Comparison of Artificial Intelligence techniques for training a Neural Network to play a game

Ovidiu-Andrei Radulescu
- 40283288

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

School of Computing

November 16, 2019

Authorship Declaration

I, Ovidiu-Andrei Radulescu, 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:

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, 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

test

Contents

List of Tables

List of Figures

1 Introduction

This report is an investigation into different Artificial Intelligence (AI) methods used in the training of Neural Networks (NN or NNs). Artificial Intelligence (AI) is intelligence demonstrated by machines, as opposed to natural intelligence shown by humans and other animals. As computers started progressing and becoming better and better at numerical calculations, their use-cases became more complex with the employment of algorithms, which are sequences of actions that resemble the way a human mind would approach a problem, thus making a computer exhibit certain capabilities of the human mind. (Wang, 2007) A Neural Network (NN), sometimes called an Artificial NN, is an interconnected system formed from simple processing elements, inspired by (but not identical to) a biological brain. Such a system learns how to perform a task by adapting or learning from a set of training patterns. This processing ability of the network is "stored in the interunit connection strengths,or weights." (Gurney, 1997)

1.1 Aims and Objectives

The aim of this project is to understand how to best create multiple Neural Networks that will ultimately be able to solve tasks, while showing the differences between them, both in implementation as well as results (efficiency, time), to ensure that future users will be able to use this as an easy start-up point for choosing an appropriate solution for their own problems.

In order to achieve these aims, technologies, both old and new, will be analysed and used, in order to provide the user with plenty of options to consider. The below steps will be taken to ensure an effective and thorough end product is delivered.

1. **A literature review** will be conducted where it will study Neural Networks and Artificial Intelligence training methods by looking into the various applications of Neural Networks and provide the technological context in the modern day through its history and development, both past, present and future. The study will break down the different kinds of training methods to find the advantages (and disadvantages)

of each, depending on their use case.

2. **An implementation methodology** will then be conducted from the literature review, along with a decision as to which environments are best suited for use within this project.

3. The Neural Networks will be **developed, implemented** and trained according to the methodology plan.

4. Consider how this project could be expanded upon in **future developments**.

1.2 Research Questions

TBD

1.3 Project Scope, Constraints

TBD

2 Literature Review

This review will discuss existing applications of Artificial Neural Networks (as there are both Biological and Artificial, for simplicity, as the topic will mostly be about Artificial Neural Networks, they will be referred to as Neural Networks or NNs).

Even though neural networks are not a new concept, they weren't very popular in the 20th century, as there was not enough computing power to run them. As computers became better and better, also new methods for training neural networks were discovered. In 2006, the discovery of techniques for **deep** neural networks produced an important change that made deep neural networks and deep learning viable options that returned excellent results and performance in areas such as speech recognition, computer vision and natural language processing (?, ?). Computing isn't the only area where NNs, as they are used in numerous fields such as engineering (Aydinalp-Koksal & Ugursal, 2008), or even medical (Baxt, 1991). The review will analyse algorithms and environments where these can be used, in order to find the best methods to achieve the tasks set.

2.1 Neural Networks Definition

To begin, we must understand and define what a neural network is.

As defined by Gershenson, an artificial neuron is a model that is inspired by natural neurons [Figure 1]. Natural neurons receive signals through synapses, when the signal received is strong enough (beyond a certain threshold), the neuron activates and emits a signal, which can in turn be sent to another synapse, and activate other neurons (Gershenson, 2003).

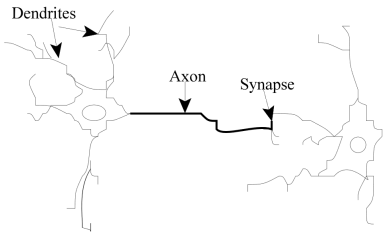

Figure 1: Biological Neuron (Gershenson, 2003)

The first emergence of interest in neural networks (also known as 'connectionist models' or 'parallel distributed processing') (Ben Kröse & Patrick van der Smagt, 1993) came after the introduction of simplified neurons (called perceptrons) in 1943 (McCulloch & Pitts, 1943). Their research was presented as a concept, components that were based off models of actual biological neurons. A perceptron works by taking in several binary inputs (shown as x1,x2 etc) in order to produce a single binary output.[Figure 2]

