Supervisor: Pat Doody

Second Reader: Rob Sheehy

Author: jonathan Quirke

May 7, 2019

Can a MAchine Learning Algorithm be used to navigate a level of super mario bros.

B.Sc. (Hons) in Computing: Software Development

Contents

[1 Abstract 3](#_Toc9850547)

[2 Introduction 4](#_Toc9850548)

[3 Artificial Intelligence 5](#_Toc9850549)

[3.1 Introduction 5](#_Toc9850550)

[3.2 Neural Network 5](#_Toc9850551)

[4 Neuroevolution 7](#_Toc9850552)

[4.1 Introduction 7](#_Toc9850553)

[4.2 Neuroevolution with Augmented Topologies (NEAT) 7](#_Toc9850554)

[4.2.1 Algorithm 8](#_Toc9850555)

[4.2.2 Crossover 8](#_Toc9850556)

[4.2.3 Mutation 9](#_Toc9850557)

[5 Convolutional Neural Networks 10](#_Toc9850558)

[5.1 Introduction 10](#_Toc9850559)

[5.2 Image Classification 10](#_Toc9850560)

[6 Reinforcement Learning 11](#_Toc9850561)

[6.1 Introduction 11](#_Toc9850562)

[6.2 Q-Learning 11](#_Toc9850563)

[6.3 Deep Q-Learning 13](#_Toc9850564)

[6.4 Double Deep q-Learning 14](#_Toc9850565)

[6.5 Duelling Double Deep Q-Learning 15](#_Toc9850566)

[7 Technologies 17](#_Toc9850567)

[7.1 Tensorflow 17](#_Toc9850568)

[7.2 Keras 17](#_Toc9850569)

[7.3 OpenAI Gym 17](#_Toc9850570)

[7.4 NES-PY 17](#_Toc9850571)

[8 Methodology and Design 18](#_Toc9850572)

[8.1 Key Research 18](#_Toc9850573)

[8.2 Research Question 19](#_Toc9850574)

[8.3 Proposal 19](#_Toc9850575)

[8.3.1 NEAT 19](#_Toc9850576)

[8.3.2 Q-Learning 21](#_Toc9850577)

[9 Applied Research 22](#_Toc9850578)

[9.1 Flappy Bird Neat Algorithm 22](#_Toc9850579)

[9.2 Super Mario Bros. OpenAI 23](#_Toc9850580)

[9.3 Image Classification 24](#_Toc9850581)

[9.4 Super Mario Bros. OpenAI using an image classifier 25](#_Toc9850582)

[10 Implementation 27](#_Toc9850583)

[10.1 Sprints 27](#_Toc9850584)

[10.1.1 Sprint 1 – Data Collection and Environment Creation 28](#_Toc9850585)

[10.1.2 Sprint 2 - NEAT 40](#_Toc9850586)

[10.1.3 Sprint 3 – Q-Learning 43](#_Toc9850587)

[11 Conclusion 55](#_Toc9850588)

[12 References 56](#_Toc9850589)

# Abstract

Finding a way to navigate a Super Mario Bros. Level using a Machine learning Algorithm. Research began into looking at NEAT as a viable training methodology to create an agent for navigation of the levels. NEAT needs either a extremely long training time or to be run in parallel to work. This turned out to be an issue with regards to hardware resources available. This was especially difficult as the project was to use Keras or TensorFlow models as the brain of the agent within NEAT.

The decision was to move to Q-learning which seemed more feasible and had numerous studies done. Q-learning as a concept is closer to that of how humans learn. This meant that the learning process would simulate how a human with no knowledge may learn to play Super Mario Bros.

The Main finding from the project was the time needed to train an agent was very dependent on the algorithm used. Duelling Double Deep Q-Learning with prioritised experience replay turned out to be the best way to train as it can train an agent in a few hours rather than several days. This is due to the way the memory of good experiences is held and replayed to the network to train.

# Introduction

Machine learning particularly reinforcement learning or neuro-evolutionary types of machine learning are particularly interesting as they do not need to be trained using any gathered dataset they create their own data by trial and error.

Using machine learning to solve problems such as navigating a Super Mario Bros. level is a good application for reinforcement learning or neuro-evolution. The process of learning is unsupervised to the point where it learns through its own trial and error. While neuro-evolution is like survival of the fittest reinforcement learning is like learning as a human.

Reinforcement learning or more specifically Q-learning was chosen as the method to use for the project. The training involved the use of OpenAI gym training environments which were then used with a number of Q-learning algorithms to find a final most effective algorithm.

The types of Q-learning used were:

* Deep Q-learning
* Duelling Double Deep Q-learning

Both of these models have their place while they both will not work for the same problems to the solved.

# Artificial Intelligence

## Introduction

Artificial Intelligence is the broad term used to describe how a computer system can appear to have intelligence. Artificial Intelligence is also referred to as machine intelligence, this is intelligence shown by computers, and this is different than the intelligence shown by humans and animals. In the field of computer science Ai is defined as the research and study of intelligent agents. These agents are able to evaluate an environment and make decisions based on the inputs. In the main stream the term Artificial intelligence is applied to systems that mimic the intelligence of humans such as learning and problem solving. (Russell & Norvig, 2009)

## Neural Network

Neural networks are made up of neurons. These neurons can take an input compute said input and give an appropriate output. A single neuron system is called a perceptron. This takes in inputs and gives an output. The output is calculated using the supplied weights from the inputs. These are useful in the representation of a dynamic function. Such as an addition function. Two plus three equals 5. The perceptron can then be thought this through learning and then dynamically create an addition function (Shiffman, 2012).

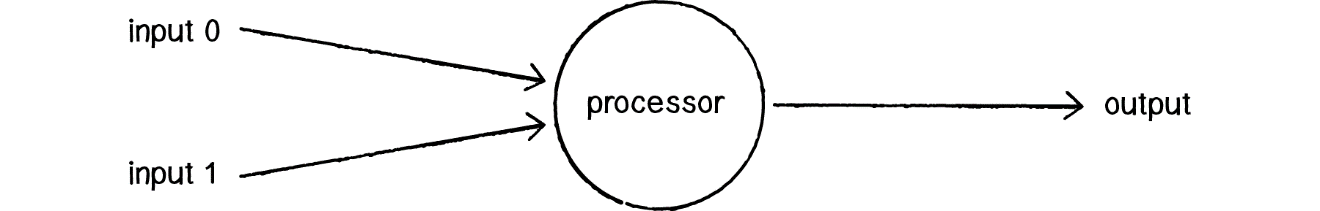


Figure 3‑1 Representation of a perceptron (Shiffman, 2012)

Neurons make up the nodes of the network. The connection between these nodes provides the Neural network a way to communicate with each Neuron. This network then can be used to solve problems through the use of weighted connections. These weights help with to create a network for specifics problem solving (Shiffman, 2012).

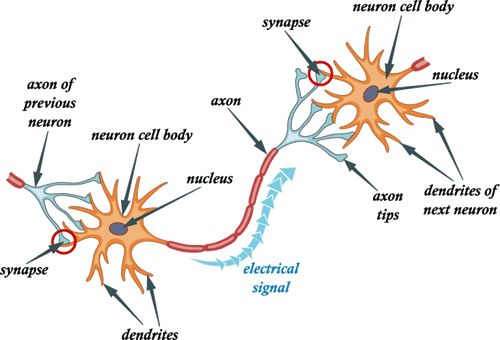


Figure 3‑2 Neuron cell (Holländer, 2018).

The representation in fig.2-3 illustrates how a neuron looks within the brain. The neurons in a neural network are a representation of the neurons located in the human brain. A neural network is computer sciences best attempt at the creation of an Artificial brain using the same kind of building blocks used within the brain.

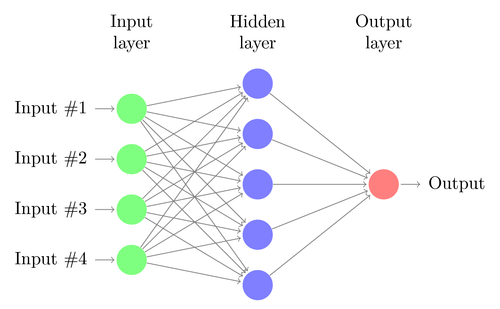


Figure 3‑3 Shows a simple Neural Network Topology (AppliedAI, 2018).

# Neuroevolution

## Introduction

Neuroevolution is a part within of the broader terms Machine Learning and Artificial Intelligence. The way Neuroevolution differs from more traditional Artificial Intelligence is through the use of an evolutionary process. Neuroevolution uses a set of algorithms that are used to simulate evolutionary response like the response in the human brain (Stanley, 2017). The difference being within the computers system these events create a neural network of pathways and nodes. This is like that of the mapping of neurons and connections in the human brain (Schiappa, 2017).

## Neuroevolution with Augmented Topologies (NEAT)

This describes how a Neuroevolutionary network augments the neural networks topology to suit the problem it is trying to solve and does so as it learns. This involves both changing the structure of the network as in the number of nodes, the connections between those nodes and change weights associated with each of those connections (Lin Chen, 2006).

This type of network uses the survival of the fittest mentality. In this way it can evolve the topology of the neural network to serve the problem you have presented it with. (Kenneth O. Stanley, 2002) This gives the chance for a more efficient neural network to be formed. This evolution takes the form of generations. Each generation takes an evolutionary step through these steps are different depending on the approach. One approach involves breeding which would take the best 2 networks formed using a fitness value to decide which networks are performing best. These 2 networks are then sent into a crossover algorithm to create offspring who are also given a random mutation to differentiate between them. This type of neural network is modelled after biological evolution. As with biological evolution the offspring of 2 parents do not always come out with the same DNA (Salih & Moshaiov, 2016).

### Algorithm

In a traditional neural network, the topology is created by the developer. The weight values on each connection are then learned through a training procedure. The situation ends up being trial and error as the developer of the neural net would need to change the topology of the network to try and find the optimal one.

NEAT does not suffer from the trial and error portion of having to change the topology of a neural network. NEAT as a way of training a neural net is an example of a topology and weight evolving artificial neural network (TWEAN). This type of topology is an attempt at not only learning the weight values through training but also an appropriate topology to be used by the neural network.

For the neural network to be used as a genetic algorithm(GA), The NEAT uses a way of keeping track of each piece in the network such as Connections, Weights and Neurons this is called a direct encoding scheme. This is different than the current way Keras works to create a network as it does not need the user to stipulate each connection and node. This is a type of indirect encoding and is used to give a more compact representation of a neural network

NEAT begins with a simple network topology which is like a perceptron. With only one input and output neurons. As the training progresses like in evolution the networks complexity will change from that simple perceptron to possibly a multilayered neural network. The complexity of the network will grow through the adding new neurons and the adding of connection between those neurons along with connecting neurons that may have not had any connections. (Stanley & Miikkulainen, 2006)

### Crossover

In genetic algorithms crossover can be also called recombination. It is a genetic operator used to combine the genetic information of two parents to create new offspring. This is one way to stochastically generate new populations. This is like how the human race reproduces. Typically, once the population is generated the offspring are then mutated using some type of mutation function (Obitko, 2011).

### Mutation

Mutation as a genetic operator is used to maintain diverse genetics for each generation. This is similar to how biological mutation happens. Mutation changes one or more of the values that make up the genetic values from an initial state (Obitko, 2011).

# Convolutional Neural Networks

## Introduction

Convolutional neural networks (CNN) are a type of neural network that are mostly used within some type of image analysis system. The neurons structure resembles a model of a biological visual cortex. CNN have many uses such as in image and video recognition, image classification and medical image analysis (Oh & Lee, 2018).

## Image Classification

Image classification is taking an image and identifying what the image is of. A classic demonstration of this is using a CNN to tell the difference between dogs and cats. To train these CNN hundreds of images of each of the classifications are needed (Computerphile, 2016).

CNN use processes such as Sobel edge detection and corner detection to remove information from the image leaving behind features to be used by the network. Each of the nodes in each layer is connected to the nodes from the layer before it and the layer after it. Each layer abstracts the image a bit more combining Sobel edge detection and corner detection based on the weights it takes a bit from each node and passes the them along. Once it reached the output node it gives a prediction based on this prediction the networks weights will be adjusted to get a better result on that images classification (Computerphile, 2016).

# Reinforcement Learning

## Introduction

Reinforcement learning is an area of machine learning along with supervised learning and unsupervised learning. (Kaelbling, et al., 1996).

Reinforcement learning is based on the rewards and punishments. The agent will receive a reward if the action performed is good and a punishment if the action was bad. This is based on how human beings learn for example if a person was to burn their hand on an oven they would know not to put their hand on the oven when it is hot. This works the other way as well with a person receiving money for doing a job correctly. This incentivises the person to do a good job and stay away from hot ovens (Gross, 2010).

In machine learning these rewards and punishments are just numbers normally with a range from negative to the positive e.g. -15 to 15.

