Evolving a Learning Agent using Neuroevolution in the FightingICE Game Framework

Deliverable 1: Final Year Dissertation
Heriot-Watt University

Robert John Dunn
H00163867
BSc Honours in Computer Science

Supervisor:
Dr Patricia A. Vargas
School of Mathematical & Computer Sciences
Heriot-Watt University

Co-Supervisor:
Dr Fabrício Olivetti de França
Center of Mathematics, Computing and Cognition
Federal University of ABC

Second Reader:
Dr Mohamed Abdelshafy
School of Mathematical & Computer Sciences
Heriot-Watt University

Declaration

I, Robert Dunn confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.
Signed:
Date:

Abstract

Neuroevolution is a popular technique for machine learning in which the topology and/or weights of an artificial neural network are adjusted by an evolutionary algorithm. The technique takes inspiration from the evolution of the biological nervous system and is a popular approach for reinforcement learning problems. One way to demonstrate the effectiveness of neuroevolution is through artificial intelligence in games. This project aims to implement a learning agent in the FightingICE platform, a two-dimensional Java fighting game organised and maintained by Ritsumeikan University, Kyoto. The agent is designed to evolve through neuroevolution to improve its performance in the game, eventually becoming competitive versus a human opponent. By implementing a neuroevolution method in a simplistic environment, we hope to evaluate the effectiveness of neuroevolution as a method of machine learning and explore the potential of our agent's performance.

Contents

1 Introduction

1.1 Motivation

The ability to learn is a fundamental attribute of intelligent behaviour [1], in which one acquires knowledge or skills through study, experience, or being taught. The field of machine-learning aims to imitate this learning attribute and apply it to machines in order to improve their performance at tasks without being explicity programmed.

Various methods of machine-learning exist which allow the agent (machine) to adapt itself in order to process new data. One of these methods is neuroevolution, which involves the weights and/or topology of an artificial neural network being adjusted by an evolutionary algorithm. The neuroevolution method is loosely based on the way the biological nervous system operates: with neurons communicating through axons, represented by nodes of the neural network with weighted connections. Neuroevolution has received huge popularity due to the fact that many artificial intelligence problems can be cast as optimisation problems, and since the method is ground in biological metaphor and evolutionary theory. [8]

One way to test the effectiveness of machine-learning methods is through artificial intelligence in computer games. Since the state of a game is usually easily determined (e.g. the hit-points of both player's characters), the effectiveness of the learning method when controlling the agent's actions can be easily evaluated. The FightingICE platform [7], is a two-dimensional fighting game written in Java by a group in Ritsumeikan University, Kyoto. The platform allows programming of artificially intelligent agents within the game and sends the agent delayed information about the current state of the game.

1.2 Objectives

This project aims to implement a neuroevolution algorithm acting on a learning agent in the FightingICE game platform. The inputs to the algorithm will be based on the agent's environment, such as the enemy's current position and the enemy's energy level, the outputs will be character actions e.g. move left, move right, attack. In order to increase the rate of the agent's evolution, an environment using incremental evolution will be used, exposing the agent to progressively more difficult challenges.

Once implemented, we will evaluate the algorithms performance versus a human opponent. We will use human volunteers of varying skill levels in the game, and record their performance versus the agent at different stages of its evolution. We will also compare the effectiveness of the algorithm to that of previous prototypes to evaluate any improvements made.

From this project, we hope to assess the effectiveness of neuroevolution as a method of machine-learning and gain the programming experience from implementing it. We also hope to find the potential to which an agent employing a neuroevolution method to learn can perform. Through comparisons of the algorithm to previous prototypes, we can also assess the progress of our algorithm over time and how improvements affect the agent's performance.

1.3 Professional, Legal, Ethical and Social Issues

1.3.1 Professional Issues

2 Literature Review

In this section of the dissertation, we will present any fundamental concepts used in the project. We will also discuss and evaluate relevant work in the field.

2.1 Machine Learning

Learning is a fundamental human function in which we modify our behaviour tendency according to experiences or teaching [1], to become better when a similar situation occurs. In the study of machine-learning, algorithms, computer applications, and systems, utilise learning to improve their performance at certain tasks. There are two main entities in the machine-learning model, the teacher and the learner. The teacher, while not always available, contains the knowledge to perform a given task while the learner has to learn the knowledge the teacher holds. [15] There are three types of machine learning:

- *Supervised Learning*
  A teacher provides the learner with a set of input and desired output pairs. The learner can then use these examples to improve its performance at the task.

