Playing video games using neural networks and reinforcement learning

Joseph Shihab Esmaail

May 2016

Abstract

This is a Technical Computer Science project, with the aim to develop a neural network that is able to learn to play video games. This document explains the projects motivations and aims, what related work exists, my current understanding of the theory and tools involved, and how the development of the project will be handled. (edit and not exceed 10 lines)

Project Dissertation submitted to Swansea University
in Partial Fulfilment for the Degree of Bachelor of Science

Prifysgol Abertawe
Swansea University

Department of Computer Science
Swansea University

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

April 29, 2016

Signed:

Statement 1

This dissertation is being submitted in partial fulfilment of the requirements for the degree of a BSc in Computer Science.

April 29, 2016

Signed:

Statement 2

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

April 29, 2016

Signed:

Statement 3

I hereby give consent for my dissertation to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

April 29, 2016

Signed:

Contents

1 Introduction

Neural Networks are a model within the field of Machine learning which is currently a major buzzword in the Computer Science Discipline, owing a lot to the recent growth of big data. It has also gained increasing public presence through efforts such as IBM's Watson, which won against two former champions in Jeopardy in 2011, and Google's DeepDream program which displayed how convolutional neural networks interpret images.
Machine learning is also a major player in our day to day lives with Apple, Microsoft, and Google all using it for their smartphone digital assistants, Google using it for their search engine, and social media such as Facebook using it to cater it's service to each individual user. This will also extend in the coming future with autonomous vehicles such as Google's self-driving car project and motor companies such as Tesla starting to incorporate self-driving systems into their cars.

This project aims to apply machine learning, in particular neural networks, to the task of learning to play video games. Video games have been an almost constantly growing phenomenon since their inception in the 1970's with Pong, recently entering the mainstream with the growth of the eSports industry which hosts tournaments with prize pools in the region of a million US dollars. The application of machine learning to video games has the potential to enable a greater interest in machine learning, as it's current implementations is very pervasive.

This dissertation will provide the reader with a background to neural networks, reinforcement learning and relating work to these and their application to video games, as well as how this project has been organised, what technologies are utilised, how the task has been approached and implemented, as well as what results are produced.

1.1 Aims and objectives

To be able to successfully produce a neural network that can play video games, the network will need a way to interact with the video games, both to receive information about the current game state and to send actions into the video game to play. So we will need methods to obtain the outputted game screen data and current score, as well as methods to input actions as a human player would via a game controller. The neural network needs to be able to interpret the input game state, and output a decision on what action to perform, this action hopefully being one that results in the best performance or score in the video game. The neural network will determine what action is the optimal one after performing a learning process using reinforcement learning, on the particular video game in question. After the network is capable of performing optimal actions to the best of its ability, the performance will be evaluated against hu-

man players to make a decision on how successful it was at learning to play the video game.

To summarise, the aim of this project is to develop an artificial intelligence agent, namely a neural network that will use reinforcement learning to learn to play a large set of video games. This can be achieved by splitting the overall task into the following steps:

- Find methods to manipulate video game inputs and outputs.
- Develop the neural network.
- Develop reinforcement learning methods for the intelligence agent.
- Train and evaluate the neural network with video games.
- GPU-Accelerate the network to improve performance.

2 Background

2.1 Related work

An early use of a neural network to play a game can be seen in the paper 'Temporal difference learning and TD-Gammon' by Gerald Tesauro. In this Tesauro applies the neural network (known as TD-Gammon) to the game backgammon, and trained itself using reinforcement learning, in particular *temporal-difference learning*. The neural network would train itself "by playing against itself and learning from the outcome" [17]. Tesauro concludes that the temporal difference learning proved to be "a promising general-purpose technique" [17]. TD-Gammon successfully learnt to play backgammon, successfully enough that "by version 2.1 TD-Gammon was regarded as playing at a level extremely close to equalling that of the best human player, and had even started to influence the way expert backgammon players played" [17]. This particular implementation of reinforcement learning to neural networks however would not be possible to implement across a broad selection of games as it requires a game with two players.

Neural networks and reinforcement learning have also been applied to obstacle avoidance in robots, which is a similar environment to that of video games. In the paper 'Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance' by Huang et al. reinforcement learning was chosen because "a complete knowledge of the environment is not necessary" [8]. When a human player starts to learn to play a video game, they also do a complete knowledge of all the elements of the game. In this case the neural network needs to be learning the video game in a similar fashion to how a human player would, so reinforcement learning is applicable here. The

results of the paper show that using neural networks and reinforcement learning can enhance learnability and allow task completion in a complex environment, which can also be translated across to the application to video games.

