Evolution of Neural Network Controllers for Gameplay Behaviours

Ryan O'Flaherty
40168766

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Games Development

School of Computing

November 9, 2017

Authorship Declaration

I, Ryan O'Flaherty, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date:

Matriculation no:

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

♣

♢

♡

♠

Contents

List of Tables

List of Figures

Acknowledgements

The Games Development course has been journey of personal development in addition to gaining a higher education by obtaining and improving skills. It has been the proverbial roller-coaster ride with several highs, lows, moments of complete disorientation, and a feeling of wishing you could do it all over again. The last four years have gone by in the blink of an eye, but have provided experiences and created friendships that will remain far beyond graduation.

I would like to thank my friends and family for ensuring that I remained positive and motivated along the way, and for their participation in the testing phases. All of the support, assistance and feedback has been invaluable to both myself and the project as a whole.

I am also hugely appreciative of my project supervisor, Dr Simon Powers. Simon has provided me with excellent support and guidance throughout the process and for that I am extremely grateful.

1 Introduction

This chapter highlights the aims and objectives for this honours project, providing a brief insight into the what is to come. It also outlines the desired deliverables for the completed project.

1.1 Project Aims and Structure

This project intends to provide a digitally playable card game that can be played by a human user against one or more artificial intelligence agents. The underlying purpose of the project is to research and demonstrate the effectiveness of artificial neural networks and evolutionary algorithms in a game of this type.

It plans to do so by using the NeuroEvolution of Augmenting Topologies (NEAT) library to co-evolve neural network structures as well as input weights to make decisions relating to in-game moves that can be made, and to learn from the results of those decisions.

This report will document how well the agent performs over a vast amount of games and topologies, presenting the results through a mixture of figures and charts. An in-depth explanation of the way in which the performance is measured will be described in the methodology section.

While the project has research at its core, it is also intended to be a game that can provide an enjoyable experience for a playing user who may not appreciate what is going on under the hood.

1.2 Overview Of Project Content and Milestones

Milestone proposals for the project are as follows:

1.2.1 Research

Neural networks and Evolutionary algorithms research is necessary to aid with the project development and to make up for current lack of knowledge

1.2.2 Game Foundation

A short-term goal of the project is to create a bare-bones version of the card game without any artificial intelligence. This has to be created as it is the foundation on which the rest of the project will reside, and as such it needs to be rigorously tested to ensure that the rules of the game have been correctly implemented without any bugs. That way, any future problems will be related to the artificial intelligence itself.

1.2.3 Implementation of Artificial Intelligence

Most likely the phase of greatest difficulty and complexity, the implementation of artificial neural networks and evolutionary algorithms is expected to be a largely time consuming task.

This will be where the game will transition from a hard-coded, bare-bones implementation to something of more interest. Once this has been successfully set up, we can begin testing and tweaking it to analyse the varying results.

1.2.4 Experiments and Testing

A large amount of games will be required to allow the evolution process to grow into something that performs to a decent level in our game. This will mean the game will have to be played multiple times, and so it will be beneficial to get other people to play-test the game too.

Each time tweaks are made in our algorithm, the game will need to be thoroughly re-tested, with the results of these tests being accurately documented to provide a solid basis for the next stage of tweaking.

The aim is to have a very large set of data to analyse and draw conclusions from in the latter stages of the project.

1.2.5 Evalution and Conclusion

The aforementioned dataset will be used extensively to deeply examine the performance of each machine learning techniques implemented throughout the experimental stage of the project, with regards to both our hard-coded solutions and the results of other attempted solutions.

2 Background

2.1 Introduction

The following section of this dissertation will go on to discuss the history of artificial intelligence within the context of video games, before going on to explain neural networks, evolutionary algorithms, and the NeuroEvolution of Augmenting Topologies (NEAT) library.

2.2 History of AI in Games

Video games have been a popular area of interest for artificial intelligence developers and researchers for many decades.

Over several tens of years, a large amount of research and development has been done in an attempt to perfect chess-playing artificial intelligence agents(Thrun, 1995), and work has more recently been put in to do the same with the game of Go.

In March of 2016, the goal of getting such an agent to compete and win at the highest level was reached when AlphaGo, a program engineered by Google, managed to overcome the Go world champion human player, Lee Sedol(Kurenkov). This was then reported as a major breakthrough for the artificial intelligence field.

With Go, developers are unable to use brute force approaches due to the vastly complex nature of the game(Granter et al.). In other words, the complexity of Go means that a strategy cannot be hard-coded to play the game, and things like search trees are not applicable either. AlphaGo's solution to this was the implementation of a neural network to learn and play the game, taking into consideration millions of game states from previous games involving human expert players.

This would theoretically result in AlphaGo performing at a similar level to that of the human players it learned from. To go above and beyond that level, and to be able to beat a champion like Sedol, AlphaGo was then put up against itself, consequently improving with every game it played(Granter et al.). It can be said that in this phase, AlphaGo literally taught itself.

In terms of Chess, a machine managed to win against a human of World Champion status named Gerry Kasparov in the late 1990s(Campbell et al., 2002). That machine was Deep Blue. In actual fact, it was the second version of IBM's Deep Blue. The first Deep Blue lost to the same opponent a year prior to it's successor's achievement in 1997(Campbell et al., 2002). That success came in the form of two wins, a loss, and three draws(IBM, a), over the course of several days. However, IBM did not use artificial intelligence with Deep Blue, relying instead on computational power and a less complex search and evaluation function(IBM, b).

