 games before substantial improvement was seen.

### 6.1.2 Phase 2

Phase 1 resulted in a model that could play pinball to a degree where Reinforcement learning could be applied to it instead of requiring further human gameplay. Copies of all scripts were made and where applicable some of these copies acted as the basis for some of the scripts below .

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 8 | 11/2/2019 | 15/2/2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Take screenshots of score | Complete |
| 2 | Clean screenshots, create training and test sets | Complete |
| 3 | Build and train model to classify score, test | Complete |

To begin the reinforcement learning phase it will be necessary to extract the game score at each time step. Rewards can then be assigned when the score increases from one time step to another. Steeper point increases, such as from completing “missions”, could result in larger rewards to encourage the model to focus on these.

A script was written to take a screenshot of the region of the game interface on which the score is displayed. Each of the 9 spaces on the scoreboard can contain the digit 0 to 9 or can be completely blank. After a number of screenshots had been collected these were broken up into their 9 individual digits and each digit was placed in a corresponding folder. 18 instances of each digit/blank were collected, 15 of each were added to a training set and the remaining 3 to a test set.

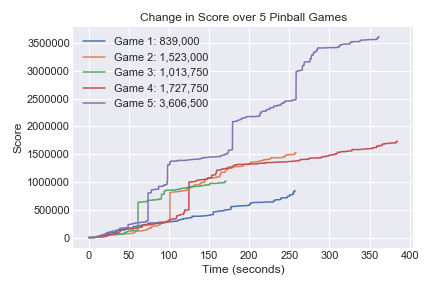
A model was trained to 100% accuracy, possible only because there is no variation whatso ever between the images of each digit. A script was written that would, during gameplay, take a screenshot of each of the individual digits in the scoreboard, feed them to the trained model and append the result to a variable, returning this variable when all 9 positions were examined. This proved to be slow and often the score would change during this process. The script was changed to instead screenshot the entire scoreboard and save it to a file. After the games conclusion the scores could then be converted from image to numerical values for use with a future reinforcement model.

Figure 6.2 – Change in score over time in 5 games played by a human

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 9 | 18/2/2019 | 22/2/2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Retrain model for additional inputs | Complete |
| 2 | Write scripts to play and process games | Complete |
| 3 | Test scripts | Complete |

The final model trained in sprint 9 could take in only four inputs; left and right flipper, plunger or do nothing. For the reinforcement phase three additional inputs would need to be added; tilt right, left and down. The entire network architecture would need to be changed to accommodate this but it was quickly found that doubling the number of nodes in each layer led to a model that could achieve similar performance to the one in sprint 9. The same 250 games were used to train this new model by appending three 0’s to the target array for each instance.

A new file was written to play the pinball game using the new model. It needed a function to apply the Epsilon Greedy algorithm and had to be able to take a screenshot of the table, the score, the Q-values returned for each button and the button pressed. To to the inability to automate this stage code was added so that a human monitor could press the Q button to give a penalty and code was also added to end the training process.

Another script was written that would take in this data and use it to build a data frame. The score image recogniser written in the previous sprint would convert the score at each timestep to digits and these could be compared, with rewards being issued where the score increased from one timestep to the next. The penalty issued by the human monitor would be used as a negative reward. All of this information could be used to apply the Bellman equation and retrain the model. This new, updated model would then be saved out to be used in the next training run.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 10 | 23/2/2019 | 10/3/2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Train and test model | Complete |
| 2 | Experiment with reward shape | Complete |

The entire process of allowing the pretrained network to play games of pinball itself and then updating the network using reinforcement learning was now in place. It was still necessary to find a reward shape that would elicit the best results from the network. Early on the network was given a reward of 0 at each timestep, 1 for an increase in score and -10 for a drain. Later different positive rewards between 1 and 10 were given depending on how large an increase in score occurred, the hope being that the network would learn to focus on higher scoring targets. Eventually the reward shape of 0 at each timestep, 0.5 for an increase in score and -1 for a drain was implemented. Normalising the values allow the network to converge faster, however the model was now training to maximise the amount of targets it hit instead of maximising score.

The 7 input model was abandoned towards the end of this sprints, which was a great disappointment, and the 4 input network was used instead. This was due to difficulties launching the ball. The human expert whose experience replay was used to train the network had always launched the ball successfully each time. However, in the reinforcement stage, it is possible that the Epsilon greedy algorithm would choose a random move instead of the best move and would choose to tilt the table, knocking the ball off centre before launch. As the network had never been exposed to a launch being made by the human expert with the ball of centre the network would do nothing. As the games were being played in real time and required the observation of a human monitor it would take too long to allow the network to train itself to launch from this position using reinforcement learning. Also, as the network was pretrained, a low value of epsilon was used, 5% or less. As such it was possible that the network would never learn to launch in this circumstance.

|  |  |  |
| --- | --- | --- |
| **Sprint Number** | **Start Date** | **Finish Date** |
| 11 | 11/3/2019 | 17/3/2019 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Train and test model | Complete |
| 2 | Write script to analyse performance | Complete |

After implementing the changes made in the last sprint the process of training a network was began again. As there is no way of automating the process at the moment it took a considerable amount of time to go through each training loop of having a game play out and then using it to update the network. 130 games of training data were provided by the end of this sprint.