In 2007 the paper 'Reinforcement learning and neural networks for Tetris' by Nicholas Lundgaard and Brian McKee saw the application of neural networks and reinforcement learning to the popular 1984 video game Tetris. They observe that video games are well suited for the application of reinforcement learning as more often than not there is a built in system to track how well a player is doing, often known as the score or points of the player. They also make the observation that in the case of Tetris "it is nearly impossible to definitively say what was a right or wrong move" [10], which can be applied also to most video games and means that the task of learning for neural networks applied to them cannot be done through supervised learning. In practice they found that "while the learning agents fail to accumulate as many points as the most advanced AI agents, they do learn to play more efficiently"[10]. Lundgaard and McKee conclude that "compared to human performance, the learning agents appear to perform well: while they can't survive forever, they perform better during game-play"[10]. This suggests that an AI player doesn't necessarily gain an unfair advantage over a human player by being capable of surviving indefinitely, they instead play at a similar pace.

The most recent and relevant work in developing neural networks to play video games comes from DeepMind, who were recently acquired by Google. Initially in their paper 'Playing Atari with deep reinforcement learning' in 2013 and then expanded upon in 2015 in 'Human-level control through deep reinforcement learning', a paper which made the front page of Nature. In these papers they refer to their neural network as a Deep Q network (DQN), which is a deep convolutional neural network trained using a variant of Q-learning. This network is then applied to Atari 2600 games via the *Arcade learning environment*, which was also developed by DeepMind. A convolutional neural network was chosen here because it "mimics the effects of receptive fields"[12], so it is more equipped for the task of pattern recognition within images. A deep network was chosen because "several layers of nodes are used to build up progressively more abstract representations of the data"[12], allowing the network to "learn concepts such as object categories directly from the raw sensory data"[12]. The results of this paper show **COMPLETE THIS PART**.

This related work confirmed that a neural network is a suitable model of machine learning to apply to the task of learning to play video games. In particular a deep convolutional neural network trained with a Q-learning variant has been proven as capable of learning to play multiple video games as opposed to just one, with no changes to the structure of the neural network.

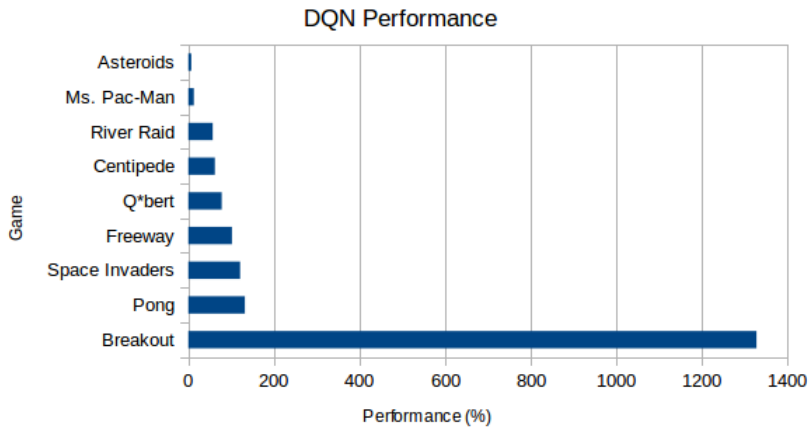

Figure 1: DQN performance with regard to human performance [12].

2.2 Neural networks

Neural networks (also known as artificial neural networks) are machine learning models based on the structure of the brain in animals such as humans.

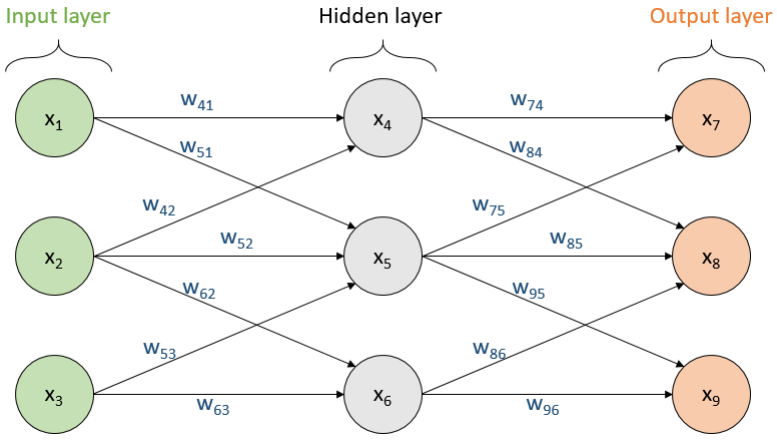

Figure 2: An example neural network.

These neural networks are, naively, a collections of connected nodes, where the nodes are referred to as neurons, and the connections between them as weights. The neurons of the network exist in layers, an input layer for inputs to the network, and output layer to give the results of the network, and poten-

tially some hidden layers in between. If a neural network has more than one hidden layer that network is then considered a Deep neural network. A neural network is a feed-forward network if all of it's connections point towards the output layer, or a recurrent network if there exist feedback connections which point backwards in the network towards the input layer. For the purpose of this explanation I will be using a feed-forward network,

The input layer of the neural network receives its data from the input to the network, this data is then passed along weighted connection(s) to neurons in the next layer of the network. All layers of a network, with the exception of the input layer, have a bias unit as well as having a summation and an activation function performed on them before the neuron data is propagated forward through to the next layer. The summation process for a neural is performed as follows:

$$x_i = \sum_j w_{ij} x_j + b \tag{1}$$

Where:

w_{ij} represents the weight for the connection from neuron j to neuron i.

x_j is the value of neuron j.

b is the value of the bias unit.