- *Unsupervised/Self-organised Learning*
  There is no teacher, so the learner learns based only on the observed relationships. A common application is finding patterns or grouping within data, using methods such as cluster analysis.

- *Reinforcement Learning*
  The agent uses goal-directed learning where a notion of reward is introduced and the agent attempts to maximise this reward. There is no teacher for the agent and no explicit model of the environment. Computational approach to learning from interaction. [3]

Another way to categorise machine-learning methods is by the output of the method. The output of the method is usually a good indication of the application of the method, for example, two classes of outputs 'spam' and 'not spam' in spam filtering.

- *Classification*
  Inputs are assigned to two or more classes (groups) of output. Typically used with a supervised learning method. One application is spam filtering.

- *Regression*
  Supervised learning problem similar to classification, however the outputs are continuous rather than discrete.

- *Clustering*
  A set of inputs to be divided into groups. Unlike classification, groups are not defined beforehand. Typically an unsupervised learning task.

- *Density Estimation*
  Finds the distribution of inputs in some space.

- *Dimensionality Reduction*
  Inputs mapped into a lower-dimensional space.

Machine-learning is being applied extensively in modern day life, though not everyone is aware of it. The personalisation of search engine results and social networking feeds are one application, providing user's with appropriate content. [4] Other examples of machine-learning application include optical character recognition (OCR) and computer vision. [5]

2.2 Neuroevolution

2.2.1 Fundamentals

Neuroevolution is a biologically-inspired method of machine learning in which an artificial neural network is evolved using an evolutionary algorithm. The biological inspiration for the method comes from the nervous system in which neurons communicate through axons, translated to machine-learning by the nodes of a neural network communicating through weighted connections.

The method has proved popular, especially in the fields of artificial life, evolutionary robotics and computer games. One reason for its popularity is the fact that neuroevolution is a form of reinforcement learning, which can be applied more generally than its counterpart, supervised learning. The popularity also stems from the fact that the method is ground in biological metaphor and evolutionary theory. The method consistently performs well in many areas of application and can handle large action/state spaces. [8]

The applications of neuroevolution are numerous and diverse. One application is the implementation of artificially intelligent agents in computer games (see section 2.2.5). Other applications include dynamic resource allocation, optimising manufacturing processes and even creating musical melodies. [6]

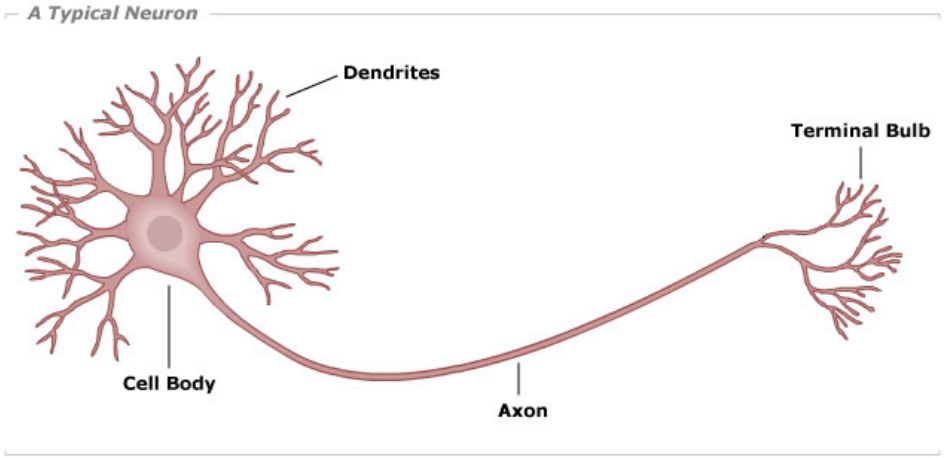

Figure 1: Simple diagram of a biological neuron.

2.2.2 Artificial Neural Networks

Artificial neural networks model the way the brain solves problems with collections of neurons communicating through axons, represented with nodes of the network with weighted connections.