2.3 What is a Neural Network?

Artificial neural networks, commonly referred to simply as neural networks, are a rough representation of the human brain. Human brains can be thought of as highly complex and non-linear systems for processing information in parallel(Haykin, 1999).

To emulate this, neural networks are built using a series of layers of network nodes. These nodes are used to represent neurons in the brain. The first is an input layer, followed by two or three hidden layers before a final output layer of nodes(Wang, 2003). The connection between these nodes is representative of axons in the brain.

In an artificial neural network, the input and output layers take data in and produce results respectively, with the processing of said data being done within the hidden layers - but how does it work?

2.3.1 Perceptrons

One type of artificial neuron (or node) is known as a "perceptron." Each perceptron receives several inputs and uses them to produce one binary output(Nielsen, 2015).

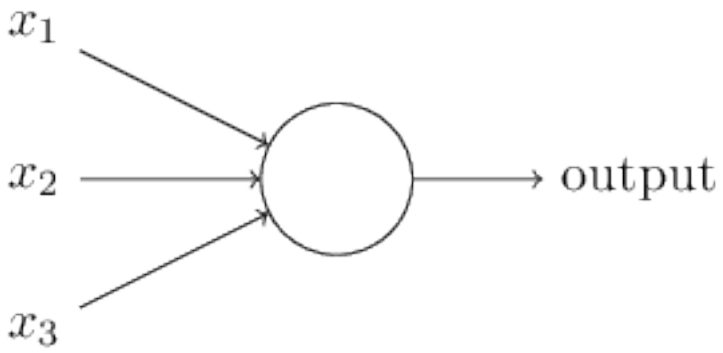


Figure 1: A Single Perceptron

Frank Rosenblatt, the scientist who developed the perceptron in the 1950s and 1960s, devised a rule for computing the output from these neurons. Using what he called "weights," the importance of each input is assessed and expressed. Each input has a weight assigned to it, and the resultant output from these inputs - either a 1 or 0 - is dependent on whether or not some threshold value is less than or greater than the sum of the weights from all of the inputs to that particular perceptron. Therefore, if the weighted sum is less than or equal to the threshold value, the output is a 0. Otherwise,

a 1(Nielsen, 2015). Both the threshold value and the input weights are real numbers. These can be tweaked to alter the decisions made by a neural network.

2.3.2 Sigmoid Neurons

Sigmoid neurons are akin to perceptrons, however, they are modified in such a way that marginal alterations in their weights and bias cause only a small change to their output(Nielsen, 2015). This crucial difference is what affords a network consisting of sigmoid neurons the ability to learn.

The inputs to a sigmoid neuron also differ from those of perceptrons. Rather than being binary (1 or 0), these inputs are any number *between* 1 and 0. Much like with perceptrons, these sigmoid neuron inputs are weighted, with an overall bias included. These can be denoted b, w_1, w_2, ... w_n where b represents the bias, and each w is an input weight. This time however, the output is non-binary. To calculate it, we use

$$\sigma(w \cdot x + b) \tag{1}$$

where σ is known as the sigmoid function, which is defined as:

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

In its full extended form, with x being used to symbolise the inputs, the output of a sigmoid neuron is calculated as

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}. \tag{3}$$

//
To understand the similarity to the perceptron model, suppose z $\equiv w \cdot x + b$ is a large positive number. Then $e^{-z} \approx 0$ and so $\sigma(z) \approx 1$. In other words, when $z = w \cdot x + b$ is large and positive, the output from the sigmoid neuron is approximately 1, just as it would have been for a perceptron. Suppose on the other hand that $z = w \cdot x + b$ is very negative. Then $e^{-z} \to \infty$, and $\sigma(z) \approx 0$. So when $z = w \cdot x + b$ is very negative, the behaviour of a sigmoid neuron also closely approximates a perceptron. It's only when $w \cdot x + b$ is of modest size that there's much deviation from the perceptron model(Nielsen, 2015).

//
The shape of a plotted σ function can be seen in Figure 2.

//
The smooth nature of σ means that minor changes in the weights and bias

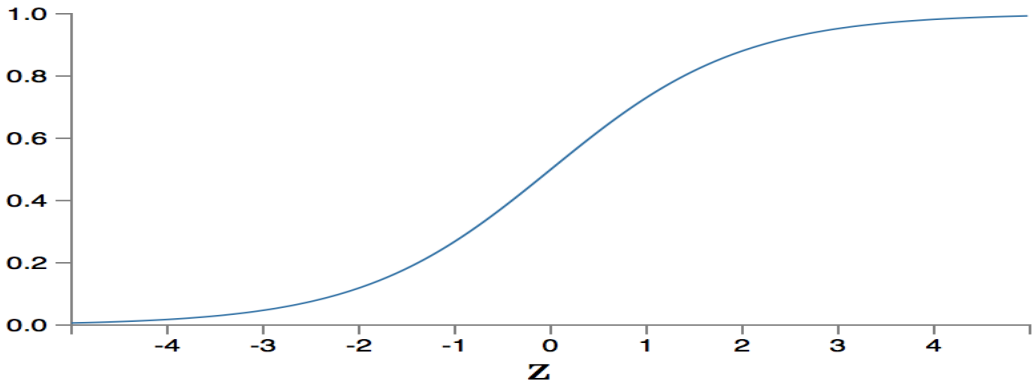


Figure 2: Sigmoid Function
(Nielsen, 2015)

- which are depicted as Δw_j and Δb_j respectively - will consequently create small changes to the output from the neuron. That change - Δoutput - can be approximated with calculus:

$$\Delta\text{output} \approx \sum_j \frac{\partial\,\text{output}}{\partial w_j}\Delta w_j + \frac{\partial\,\text{output}}{\partial b}\Delta b \tag{4}$$

The sum is over all of the weights, w_j, and $\partial\,\text{output}/\partial w_j$ and $\partial\,\text{output}/\partial b$ denote partial derivatives of the output with respect to w_j and b, respectively.