After this summation process is applied to a neuron an activation function is usually applied. The most commonly applied activation function, and the one i will use for this explanation, is a Sigmoid function which is visualised in Figure 3. A Sigmoid activation would be performed as follows:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{2}$$

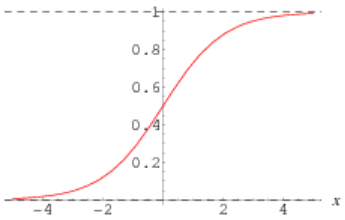

Figure 3: Graph of the Sigmoid function [2]

The summation and activation function for a neuron can be combined into one calculation for the final output value of a neuron as:

$$output(x_i) = \sigma(\sum_j w_{ij} x_j + b) \tag{3}$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-\sum_j w_{ij} x_j + b}} \tag{4}$$

Once the values have been propagated to the output layer and the neurons in the output layer have had summation and activation performed on them, we have the output of the neural network.

There are many different types of neural networks as well as just feed-forward and recurrent, for the purposes of this dissertation, I am using a convolutional neural network. Convolutional neural networks are design to "recognise two-dimensional shapes with a high degree of invariance to translation, scaling, skewing, and other forms of distortion" [6]. These convolutional neural networks use the main concepts of shared weights and biases, local receptive field, and very often, max pooling. The input for this kind of network would be the pixels of an image, for example, a 25x25 greyscale image would have an input layer size of 25x25, a sequence of 3 25x25 greyscale images would have an input layer size of 25x25x3.

Unlike a normal neural network, convolutional neural networks have layers known as convolutional layers, which are either a 2 or 3 dimensional structure. These convolutional layers use local receptive fields which connect "the input pixels to a layer of hidden neurons. However we won't connect every input pixel to every hidden neuron. Instead we only make connections in small, localised regions of the input image"[14].

Figure 4 visualises how a local receptive field of size 5x5 with a stride length of 1 works. The stride length is the amount of pixels we shift the local receptive field when applying pixels to the next hidden layer neuron. The size of the local receptive field is also then the size of the shared weights for the convolutional layer. These shared weights are used for every local receptive field in the current layer as well as a shared bias. The process for summation and activation of the result of a convolutional layer can expressed as [14]:

$$\sigma(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} x_{j+l,k+m}) \tag{5}$$

Where a 5x5 local receptive field is being applied for the j^{th}, k^{th} hidden neuron with σ as the activation function, $w_{l,m}$ as the shared weights, and x represents the value of the input neuron [14].

Convolutional neural networks can also have pooling layers, which can be "used immediately after convolutional layers" [14]. While these layers do no actual learning, they can perform a process known as max pooling. A max pooling layer will take the output of the convolutional layer (size $N * N$), extract the maximum of a $k8k$ region, and output a $\frac{N}{k} * \frac{N}{k}$ layer [5].

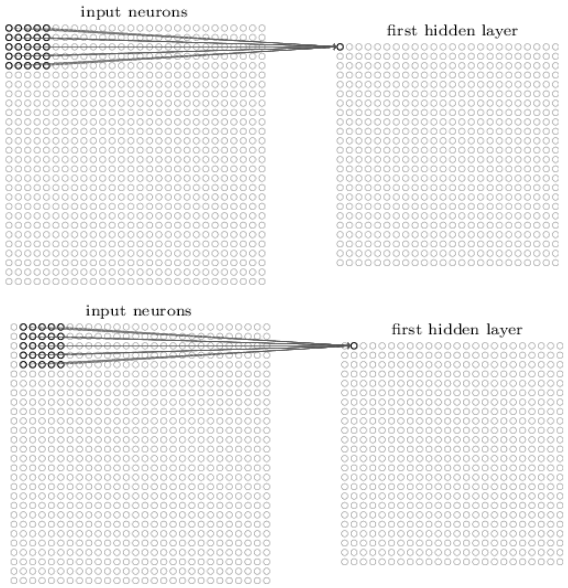

Figure 4: How local receptive fields work [14]

A Convolutional network can be made up of a sequence of convolutional layer followed by pooling layers and then normal feed-forward neural network layers such as fully connected layers, where every neuron in the current layer is connected to every neuron on the previous layer. You can also omit the pooling layer, particularly in the case of applying to low resolution images, and just use a sequence of convolutional layers followed by normal feed-forward layers.

The details covered so far only cover the forward propagation of a neural network, the network needs to also propagate backwards and tune parameters such as weights and biases to present a more meaningful output to given input, also known as learning. There are, generally speaking, three kinds of learning that can be applied in machine learning, supervised learning, unsupervised learning, and reinforcement learning.
Supervised learning requires "the availability of a teacher or supervisor who classifies the training examples into classes" [4]. For unsupervised learning the network "must identify the pattern-class information as a part of the learning process" [4]. Reinforcement learning uses the idea of states, actions and rewards, and encourages the network to perform actions in states that produce the greatest reward.