## Q-Learning

Q-learning can be done with the use of a Q-table which for small environments keeps track of the entire environment state. This generally stores the environment state, action preformed and reward (Kaelbling, et al., 1996). This is then used to decide the best actions to get to the goal. As in the Mouse example.

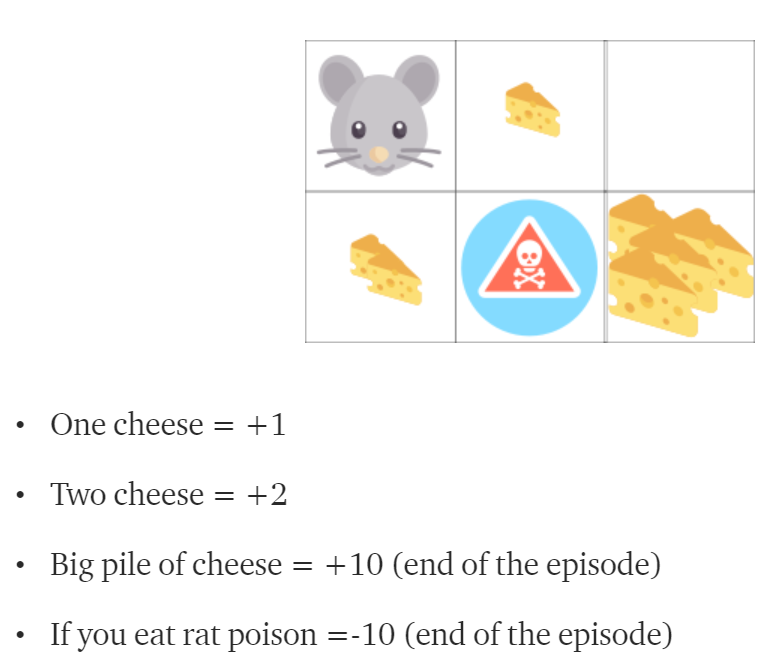


Figure 6‑1 Example game for Q-Learning with Q-Tables (Simonini, 2018)

Figure 5-1 shows the game the agent is trying to get to the big pile of cheese without hitting the poison which ends the game. To begin with the game is played using random exploration to find the best route. The rewards for collecting the cheese, the poison and Big pile of cheese are listed based on these the agent must learn what is the best path to maximise the total reward. It does this by following the below process.

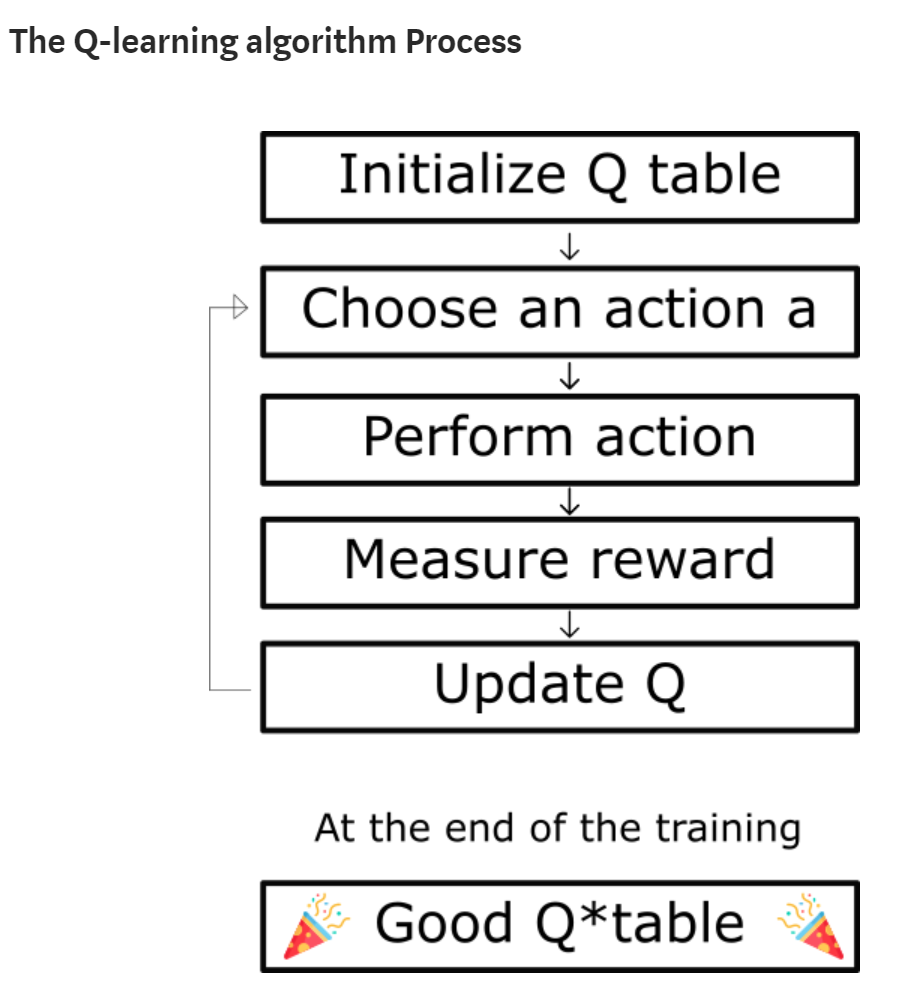


Figure 6‑2 Q-Learning Algorithm process (Simonini, 2018)

Following this process, the Q-table gets updated every time an action is preformed using an update formula. This is based around the action and reward for a given state.

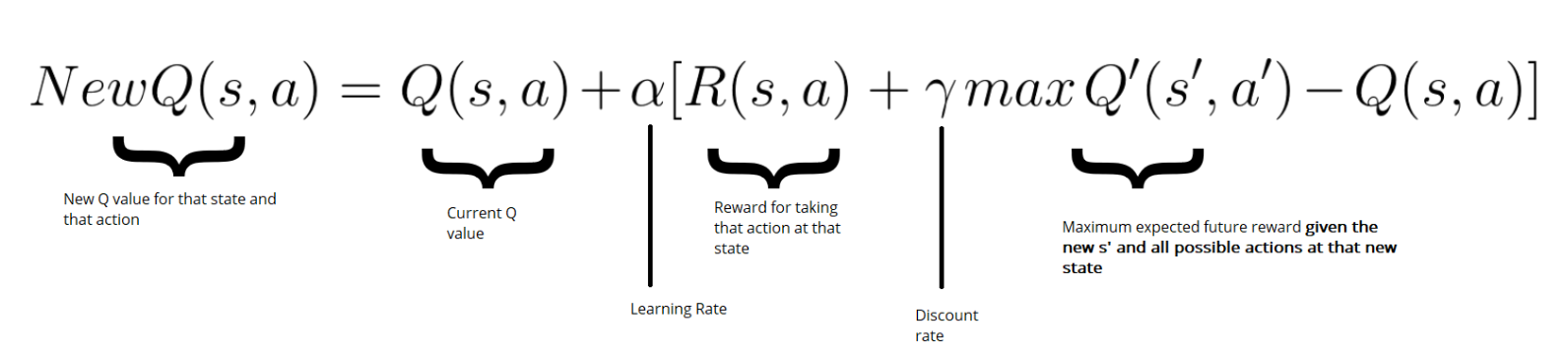


Figure 6‑3 Q value calculation (Simonini, 2018)

Based on the output of the figure 5-3 will be used to update the Q-table for the given action at the current state.



Figure 6‑4 Updated Q-Table (Simonini, 2018)

From the output in Figure 5-3 the table in Figure 5-4 gets updated to reflect how good of an action that was taken was to the goal. This will accumulate at each state and action until an optimal route has been found (Simonini, 2018).

## Deep Q-Learning

Deep Q-learning (DQ) is when Q-learning using a deep neural network instead of a Q-table this is especially a good in environments that have an environment with a huge number of states such as a video game or robotics where the states are possibly infinite. These are a more typical situation where Q-Learning would be used for real world applications (Simonini, 2018).

DQ uses a deep neural network as way to approximate Q-Values for the possible actions the agent could take. From this the biggest value that is returned is used as the current best action taken (Zhu, et al., 2017).

DQ uses exploration as a way to learn from the actions it takes this involves doing a series of actions and recording it into a memory. This can then be played back to the neural network later for training. Exploration usually starts at 100% random exploration this has a decay rate set in the parameters of the algorithm. Generally, the rate of exploration will get capped at a lowest point usually around 1%. This allow for new learning to continue even after the exploration should be done.

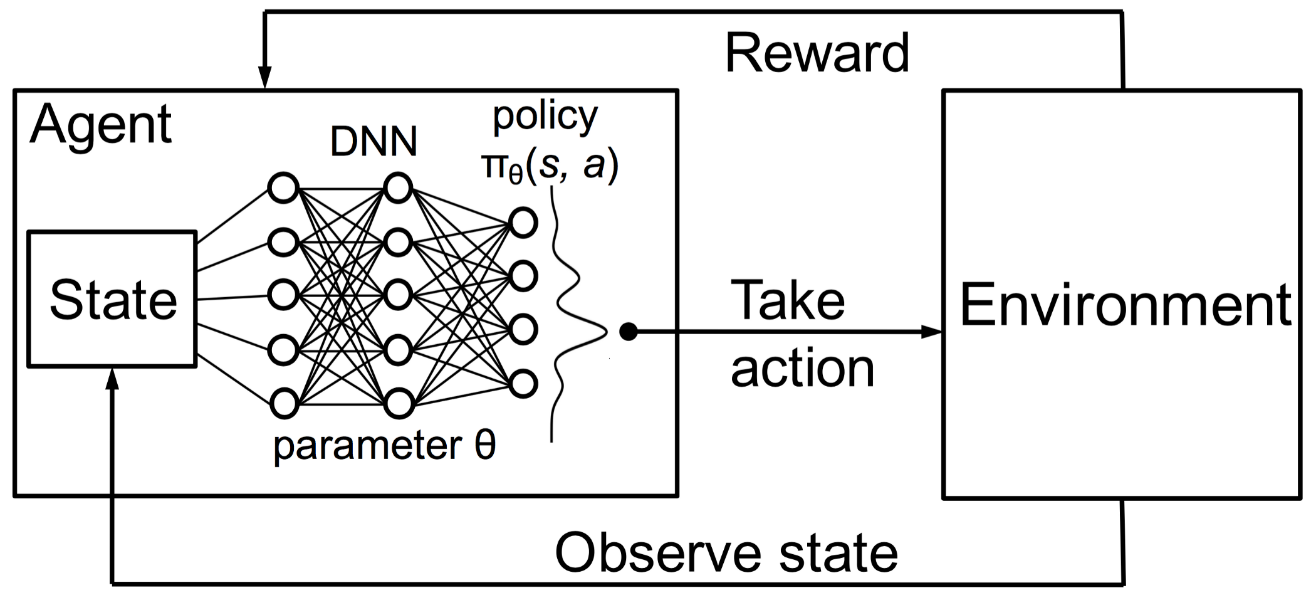


Figure 6‑5 Flow of a Deep Q-learning set up (Trivedi, 2018)

## Double Deep q-Learning

Double Deep Q-learning uses two networks instead of using one network to estimate the given Q-value this allows for the estimation of Q-values using both neural networks.

Double Deep Q-Learning (DDQ) is used to handle a problem with overestimating Q-values. This is a problem with calculating the TD target as it is not known that this is the best action for the next state is the action represented by the highest Q-value.

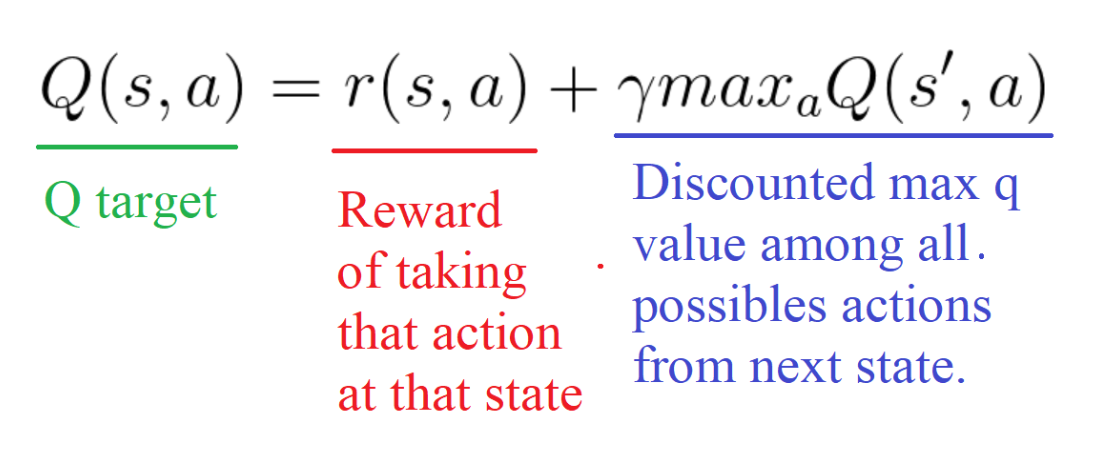


Figure 6‑6 TD Target calculation (Simonini, 2018)

The accuracy of the Q-value is based on the what actions have been performed and the what neighbouring states have been explored. While this is the case DDQ can mitigate this by using two networks to calculate the action for the next state using one network and calculate the target Q-value for taking that action at the current state with the other (Hasselt, 2010).