As Δoutput is a linear function of the changes Δw_j and Δb_j in the weights and bias, this linearity makes it easy to choose small changes in the weights and biases to achieve any desired small change in the output. So while sigmoid neurons have much of the same qualitative behaviour as perceptrons, they make it much easier to figure out how changing the weights and biases will change the output.

//

2.4 Evolutionary Algorithms

As the name might suggest, an evolutionary algorithm is one that evolves. It does so to encourage finding the most optimal solution to a problem. A vast amount of varying evolutionary algorithms exist, but at the core of them all is the same principal idea: "given a population of individuals the environmental pressure causes natural selection (survival of the fittest) and this causes a rise in the fitness of the population"(Eiben and Smith, 2015). As a result, the population adapts over time to its environment.

What is the process of evolution? The generational cycle works as follows:

2.4.1 Representation

Step one when defining an evolutionary algorithm is to bridge the gap between the "real world" and the "evolutionary algorithm world"(Eiben and Smith, 2015). That is, to link what are known as phenotypes, to representative genotypes.

- Phenotype:
  - A solution to a problem
- Genotype
  - Chromosome used to represent the solution to a problem

Representation refers to specifying which genotypes equate to each phenotype(Eiben and Smith, 2015). For example, if an integer is the solution to a problem, it is the phenotype, and a binary representation of that particular integer would be the genotype relating to that phenotype.

A genotype (or chromosome) is made up of genes. Values are assigned to each gene, and may be referred to as alleles. These can be of any type, or even a mixture. Types include binary strings, integers, real values, permutations and symbols. So in the example above, each gene would be either a 1 or 0, combining to form the integer as an overall chromosome.

Sometimes it may not be quite so straightforward, and genotypes need to be explicitly mapped to phenotypes, as seen in Figure 3.

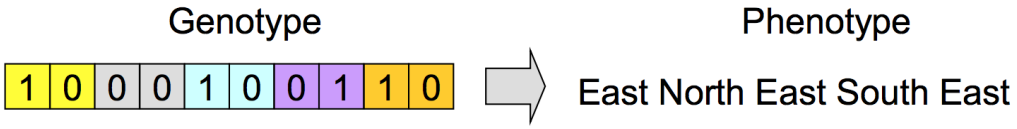


Figure 3: Genotype vs Phenotype

2.4.2 Initialisation

In the beginning, we start off with a population comprised completely of random chromosomes. This is likely to yield very poor results, however there is always a chance that some may be better. These individuals must then be evaluated.

2.4.3 Fitness

The fitness of an individual is what defines how fit for purpose it is. This measurement of quality will be defined differently for every algorithm, depending entirely on the context of problems it is being used to solve. In this project, the fitness will correlate to the amount of illegal moves the agent attempts to make, and how many times it is forced to increase its hand rather than decrease it. However, as the game of switch is largely down to luck, this will have to be taken into account to incorporate some form of leniency to fitness calculations.

The role of an evaluation function (or fitness function) to encourage improvements by defining what an improvement is(Eiben and Smith, 2015). This lays the foundation for selection.

2.4.4 Selection

The process of choosing individuals, based on their fitness, from the population to become parents to the next generation of individuals is called selection(Eiben and Smith, 2015). This is the driving force behind progressive evolution as it biases selection towards individuals of higher quality.

2.4.5 Crossover

Sometimes referred to as recombination, crossover is an operation that merges genes from two parent genotypes together into one or two offspring genotypes(Eiben and Smith, 2015).

Crossover can be defined in a few ways. There is one-point crossover, where a randomly chosen point along the length of a chromosome determines which genes are passed on from that particular genotype, and the rest will come from the other parent. This is demonstrated in Figure 4.

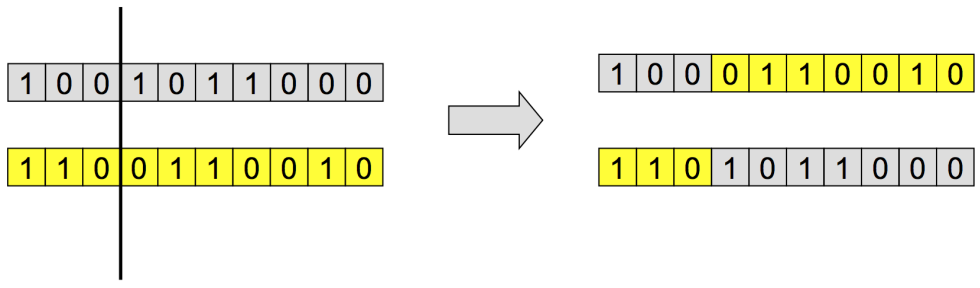


Figure 4: One-Point Crossover

Next, n-point crossover is when more than one (n) point is chosen, and chromosomes can be split up in different ways, as seen in Figure 5.