To actually be able to utilise any of these learning methods, the neural network needs to apply an algorithm to actually perform the learning process, the most popular of which is backpropagation. The output of the neural network

is highly unlikely to be correct on the first run of the network, due to the weights and biases of the network being assigned randomly at the start. We require a method to bring the actual output of the network closer to the desired output. This is backpropagation and is done using gradient descent.
Once the neural network has been run on several pieces of input data, we want to see how different the actual output is from the desired output. This error is calculated as [11]:

$$E_{total} = \sum \frac{1}{2}(target - output)^2 \tag{6}$$

$$E_{O_i} = \frac{1}{2}(target_{O_i} - output_{O_i}) \tag{7}$$

Equation 6 is for the total error of all output neurons, and equation 7 is for the error for one output neuron i.

The idea behind backpropagation is to find how much to change the weights and biases so that the actual output can be brought closer to the desired output. The process for this differs depending on if it is applied to an output layer, or to a hidden layer of the network.

When applying it to an output layer we want to find how much to change the weight w_{ij}, that connects a hidden neuron j to an output neuron i, to correct our output. So we need to calculate the change in E_{total} with regard to w_{ij}, this can be done with derivatives.

$$\frac{\partial E_{total}}{\partial w_{ij}} \tag{8}$$

$$\frac{\partial E_{total}}{\partial w_{ij}} = \frac{\partial E_{total}}{\partial out_{O_i}} * \frac{\partial out_{O_i}}{\partial net_{O_i}} * \frac{\partial net_{O_i}}{\partial w_{ij}} \tag{9}$$

Here we apply the chain rule to (8) to get (9) [11]. net_{O_i} is the net output of the output neuron i before the activation function has been applied.

Next we calculate the change in the E_{total} with regard to out_{O_i}, the change in out_{O_i} in regard to net_{O_i}, and the change in net_{O_i} with regard to w_{ij} [11].

$$\frac{\partial E_{total}}{\partial out_{O_i}} = -(target_{O_i} - out_{O_i}) \tag{10}$$

$$\frac{\partial out_{O_i}}{\partial net_{O_i}} = out_{O_i}(1 - out_{O_i}) \tag{11}$$

$$\frac{\partial net_{O_i}}{\partial w_{ij}} = out_{H_j} \tag{12}$$

$$\frac{\partial E_{total}}{\partial w_{ij}} = -(target_{O_i} - out_{O_i}) * out_{O_i}(1 - out_{O_i}) * out_{H_j} \tag{13}$$

In (11), the derivative of the activation function(sigmoid) has been calculated which is "the output multiplied by 1 minus the output" [11].

(13) can then also be written with the *delta rule* [11].

$$\delta_{O_i} = \frac{\partial E_{total}}{\partial out_{O_i}} * \frac{\partial out_{O_i}}{\partial net_{O_i}} = \frac{\partial E_{total}}{\partial net_{O_i}} \tag{14}$$

$$\frac{\partial E_{total}}{\partial w_{ij}} = \delta_{O_i} out_{H_j} \tag{15}$$

When applying backpropagation with gradient descent to a hidden layer, the process can be sped up by making use of the delta rule. We want to find the change in E_{total} with regard to w_{ij}, a weight connecting neuron j (can be in either input or hidden layer, depending on position in network) to neuron i in a hidden layer. [11]

$$\frac{\partial E_{total}}{\partial w_{ij}} = \delta_{H_i} * \frac{\partial net_{H_i}}{\partial w_{ij}} \tag{16}$$

$$\frac{\partial net_{H_i}}{\partial w_{ij}} = out_j \tag{17}$$

$$\frac{\partial E_{total}}{\partial w_{ij}} = \delta_{H_i} * out_j \tag{18}$$

Now to get our new value of w_{ij}, w_{ij}^+ we perform the following [11]:

$$w_{ij}^+ = w_{ij} - \eta * \frac{\partial E_{total}}{\partial w_{ij}} \tag{19}$$

With η being a learning rate for the neural network.

The learning method that will be used for this dissertation is backpropagation but implemented using a method known as RMSProp [7] a variant of Stochastic gradient descent. Stochastic gradient descent differs from normal gradient descent by applying to a random subset of training data as opposed to all of it. This approach is advantageous when you have a very large training set and/or a large amount of weights to update. RMSProp then builds upon this, it will "divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight" [7]. This particular method has been chosen as it has been proven to work in the application of neural networks to video games in the paper 'Human-level control through deep reinforcement learning' [12].

2.3 Reinforcement learning

The learning method that will be utilised for the neural network in this project is Reinforcement learning. Reinforcement learning, as mentioned above, uses the idea of states, actions, and rewards to encourage the network to perform actions that produce the best possible reward for their current state. It is a learning model reminiscent of how animals learn, "learning what to do - how to map situations to actions - so as to maximise a numerical reward signal" [16]. A machine learning model using reinforcement learning is able to enter an unknown situation and "learn from it's own experience" [16].