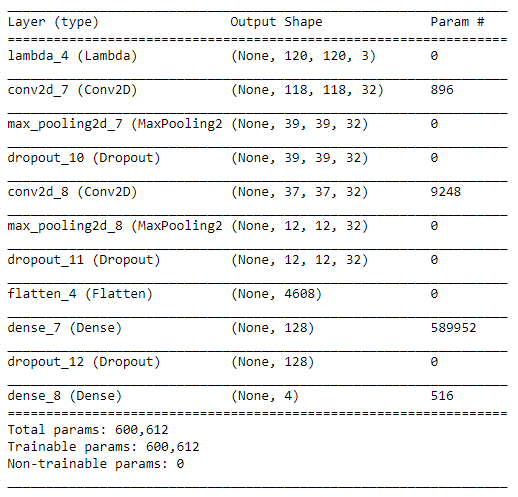
A script was written that would take in data from each game and plot it to a series of graphs, showing the maximum score per game, the number of total rewards per game and the number of rewards per ball played. This could be used to monitor the performance of the network.

Figure 6.3 – Summary of the final network architecture

# 7. Findings & Conclusions

Through the use of imitation learning, whereby an agent was trained using examples from expert human gameplay, the agent was able to play a game of pinball. The agent could launch the ball onto the scoring area and would often activate a flipper to drive the ball up the table and away from the drain. However, the agent required a substantial amount of data before it could achieve this. The final imitation learning model was trained on 250 games and still performs vastly below human level, often failing to hit the ball while it rolls down a flipper towards a drain (Nagle, 2019). Providing further training data may improve the performance of this agent.

The final architecture settled upon for the Neural Network was surprising uncomplex. It is possible that a more advanced architecture might achieve a better performing agent but in general this architecture seemed more than sufficient. Generating and providing more data could be achieved by asking multiple expert humans to submit their gameplay data. The agent with then, at each time step, predict the policy of the group average instead of that of any one specific player. It is unclear how much more data would be required to construct an agent capable of equalling or beating the average score of a human participant. This is dependent on the task at hand; for example 15 hours of expert gameplay was required to train an agent to play Mario Kart to a stage where it could achieve a race winning time on a number of levels (Bling, 2017).

Successfully launching the ball onto the scoring area was a major achievement which would be near impossible by an agent acting randomly. It would take thousands of iterations for an agent acting in such a fashion to do so and thousands more before that behaviour was locked in using reinforcement learning. As such, once the imitation learning agent was able to launch the ball consistently and hit it with the flippers on most occasions the first phase of the ended.

The initial intention with the Reinforcement Learning phase was to examine if the agent could improve on its performance and learn new behaviours which had not been exhibited in the training examples of the human expert. The agent was expanded to execute new functionality, namely the ability to nudge or tilt the table, functionality which had not been shown to it in the human gameplay.

The agent was now adapted so that it could tilt or nudge the table, behaviour which had not been demonstrated by the human expert in their gameplay. However this line of investigation had to be abandoned. The human expert had launched the ball onto the scoring area from the same position in all 250 of their games but in the Reinforcement Learning phase it was possible for the agent to tilt the table before launching the ball. The agent no longer “knew” how to launch the ball once it had been moved off the centre of the launching pin. To correct this it would be necessary for the human expert to provide examples of launching the ball from positions where it is off centre; by first tilting the table before launching. This problem had not been anticipated in the initial training phase and there was insufficient time to provide the necessary examples to the base model.

As such it was necessary to remove the functionality to tilt the table and instead observe if the model could improve in performance over the base model. Before doing so however it was observed that the agent was exploring this functionality and the network was changing its Q-value for executing this functionality. There are certain conditions in which, without tilting, it is impossible to prevent the ball from draining, an event that provides a large negative reward to the network. It was the aim of this project to examine if, over time, the network would penalise pressing any other button or doing nothing in these cases and learn, randomly, that tilting the table led to states where the network received a reward instead of a state where the network was penalised. As mentioned above, it would be necessary to provide instances of the ball being launched from an off centre position to account for the Reinforcement Learning agent randomly tilting the table before launching to overcome this. Another solution would be to introduce a timer and a method of checking if the ball has been launched. If a certain time elapses a negative reward would be given, the network retrained on this information, and the game restarted. Over time this would penalise tilting the table before the ball has launched or doing nothing while the ball is still in the launch chute.