The most basic version of an ANN (artificial neural network) is a perceptron. A perceptron is a simple machine which takes a variable number of input nodes which connect to a single output node. [16] The concept of the perceptron was extended by means of the the multi-layer feed forward network, which incorporates extra 'hidden' layers of nodes between the input and output. The more hidden layers there are, and the more nodes in these layers, the more complex behaviour a network can experience. [17]

Neural networks can be employed in a variety of ways, for both supervised and unsupervised/reinforcement learning problems. Networks can have an associative (content-addressable) memory and can also implement parallel processing across individual nodes. [2] Common applications for neural networks include character recognition, image compression and stock market prediction. [18]

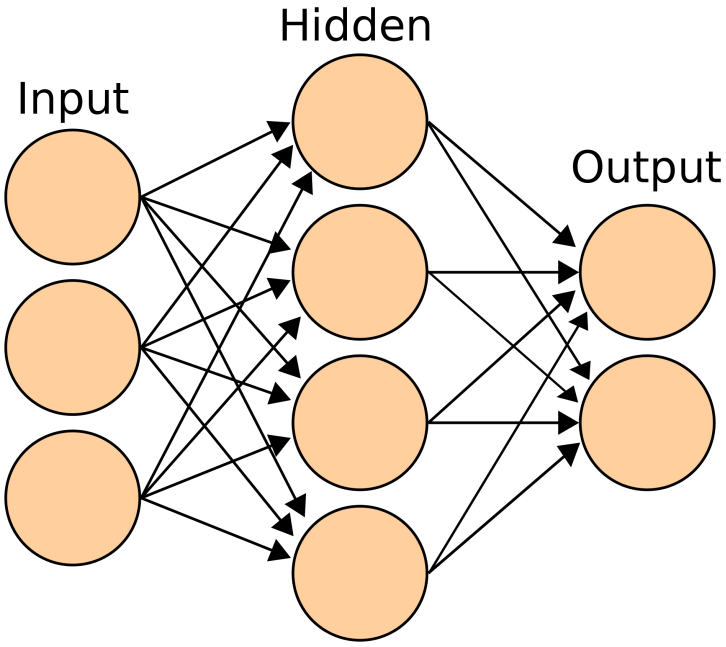

Figure 2: Multi-layer feed forward network with labelled layers.

Activation Function

In order to calculate the output of a certain node in a network given the input or set or inputs, we use an activation function. [19] This output of the node is usually saturated to a value between minus and positive one by the function and then scaled appropriately when output. [15]

The simplest activation function is the step function [2], where if the weighted sum of the inputs to the node falls below a certain threshold the output is zero, else the output is one. This can be thought as the node either sending signals or not sending signals. The step function is used in perceptrons and often shows up in other models. [16] Since there are only two possible outputs of nodes using the step function, it is common to use different, more general functions.

One of these more general, non-linear activation function is the hyperbolic tangent function tanh, shown in figure 3. This allows nodes to have an activation output of a real number between negative one and one. The usage of these more general functions allows us to exhibit more complex behaviour within the network.

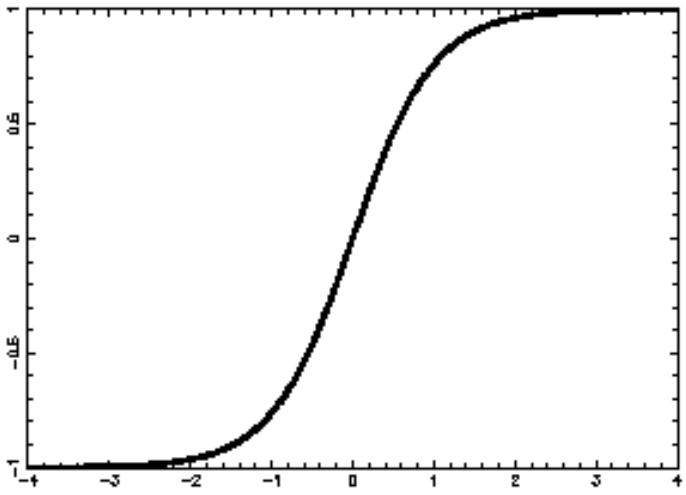

Figure 3: tanh activation function plot.

2.2.3 Evolutionary Algorithms

Evolutionary algorithms are methods of optimisation, where potential solutions to the problem are seen as individuals of some population. Each individual in the population is assigned a fitness value which is calculated according to the performance of the provided solution, and the algorithm works to find the best (fittest) of these solutions. Using techniques inspired by biological evolution such as reproduction, mutation, recombination, and selection, new populations of solutions can be generated and assessed. Using evolutionary algorithms allows fine tuning of the search space through constants such as rate of reproduction and mutation rate. [Elitism?GAs?]