The particular method of reinforcement learning that is implemented here is Q-learning. In Q-learning we attempt to find the optimal action for each state to maximise the reward gained. This kind of learning very closely matches the basic idea of how to learn to play a video game. The way that Q-learning works is that "for every possible state, every possible action is assigned a value which is a function of both the immediate reward for taking the action and the expected reward in the future based on the new sate that is the result of taking that action" [18]. The value update function for this, as found in [18], is as follows:

$$Q(x,u) := (1-\alpha)Q(x,u) + \alpha(R + \gamma maxQ(x_{t+1}, u_{t+1})) \tag{20}$$

Where,

Q is the expected value of performing action u in state x,

R is the reward,

α is the learning rate,

γ is the discount factor.

This dissertation will use the neural network as a function approximator for Q-learning. This means that each input to the network (game screen data) will be treated as a state, and the output of the network (the action to perform in game) will be treated as an action, with an associated reward. The value update as seen in equation 20 will then be applied. The neural network used is therefore referred to as a Deep Q network.

3 Project Management

3.1 Software life-cycle

The software life-cycle used for this project is the agile development model, in particular the scrum model. The choice of agile for this project is advantageous compared to a more traditional model such as waterfall. This is because in

the Waterfall model "as the requirements change or as requirements problems are discovered, the system design or implementation has to be reworked and retested" [15]. This would be particularly problematic and result in a prolonged development process if and when errors occur, especially in earlier stages as it may in a undergraduate project such as this.

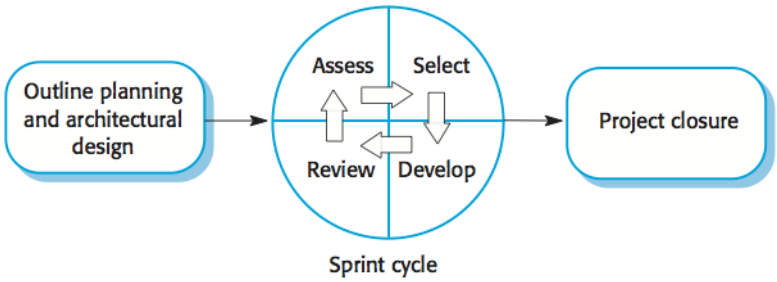

Figure 5: The Scrum development process [15]

The Scrum method is designed for small development teams, as a result many elements of scrum have been scaled down or omitted completely for application to this project. For example, there are several different roles for development team members such as a Scrum master, in this case the roles have been removed altogether.

Development in the scrum method is performed as displayed in figure 5, beginning with the outline planning stage. Here we establish the project's general objectives and design the software architecture. In this case, this stage was undertaken during the initial document. Next follows the main area of scrum, the sprint cycle. It is within this sprint cycle that the actual development is performed, and is done so in a series of fixed-length sprints. These sprints are then divided into four phases:

1. **Assessment phase**: A list of tasks to be performed on the project is reviewed, priorities and risks are assigned to them. In the review of tasks, more may be added.
2. **Selection phase**: The task or tasks to be performed this sprint cycle are chosen.
3. **Development phase**: Progress is reviewed at regular intervals, and task priorities can be potentially re-assigned.
4. **Review phase**: Tasks performed this cycle are now reviewed.

The final stage of scrum is then project closure, this wraps up the project with required documentation being produced and lessons learnt from the project

are assessed. The project closure process is this final dissertation document for the project.

I feel that scrum has been beneficial to the development of this project as it broke it down into smaller digestible sprints. Each sprint had clear goals set out, prioritised and then reviewed. It has also meant that there is less chance for any delay in development as there have been regular stops to review and reassess the tasks being performed.

3.2 Development

This project is developed in the C++ language. The development will be performed within the Windows Operating System and the project will be compiled and run within Linux. Windows is being used for development because it allows for usage of the Visual studio 2013 integrated development environment. This is preferable as it offers a built-in compiler and debugger for C++ as well as Intellisense code completion. Linux is then used for the final compilation and running of the project as tool to access video games and their data, the Arcade learning environment, is not compatible with Windows.

All code produced for this project will conform to 'Bob's concide coding conventions' [9] as outlined by Dr. Robert S Laramee of Swansea University.