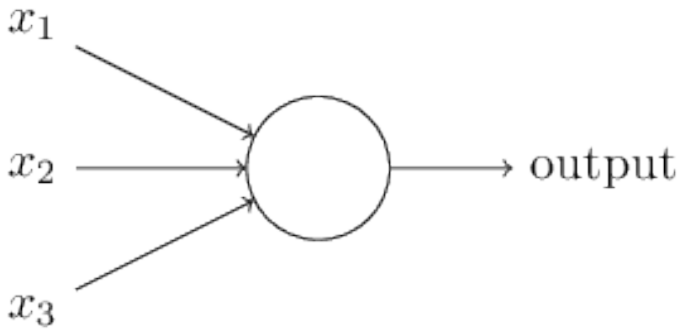

Figure 2: Perceptron Model (?, ?)

This model was improved and developed further in the 1960s by Frank Rosenblatt. He introduced weights (real numbers that express the importance of the input in relation to the output) (Rosenblatt, 1958). Because the neuron's output can only be 0 (neuron isn't activated) or 1 (neuron is activated), this can be determined through the sum of the weights being greater or less than a set threshold value (also a real number that is given as a parameter). This is called an activation function.

$$output = \begin{cases} 0 & if \quad \sum_j w_j x_j \leq threshold \\ 1 & if \quad \sum_j w_j x_j > threshold \end{cases} \tag{1}$$

At a glance, one neuron on its own isn't very impressive and it can only do so much through firing or not depending on the received input. One neuron by itself isn't learning anything either, as the output will never vary if you give the same input over and over (threshold is still the same). There are two immediate ideas that surface into how to make a neuron useful and

interesting: finding a way to make it learn, and grouping multiple neurons together that will form a (neural) network. (Marsland, 2015)

2.2 Classifying Neural Networks

Having a model of a NN is only half of what a NN is and does. The big selling point of a NN is it's learning capabilities, also referred to as machine learning, and its similarities to how humans learn. The definition given by Mitchell (Mitchell & Thrun, 1993) is that given a task, training experience, and a performance measure, a computer is said to learn if its performance of the task improves with experience.(Thrun & Pratt, 2012)

These algorithms build a mathematical model based on sample data (also called training data) in order to generate decisions without any additional human programming. Based on the machine learning methods NNs can be split into the following categories:

- Supervised Learning, where an algorithm builds its model by having both the inputs and the desired results (used in speech recognition, handwriting recognition, etc.)
- Unsupervised Learning, where an algorithm builds its model by having only the inputs. Here, there is no correct answer from the start.
- Reinforcement Learning, where an algorithm is built around maximising its reward, based on rules given.

2.3 Layered Neural Networks

As previously noted, a single neuron can only do so much in order to make a decision, and a very simple one at that, but a complex network can make more complex decisions. Neural Networks are split between 'layers', the most simple one having 3 layers: an input layer, a hidden layer, and an output layer. All networks need to have 1 input layer and 1 output layer, but the hidden layer can be a group of layers, which is where all the computation is done.

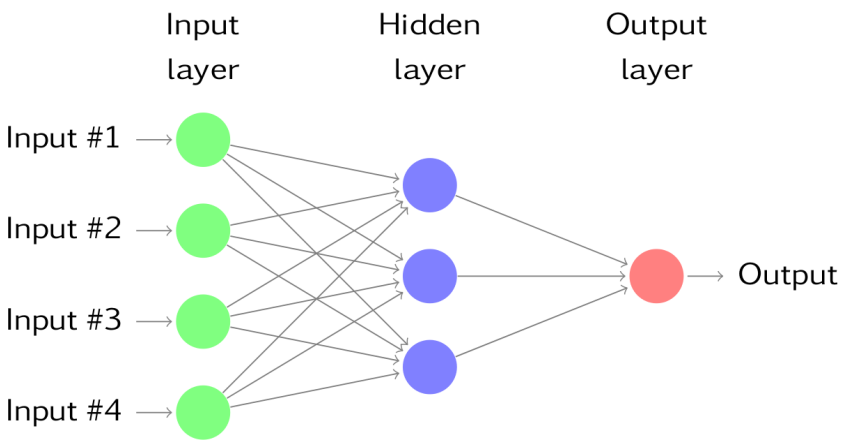

Figure 3: Shallow Neural Network (Rob J Hyndman & Athanasopoulos, 2018)

2.3.1 Shallow Neural Networks

One might think of a neural network as a very complex system of layers, but in truth, the simplest neural network, also called a shallow NN, has an input and an output layer, as well as one (shown in [Figure 3]) or sometimes two hidden layers.