2.2.4 NEAT (NeuroEvolution of Augmenting Topologies)

NeuroEvolution of Augmenting Topologies (NEAT) is a method of neuroevolution developed by Ken Stanley in 2002, which involves both the weights and the topology of the ANN being altered. The method is a genetic algorithm and involves applying the following three techniques:

1. Track genes with history markers to allow crossover among topologies
2. Apply speciation to preserve innovations
3. Developing topologies incrementally from simple initial structures

A notable extension of NEAT which could potentially be appropriate for the project is HyperNEAT. HyperNEAT is a hypercube-based extension of the method developed by the Evolutionary Complexity Research Group at UCF.

2.2.5 Neuroevolution in Games

How neuroevolution can be used to evolve learning agents in games, usually agent is optimising some value e.g. HP. How fitness in games is evaluated, giving some examples. How neuroevolution fares compared to other learning algorithms. Present examples of projects implementing neuroevolution to learn.

2.2.6 Incremental Evolution

When an agent is provided with a complex behaviour, it may have trouble evolving to perform at its potential. Using incremental evolution, the behaviour can be learnt incrementally with tasks gradually increasing in difficulty. This form of evolution proved effective in at least one implementation [11]. The agent's task was a prey capture task: the agent moves through the environment and must catch its prey before the set number of time-steps. With increasingly difficult tasks, the agent was able to rapidly improve its performance, and skip many potential generations of evolution.

2.3 FightingICE

2.3.1 Game Platform

FightingICE is a Java based game platform organised and maintained by Intelligent Computer Entertainment Lab., Ritsumeikan University. The game is based in an arena where two fighters are competing versus each other, attempting to deplete the other's hit-points while preserving their own. The FightingICE platform was designed to allow easy development and evaluation of artificially agents in the game for research or hobby purposes. Once implemented, an agent receives information about the state of the game from the platform periodically, such as the opponent player's location and current energy levels. A delay is added to this game information in order to simulate the delay a human player would experience from reaction time.

2.3.2 Game Agents

There are four characters available in the game: Zen, Garnet, Lud, and Kfm. Each of the characters is capable of moving, performing attacks, and combining these attacks into unique combos. The game starts with each player at 0 hit-points and ? energy. Once a player successfully connects an attack against its opponent, the player's energy is increased and the opponent's hit-points decrease. In order to compete against an opponent, the character must dodge opposing attacks and make effective use of energy to land attacks and reduce the opponent's hit-points.

2.3.3 Related Projects

Discuss and reference relevant projects which have been completed using the FightingICE framework. Discuss what has been achieved in said projects and the potential further research that can be undertaken.

3 Organisation

In this section we present the decision process on how to approach the task including analysis of the task itself, analysis of the requirements, and analysis of the risks. A gantt chart of the project timeline is also included.

3.1 Project Task Analysis

The objective of this project is to evolve an agent in the FightingICE game platform with a neuroevolution method. The agent will be evolved to the point of being competitive versus a human opponent.

In order to implement the neuroevolution method, we will first need an artificial neural network to control the actions of the agent. The neural network will need to take as input various factors from its environments such as the location of the opposing player, the energy of the opposing player and whether the opponent is performing an attack. The outputs of the neural network will be actions for the character, for example move left, jump, attack. Once a neural network controlling the agent has been implemented, the network must be evolved to imitate the learning process. This will involve deciding on a suitable evolutionary algorithm and the encoding of the network's genotype. It will also need to be decided whether only the weights of the network are evolved, or also augmenting the topology (NEAT) and/or altering the activation function.

To improve the rate of evolution for the agent, we will create an incremental evolution environment where the agent is exposed to tasks of gradually increasingly difficulty. To implement this environment, we will limit the capabilities of the training opponent and gradually return them.

To test whether the agent's performance has improved to the point of being competitive versus a human opponent, we will find volunteers of varying skills to face our agent in the game. The results of the player versus our agent at various stages of its evolution can then be evaluated to assess the effectiveness of the neuroevolution algorithm which we implemented.