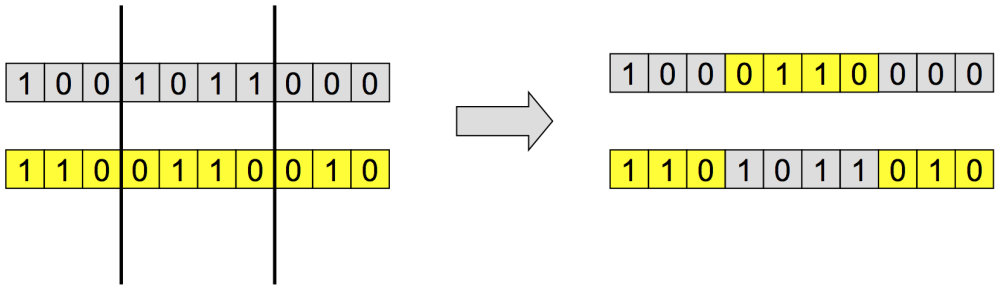


Figure 5: N-Point Crossover

Another type of crossover is called uniform. In this case, each gene of the offspring is randomly selected by deciding which of the two parents to inherit from. This is demonstrated in Figure 6.

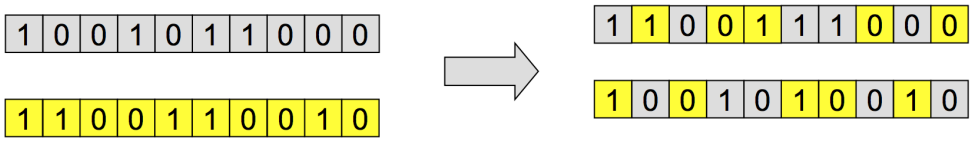


Figure 6: Uniform Crossover

In the case of arithmetic crossover, an average of the two parent genes is calculated and used for the child gene. This is useful if the genes consist of real numbers. Figure 7 depicts this.

All of the above techniques are used in binary and integer chromosome representations. Permutations cannot be recombined using any of these techniques, but are beyond the scope of this project.

2.4.6 Mutation

When applied to a genotype, mutation returns a slightly mutated offspring(Eiben and Smith, 2015). It can create new genes in the population, which in turn diversifies the population.

Like with crossover, there are different ways to perform mutation. In the case of binary chromosomes, we would allow each gene to 'flip' from a 1 to a 0 or vice versa, as demonstrated by Figure 8. Each gene has a probability

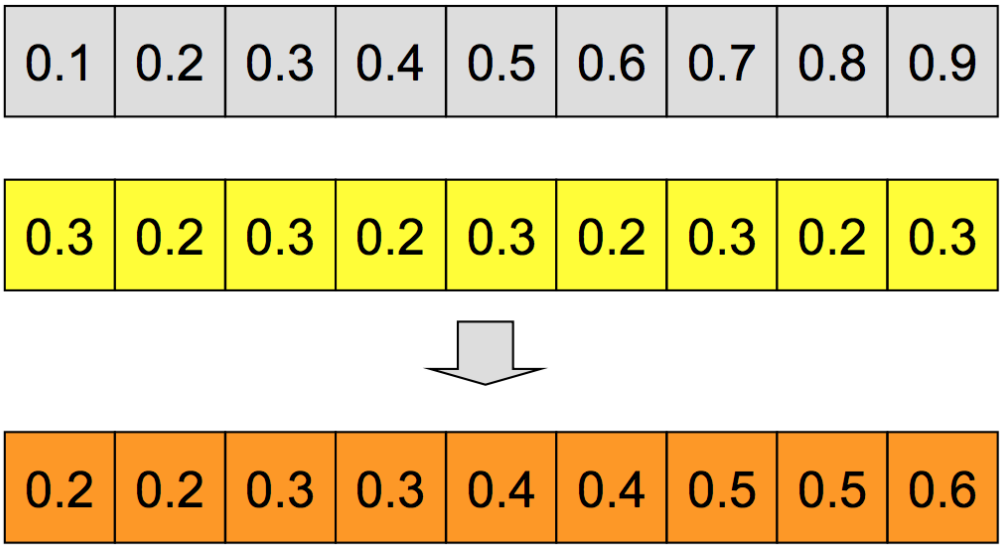


Figure 7: Arithmetic Crossover

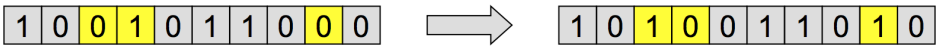


Figure 8: Binary Mutation

of being mutated in this way, which will often be calculated as $1/n$ with n being representative of the chromosome length.

Integers are slightly different. The probability is remains the same, but the difference is that a set of possible numbers, for example 0 to 9, is set up and when a gene is chosen for mutation, a number within that range is randomly generated and used in the offspring. This is displayed in Figure 9.

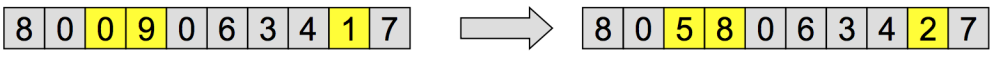


Figure 9: Integer Mutation

Again, permutations work in a completely different way, but are not considered as part of this project.