As shown in the example NN, this neural network takes in 4 inputs and returns one output. The arrows that go from the input layer to the hidden layer shows that all the inputs are given to all the neurons of the network, and then all neurons return an output.

2.3.2 Multi-Layer Neural Networks

Multi-Layer NNs are more complex versions of the basic (shallow) model, by using more layers in the hidden layer. As all the computation is done by the hidden layer, multiple layers allow a neural network to do more abstract, complex computations.

2.4 Activation Functions

After talking about the structure of a NN, the next step is learning about the decision-making aspect. Previously, we mentioned that a neuron activates or not depending on the calculation of the sum of the input weights.

Determining this is the purpose of the activation function.

2.4.1 Step Function

The step function is the simple "if - then - else" version of activating a neuron, as seen in Equation 1. As such, this can be the function to use for a binary application, something that will return "yes/no" to a query. But then, if you want something more advanced and the output required has more than two options, the step function is no longer giving you a definite answer. For example, a NN that can tell you in which direction to go depending on the 4 cardinal points, if two of the neurons will fire North and West, it is unclear which one to pick: N? W? Or perhaps NW? For this scenario (which is a more real-life scenario anyway), the NN needs to give intermediate (analog) activation values rather than binary ones. (Sharma, 2017)

2.4.2 Linear Function

A linear function (eg. f(x) = ax +b) is a function whose graph is a straight line. This way a NN will give a range of activations, where you can pick the biggest value (or values). But as a linear function is a constant function, if multiple layers in the same NN use linear functions, as the data moves through the functions, combining linear functions, is the equivalent of just a single linear function, meaning they can all be condensed into one single layer.

2.4.3 Sigmoid Function

A sigmoid function(e.g. f(x) $=\frac{1}{1+e^x}$) is a function who will produce a result between 0 and 1. The sigmoid function is one of the most used functions, and is most useful for training data between 0 and 1 (as bigger numbers of x will take the results to the extremes of the graph curve)[Figure 4].

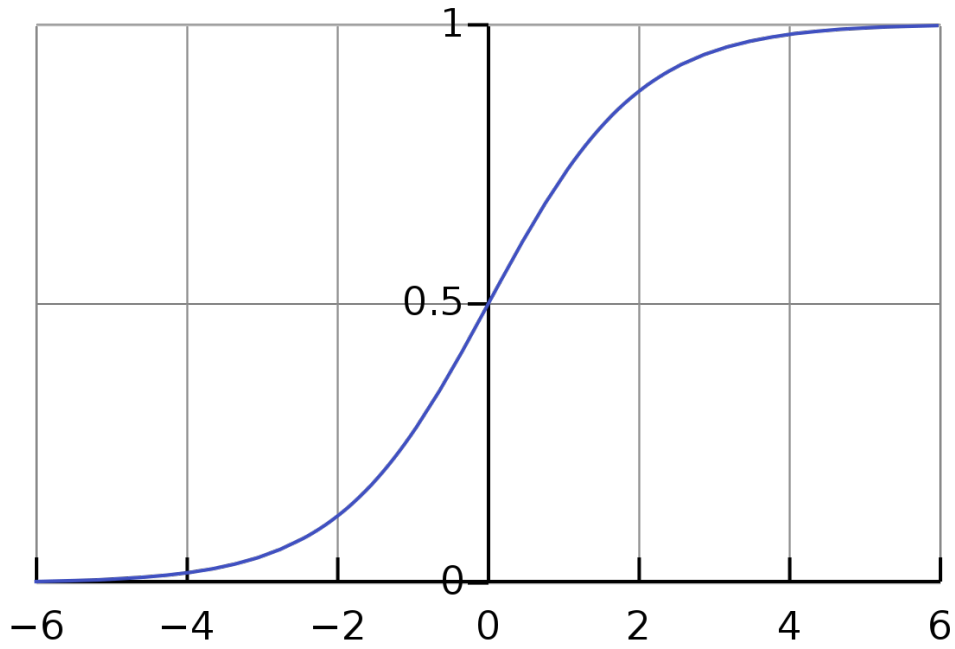

Figure 4: Sigmoid Function Graph (Nikhil Ketkar, 2017)

3 Supervised Learning

3.1 Backpropagation

Backpropagation (backward propagation of errors) is a widely used supervised learning algorithm, that through the use of an error function, the algorithm calculates the gradient of the function and determines if the NN has made a mistake when a prediction was made.