3.2 Requirement Analysis

3.2.1 Table of Requirements

No.	Requirement	Priority	Predecessors
1	Implement a neural network to control agent's behaviour	High	
1.1	Initialise a network with random weights, test if the sensors can be read and the actions can be output	High	
1.2	Implement a simple rule-based agent for the agent to compete versus	High	1.1
1.3	Test and execute at least one agent from the FightingICE AI competition	Medium	
2	Implement an appropriate evolutionary algorithm to evolve the neural network	High	1
2.1	Test whether a simple evolutionary algorithm evolving the weights of the network improves the agent's performance	High	1
2.2	Adapt and use the HyperNEAT method, test whether it improves on the previous algorithm	Medium	1, 2.1
3	Create an incremental evolution environment for the agent	Medium	1, 2
3.1	Limit the capabilities of the opponent and incrementally return them to test whether speed of evolution is improved	Medium	1, 2
3.2	Evolve against a simple agent, replace opponent with best agent after convergence and continue	Medium	1, 2
4	Evaluate the agent's performance versus a human opponent at various stages of its evolution	High	1, 2

3.2.2 Requirements Textual Descriptions

1 - Implement a neural network to control agent's behaviour

To implement an agent in the FightingICE platform controlled by a simple neural network.

1.1 - Initialise network with random weights, test if sensors can be read and actions can be output

In order to test whether sensors can be read and actions can be output, initialise a simple neural network with random weights to control the agent. Ensure game information is being received and number of outputs nodes match character actions.

1.2 - Implement a simple rule-based agent for the agent to compete versus

Create agent in FightingICE which follows simple rule-based logic. Use as an opponent for the neuroevolution agent while it is evolving.

1.3 - Test and execute at least one agent from the FightingICE AI competition

Find agents from the FightingICE AI Competition and test them to see strategies of other programmer's work. Attempt to read and understand the code.

2 - Implement an appropriate evolutionary algorithm to evolve the neural network

Find and implement an appropriate evolutionary algorithm to evolve the ANN of the agent. Decide on implementation based on performance evaluation.

2.1 - Test whether a simple evolutionary algorithm evolving the weights of the network improves the agent's performance

Implement a simple EA such as hill-climbing or an algorithm using only mutation to evolve the agent's neural network. Test whether the evolution improves the agent's performance and if so, to what extent.

2.2 - Adapt and use the HyperNEAT method, test whether it improves on the previous algorithm

Replace the previous evolutionary algorithm with an implementation of the HyperNEAT or other appropriate neuroevolution method. Test the new implementation versus the rule-based agent to test whether it improves on the previous algorithm.

3 - Create an incremental evolution environment for the agent

Create an incremental evolution environment to speed the process of evolution for the agent.

3.1 - Limit the capabilities of the opponent and incrementally return them to test whether speed of evolution is improved

Initially, completely limit the capabilities of the agent's opponent and gradually return these capabilities over time. Evaluate whether gradually exposing the agent to these capabilities improved the speed of evolution of the agent.

3.2 - Evolve against a simple agent, replace opponent with best agent after convergence and continue

Begin evolution with simple rule-based opponent. Once convergence in evolution occurs, replace the agent's opponent with the current best agent. Repeat this process incrementally and compare performance of each opponent with that of its successors.

4 - Evaluate the agent's performance versus a human opponent at various stages of its evolution

Test the performance of different agents at different stages of their evolution against human players. Attempt to find volunteers that rank at different skill levels from novice to expert. From this we can assess whether the agent was evolved to become competitive versus a human opponent.

3.3 Performance Assessment

3.3.1 Table of Project Prototypes

Objective	Date	Assessment
Prototype 1: Agent using neural network to sense environment and output simple actions in FightingICE	15/12/2016	Agent can proficiently perceive its environment and output simple actions
Prototype 2: Agent controlled by neural network being evolved by simple evolutionary algorithm, altering only the weights of the network	20/01/2017	Evaluate agents performance versus simple AI opponent at different stages of its evolution
Prototype 3: Agent evolved with HyperNEAT or other appropriate neuroevolution method	10/02/2017	Evaluate agent's performance versus AI and compare whether method was more effective than simple EA
Prototype 4: Agent with appropriate learning method implemented in incremental evolution environment	10/03/2017	Agent's environment is adapted to utilise incremental evolution. Agent's performance assessed and compared versus learning without incremental environment.