2.4.7 Replacement

This is the part where individuals are removed from the population to make way for a generation of new and hopefully improved genotypes. We could just remove the oldest genotype in the population, but this could be one with a high fitness! This can also be done randomly, but again there's a risk that we could be removing individuals of high quality. The other way is to determine which individuals to remove using the fitness. We could just get rid of the least fit genotypes, which could lead to the population improving very quickly, however it could also lead to premature convergence.

2.5 NeuroEvolution for Augmenting Topologies (NEAT)

"NEAT is a method for evolving speciated neural networks of arbitrary structures and sizes. NEAT leverages the evolution of structure to make neuroevolution more efficient"(Stanley and Miikkulainen, 2002b).

It is claimed to result in significantly faster learning than neuroevolution techniques using fixed network topologies.

In NEAT, the mutation phase can alter network structures as well as connection weights(Stanley and Miikkulainen, 2002a). While connection weights are mutated in the same way as described in the previous section, network structure can be mutated in two ways:

- Add Connection
  - A new connection gene is added, linking two nodes that were not connected beforehand
- Add Node
  - A connection that already exists is split and the new node replaces the old connection. That previous connection is disabled and the genome gains two new connections.

3 Literature Review

Video games are a field that has been used as a catalyst for research and development in artificial intelligence due to the relatively risk-free nature of it when compared to other areas where AI might be used for quite some time now, but it was only recently that a program was able to beat a world class human player in the game of Go(Kurenkov). This is despite a huge amount of work and time being injected into developing these types of game-playing agents with the desired result of beating the best human players in the world at Chess, and more recently, Go.

This project aims to create a digital version of the card game Switch. While playing, each competitor has no idea what cards are held by their opponent(s). They only know what cards they themselves hold, and what the most recently played card was. They will also know whose turn it is and what direction the play is going (if there are multiple remaining opponents).

Solutions for Chess and Go might use search trees for decision making, however this requires knowing the full state of the game, including the location of every game piece on the board. This is what is known as "perfect information." However, in card games such as Poker or Switch, there are unknowns such as the hands other players have or the value of cards that are face down. Therefore, we are faced with *"imperfect information."*

3.1 AI for Playing Games

When researching neural networks in relation to games, a stand-out is MarI/O. There is a video on YouTube showcasing its solution after 34 generations using NEAT. It is described on the page of that video as the following:

"MarI/O is a program made of neural networks and genetic algorithms that kicks butt at Super Mario World"(SethBling, 2015).

Despite that description, the intelligence of MarI/O is questionable. Although it has mastered the art of manoeuvring its way across the level in question, that is all it has done. In other words, if we were to take the same agent and run it on another Super Mario level, it wouldn't do so well. It has figured out how to pass through the level by continually moving right, and jumping at optimal times to avoid enemies and gaps in the map. Albeit successful, this is not a strategy that could be deemed as intelligent.

For Alan Turing to consider a machine intelligent, it must be able to act in a way that would be indistinguishable from the way a human would act. The Turing Test, created by Turing himself in 1950, was an imitation game whereby a human would hold textual conversations with another human, and

Figure 10: MarI/O
(SethBling, 2015)

with a computer. If the testing human is unable to successfully differentiate between the two based on interrogation within those conversations, then Turing would deem it unreasonable not to call the computer intelligent(Turing).

Although dealing in a different realm altogether, applying the same principle to MarI/O and questioning if the strategy it presents could pass a similar test of human judgement to decide whether or not it is a strategy that is likely to have been played by a human, then we cannot possibly declare MarI/O's technique as an intelligent solution.

3.2 Why Neural Networks?

This project has dealt a hand presenting a problem in which a decision making process will have to make use of imperfect information. Not only can neural networks cope with this, in fact, they excel in situations of imperfect information.

Without a neural network, any given scenario within a problem would need to have some kind of hard-coded solution that is step-by-step in nature. Through a learning process, a neural network can find solutions to these scenarios on its own, via exploration of the possibilities thrown up by its decision making process over several generations.

Imagine having to write the code for a solution to every single possible game state in Chess. For every move, you would have to write a solution for it as many times as there are possible opportunities for that particular move, i.e, every possible board configuration in which that single move is legal. This would take an unthinkable amount of time, and in reality is probably not even possible. Providing a coded solution for every possible board configuration would require a huge amount of code segments declaring that if some state condition is true, then do move this piece to this location. In doing so, we would find ourselves with an enormous file size, requiring unrealistic amounts of memory to run the code.

On top of that, it could take the machine a long time to search through all of the conditions to find the one that applies to the situation at hand.

In Chess, there are 400 configurations possible after each player has made a single move each, and over 72,000 after two each(Fuhriman). This continues to grow at an exponential rate, making it easy to see why implementing a fully coded strategy is unrealistic. Using a neural network and allowing it to analyse every situation it encounters on its own, deciding what moves to make and learning from the results is a far better idea.

Additionally, hard-coding solutions to given situations would make the game predictable, and could also lead to making moves that are not actually the best for that current game state. The flexible nature of a neural network provides the potential to reach better solutions that hard-coding might miss out on.

In our Switch card game for example, there is a rule that states if the previous player played a 2, you must pick up two cards unless you have another 2. Knowing this, and wanting to force our opponent to inherit more cards, we might hard-code a strategy that says, "if you have a 2 and it's available to be played, play it." This could cost us a chance to win the game though!