This algorithm is based on an evolutionary approach, as an untrained NN doesn't know anything about what it needs to do, but through exposure (inputs) and a lot of trial and error (the training), the NN learns from the data provided, this being reflected in the weights (the way the NN alters the data through its layers), the more the NN is trained, those weights get calibrated and can make more accurate predictions.

Backpropagation works by propagating the input data **forward** through the NN layers, towards making a decision, and then **backpropagates** the error information of the prediction, in reverse through the layers, in order

to adjust the weights.(Chris Nicholson, 2019) The steps the algorithm takes are:

- NN makes a prediction about the data
- This is measured through a function
- The error is backpropagated to adjust the weights

4 Unsupervised Learning

4.1 Genetic Algorithms

Genetic Algorithms are based on the real-life process of natural evolution (Agoston E. Eiben, 2015). Eiben notes that this approach is not surprising at all, as the power of evolution is present everywhere in the world, through its diversity and adaptability to any environment and specific issues. This inspiration is closely tied to the fundamental principle of trial-and-error, as it takes many generations to adapt a species to its full potential.

In a conventional approach to programming, the programmer tells the computer what it needs to do, in broken down simple tasks that can be easily performed by a computer, just like in mathematics or physics, which programming borrows a lot from. This conventional model can be considered an exact science, or a deterministic model, which means that no matter how many times you run such a program, the result will always be the same (6 multiplied by 7 will always be 42).

But in a genetic algorithm, a programmer does not tell the computer how to solve a problem. In place of this, the program figures out its solution through adapting over multiple iterations to the environment's needs. This is called a stochastic approach, meaning the results of running such a program multiple times will not always be the same (you might take different busses when going to work).

A genetic algorithm's purpose is to optimise a set of parameters. This information is encoded in a bit string of fixed length, called the parameter string, or individual (Goldberg & Holland, 1988). Each individual is a different possible solution to the problem, the solution's quality being measured

through its fitness value (given by a fitness function specific to the problem). The evolution algorithm is a cycle of selection, crossover and mutation. By having an initial population, also called the first generation (usually randomly generated solutions), we want to select the best (fittest) individuals to be part of the parent pool for creating new offspring.

The crossover is the phase where offspring are created, this can be done through various means (list), this is the most significant phase of the algorithm as it dictates the future of the solutions.

Mutation is the step where some of an offspring's genes are randomly modified, as to incentivise gene diversity. Mutation happens through a user defined mutation probability and can be done through a number of ways (inverting a bit in a binary gene, using uniform or non-uniform random values for integer or float genes, etc.).

Both the mutation probability and mutation size are usually set low, as using too much randomness defeats the purpose of the algorithm, turning it into a simple random function.

By doing this, evolutionary algorithms consider multiple points in the search space concurrently, reducing the risk of converging to local optima (Branke, 1995). Even though there is a significant degree of probability applied to every step, by favouring the fitter individuals, the most relevant areas of the search space are explored.

4.2 NEAT

Neuroevolution of Augmenting Topologies (NEAT) is a genetic algorithm used in the generation of evolving NN (Stanley & Miikkulainen, 2002). The main feature that NEAT brings is that it doesn't evolve only the weights of the NN, but also modifying the nodes and their connections, thus changing the very structure of the NN. In a normal scenario, the structure of the NN is selected by the user based on previous knowledge, but a predefined structure that is made to accommodate many problems isn't the most optimal structure for a specific problem. Through NEAT, this would not be the case, as the structure would evolve to fit in more precisely with the problem given. The three main components to how NEAT operates are:

- Crossover using history markers. Stanley explains that by randomly crossing over the genomes of two NN will result in networks that are mutated wrong and non functional (see figure). In biology, this idea is taken care of through homology, which is the "alignment of chromosomes based on matching genes for a specific trait" (Heidenreich, 2019). By doing this, there is less chance of error than by using random crossover. By marking evolutions with a historical marker, each gene can be aligned with a similar one in order to have a higher chance of a working crossover [Figure 5].

- Speciation, is a solution to the idea that most new evolutions are not good ones, because by adding new connections or nodes before optimising the weights will lead to lower performing individuals. By using speciation, the population is split into several 'species' based on their similarities (in topology or connections).