The agent played 130 games and following each on the network was retrained using the data from that game. 130 games is far too few to show any substantial progression though some metrics did show an upward improvement. Each game took an a great deal of effort to process as a human observer had to monitor the game and manually assign a penalty to the agent by pressing a button when the ball drained. One solution to that would have been training another neural network to determine if the ball was still in play or not. This would have added considerable processing overhead, slowing down the rate at which game data could be captured, and was complex enough to have been an entire separate project by itself. It is an area for future study.

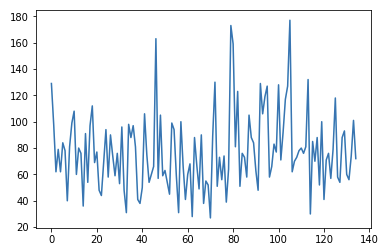
The agent, over the limited amount of self play achieved, showed a significant change in its behaviour in comparison to the base model trained on 250 human expert games. The agent seemed to be “training” itself out of certain negative behaviour, such as letting the ball slide slowly over the flipper and drain. This was achieved by the agent randomly pressing the flipper while the ball was rolling over it instead of going with the networks recommendation of doing nothing. As mentioned 130 games was far too few to show a consistent correction to this problem but shows that the network was improving on the base network generated from human gameplay.

Figure 7.1 – Total rewards per game in the reinforcement phase, showing a slight upward trend.

Also of interest is the “spamming” behaviour which the network has learned, where it constantly presses on the flipper buttons. This is due to the network randomly pressing a button while the ball is heading towards a target. When hit the target gets a reward which is decayed over all actions leading up to that point. The random press of a flipper when the ball is nowhere near it therefore get a reward. This behaviour is similar to a number of other agents trained entirely on reinforcement learning, such as the Breakout agent implemented by Deepmind (Mnih, et al., 2013) or the Atari Video Pinball agent implemented using code created by Nathan Sprague (eldubro, 2017). This behaviour shows the limitations of using reinforcement learning for control tasks and the difficulty of assigning credit to actions over time. Correctly attributing a reward to a button press that resulted in an increase in score (preventing the ball from draining) and not to a random button press is a difficult task which has not yet been addressed in the field of reinforcement learning.

This project has highlighted the large difficulty in using Reinforcement Learning where there is no forward model, no ability to speed up the simulation or know way of accessing the simulations internals, such as the game score and ball position. A forward model would allow the agent to train as it is playing the game instead of needing to do these separately. A forward model is a process whereby the network can also return the next game state and this can be used in the Bellman equation to calculate the reward for the current state. As this does not exist for 3D pinball for Windows, Space Cadet, the game must play out in full before the network can be retrained. The inability to speed up the simulation, as was done with StarCraft II so that the Deepmind agent trained on it could play over 200 years worth of games in a few weeks, and the fact the game internals can not be accessed limit the ability to automate the data collection. It is for this reason that research into Artificial Intelligence in games has been limited to a number of environments which offer these advantages, such as VizDoom for Doom and OpenAI Gym for games such as Breakout, Video Pinball and Super Mario Bros.

This project replicates the considerations that must be made when trying to implement Reinforcement Learning in an area which does not have an advanced environment prebuilt. In such cases it may be best to forgo Reinforcement Learning and instead train an agent on expert human input, possibly on the input of multiple humans to speed up collection of and diversify the training data. Further research could be done into comparing the performance of an agent in a game using data from a single user and data from a wide selection of users.

The Reinforcement Learning phase agent has shown that it can adapt the behaviour of a human expert. Had an attempt been made to train an agent to play Space Cadet using only Reinforcement Learning it is possible that after hundreds of games it would still be unable to launch the ball or activate the flippers. Bootstrapping a model with expert human examples can be beneficial in instances where the limitations of this project are present. A further implementation is in areas where Reinforcement Learning has made limited success, such as in stock market predictions. An examination could be made of training a network on the buy and sell orders of professional human traders and using this to bootstrap a Reinforcement Learning agent. The agent would make immediate progress through the use of heuristic rules used by humans but would also be free to explore and learn new behaviour. Training should not be limited by large mistakes or hurdles at the start, such as trying to launch the ball in an accurate pinball game (it is much simpler to launch the ball in older video pinball games than it is in Space Cadet).