If we only had two cards left, both of the same suit and matching the suit of the most recently played card, one being a 2 and the other being a 7, we could play the 7 first, and that would allow us to play a run of that suit. This would allow us to play both cards and win the game. A neural network might learn that playing a 7 is better than deploying a 2, but our hard-coded strategy may not.

Another implementation of artificial neural networks, albeit eerie, is Alter(Smith). Replicating the upper-body of a human, Alter is a robot that can almost be described as being *alive*.

Technologically, it consists of 42 pneumatic actuators and a "central pattern generator" with a neural network used to let the robot develop movement patterns(Smith). The network receives input in the form of sensory data relating to temperature and humidity as well as proximity and noise(Orf).

Although it does not behave in a human-esque manner, Alter is continually perceiving the nearby environment, and reacting to it in such a way that is completely uncontrolled by a human, and not pre-programmed. This is how it somewhat provides the illusion of life.

RoboCup(RoboCup) is a robotics competition whereby robots compete in a game of football (soccer). The competition was launched by Japanese researchers in 1993, who soon after found themselves inundated with requests from other nations to make the project an international joint effort(RoboCup).

The robots that participate in these competitions vary in size and shape. Thus, detecting fellow robots visually is not a simple task. It is however a job for a neural network of multi-layer perceptrons(Kaufmann et al., 2004). Robots should also be able to recognise team-mates and tell them apart from opposing players.

In terms of playing the game, it is infeasible due to the dynamic nature of the game to consider all situations ahead of time when programming the robotKitano et al.. Machine learning is therefore a necessity for RoboCup participation.

3.3 Why Evolve Neural Networks?

Artificial neural network evolution has demonstrated positive results with tasks involving reinforcement learning, and has performed especially well with those that include hidden state information(Stanley and Miikkulainen, 2002a). This appeals to the needs of this project as it deals with unknown game aspects, such as the cards in other people's hands.

The evolution of neural networks has demonstrated a large degree of potential when coupled with tasks solvable by reinforcement learning techniques(Stanley and Miikkulainen, 2002a). It outperforms the basic methods of reinforcement learning against tasks that are considered benchmarks, and is therefore a justifiably sought after means of decision making.

Another arcade-style game that has been used in the research of neural

networks is PAC-MAN. In a particular paper introducing the concept of trying to evolve a player for the retro classic, a simple ghost avoidance strategy was given a hard-coded implementation for comparison purposes. Perhaps expectedly, the results were underwhelming. The aim was to demonstrate that the neural networks were learning something more sophisticated than such a simple strategy(Lucas, 2005). Evolving neural networks and allowing the controller to develop its own strategies, as opposed to hard-coding them, produced universally superior results.

To demonstrate the significance of evolved neural networks in the real world, let's look at an open source software library that is used worldwide.

Google initially developed TensorFlow, the successor to the DistBelief system they used from 2011(Abadi et al., 2016), for research in neural networks and machine learning(Google), but has since grown into a much wider-serving interface. It has found itself deployed in complex computational areas such as speech recognition and natural language processing, as well as image recognition(Abadim et al., 2015).

Google themselves use TensorFlow for a variety of different projects; Massively Multitask Networks for Drug Discovery and RankBrain are examples of large-scale deep neural network projects owned by Google, used for drug discovery and information retrieval respectively(Google). Other companies which benefit from the use of TensorFlow include Snapchat, Nvidia, Twitter, Intel, Dropbox, Ebay and Uber to name a few.

Evolved neural networks are not without disadvantages. Their black-box nature means that we don't get an understanding of why it makes certain decisions or takes particular actions, and it can be difficult to modify behaviours. Furthermore, there is the problem of potential overfitting, as previously described with the MarI/O(SethBling, 2015) example where the agent has perfected that particular Mario map, but if it were to be placed in a new environment, the same strategy would not suffice. It would be the same as introducing a robot of never before seen size or shape in our RoboCupKaufmann et al. (2004) example, without allowing retraining with images of the new robot machine.

3.4 Why Co-Evolve Topologies and Weights?

Much like when we compare evolving neural networks to standard reinforcement learning techniques, evolving network topologies often performs in a significantly superior manner to the alternative - in this case fixed topology neural network evolution(Stanley and Miikkulainen, 2002a).

There is always a chance with evolving topologies of making the search overly complex, but in contrast to that possibility there is also the potential to find the optimal amount of hidden nodes for any given problem by itself, which would save some time(Stanley and Miikkulainen, 2002a). It is also possible for NEAT to reduce the complexity of a network's structure when evolving the topology.

Simple networks are where NEAT begins, before expanding the search space when necessary(Stanley and Miikkulainen, 2004). This flexibility is what allows it to find far more complex controllers than evolution networks with a non-negotiable topology.

3.5 Summary

Evolving artificial neural networks, including structural network evolution, allows us to create solutions that perhaps would not be feasible to build with step-by-step hard-coding approaches. For a project such as this where we cannot write a fool-proof strategy to deal with any given situation within the game, it is ideal to have tools such as NEAT.

Despite the disadvantages of evolved neural networks, the negatives are outweighed by positives and the undisputed fact is that they are able to achieve things that developers alone cannot.

As Switch is largely down to luck like most (if not all) card games, there is never going to be a perfect solution. However, evolving neural networks is a lot more capable of finding a consistently positive solution than a programmer implementing a strategy in advance could ever be.