- Minimal Structure. The main goal of Stanley & Miikkulainen in their paper about NEAT was to create a framework that could evolve minimal networks. By starting with a minimal amount of nodes and connection, the evolving complexity of the network will only happen if it is found to be useful.

4.3 CPPN

Compositional Pattern Producing Networks (CPPN), first introduced by K.O. Stanley (Stanley, 2007), the same author behind NEAT, CPPNs are an extension of NEAT. At the base level a CPPN works in the same way as NEAT, one of the major differences of CPPN is its activation function. While usually the same activation function is used in every node, CPPN can allow each node to have their unique activation function. By having different activation functions in a NN, this allows the NN to create an output that is "patterned, symmetrical and unique" (Wolfe, 2018). As such, because of the ability to create complex geometries, a CPPN is used in the generation of 2D and 3D shapes. The inputs to a CPPN, usually being x, y (and z for 3D) coordinates of a pixel, the CPPN then outputting a value (rgb for colour or

binary for b&w) for the pixel's colour [Figure 6].

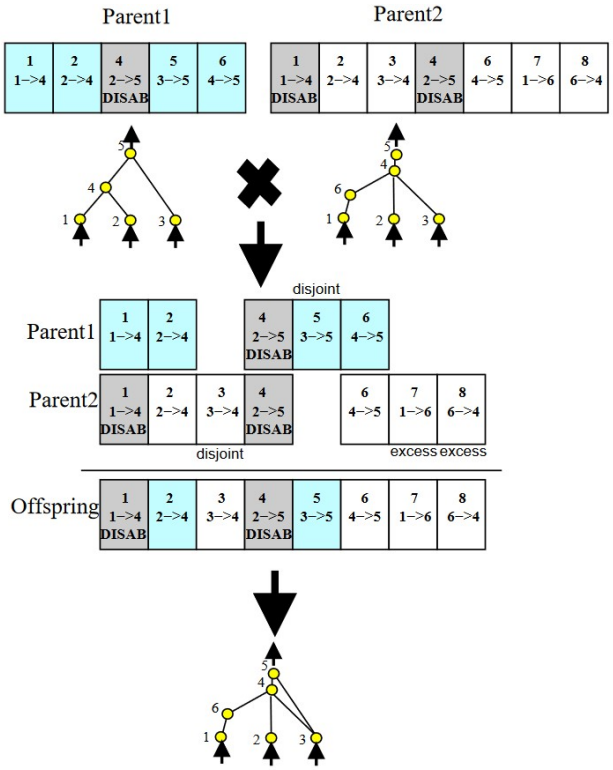

Figure 5: NEAT crossover using markers (Stanley & Miikkulainen, 2002)

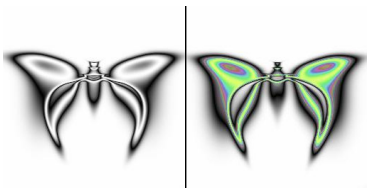

Figure 6: CPPN generated b&w and colour image (*Picbreeder*, n.d.)

4.4 Reinforcement Learning/ Deep-Q Learning

As defined by Sutton & Barto, "reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal" (Richard S. Sutton & Andrew G. Barto, 1998). For this, the learner (the agent program) isn't taught which actions to choose, but rather find out which actions maximises its reward through trial and error. In some

cases, by choosing actions, these choices may affect future rewards past the immediate reward. These basic rules of reinforcement learning, are the basis for (non deep) Q-Learning.

In Q-Learning, through the uses of functions, an agent can map every action and its reward, and choose accordingly. But in a real-life scenario, mapping every action isn't a feasible solution, as there can be millions of actions that would need to be mapped, and not enough time or memory to do so. By using a neural network (creating a Deep Q Network as a result) in order to approximate the result of a Q-Learning function.