## Duelling Double Deep Q-Learning

Duelling Double Deep Q-Learning (DDDQ) is an improvement on DDQ. This is an improvement on network topology where the network splits to do two calculations then is brought back together using a special aggregation layer. The calculations are used once aggregated to give a Q-value like the previous Q-Learning algorithms. This uses the formula in Figure 5-7 below (Simonini, 2018).

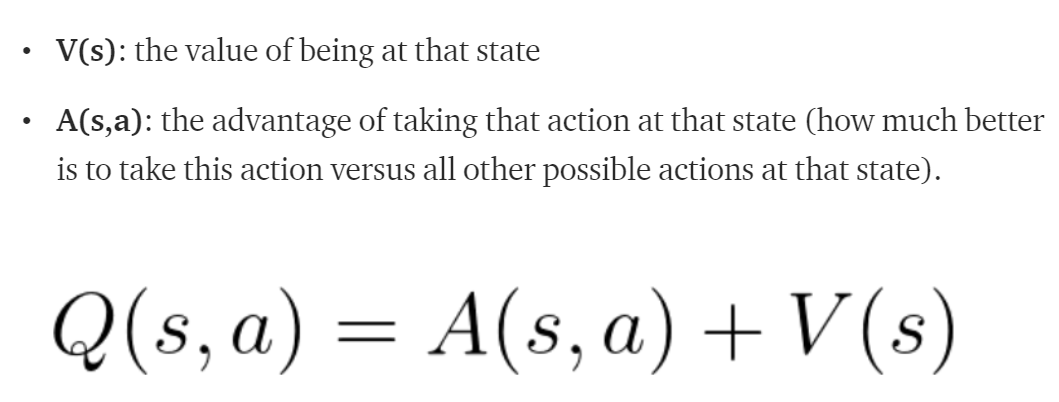


Figure 6‑7 Calculation of a Q-value for DDDQ (Simonini, 2018)

The split within the network calculates A (s, a) at one side and V (s) at the other and is then brought back together to give the final Q-value using an aggregation layer (Simonini, 2018).

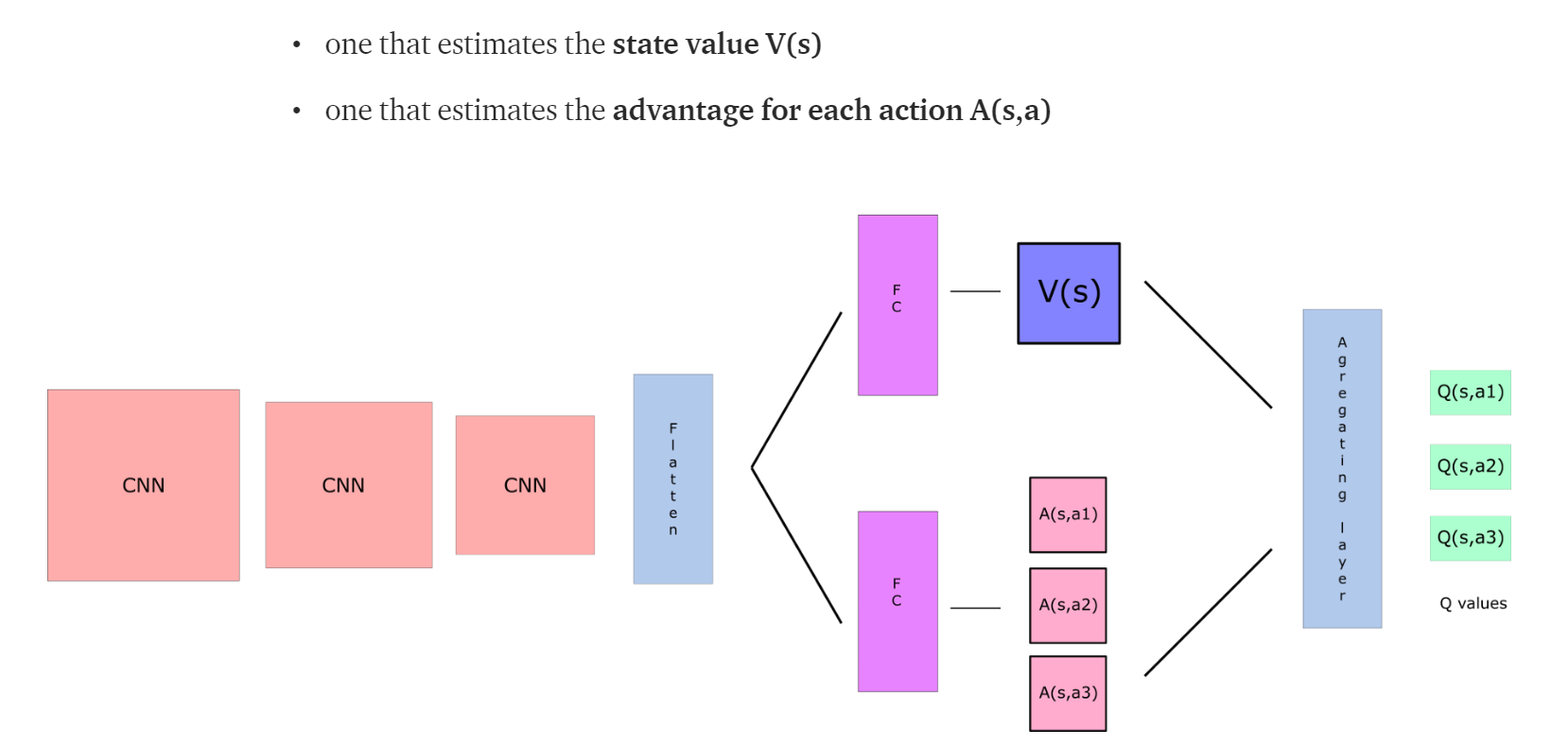


Figure 6‑8 DDDQ network topology (Simonini, 2018)

The reason to calculate the Q-value this way is to decouple the estimation of Q-values allowing the network to learn which states are valuable without having to learn the outcome of all the actions at each of the states (Simonini, 2018).

# Technologies

## Tensorflow

TensorFlow is an API used to aid in the creation of artificial intelligence (AI). TensorFlow is an open source AI created by the Google Brain team. TensorFlow using a data flow graphs for numerical computation. The graphs nodes are each a mathematical operation. The graphs edges are represented in multidimensional data arrays these are known are tensors. These tensors represent the flow between nodes. With how flexible the architecture of TensorFlow is, it can be deployed to multiple CPUs or GPUs in many different hardware configurations (Google, 2018).

## Keras

Keras is a high-level AI API. Keras is mainly used on top of TensorFlow it helps with rapid prototyping and experimentation of AI. Keras is written in Python. While being very easy to get going with Keras can be used to make very complex neural networks (Keras, 2018).

## OpenAI Gym

OpenAI gym is a benchmarking application for general intelligence. OpenAI gym allows for a framework where multiple different intelligence types can be plugged in and used to perform a function. This in turn will give out a metric which can be used to gauge the performance of the different AI types (Open AI, 2018).

## NES-PY

NES-PY is an implementation of Open AI gym. This uses Open AI gym to allow the use of Nintendo Entertainment System (NES) games for benchmarking of general intelligences with these games (Kauten, 2018).

# Methodology and Design

## Key Research

Research began in the previous chapters looking at how an artificial intelligence (AI) can be used to solve simple problems all the way up to how a neuroevolution of augmented topologies (NEAT) AI can be used to solve complex problems. While NEAT is the focus of this project learning how traditional AI are used and how they are designed will create a better understanding of what is happening while the NEAT AI is training. Some of the most interesting areas looked at was when it came to the multiple different types of AI. All these types of AI have different purposes such as a convolutional AI which is most often used for image classification. This kind of network will be useful in the creation of the NEAT algorithm as for this project the network will be taking in an image and doing some type of classification to it. The classifications would be the button presses in the game such as jump, run, left, right, up and down.

Using NEAT will involve many different aspects of neural networks such as the use of mutations and crossover which are used in Neuroevolution or genetic algorithms (GA). Neuroevolution would encompass NEAT, this is because Neuroevolutionary AI and GA tend not to change the topology they start with. Neuroevolution and GA only use the weights as its genetic makeup. This is how NEAT differs from them. NEAT uses the nodes, layers, connections and weights as its genetic makeup. This can make the NEAT algorithm create more efficient AI.

During the course of the research it was found to not be feasible in the time to create an algorithm for NEAT based around Keras or TensorFlow. This then led the research into Reinforcement learning algorithm as these seem to be more suited to the problem and TensorFlow and Keras could be used in the creation of the Neural networks being used. While the research had led down another path the same information will be used for training the Agent to perform the tasks as what would have been used in the NEAT algorithm. This includes the image input and the output of what movement should be performed also using the same type of fitness function to give an action for the value performed. For the most part the environment of training won’t be changed just the Machine learning algorithm being used to perform the training.

As reinforcement learning has many different algorithms to use a certain amount of them need to be evaluated to see if how efficient they are at training an agent for the specific purpose.

## Research Question

Can a machine learning algorithm be used to navigate a Super Mario Bros. Level?

## Proposal

### NEAT

The proposal for this project is to have a Neuroevolution of augmented topologies AI solve a level or several levels of the Nintendo Entertainment System (NES) game Super Mario bros.

To do this the use of an emulator with the ability to capture the image frames from the screen is needed. Also involves the use of several frameworks such as TensorFlow, Keras, Open AI gym and NES-PY. The initial application will be a simple running of the emulator where a simple convolution neural network (CNN) will take in the output from the emulator and return an action between with a list of floating point numbers. From this the maxarg number will be selected to give the action to perform.

The current objective is to get a version of NEAT working with Keras where the connections between nodes cannot be specified. This means for the time being weights, number of nodes and number of layers are what can be specified for genetic makeup.

For this algorithm to work a fitness metric will need to be calculated. This can be done through NES-PY as it will give information such as X position which signifies the distance the character has travelled in the level.

While training the NEAT algorithm will create generations of neural networks. The generations will be split into offspring of which the fittest will be picked through a fitness metric which will be calculated using speed across the level and time alive while making progress. Once the fittest is picked using the fitness value it will then spawn a new generation of offspring. Each of these offspring will receive a random mutation which will have a new neural network topology who will respond differently to different stimuli but will be related to the topology that spawned it. The offspring will then in turn continue the cycle by picking its current generations fittest and mutating that to create another generation.

In the future the hope would be to use pure TensorFlow to add the functionality of the connections. This is only going to be possible if time is permitting. This would give the full functionality of a NEAT AI where all stated genetic makeup can be used.

The plan will also include to have multiple players running at the same time to speed up training. This would involve having multiple of a generations running the levels at the same time.

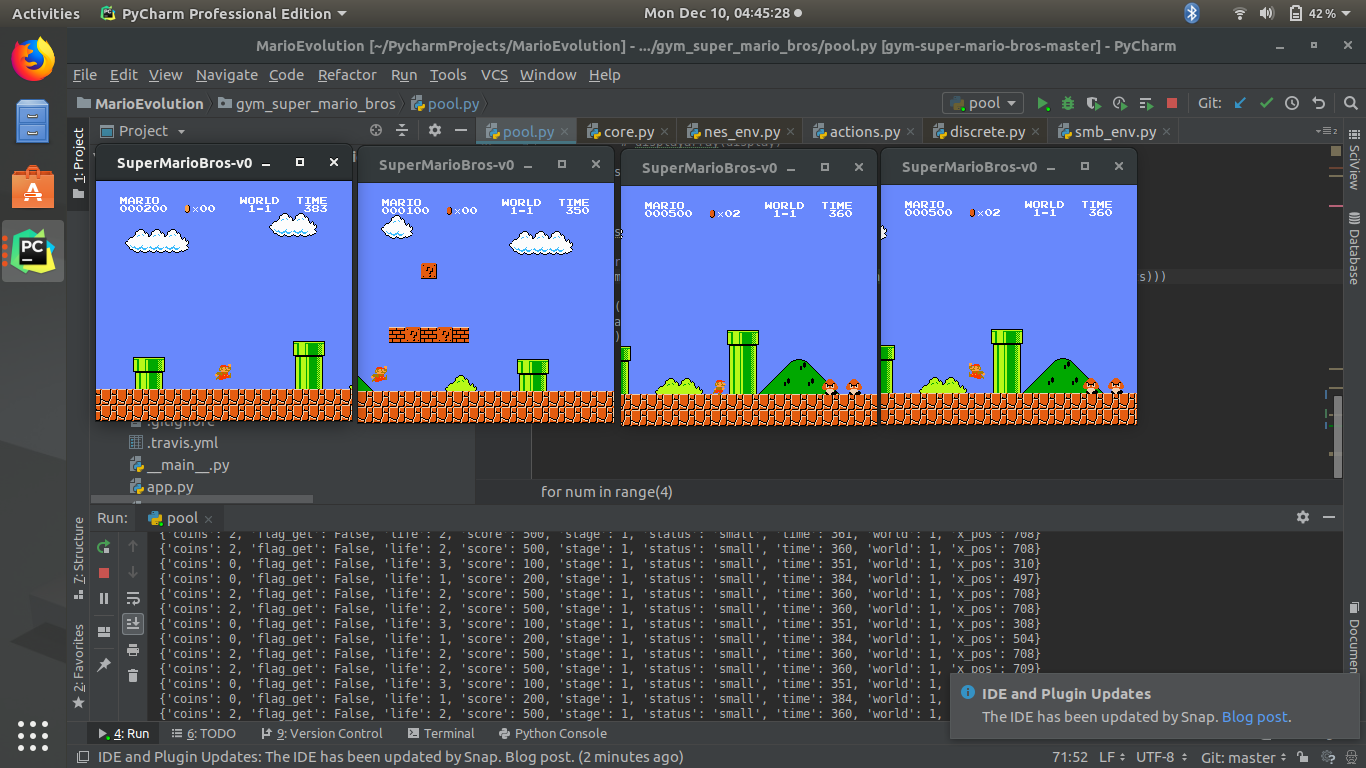


Figure 8‑1 Example of multiple Players in parallel

The figure above shows an example of multiple players running in parallel. This can be done also where not all the windows would be visualised some could be run without rendering which should speed up training significantly.