4 Methodology

4.1 Language and Libraries

The programming language used to build this project is C++. It is used in collaboration with the NeuroEvolution of Augmenting Topologies (NEAT) library.

4.2 Neural Network

4.3 Evolutionary Algorithm

4.4 Fitness Evaluation

5 Results

A compilation of results from all of the test phases.

5.1 Result 1

.

5.2 Result 2

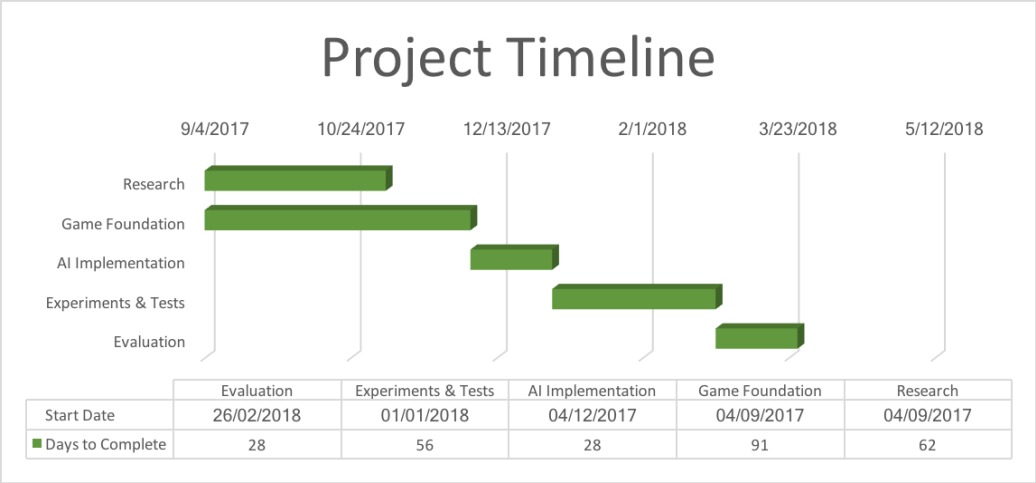
.

5.3 Result 3

.

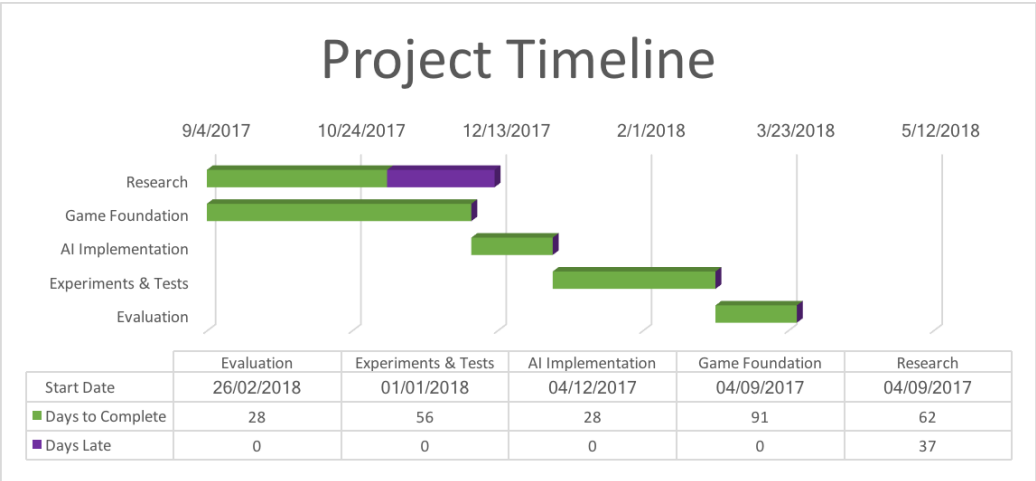
6 Critical Evaluation

- Gantt chart from beginning of project
- Gantt chart from week 9 milestone (ie now, when lit review is done)
- Gantt chart at Christmas/New Year
- Completed project gantt chart



	Evaluation	Experiments & Tests	AI Implementation	Game Foundation	Research
Start Date	26/02/2018	01/01/2018	04/12/2017	04/09/2017	04/09/2017
Days to Complete	28	56	28	91	62

Figure 11: Original Project Timeline Gantt Chart



	Evaluation	Experiments & Tests	AI Implementation	Game Foundation	Research
Start Date	26/02/2018	01/01/2018	04/12/2017	04/09/2017	04/09/2017
Days to Complete	28	56	28	91	62
Days Late	0	0	0	0	37

Figure 12: Updated Project Timeline Gantt Chart

7 Conclusion

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devlin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, and Google Brain. TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 2016.

Martín Abadim, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Judlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

Murray Campbell, Joseph A. Hoane, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134:57–83, 2002.

A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015.

David Fuhriman. How Many Possible Move Combinations are there in Chess. URL `http://www.bernmedical.com/blog/how-many-possible-move-combinations-are-there-in-chess`.

Google. Tensorflow. URL `https://www.tensorflow.org`.

Scott R. Granter, Andrew H. Beck, and David J. Papke Jr. AlphaGo, Deep Learning, and the Future of the Human Microscopist.

Simon Haykin. *Neural Networks - A Comprehensive Foundation*. Prentice Hall, 1999.