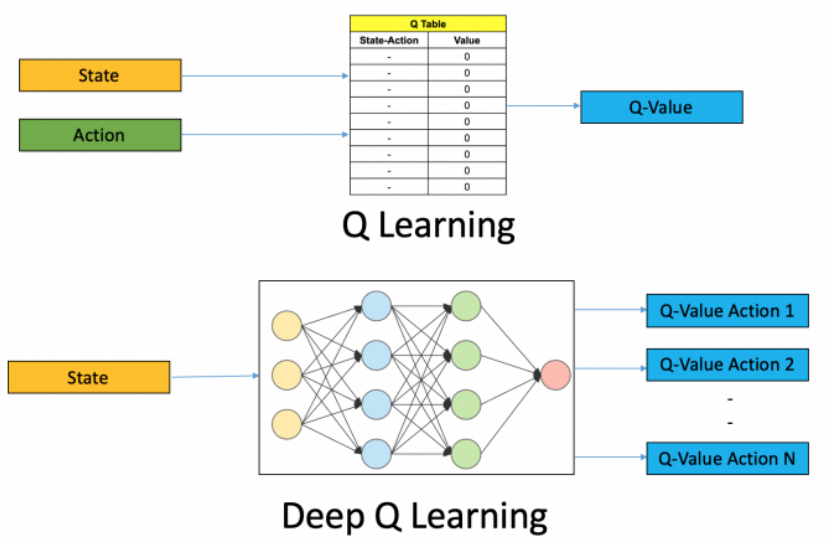

Figure 7: Q-Learning and Deep Q-Learning (*Picbreeder*, n.d.)

References

Agoston E. Eiben. (2015). *Introduction to evolutionary computing* (Second edition. ed.). Springer.

Aydinalp-Koksal, M., & Ugursal, V. I. (2008). Comparison of neural network, conditional demand analysis, and engineering approaches for modeling end-use energy consumption in the residential sector. , *85*(4), 271 – 296. Retrieved from http://www.sciencedirect.com/science/article/pii/S030626190600136X doi: https://doi.org/10.1016/j.apenergy.2006.09.012

Baxt, W. G. (1991). Use of an artificial neural network for the diagnosis of myocardial infarction. , *115*(11), 843–848.

Ben Kröse, & Patrick van der Smagt. (1993). An introduction to neural networks.

Chris Nicholson. (2019). *A beginner's guide to backpropagation in neural networks.* Retrieved from http://skymind.ai/wiki/backpropagation

Gershenson, C. (2003). *Artificial neural networks for beginners.* Retrieved from https://login.ezproxy.napier.ac.uk/login?qurl=https%3A%2F%2Fsearch.proquest.com%2Fdocview%2F2091242675%3Faccountid%3D16607

Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. , *3*(2), 95–99.

Gurney, K. (1997). *An introduction to neural networks.* Taylor & Francis, Inc.

Heidenreich, H. (2019). *NEAT: An awesome approach to NeuroEvolution.* Retrieved from http://hunterheidenreich.com/blog/neuroevolution-of-augmenting-topologies/

Marsland, S. (2015). *Machine learning : an algorithmic perspective* (2nd ed. ed.). Boca Raton : CRC Press.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. , *5*(4), 115–133. Retrieved from https://doi.org/10.1007/BF02478259 doi: 10.1007/BF02478259

Mitchell, T. M., & Thrun, S. B. (1993). Explanation-based neural network learning for robot control. In *Advances in neural information processing systems* (pp. 287–294).

Nikhil Ketkar. (2017). *Deep learning with python: A hands-on introduction.* Apress.

Picbreeder. (n.d.). Retrieved from `http://picbreeder.com/`

Richard S. Sutton, & Andrew G. Barto. (1998). *Reinforcement learning: an introduction.* MIT Press, IEEE Xplore.

Rob J Hyndman, & Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2nd edition ed.). Retrieved from `https://Otexts.com/fpp2/`

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. , *65*(6), 386.

Sharma, A. (2017). *Understanding activation functions in neural networks.* Retrieved from `https://link.medium.com/1H2Vt6LQoZ`

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. , *8*(2), 131–162. Retrieved from `https://doi.org/10.1007/s10710-007-9028-8` doi: 10.1007/s10710-007-9028-8

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. , *10*(2), 99–127.

Thrun, S., & Pratt, L. (2012). *Learning to learn.* Springer Science & Business Media. (Google-Books-ID: X_jpBwAAQBAJ)

Wang, P. (2007). Three fundamental misconceptions of artificial intelligence. , *19*(3), 249–268. Retrieved from `https://doi.org/10.1080/09528130601143109` doi: 10.1080/09528130601143109

Wolfe, C. (2018). *Understanding compositional pattern producing networks.* Retrieved from `https://towardsdatascience.com/understanding-compositional-pattern-producing-networks-810f6bef1b88`

Appendices

A Project Overview

A.A Example sub appendices

...

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.