3.3 Project timeline

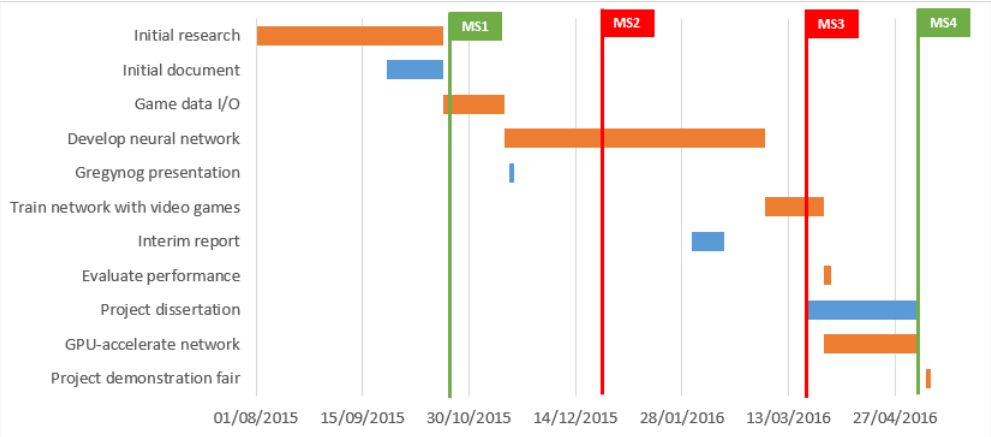

Figure 6: Proposed (and revised) project schedule.

Label	Milestone	Reason
MS1	End of initial phase, start of sprint cycle.	Scrum development model.
MS2	Start of Christmas break.	Allows more time dedication to project.
MS3	Start of Easter break.	Allows more time dedication to project.
MS4	End of project closure phase.	Scrum development model.

The project schedule, as seen in figure 6, has 2 kinds of tasks. Tasks set by the Computer Science department are those with blue bars, and tasks as set out by myself are those with orange bars. There are also 2 kinds of milestones, green milestones which represent the end of the intial and closure phases of scrum, and red milestones which represent the two holiday breaks in University term time. This breaks have allowed me to allocate more time to the development of the project and/or completion of documentation for the project.
The bulk of my time in this project has been dedicated to the development of the neural network in C++. This is both because it is the most integral component of this project, and because I required to learn the C++ language as I had no prior usage of it until this project.

3.4 Risks

The risks of this project are all relating to some kind of down time. In the case of myself getting ill, be it in general or from migraines, or a close family member becoming ill, it results in slowed progress and can be mitigated by keeping up-to date with sprint tasks. The other kind of downtime comes from hardware downtime, either my own or on the end of a service such as GitHub, on which I host my version control and backups. These issues could result in loss of project data, potentially crucial, and can be mitigated by keeping sufficient backups of files, in several places, and having backup hardware.

These risks can be seen in the risk table in figure 7, showing the probability and impact of a risk scaled from 1 to 10, with 1 being low and 10 being high. A risk score is calculated for each risk then, as the product of the probability and impact of the risk, and the risks are ordered by this risk score.

Of these risks, some did come in to play during the development of the project. **COMPLETE THIS SECTION***COVER MIGRAINES COMING ON MORE FREQUENTLY !!!!!! AS WELL AS REMOVING TOO AMBITIOUS RISK IN THE INTERIM AND THEN THE PROBLEMS FACED.*

Risk	Probability	Impact	Risk score	Mitigation
Illness of close family member that will require my care.	5	6	30	Have good time management to afford missed days and/or catching up.
GitHub has a service failure.	3	7	21	Keep sufficient offline backups of project development
Development slowed due to Migraines.	6	3	18	Have good time management to afford missed days and/or catching up.
General illness.	5	3	15	Have good time management to afford missed days and/or catching up.
Hardware failure.	2	5	10	Keep sufficient backups of files, and spares for components.

Figure 7: Risk table

4 Technologies

4.1 C++

This project has been developed using the C++ language. This choice was made as C++ is both object-oriented, and low level. As well as this C++ is compatible with CUDA, so it would be possible to be able to GPU-accelerate the neural network.

As this was my first time using the C++ language, there were a few hiccups along the way. For example object scope, new neurons are created within loops, as many are potentially being made for each layer. These neurons are then stored in vectors for their layer, however when trying to reference these neurons later, they no longer existed. This is because when C++ left the loop, garbage collection was performed to remove the neuron objects as they went out of scope. **As I am more experienced in Java, I did not expect this issue, so after realising what was causing the issue, the object scope, I fixed it by using the New keyword. This new keyword means that all garbage**

collection is performed manually, and scope is no longer an issue.

4.2 Git

In this project, Git has been used for both version control and backups with the service GitHub being used for the repository. To perform the version control, local repositories will be created on development devices. There are also two main branches, master for working code, and dev for code that is being implemented and not yet confirmed to works. Whenever an existing file has been edited or a new file has been created, the following steps will be carried out in the operating system command line (when located in repository directory):

```
git add FILE
git commit -m ``comment on changes"
git push
```

This means I can track what changes have been made for each update in the commit message, and have the ability to safely roll back if the implemented changes are no good.

4.3 The Arcade learning environment

The neural network needs a way of interfacing with video games, and this can be done using a tool know as the Arcade learning environment. The arcade learning environment is built an Atari 2600 games console emulator known as Stella, so it is limited to only allowing interface to games for that games console. This environment was created alongside the paper 'The Arcade learning environment: An evaluation platform for general agents' [3] by DeepMind, and was used in both aforementioned DeepMind papers. It has been made freely available online for use by "researchers and hobbyists to develop AI agents for the Atari 2500" [3].