Due to the necessity of a forward model, a faster simulation which would allow for an attempt at pure Reinforcement Learning, and access to the games internals, it would be interesting to suggest Space Cadet as a benchmark for AI study. The game is advanced enough to challenge a robust agent, especially with complex missions, yet simple enough in that it doesn’t contain multi-ball or moving parts. Julian Togelius has operated a number of competitions in the past in the area of self driving cars and Super Mario Bros (Yannakakis & Togelius, 2018) and has suggested competitions as a way of furthering AI research in general and specifically in the field of games (Togelius, 2018). The creation and promotion of a competition to build an agent to play Space Cadet would necessitate the creation of an environment that would overcome many of the problems faced in this project. Alternatively, using a game engine such as Unity which has its own Machine Learning tools, agents could be submitted to a competition to play on tables created specifically for that competition. There are general similarities from table to table that allow professional human players to score well on tables they have never seen before and replicating this for a virtual competition could see pinball contribute to the study of General Artificial Intelligence.

# 8. References

Arulkumaran, K., Cully, A., & Togelius, J. (2019). *AlphaStar: An Evolutionary Computation Perspective.*

Bling, S. (2017, November 5). *MariFlow - Self-Driving Mario Kart w/Recurrent Neural Network.* Retrieved from Youtube: https://www.youtube.com/watch?v=Ipi40cb\_RsI

Drillot, N., & Petry, J. L. (Directors). (2016). *Wizard Mode* [Motion Picture].

eldubro. (2017, September 29). *Deep Q Network learning to play Video Pinball.* Retrieved from Youtube: https://www.youtube.com/watch?v=l5o429V1bbU

Haykin, S. (1999). *Neural Networks: A Complete Foundation.* Pearson.

Hebb, D. O. (1949). *Organization of Behavior.*

Irons, M. L. (2005). *Space Cadet Pinball Strategy*. Retrieved from Mark Irons' Home Page: http://www.rdrop.com/~half/General/GameTips/space.cadet.html

LipowitzFilms. (2018, March 10). *Neural Network Pinball.* Retrieved from Youtube: https://www.youtube.com/watch?v=DBAkWK8P5SM

Luger, G. F. (2002). *Artificial Intelligence: Structures and Strategies for Complex Problem Sovling.* Essex: Pearson.

McCulloch, W. S., & Pitts, W. H. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 115-133.

Metcalf, A. (2011). *Pinball AI.* Retrieved from YouTube: https://www.youtube.com/watch?v=NCyTAzCbbuw

Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.*

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusa, A. A., Veness, J., Bellemare, M. G., . . . Hassabis, D. (2015). Human-level control through deep reinforcement. *Nature*.

Nagle, J. (2019, May 7). AI Pinball Agent. Retrieved from https://youtu.be/fCpSWevygBg

neoseeker. (2003). *weak flippers*. Retrieved from neoseeker: https://www.neoseeker.com/forums/3624/t191286-weak-flippers/

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann Publishers.

Parodi, F. (2015, September 1). *Evolutionary Algorithms | AI playing 30000 frames of Pinball.* Retrieved from YouTube: https://www.youtube.com/watch?v=yyrEMOAwxic

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017, May 15). Curiosity-driven Exploration by Self-supervised Prediction.

Rohlfshagen, P., Liu, J., Perez-Liebana, D., & Lucas, S. M. (2018). Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game. *IEEE Transaction on Games*, 233-256.

Rosenberg, A., & Somappa, G. (2017). Pixel to Pinball: Using Deep Q Learning to Play Atari.

Rosenblatt, F. (1957). *The Perceptron - A Perceiving and Recognizing Automaton.*

Ruben, A. (2018). *Pinball Wizards: Jackpots, Drains and the Cult of the Silver Ball.* Chicago, Illinois: Chicago Review Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*.

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach.* New Jersey: Pearson.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., . . . Hassabis, D. (2017). *MasterIng Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.*

smallplanet. (2018). *SmallPlanetiOS.* Retrieved from Github: https://github.com/SmallPlanetiOS/smallplanet\_Pinball

sodeepdude. (2017). *Comparison of human scores and “human scores” in Atari*. Retrieved from sodeepdude: https://eject.com.au/sodeepdude/comparison-of-human-scores-and-human-scores-in-atari/

sodeepdude. (2017). *DQN learns video pinball*. Retrieved from sodeepdude: https://eject.com.au/sodeepdude/dqn-learns-video-pinball/

Sprague, N. (2017). *Theano-based implementation of Deep Q-learning*. Retrieved from Github: https://github.com/spragunr/deep\_q\_rl

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning 3*.

Sutton, R. S., & Barto, A. G. (2017). *Reinforcement Learning: An Introduction.* Cambridge, Massachusetts: MIT Press.

Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of teh ACM*.

Togelius, J. (2018). *Playing Smart.* Cambtidge, MA: MIT Press.

Turing, A. (1950). Computing Machinery and Intelligence. *Mind*, 433-460.

Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games.* Springer.