### Q-Learning

Q-Learning is a more advanced type of machine learning algorithm. The algorithm being used with this project uses a set neural network topology. This network is then trained using the Q-learning algorithm to the solve the given problem.

The project will use a Q-learning algorithm obtained from The Free Code Camp website (Simonini, 2018). Other parts of the tutorial will be used to create the final algorithm that will be used.

The proposal is to use Q-Learning algorithm to navigate levels in Super Mario Bros. This will take experimentation on what should be the input, output and how the output will be taken and used as an action.

A Few types of Q-Learning algorithms will be looked at from Deep Q-learning (DQ), Double Deep Q-Learning (DDQ) and Duelling Double Deep Q-learning (DDDQ). DDDQ can also be augmented using prioritised experience replay.

The focus would be to use these different Q-learning algorithms and use the best one for the navigation of Super Mario Bros. game.

# Applied Research

## Flappy Bird Neat Algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| Prototype  Number | Prototype Name | Start Date | Finish Date |
| 1 | Flappy Bird NEAT implementation | 10/10/2018 | 13/10/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Install Pycharm | Complete |
| 2 | Install Python 3.6 | Complete |
| 3 | Install Keras and Tensorflow | Complete |
| 4 | Clone Neat algorithm Flappy Bird Code  https://github.com/erilyth/Flappy-Bird-Genetic-Algorithms | Complete |
| 5 | Run Flappy Bird to Verify Installation of dependencies | Complete |
| 6 | Step Through code to see what each part is responsible for | Complete |
| 7 | Document Areas of interest within the Flappy Bird Code | Complete |

Locating information on how others have implemented a NEAT Algorithm. This involved searching for projects that have used such and algorithm and stepping through the code to find how it works. Also helpful was the information surrounding the dependencies used in this project as they could be used within this project moving forward.

## Super Mario Bros. OpenAI

|  |  |  |  |
| --- | --- | --- | --- |
| Prototype Number | Prototype Name | Start Date | Finish Date |
| 1 | Super Mario Bros. OpenAI | 15/10/2018 | 18/10/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Install Pycharm | Complete |
| 2 | Install Python 3.6 | Complete |
| 3 | Clone Super Mario Open AI code  https://github.com/Kautenja/gym-super-mario-bros | Complete |
| 4 | Run Code to verify | Complete |
| 5 | Step Through code to see what the main pieces are responsible for | Complete |
| 6 | Read Documentation to learn how the get the information out | Complete |

Using Super Mario OpenAI gym gives a good starting point for the project. It includes all the information needed for training. This includes information like x location in the current level along with a multitude of other information such as when Mario has died or eaten a mushroom. The API can also return an image of the screen. This can then be used as the input in the NEAT AI.



Figure 9‑1 Image capture from Super Mario Open AI

## Image Classification

|  |  |  |  |
| --- | --- | --- | --- |
| Prototype Number | Prototype Name | Start Date | Finish Date |
| 1 | Image Classification Cat vs Dogs | 22/10/2018 | 25/10/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Install Tensorflow and Keras | Complete |
| 2 | Download Image Classification Code  <https://gist.github.com/fchollet/0830affa1f7f19fd47b06d4cf89ed44d> | Complete |
| 3 | Download Image files to be classified  <https://www.kaggle.com/tongpython/cat-and-dog> | Complete |
| 4 | Run Code to train the network | Complete |
| 5 | Step Through code to see what the main pieces are responsible for | Complete |
| 6 | See how changing each how many epochs and the model set up changes the accuracy of the model. | Complete |

Image classification can be used to give an output based on the image it has been given. This image classifier was given a dataset of cats and dogs and from that training it would then be able to tell the difference between them. With 5 epochs the current model has an accuracy of 65 %.

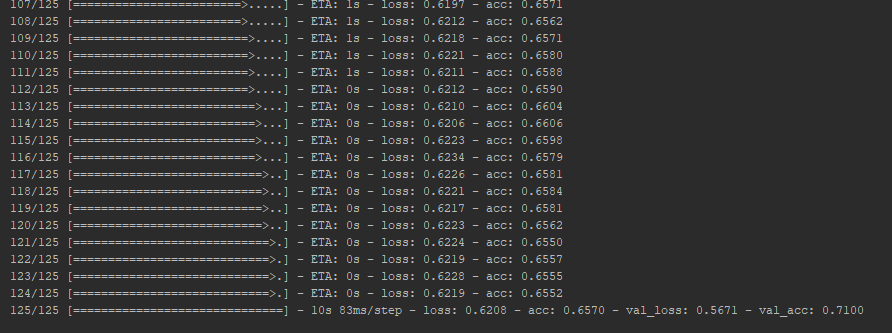


Figure 9‑2 Image Classification Output

## Super Mario Bros. OpenAI using an image classifier

|  |  |  |  |
| --- | --- | --- | --- |
| Prototype Number | Prototype Name | Start Date | Finish Date |
| 1 | Data Collection | 29/01/2018 | 01/02/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Install Pycharm | Complete |
| 2 | Install Python 3.6 | Complete |
| 3 | Download Image Classification Code  <https://gist.github.com/fchollet/0830affa1f7f19fd47b06d4cf89ed44d> | Complete |
| 4 | Clone Super Mario Open AI code  https://github.com/Kautenja/gym-super-mario-bros | Complete |
| 5 | Add image classification Code to Super Mario Gym | Complete |
| 6 | Change network topology and remove training | Complete |
| 7 | Add code to calculate Action converting numbers 0 – 1 to a whole number from the number of given actions | Complete |
| 8 | Run code to verify working | Complete |
| 9 | Debug Code to see outputs | Complete |

Starting with Super Mario Open AI gym the code was changed to create a convolutional neural network that would take in an image of the correct size that is being outputted by the emulator. This is then used to give a prediction these predictions are between 0 – 1 which will not work for the given actions. This needs to be mapped to the number of actions possible for instance is the number of actions was 5 then 0 – 1 would need to be divided up into 5 equal parts each mapped to a whole number between 0 and 4.

# Implementation

## Sprints

This chapter will describe the development process undertaken. The key phases of development will involve the following steps

* Data collection and Training and replay environment
  + gathering gameplay data such as button presses and screen capture. Using an emulator e.g. Bizhawk, Python
  + Creating an environment for Training and Playback of trained models.
* NEAT – Creating a NEAT framework
* Q-learning environment – data needed to perform q-learning
  + Q-Learning – my own algorithm
  + Deep Q-Learning – medium code changed to use my environment
  + Duelling Deep Q-Learning with Prioritised Experience Replay – Medium code
* Running a saved model – Saving the model and running in an environment

### Sprint 1 – Data Collection and Environment Creation

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 1 | Data Collection And Environment | 21/01/2019 | 01/02/2019 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | CSV File formatting | Complete |
| 2 | Created Program that can capture button presses from an Xbox Controller. This will use a python package called input which has been modified to allow for the code to be nonblocking. | Complete |
| 3 | Add Code to Capture Screen output from the emulator Bizhawk. | Complete |
| 4 | Create a function that adds the output from the controller to a CSV file and saves the screen grabs to a Folder | Complete |
| 5 | Cleaning the CSV file | Complete |
| 6 | Add code to load the Data for use in training a neural network | Complete |
| 7 | OpenAI Gym Environment | Complete |
| 8 | Find a NES OpenAI gym Environment | Complete |
| 9 | Create multiple lists of actions that can be performed within the game. | Complete |
| 10 | Create a randomise function that picks actions at random | Complete |
| 11 | Capture meta data from the env eg. x in level, x and y on screen and the image displayed | Complete |
| 12 | Create a function to change the image into greyscale. | Complete |
| 13 | Create a function that crops the image around the Mario character using his x and y screen coordinates | Complete |

From research carried out this is the best way to gather this type of data. This kind of data needs to be gathered in a way that it can be given to a neural network for training, so a CSV file and images seem the best way to do this.

Task 1:

Creating a CSV file layout column structure such as the heading of each column. This needs to be easily read into the program so using what the button presses are eg,

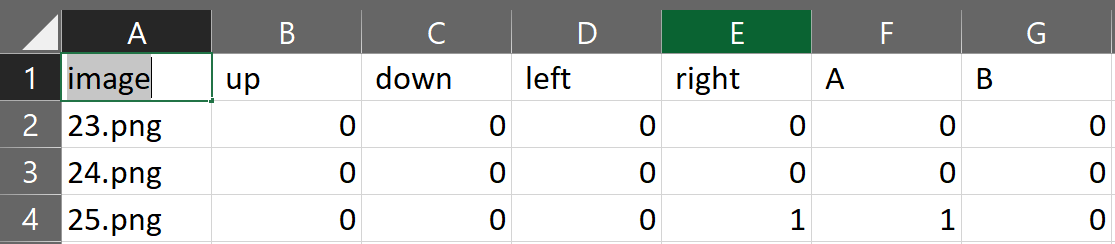


Figure 10‑1 Example of a CSV file structure

Each row will then signify the image clicked and the buttons being pressed at that moment. As you can see above the buttons being pressed in the last line are ‘right’ and ‘A’.

Task 2:

Capturing Xbox Controller input. To capture gamepad/Xbox controller button input a library called inputs was needed. Inputs as a library is extremely handy but has a flaw that caused issues for real time capture for me. This was that when no event or button press had happened it blocked the program from continuing which meant that if a button was not being pressed the screen was not being captured

While trying to resolve the problem I came across a pull request on the repository for the Inputs library which when implemented into my code worked for me prevent the blocking of the program.

The inputs from the controller are recorded into a list e.g.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Image | Up | Down | Right | Left | A | B |
| ‘’ | 0 | 0 | 0 | 0 | 0 | 0 |

This is then will then subsequently be written to a CSV file. The image at the start will signify the image that was displays as the action was being performed.

Task 3:

Screen capture is performed by a library called Pillow. Pillow has a function ImageGrab.Grab that can take in the position and size of an image that you would like to capture e.g.



Figure 10‑2 Code snip for capturing screenshots

This code takes a screenshot 56 pixels from the top of the screen with a pixel size of (1172, 898) this is taken of the whatever is in that section of the screen. This image is then saved to a folder for training later.

Task 4:

To add the functionality of creating a CSV file, A folder to store the CSV file and the images needed in this dataset. The library CSV this can create a CSV file and add rows to it. I also used the built in OS library from the python 3.6 API to create folders.

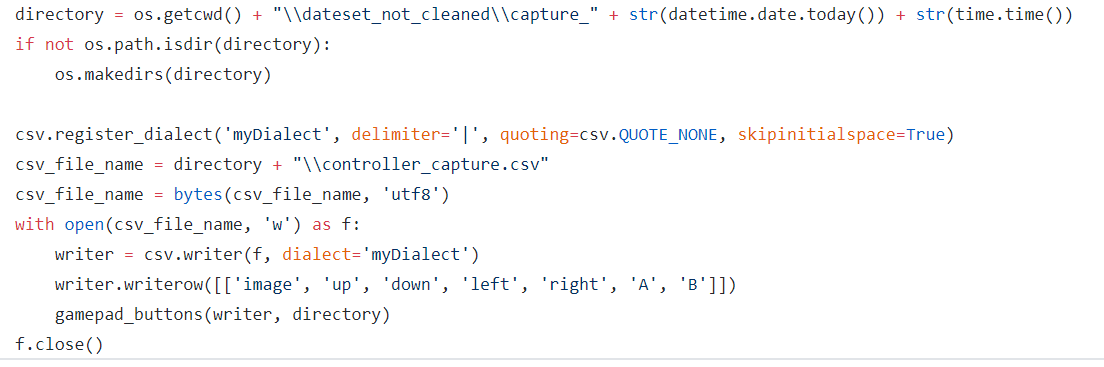


Figure 10‑3 Code to create a CSV file the image name and action stored

Task 5:

Cleaning the CSV file data and Image data.