IBM. Deep Blue, a. URL `www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/`.

IBM. Deep Blue - Frequently Asked Questions, b. URL `https://www.research.ibm.com/deepblue/meet/html/d.3.3a.html`.

Ulrich Kaufmann, Gerd Mayer, Gerhard Kraetzschmar, and Günther Palm. Visual Robot Detection in RoboCup using Neural Networks. *RoboCup 2004: Robot Soccer World Cup VIII*, 2004.

Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. RoboCup - A Challenge Problem for AI. *AI Magazine*, 18(1).

Andrey Kurenkov. A Brief History of Game AI. URL `http://www.andreykurenkov.com/writing/a-brief-history-of-game-ai/`.

Simon M. Lucas. Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games*, 2005.

Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

Darren Orf. This Robot That Runs Entirely off a Neural Network is Creepy as Hell. URL `https://gizmodo.com/this-robot-that-runs-entirely-off-a-neural-network-is-c-1784649778`.

RoboCup. RoboCup. URL `http://www.robocup.org/objective`.

SethBling. MarI/O, 2015. URL `https://www.youtube.com/watch?v=qv6UVOQ0F44`, source code: `https://pastebin.com/ZZmSNaHX`.

Mat Smith. Japan's Latest Humanoid Robot Makes its Own Moves. URL `https://www.engadget.com/2016/07/30/japan-humanoid-alter-robot/`.

Kenneth O. Stanley and Risto Miikkulainen. Efficient Evolution of Neural Network Topologies. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002a.

Kenneth O. Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 2002b.

Kenneth O. Stanley and Risto Miikkulainen. Evolving a Roving Eye for Go. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004.

Sebastian Thrun. Learning to Play the Game of Chess. *MIT Press*, 1995.

Alan Turing. The Alan Turing Internet Scrapbook. URL `http://www.turing.org.uk/scrapbook/test.html`.

Sun-Chong Wang. Artificial Neural Network. *Interdisciplinary Computing in Java Programming. The Springer International Series in Engineering and Computer Science, vol 743*, 2003.

Appendices

A Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project:

Evolution of Neural Network Controllers for Gameplay Behaviours

Overview of Project Content and Milestones

The idea is to implement a card game with four players. One of the players is the human, another is an AI agent that has no idea how to play the game, and the other two are hard-coded to know the rules and how to play. The intention is for said card game to be Switch, however this is subject to change if the rules are found to be too difficult for the scale of the project – in which case a simpler game will be substituted in.

The agent then learns how to play by trying to make moves based on neural networks. Initially this will be totally random but after the first generation of the algorithm cycle, it will be based on the chromosomes with the highest fitness, which should then begin to provide better results. These moves can be blocked if they are not legal. There'll be a scoring system for the agent that will be negatively affected by illegal moves and it will then use this to learn how to do better the next time it plays. The scoring system will also see the agent penalised for losing or not winning. This will be what our fitness is based on.

It is worth noting that how successful you are in a game of Switch depends entirely on the hand you're dealt, and how your opponents play the hands they are dealt. A lot of the game is about luck, and so negatively affecting the agent's score should take this into account and deploy some leniency.

The project will make use of the NeuroEvolution of Augmenting Topologies (NEAT) library and will most likely be coded in C++. It will use neural network controllers, co-evolving weights and topologies.

The Main Deliverable(s):

- A playable card game that incorporates an Artificial Intelligence agent that must learn how to play the game from scratch based on a score system that penalises the agent for illegal or costly decisions.
- Experimental research into improving the performance (in terms of score) or speeding up the learning process of the agent.
- A report into what positively or negatively affects the agent, and what causes the effects that it has including experiment results using charts and figures. Changes will be made by varying parameter settings of the evolutionary algorithm in a systematic way.

The Target Audience for the Deliverable(s):

Whilst the final product will be a playable game, it will really be aimed more at being experimental research into Artificial Intelligence techniques and, more specifically, evolving neural network controllers for playing games. Thus, the audience most likely to be interested in the project are those who also want to look into artificial intelligence agents.

The Work to be Undertaken:

- Design and build a game of Switch without the AI agent
- Thoroughly test the bare-bones game to ensure it works perfectly without bugs
- Research neural networks and evolutionary algorithms
- Implement the AI agent
- Experiment with a few different techniques and test how they perform in terms of improving or decreasing the agent's intelligence/performance in game.

Additional Information / Knowledge Required:
Neural networks and evolutionary algorithms

Information Sources that Provide a Context for the Project:

- Lubberts, & Miikkulainen (2001). Co-Evolving a Go-Playing Neural Network.
- Stanley, Bryant, & Miikkulainen (2005). Evolving Neural Network Agents in the NERO Video Game. IEEE Press.
- Thrun (1995). Learning to Play the Game of Chess. MIT Press.

The Importance of the Project:
Exploring possibilities and limits of AI in games, particularly evolved controllers which do not have to be hard-coded.

The Key Challenge(s) to be Overcome:

- Complete lack of knowledge and experience with Artificial Intelligence techniques

B Second Formal Review Output

Insert a copy of the project review form you were given at the end of the review by the second marker

C Diary Sheets (or other project management evidence)

Insert diary sheets here together with any project management plan you have

D Appendix 4 and following

insert content here and for each of the other appendices, the title may be just on a page by itself, the pages of the appendices are not numbered, unless an included document such as a user manual or design document is itself pager numbered.