3.4 Risk Assessment

3.4.1 Table of Risks

No.	Risk Name	Probability	Response
1	Laptop failure	Low	Online storage (github), backups
2	Difficulties with FightingICE platform	Medium	Consult tutorials and relevant documentation from website
3	Difficulties with Java programming	Low	Spend more time familiarising with language and practising

4 Appendices

4.1 Images

Neuron - https://online.science.psu.edu/sites/default/files/bisc004/content/neuron.jpg
ANN - https://upload.wikimedia.org/wikipedia/commons/thumb/e/e4/Artificial_neural_network.svg
Artificial_neural_network.svg.png

5 References

References

[1] Michalski, R., Carbonell, J. and Mitchell, T. (1983). Machine Learning: An Artificial Intelligence Approach. pp. 5-20.

[2] Wilde, P (1997) **Neural Network Models** [book] P4-P14

[3] Barto, S. and Sutton R. (1998). Reinforcement Learning: An Introduction. pp. 23-30.

[4] McCallum, A., Nigam, K., Rennie, J. and Seymore, K. (2001). A Machine Learning Approach to Building Domain-Specific Search Engines. [online] Available at: http://www.kamalnigam.com/papers/cora-ijcai99.pdf [Accessed 23 Nov. 2016]

[5] Wernick, M., Yang, Y. and Strother, S. (2010). Machine Learning in Medical Imaging. [online] Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4220564/ [Accessed 23 Nov. 2016]

[6] Neural Networks Research Group. (2014). Research on Neuroevolution Applications. [online] Available at: http://www.cs.utexas.edu/users/nn/pages/research/ne-applications.html [Accessed 23 Nov. 2016]

[7] Intelligent Computer Entertainment Lab, Ritsumeikan University. FightingGameAICompetition. [online] Available at: http://www.ice.ci.ritsumei.ac.jp/ ftgaic/ [Accessed 23 Nov. 2016]

[8] Risi, S & Togelius, J (2015) **Neuroevolution in Games: State of the Art and Open Challenges** [online] Available at: https://arxiv.org/pdf/1410.7326.pdf [Accessed 03 Nov. 2016]

[9] Pace, A (2014) **Improving AI for simulated cars using Neuroevolution** [online] Available at: *http://commerce3.derby.ac.uk/ojs/index.php/gb/article/view/3/1* [Accessed 07 Nov. 2016]

[10] Lampropoulos, A (2005) **Machine Learning Paradigms** [online] Available at: *http://file.allitebooks.com/20150722/Machine%20Learning%20Paradigms-%20Applications%20in%20Recommender%20Systems.pdf* [Accessed 18 Nov. 2016]

[11] Gomez, F & Miikkulainen, R (1997) **Incremental Evolution of Complex General Behaviour** [online] Available at: *http://nn.cs.utexas.edu/downloads/papers/gomez.adaptive-behavior.pdf* [Accessed 18 Nov. 2016]

[12] Batsford, T (2014) **Calculating Optimal Jungling Routes in DOTA2 Using Neural Networks and Genetic Algorithms** [online] Available at: *http://commerce3.derby.ac.uk/ojs/index.php/gb/article/view/14/12* [Accessed 18 Nov. 2016]

[13] Hausknecht, M & Lehman, J & Stone, P (2014) **A Neuroevolution Approach to General Atari Game Playing** [online] Available at: *https://www.cs.utexas.edu/ mhauskn/papers/atari.pdf* [Accessed 15 Nov. 2016]

[14] Braylan, A & Hollenbeck, M & Meyerson, E & Miikulainen, R (2015) **Reuse of Neural Modules for General Video Game Playing** [online] Available at: *https://arxiv.org/pdf/1512.01537v1.pdf* [Accessed 17 Nov. 2016]

[15] Dehuri, S & Ghosh, S & Cho, S-B (2011) **Integration of Swarm Intelligence and Artificial Neural Network** [book] P1-P23

[16] Widrow, B. and Lehr, A. (1990). 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. [online] Available at: https://pdfs.semanticscholar.org/8b73/adda1-5fa71b0a35ffedb899d6a72d621923b.pdf [Accessed 24 Nov. 2016]

[17] Hornik, K. (1989). Multilayer Feedforward Networks are Universal Approximators. [online] Available at: http://deeplearning.cs.cmu.edu/pdfs/Kornick_et_al.pdf [Accessed 24 Nov. 2016]

[18] Stanford University. (2013). Applications of neural networks. [online] Available at: http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Applications/index.html [Accessed 24 Nov. 2016]

[19] Hornik, K. (1990). Approximation Capabilities of Multilayer Feedforward Networks. [online] Available at: http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf [Accessed 24 Nov. 2016]