**Step 1:** Remove Images before gameplay has started. All images that are before the player takes control of Mario are not used. This include any image captured before.

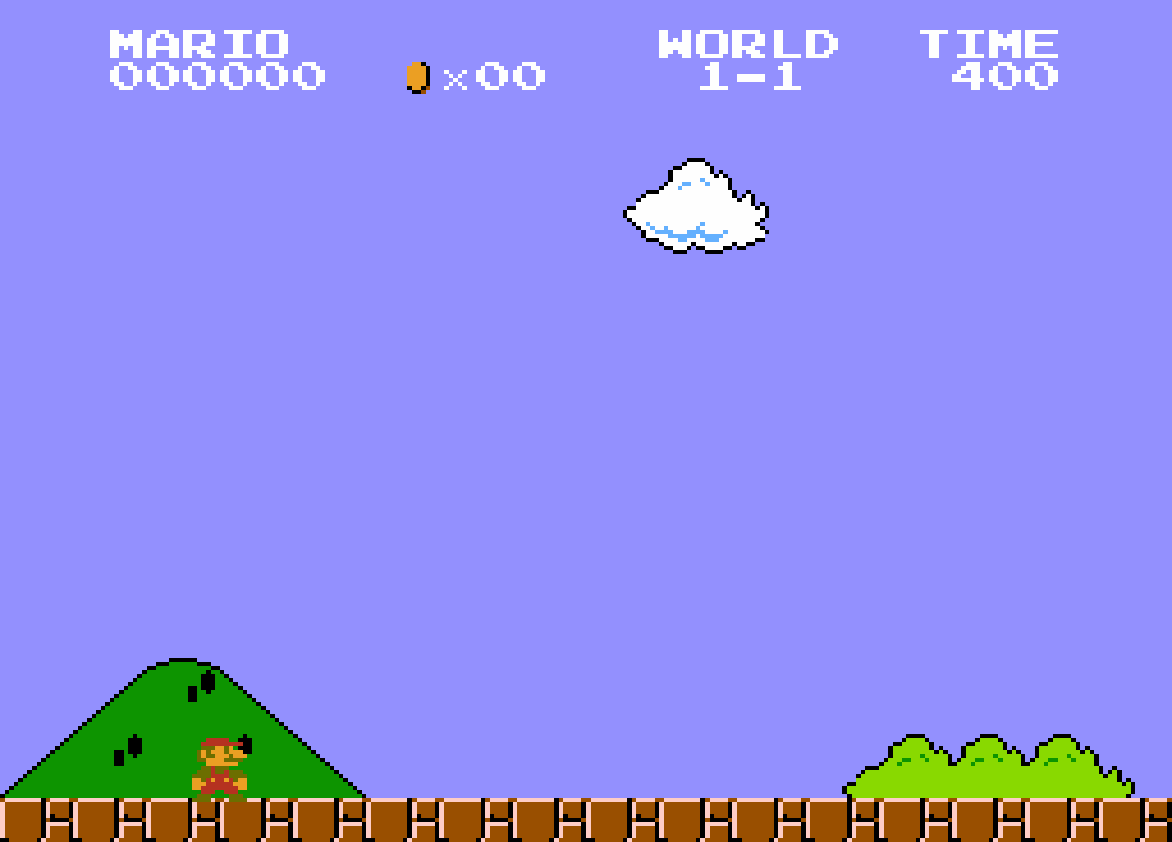


Figure 10‑4 First frame where the player can take control of mario

**Step 2:** Remove any rows for images that are before the image displayed above as they will not contain any actions or data that can be used in training

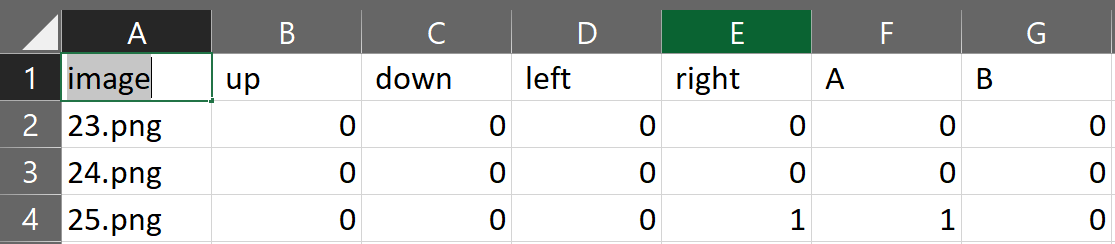


Figure 10‑5 First row is the matching row to the screen grab in fig. 8-3

This shows the rows for every after the time the player can take control of Mario.

Task 6:

Loaded of the dataset. This involves several libraries such as Pandas for loading the CSV files, CV2 for loading the images and the OS library is used to simplify the file address using os.getcwd(). These datasets will be then loaded and used to train a network. Using the CSV file as the y values (expected outputs) and the images as the x values (Inputs)

Task 7:

OpenAI gym has several different environments that could potentially be used for the purpose of a Super Mario Bros. environment such as the OpenAI gym retro. This environment can load retro game roms into its emulation system and display the game using the OpenAI environment patterns and a retro game emulator.

This makes it quite easy to work with as the documentation that applies to vanilla OpenAI gym also applies to OpenAI gym retro.

This was a problematic emulation system as some information could not be gathered in an easy way from the emulator such as the Mario character’s position on screen.

Task 8:

As OpenAI gym retro was not suitable nes-py OpenAI gym environment looked promising as it had a version built specifically for Super Mario bros.

Nes-py is a OpenAi gym Nintendo Entertainment system (nes) emulation system that uses nes roms so they can be used to train AI on a specific game. The nes-py environment gives access to memory locations in the rom that are specific to elements such as when Mario dies, his x and y coordinates and a flag for if Mario has completed a level.

These elements are useful for training in that they can be used to define the environment. As well as used to define the reward function for training.

This meant the information needed to train an AI would be easier to access and use for that purpose. The environment came with predefined configuration. This was several action lists and set environment variables. Some of this would need to change to fit the training to be done.

Task 9:

Creating multiple different action lists for use with the nes-py OpenAI gym environment. This involves creating a file with python list of python lists with the action in the inner list

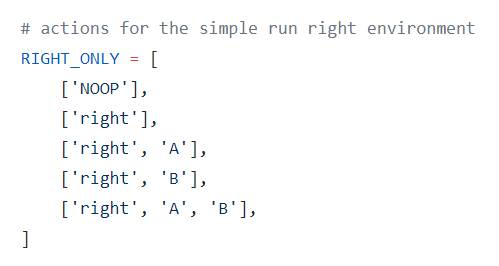


Figure 10‑6 Example of an Action list

In all four action lists were created. Figure 9-5 is the simplest action list and each one added in more complex actions finishing with. The most complex action list with 14 actions in it

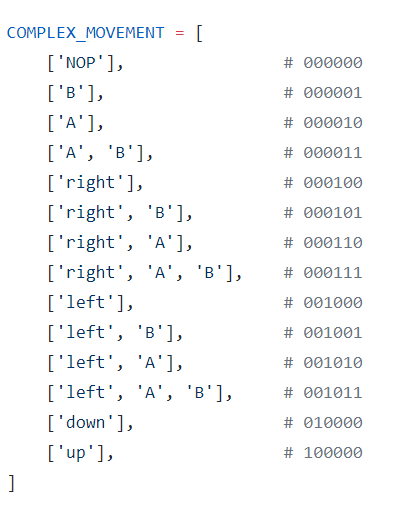


Figure 10‑7 Complex action list containing 14 actions

This a larger list giving much more options for exploration and the ability to learn the environments.

Task 10:

The random action function needs to take in the number of actions being used by the environment for instance the RIGHT\_ONLY action list will need an option for 0-4. The function will return a random integer number from 0-4

This will work since the environment is created using the action list as a parameter in its constructor. Meaning to apply the specified actions it needs to only receive a integer to perform the action

action = random.randint(1, len(possible\_actions)) – 1

Figure 10‑8 Code to randomly select an action

Possible actions is a list of actions that the agent could perform. The action returned is then given to the “env.step(action)” and the environment will then perform that action.

Task 11:

Capturing the meta data from the environment can be gathered at each step. A step is when an action is performed. The step returns a tuple of variables the current state, reward, done, info. Info is a dictionary with a number of datapoints stored within it. As shown in Figure 9-7.

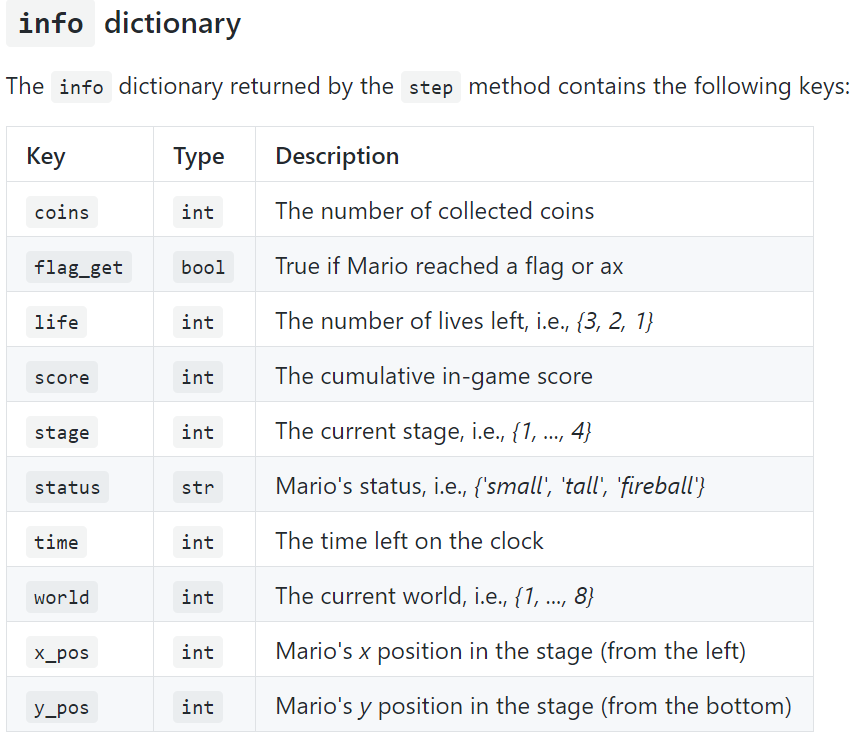


Figure 10‑9 List of information returned in the Info dictionary

Task 12:

A function is needed to convert the current state of the environment which is the image being displayed in Figure 9-8.

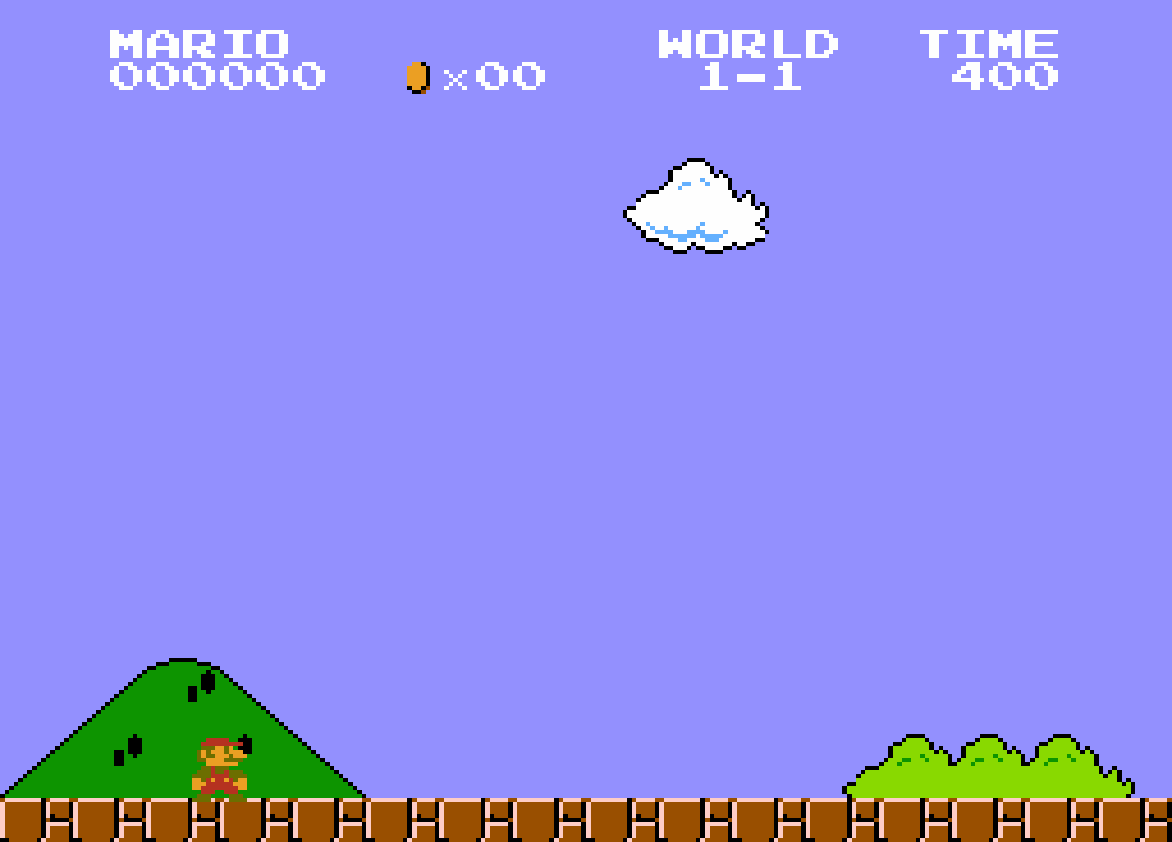


Figure 10‑10 Image that represents the state of the environment

This needs to be converted to greyscale as colour does not add extra information but will add the need for more power needed to process it. The function needs to take in the image and return an image that is greyscale. The same as in Figure 9-9