The arcade learning environment is compatible with a languages such as C++, Java, and Python and presents several interfaces for these languages to communicate with it and therefore with the video games. **CHECK THE MANUAL, THE FIFO PIPS ARE DIFFERENT TO WHAT IS BEING USED NOW?!?**

5 Analysis of problem

5.1 Video games

To be able to apply the neural network to a video game, we will need access to control the video game, as well as the game screen data. The first option

for this was the use open-source video games that specifically targeted the PC platform. These would allow access to control the video games with some modification however they may not have necessarily given us a straight forward way to access information on the game screen. The programming language to be able to interface with these games would depend entirely on the source code for the video game being chosen.

Another option I had was to use video game console emulation software, such as the 'BizHawk emulator'[**INITIAL BIZHAWK REFERENCE**] which can emulate several consoles. Communication to video games being played within this emulator would be through the it's memory addresses. This can be done via scripting in *Lua*, however it would be very cumbersome, as the memory addresses are different on a game-by-game basis, and they aren't very well documented.

Building on top of the idea of using video game console emulation software, the Arcade learning environment was considered and then later chosen as the way to access video games. This is due to it's use in existing work by DeepMind, and that it allows a unified interface for many different video game, albeit limited to the Atari 2600 console.

5.2 Intelligence agent

The intelligence agent in this case will be where the neural network and the interface for arcade learning environment interface will be instantiated. In this agent they will then communicate with each other, while going through the algorithm for performing Q-learning. This intelligence agent is what will be called from the final programs *main* method, and it will handle the rest.

As outlined in the work from DeepMind in regard to a Deep-Q learner, the intelligence agent will use something known as *exploration-exploitation*. **DISCUSS HOW THIS WORKS, AS WELL AS THE EPOCHS AND HYPERPARAMETERS.... LIFT FROM PAPER AND REFERENCE**

Could include the algorithm with a reference, and then discuss

5.3 Preprocessor

Game screens are output from the Arcade learning environment as size 210x160 with 8-bit hexadecimal values for a 128-color palette specific to the NTSC version of the Atari 2600. The neural network will take greyscale images of size 84x84 (as explained in section *Network structure*) as input, therefore the image we receive from the environment needs to be preprocessed. First we need to

select a region of interest within the image, to reduce the height of the image from 210 to 160. The image will then be downscaled by a factor of 0.525 to get the correct size of 84x84. Finally the 8-bit hexadecimal values will be converted into greyscale integer values using a look-up table.

5.4 Network network

The chosen topology for the neural network for this project is a deep convolutional neural network. This is due to both their proven ability to work well with image recognition problems, and the usage in the work of DeepMind.

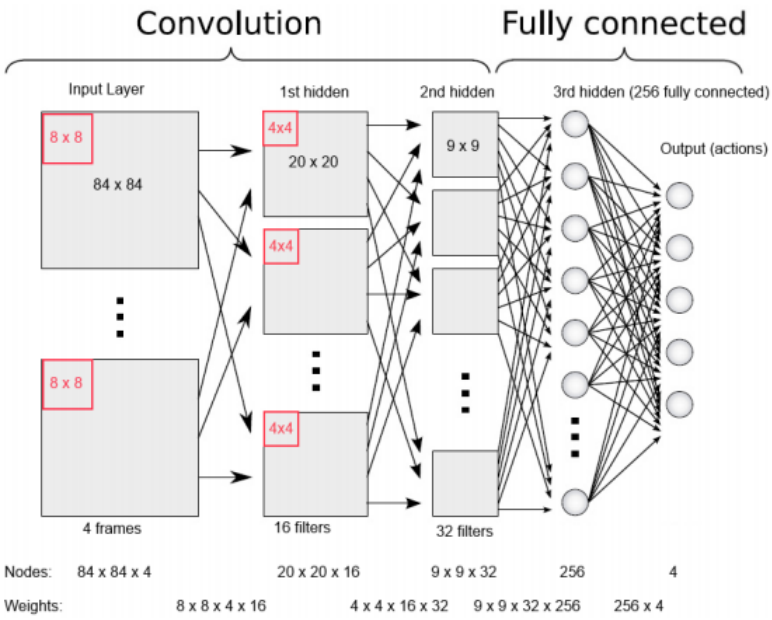

Figure 8: DeepMind original topology [1]

In the 2013 paper from DeepMind [13], the network structure seen in figure 8 was used. This was later extended to the network structure seen in figure 9 in 2015 when published in Nature.