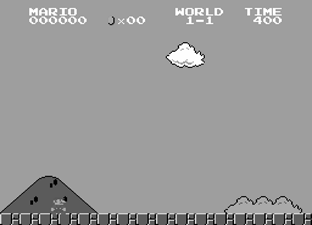


Figure 10‑11 Image converted to grayscale

The function used to convert the image to grey is

def convert\_to\_gray\_scale(img):

return np.dot(img, [0.299, 0.587, 0.114])

Figure 10‑12 Code to convert to Grayscale

This takes in the image and converts it directly to a grayscale image. Like the image shown in Figure 9-10.

Task 13:

Cropping the image around Mario needed to take in the Mario’s x and y coordinates on the screen. While the Info dictionary variable returns a x and y value the x value cannot be used for this purpose the y value can the reason this x value cannot be used is because it is the x position in the level as in the distance travelled in the level not the x value on the screen.

To get this value the environment needed to be unwrapped and the variable needs to be pulled from within the private class.

x = env.unwrapped.\_left\_x\_position

Figure 10‑13 Code for retrieving the x value

This is the code used to obtain the correct value this represents the x value from the left of the screen. This is exactly what is needed to know exactly where Mario is on the screen. The code needs to also know the size the image will be and the offset that needs to be used. An offset is used to decrease the amount of the floor in the image and the amount Mario can see in front of him.

def process\_image(image, x, y, h, w):

image = image[y:y + h, x:x + w]

image = convert\_to\_gray\_scale(image)

return image

Figure 10‑14 Code to Crop image around Mario

The above function in Figure 9-13 is used to crop the image around Mario it also includes the function to convert the image to greyscale. At each step the x and y values along with the height and width the image will be are passed into the function to create the below image.

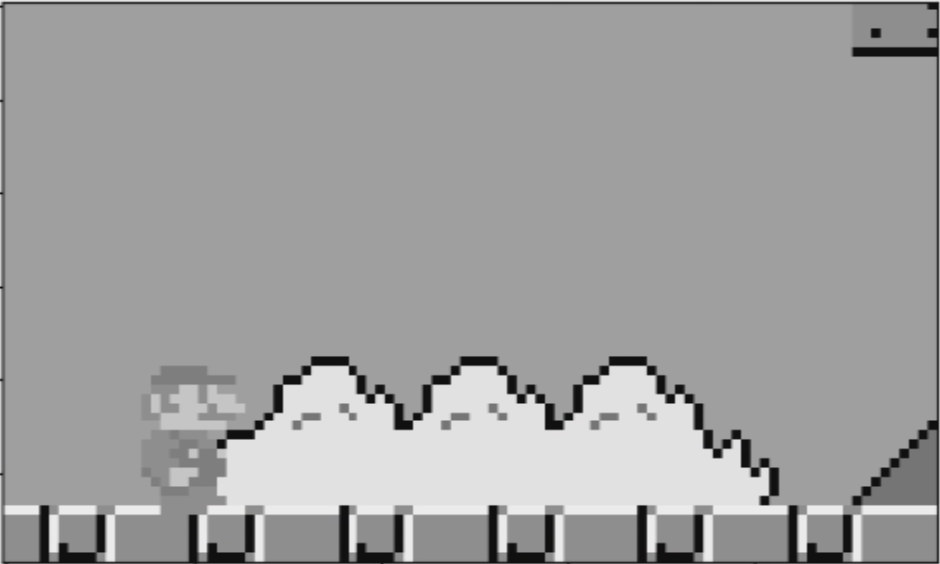


Figure 10‑15 Mario Cropped image

The image is skewed towards the front as in the game Super Mario Bros. game Mario rarely needs to go back or see what is behind him.

### Sprint 2 - NEAT

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 1 | Create a Frame work for NEAT | 04/02/2019 | 15/02/2019 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Design a basic neural network | Complete |
| 2 | Create a function that can run multiple environments in parallel | Complete |
| 3 | Create a function that randomizes neural networks based on a seed | Complete |
| 4 | Create a function that can run multiple neural networks in parallel | Incomplete |
| 5 | Transfer fitness statistics and topologies from the main thread to each other thread | Incomplete |
| 6 | Create functions for mutation and crossover | Incomplete |

Task 1:

Designing a basic neural network with one convolutional layer, a flatten layer and Dense layer for output. This was the basis for the first generation of agents in the NEAT algorithm.

All the layers that would be added through mutation would be added in between the flatten layer and the output dense layer. As such the convolutional layer and output would remain the same throughout.

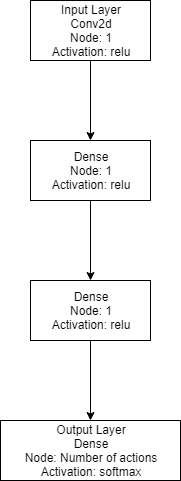


Figure 10‑16 Visualisation of the simple neural network

Figure 9-16 shows a neural network starting point for the NEAT algorithm to work with. The network contains four layers an input, output and two hidden layers.

Task 2:

Running multiple threads in python need the use of a pool which can be used to run multiple processes simultaneously. This was used to create a way to run multiple environments at the same time. Up to 8 was feasible with my current computer hardware.

Task 3:

Creating a random neural network involves deciding on where the random layers should be placed. The place decided was just before the output layer leaving the other layers where they are.

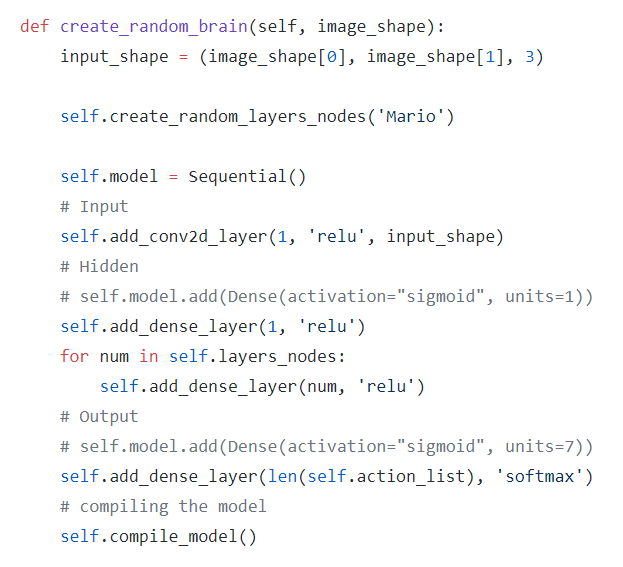


Figure 10‑17 Code for creating a random neural network

The code shows how a loop around the a function that adds dense layers to the model si then looped through however many items are in the layers\_nodes list this then adds the number of nodes that is contained in it to the layer.

Task 4:

It was found through the development process that running multiple parallel processes using python and having each one sending information back to the main process is not feasible with the current python language libraries. For this reason and the feasibility struggles with creating a NEAT algorithm using Keras and OpenAI gym was not going to work. NEAT was abandoned as a way of training an agent for using reinforcement learning.

### Sprint 3 – Q-Learning

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 1 | Q-Learning Environment | 18/02/2019 | 01/03/2019 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Create a basic Q-Learning algorithm using epsilon greedy. | Complete |
| 2 | Find and convert Deep Q-Learning code to work with the Super Mario Bros. OpenAI environment. | Complete |
| 3 | Use the Deep Q-Learning with Super Mario Bros. to train an agent. | Complete |
| 4 | Use the Deep Q-Learning model to train on a simpler game e.g. Flappy Bird | Complete |
| 5 | Change algorithm to Duelling Double Deep Q-Learning | Complete |
| 5 | Train model using the new Algorithm | Complete |
| 6 | Save and Load the model for use after training | Complete |

Task 1:

Creating a Q-Learning algorithm started with looking for tutorials and articles with either pseudo code or an example of working code. The code and some examples were found. It was a blog post containing multiple pieces of information on how reinforcement learning, and Q-Learning works.

y = 0.95

eps = 0.5

decay\_factor = 0.999

r\_avg\_list = []

for i in range(num\_episodes):

s = env.reset()

eps \*= decay\_factor

if i % 100 == 0:

print("Episode {} of {}".format(i + 1, num\_episodes))

done = False

r\_sum = 0

while not done:

if np.random.random() < eps:

a = np.random.randint(0, 2)

else:

a = np.argmax(model.predict(np.identity(5)[s:s + 1]))

new\_s, r, done, \_ = env.step(a)

target = r + y \* np.max(model.predict(np.identity(5)[new\_s:new\_s + 1]))

target\_vec = model.predict(np.identity(5)[s:s + 1])[0]

target\_vec[a] = target

model.fit(np.identity(5)[s:s + 1], target\_vec.reshape(-1, 2), epochs=1, verbose=0)

s = new\_s

r\_sum += r

r\_avg\_list.append(r\_sum / 1000)

Figure 10‑18 Basic Q-Learning code for OpenAI gym environments

The code in Figure 9-18 is only useful for very simple environments such as Cartpole. This as such would not work with Super Mario Bros. This was shown by running the algorithm for several hours and the result being worse than running random actions.

Task 2:

A tutorial on Towards Science was used as the basis for creating a Deep Q-learning (DQ) training environment. This was originally built for vizDoom so most of the functionality surrounding how the Algorithm interacts with the learning process needed to be change. Like how the environment made steps, returned the state, how actions were set up and the image processing with regards to cropping. Most of the rest of the code could stay the same.

The main issues came when trying to keep the cropping around the Mario character working correctly. As it happened when Mario fell in a hole the crop would break since Mario’s Y value would become unusable. To prevent code was added to constrain the crop from going outside of the image. This meant the image size would always stay at what it was supposed to.

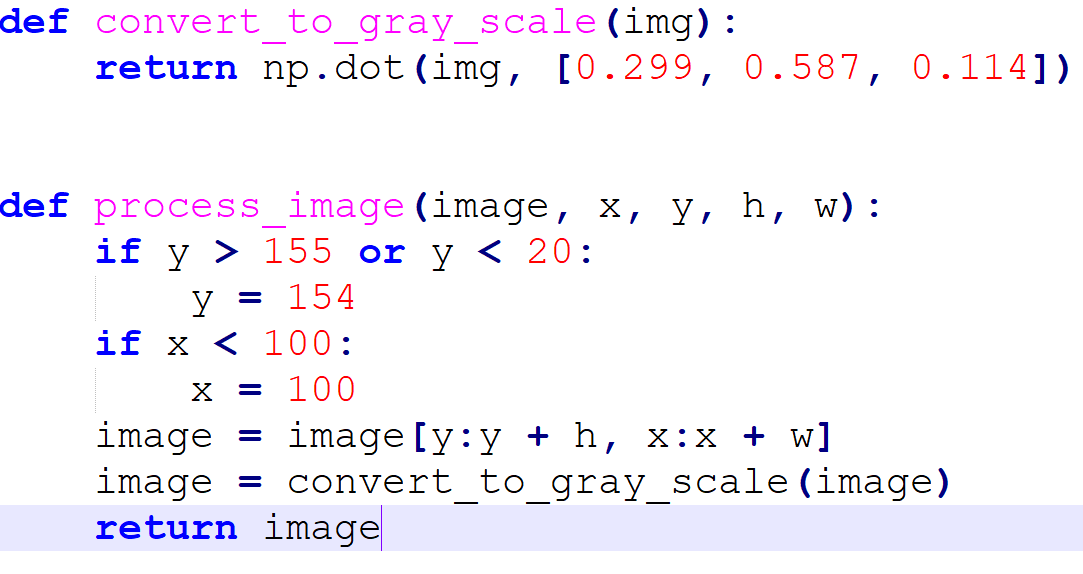


Figure 10‑19 Image Processing and Crop Constraining

In the above code the image is converted to grayscale and it is also cropped to around the x and y values it receives. The h and w parameters are the height and width of the image. The two if statements are where the constraining happens this prevents the x value from going off the sides if Mario moves backwards on the screen and the y prevents the cropped image going off the screen when Mario drops down a hole.

The next addition was to add code that would kill Mario off if he were to get stuck. This code kills Mario and gives a reward of -15 the lowest reward in the environment. This is to keep training going and to prevent episodes going on for longer than is needed. The code checks every 480 steps if he is in the same spot.

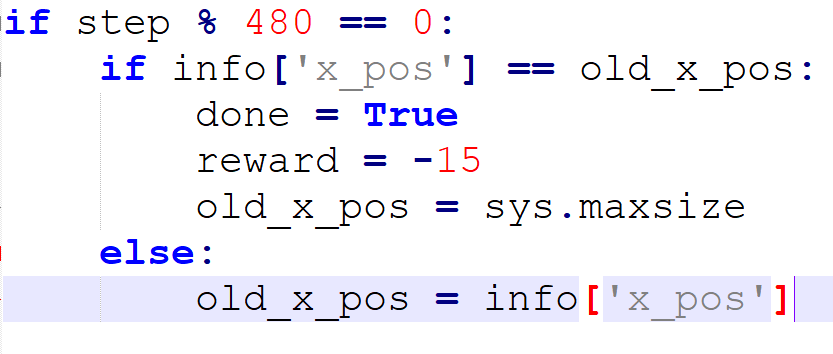


Figure 10‑20 Code to prevent the agent getting stuck

The above code checks Mario’s x positions every 480 steps and if it has not changed in that time the environment will be reset and the Mario will get a reward of -15. The old\_x\_pos is then reset to a large integer number that it would not be feasible for x to reach

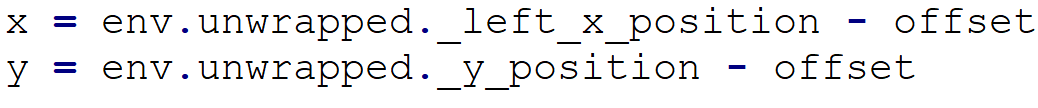


Figure 10‑21 code to receive the x and y coordinates from the environment

This code is used with the cropping of the image to give the specific x and y at every step in the progression of training. It is needed after every env.reset() and env.step(action). This is when those values are changed and need to be updated to allow for the cropping to function correctly.

Task 3:

Now to start training an agent using DQ to navigate a Super Mario Bros. Level. This is a long process will take several hours if not days to complete. The main part is that the data gathered will help with further research and the effectiveness of this a way of training an agent for this purpose.

While the algorithm has been shown to work mixed results were found when using it with Super Mario Bros. It seemed to be going well while training then once training was done the trained model would get stuck at the same point each time. As it was learning it was left to train for 4 days and the results were not great it got further but then consistently jumped down the same hole. The model had shown signs of training but seemed to lack enough of a memory of the level to be able to keep going.

The chart above shows the training and the consistency that was shown towards the end of the training but as can also be seen the fitness never reached a level of completion.

A suggestion was made to take a step back to a simpler game and see if that could shed some light on the issue. The game chosen was Flappy bird as it is a simpler game with only one action that can be on or off.

Task 4:

Use the DQ with Flappy Bird as it is a simpler game

Using the DQ with Flappy bird should have been an easy task but it turned out the best way to do it was to use a user created OpenAI gym Flappy Bird game built using Pygame. Pygame is a game engine library for python.

The first thing to do is to clone Pygame Learning Environment (PLE). The cd into the cloned folder and install PLE

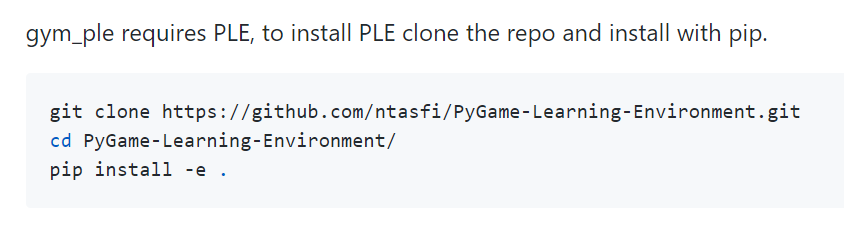


Figure 10‑22 PLE Installation Instructions

Once that is installed it needs to have gym-ple installed this can be got from pip using the pip install command.

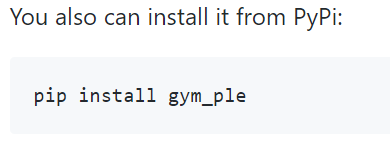


Figure 10‑23 Command to install gym-ple

This will allow the use of the OpenAI gym commands such as env.reset and env.step

Once installed the DQ needs to be edited to work with this specific OpenAI gym environment. This involves changing the image size for resize to go into the Neural Network and the way the image is cropped.

As the flappy bird character does not move right or left on the screen the cropping was basic and did not need what was done for Super Mario Bros. Instead the image was cropped into the centre of the image right up to the back of the character and then out in front till only one pipe could be seen on the screen

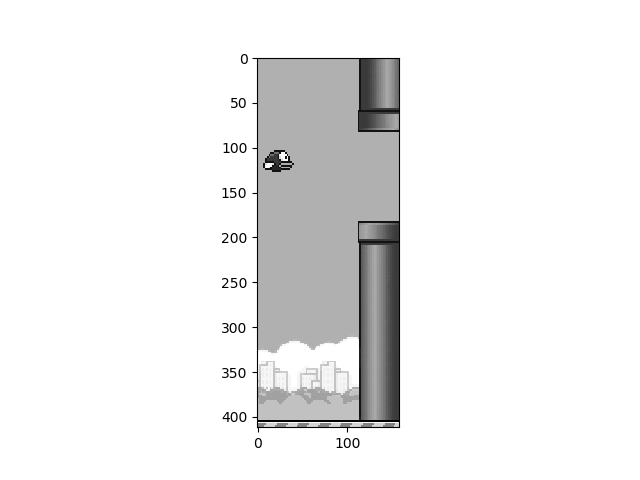


Figure 10‑24 Cropped Flappy Bird image

This is the image that the agent sees when it is trying to decide to jump or drop. Other than that everything else is the same as it was in the Super Mario implementation excepts for the Super Mario Bros. specific code such as the code that kills the agent if it stays in the same spot. Also, the code to pull the x and y coordinates from the environment.

While training a problem was observed where the random exploration was not getting through pipes all that often meaning that it did not seem feasible that the agent would ever learn navigate through the pipes. Even after training for several days the agent never got more than ok at the game.

The highest score recorded was 26 after training while replaying the agent. The belief is also that the random nature of the generation of pipes was causing significant issues with the learning process.

Task 5:

With how the Flappy Bird experiment turned out changing to a more complex algorithm seemed like the next step to completing the project. While a Duelling Double Deep Q-learning (DDDQ) project was located the project needed to be converted like the previous tutorial code to be used with OpenAI this meant transplanting the current code from the DQ project and putting it in the new project with all the previous tweaks.

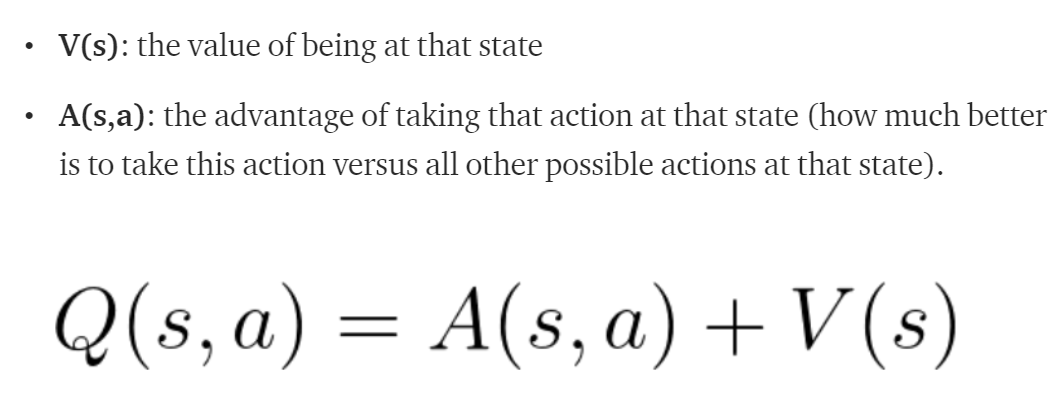


Figure 10‑25 Q-value equation (Simonini, 2018)

While that was all that was needed for the development to get the project training took some work as the network was different and the memory storage was completely changed from the DQ algorithm.

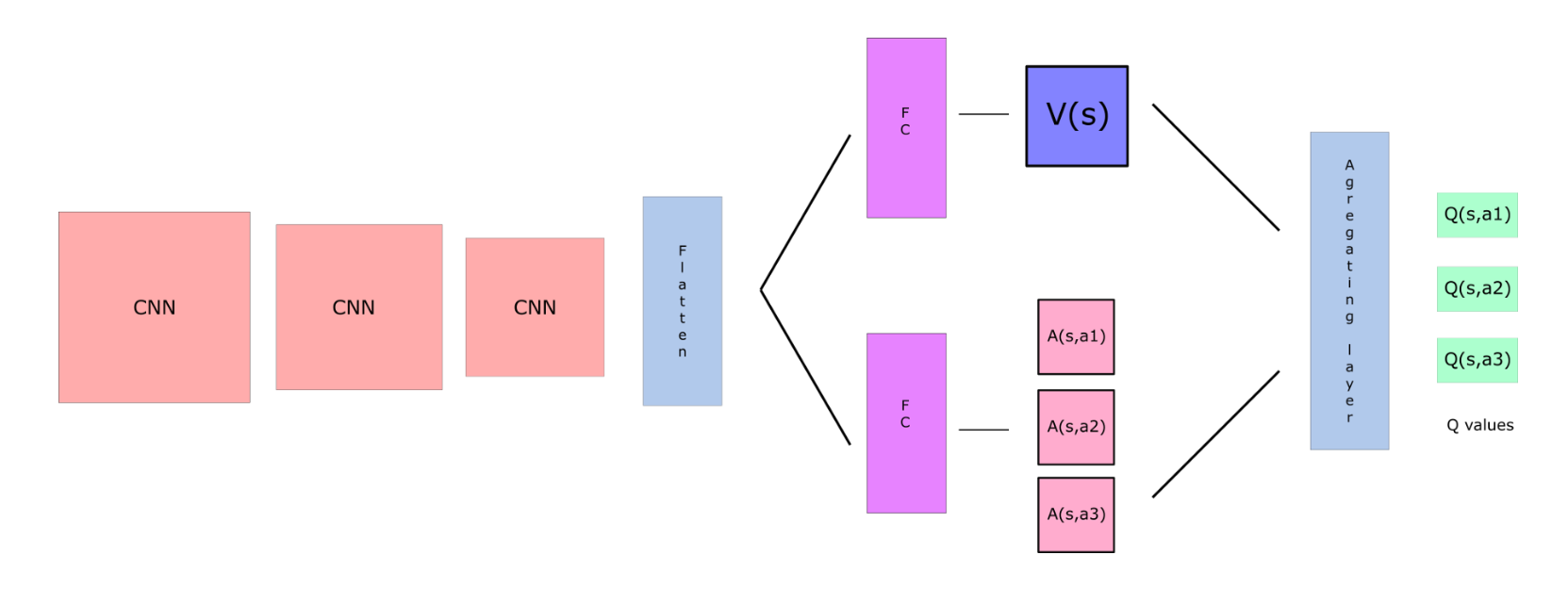


Figure 10‑26 The network topology for DDDQ (Simonini, 2018)

The above image shows the new topology in the DDDQ algorithm. This takes in the current state then split to calculate separate values and then is brought back together with an aggregation layer giving a list of Q-values which represent the best action that can be made.

Prioritized experience replay is also being used within the DDDQ project which adds a new way to storing the memory in the algorithm. This uses a sum tree which is a type of tree data structure which keeps the better memories while removing the worse memories this helps to prevent the agent from forgetting what it has previously done to get the rewards.