For this implementation, I have chosen the 2013 DeepMind topology, as it has been proven to work in this application to a broad set of video games, and is less complex than it's 2015 counterpart in terms of computation. This topology takes in four greyscale images of size 84x84, and then applies four filters of size 8x8, with stride of 4 **MAY NEED TO EXPLAIN STRIDE IN EARLIER SECTION**, to produce sixteen feature maps of size 20x20 in the second layer. These filters are applied as local receptive fields, which as explained in

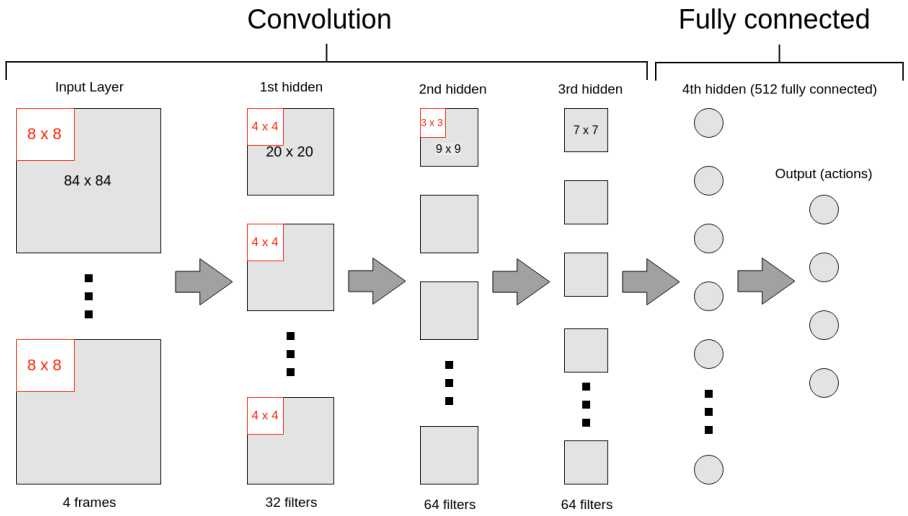

Figure 9: DeepMind extended topology

the earlier section on neural networks, sweep across the image with many pixels of the image, or neurons in the case of the network, sharing the weights of the filter(s).**This could be explained above instead if not already?!?** The second layer then is also a convolutional layer, and applies two filters of size 4x4, with stride 2, to produce thirty-two feature maps of size 9x9 in the third layer. This third layer is then connected to the fourth layer, which is a fully connected layer of size 256. **Explain FullConn Layer in NN theory?!?!** The fourth layer then connects to the final layer which is fully connected and is of varying size depending on the amount of actions available in the game the network is being applied to.

It's worth noting that this convolutional neural network does not have any pooling layers. This is due to the nature of the images being used for this network application.

Example image from atari, maybe have a real-life photo to compare against (if so, make it applicable e.g. of a game controller or console

5.5 Learning

The network will learn overall using the reinforcement learning method of Q-learning, making it a deep q-network. The backpropagation learning for the network will use a technique known as RMSProp **REFERENCE THE HINTON SLIDES**.

Explain how the Q-Learning works, reiterate the main formula for it, and pull the Q-learning part of main algo/agent

Explain how RMSProp works in theory as well as how it fits in

6 Implementation sprints

Use algorithm figures here to show how things have been implemented! As well as the dates from GDrive for sprints

6.1 Methods to manipulate I/O

6.2 Preprocessor

6.3 Network generation

6.4 Network feedforward

6.5 Backpropagation

6.6 Replay Memory

6.7 Artificial Intelligence Agent

7 Reflection of results

8 Suggestions for further work

References

[1] Korjus et al. *Replicating the paper playing Atari with deep reinforcement learning*. Tech. rep. University of Tartu, 2014.

[2] Wolfram Alpha. *Sigmoid Function*. URL: `http://mathworld.wolfram.com/SigmoidFunction.html` (visited on 13/10/2015).

[3] M. G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279.

[4] LiMin Fu. *Neural Networks in Computer Intelligence*. New York, NY, USA: McGraw-Hill, Inc., 1994. ISBN: 0079118178.

[5] Andrew Gibiansky. *Convolutional Neural Networks*. URL: `http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/` (visited on 14/10/2015).

[6] Simon Haykin. *Neural Networks, A Comprehensive Foundation*. 2nd ed. Prentice Hall, 1999. ISBN: 0139083855.

[7] Geoffrey Hinton. *Overview of mini-batch gradient descent*. URL: `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf` (visited on 08/01/2016).

[8] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo. "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance". In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*. Vol. 1. IEEE. 2005, pp. 85–89.

[9] Robert S Laramee. "Bob's Concise Coding Conventions (C3)". In: ().

[10] Nicholas Lundgaard and Brian McKee. *Reinforcement learning and neural networks for Tetris*. Tech. rep. Technical Report, University of Oklahoma, 2007.

[11] Matt Mazur. *A Step by Step Backpropagation Example*. URL: `http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/` (visited on 12/10/2015).

[12] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[13] Volodymyr Mnih et al. "Playing Atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[14] Michael A. Nielson. *Neural Networks and Deep Learning*. URL: `http://neuralnetworksanddeeplearning.com/` (visited on 12/10/2015).

[15] Ian Sommerville. *Software Engineering*. 9th ed. Harlow, England: Addison-Wesley, 2010. ISBN: 978-0-13-703515-1.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2011.

[17] Gerald Tesauro. “Temporal difference learning and TD-Gammon”. In: *Communications of the ACM* 38.3 (1995), pp. 58–68.

[18] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

A Appendix: Class diagrams