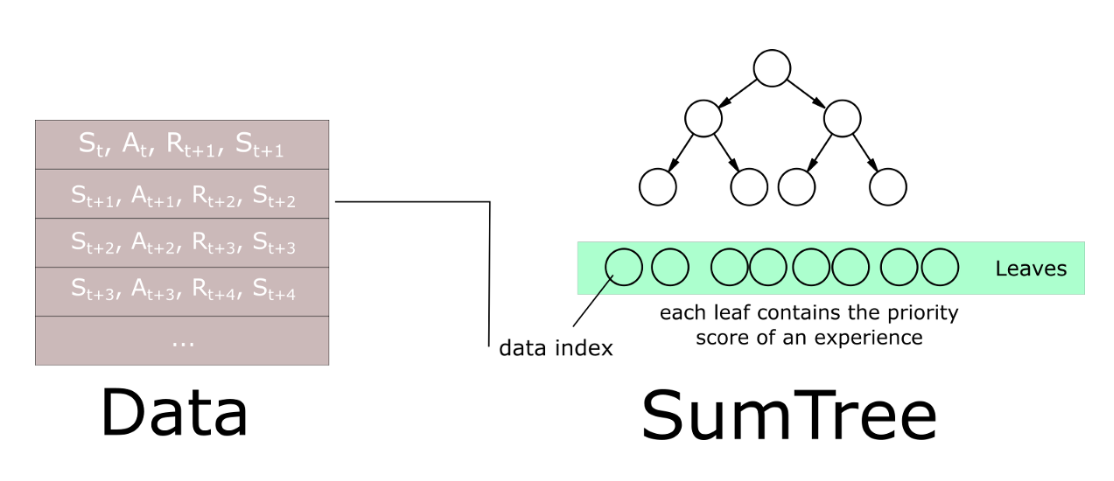


Figure 10‑27 Representation of the Data within the Sum Tree

As can be seen above the Sum Tree stores the data in a way that the best memories are quick to access with the topology of the data structure. It resembles that of a sorted binary tree which is a fast data structure to search through.

The algorithm was set off training to see how it would get on. At about 6 hours in it was completing the level and at 8 hours in it was consistently completing levels. This was a significant reduction in training time as DQ did not even reach the completion of a level.

Figure 10‑28 Total fitness over episodes

The above graph shows the progression through the game. It looks pretty similar until you get closer to the end where the values get much more consistent and closer together. This is where you can see it is using the model more than the random exploration.

Task 6:

To Save and load the models is actually not much code at all first the a saver object needs to be created using tf.train.Saver()

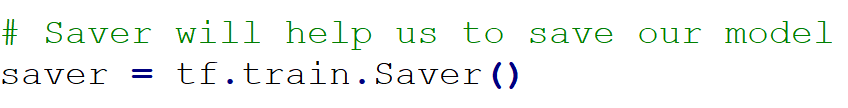


Figure 10‑29 Saver code for TensorFlow models

Once every five episodes the model gets saved this prevents the loss of training a crash or hardware failure of some sort. Means at most five episodes of training will be lost.

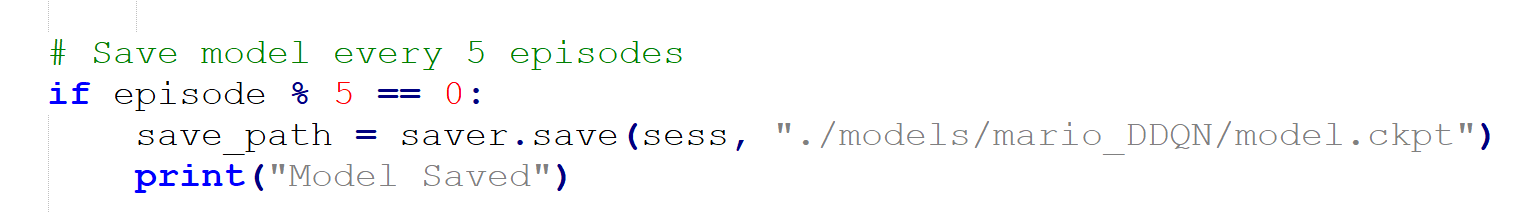


Figure 10‑30 Code to save the model

Once the code is saved the next thing to do is to load the model to try it just purely on its own merits with no random exploration.

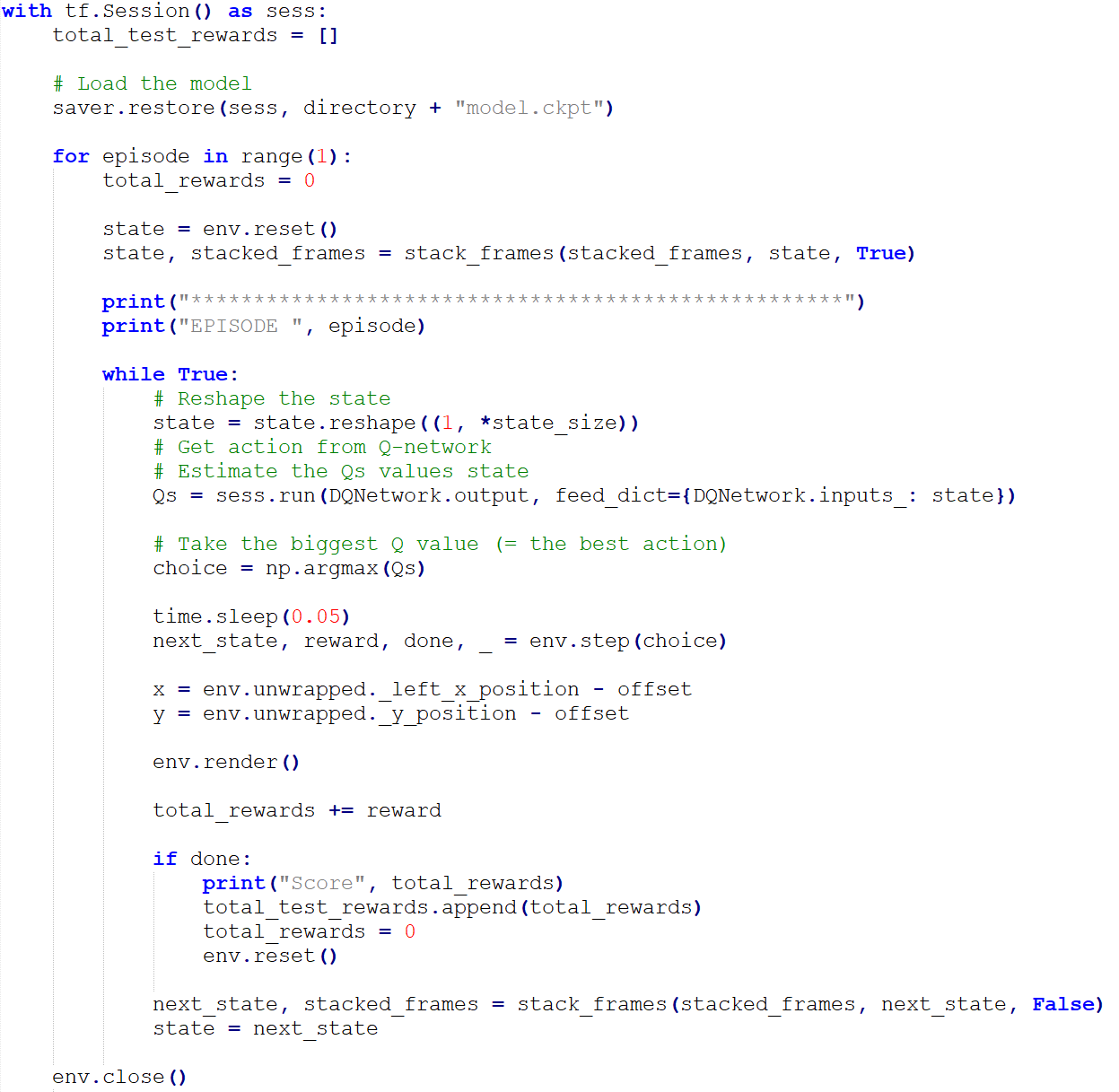


Figure 10‑31 This code loads the model the plays the game using the model

The above code loads the model using saver.restore() into the sess variable. This then can be used to take in the current environment state and play Super Mario Bros. If trained fully the agent will consistently complete the first level of Super Mario Bros. until stopped.



Figure 10‑32 Screenshot of the environment

# Conclusion

Using a Machine Learning algorithm to create an agent that can navigate levels of Super Mario Bros. can be seen a stepping stone for this type of algorithm to be used in other types of navigation. Such as drone flight, automated vacuums and car navigation systems. Using the same premise as used above the algorithm could be changed to use a different set of inputs, rewards and should give mostly the same level of results.

The project itself was a complete success using Reinforcement learning specifically Duelling Double Deep Q-Learning. This is quite a complex Reinforcement learning algorithm as it has many components that are used to work out the Q-Values also the use of a prioritised experience replay was by far the most helpful to the progression of training.

Figure 11‑1 Total fitness over episodes

As can be seen in this graph the training starts out quite bad with the odd blip brining a good episode. The graph shows how the algorithm brings the training into a consistent level after about 2000 episodes and is fully trained after 5000.

The project takes a long time to train which makes it hard to evaluate whether the algorithm worked or not this as a result ended with multiple algorithms along the reinforcement learning vein being considered until finding one that worked and gave the above results.

# References

AppliedAI, 2018. *Dark side of neural networks explained.* [Online]   
Available at: https://blog.appliedai.com/how-neural-networks-work/  
[Accessed 4 12 2018].

Computerphile, 2016. *Neural Network that Changes Everything - Computerphile.* [Online]   
Available at: https://www.youtube.com/watch?v=py5byOOHZM8  
[Accessed 12 11 2018].

Google, 2018. *tensorflow.* [Online]   
Available at: https://github.com/tensorflow/tensorflow  
[Accessed 1 12 2018].

Gross, R., 2010. Psychology: The Science of Mind and Behaviour, 6th edition. 6 ed. s.l.: Hodder Education Publishers.

Hasselt, H. v., 2010. *Double Q-learning.* s.l., NIPS.

Holländer, B., 2018. *Natural vs Artificial Neural Networks.* [Online]   
Available at: https://becominghuman.ai/natural-vs-artificial-neural-networks-9f3be2d45fdb  
[Accessed 4 12 2018].

Kaelbling, L. P., Littman, M. L. & Moore, A. W., 1996. Reinforcement Learning: A Survey. *Journal of Articial Intelligence Research,* 4(1), pp. 237-285.

Kauten, C., 2018. *NES-PY.* [Online]   
Available at: https://github.com/Kautenja/nes-py  
[Accessed 10 10 2018].

Kenneth O. Stanley, R. M., 2002. Evolving Neural Networks through. *The MIT Press Journals ,* 10(2), pp. 99-127.

Keras, 2018. *Keras documentation.* [Online]   
Available at: https://keras.io/  
[Accessed 2 12 2018].

Lin Chen, D. A., 2006. NeuroEvolution of Augmenting Topologies with Learning for Data Classification. Shandong, China, IEEE.

Obitko, M., 2011. *Crossover and Mutation.* [Online]   
Available at: http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php  
[Accessed 13 11 2018].

Oh, B. & Lee, J., 2018. *A case study on scene recognition using an ensemble convolution neural network.* Chuncheon-si Gangwon-do, Korea (South), Korea (South), IEEE.

Open AI, 2018. *Gym.* [Online]   
Available at: https://gym.openai.com/  
[Accessed 13 11 2018].

Russell, S. J. & Norvig, P., 2009. *Artificial Intelligence: A Modern Approach.* 3rd ed. s.l.:Prentice Hall.

Salih, A. & Moshaiov, A., 2016. Multi-objective neuro-evolution: Should the main reproduction mechanism be crossover or mutation?. Budapest, Hungary, IEEE.

Schiappa, M., 2017. *Man vs machine: comparing artificial and biological neural networks.* [Online]   
Available at: https://news.sophos.com/en-us/2017/09/21/man-vs-machine-comparing-artificial-and-biological-neural-networks/  
[Accessed 2 October 2018].

Shiffman, D., 2012. *The Nature Of Code.* 1st ed. New York: Self.

Simonini, T., 2018. *An introduction to Deep Q-Learning: let’s play Doom.* [Online]   
Available at: https://medium.freecodecamp.org/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8  
[Accessed 23 02 2019].

Simonini, T., 2018. *Diving deeper into Reinforcement Learning with Q-Learning.* [Online]   
Available at: https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe  
[Accessed 23 03 2019].

Simonini, T., 2018. *Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets.* [Online]   
Available at: https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682  
[Accessed 23 03 2019].

Stanley, K. O., 2017. *Neuroevolution: A different kind of deep learning.* [Online]   
Available at: https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning  
[Accessed 2 October 2018].

Stanley, K. O. & Miikkulainen, R., 2006. Evolving Neural Networks through Augmented Topologies. *Evolutionary Computation,* 10(2), pp. 99-127.

Trivedi, C., 2018. *Using Deep Q-Learning in FIFA 18 to perfect the art of free-kicks.* [Online]   
Available at: https://towardsdatascience.com/using-deep-q-learning-in-fifa-18-to-perfect-the-art-of-free-kicks-f2e4e979ee66  
[Accessed 24 03 2019].

Zhu, J., Song, Y., Jiang, D. & Song, H., 2017. A New Deep-Q-Learning-Based Transmission Scheduling Mechanism for the Cognitive Internet of Things. *Jiang Zhu ; Yonghui Song ; Dingde Jiang ; Houbing Song,* 5(4), pp. 2375-2385.