Artificial neural networks

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Information and Communication Technology

Riihimäki

2018

Nika Korpi

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Nika Korpi **Vuosi** 2018

Työn nimi Neuroverkot

Työn ohjaaja Petri Kuittinen

TIIVISTELMÄ

Työssä on tavoitteena perehtyä neuroverkkojen toimintaan. Neuroverkot on haastava aihe ja nopealla katsauksella jopa mysteerinen, joka oli myös motiivi aiheen valitsemiseen.

Työssä on perehdytty miten neuroverkot matemaattisesti toimivat ja myös ohjelmoitu neuroverkkoja hyödyntäviä ohjelmia. Ensimmäinen ohjelma perehtyy neuroverkkojen perus rakenteeseen. Toinen ohjelma kouluttaa neuroverkon käyttämällä gradientin laskeutumista. Kolmannessa ohjelmassa koulutetaan syvä neuroverkko gradientin laskeutumisella. Neljännessä ohjelmassa opetetaan neuroverkko pelaamaan breakout peliä käyttämällä evoluutioalgoritmia.

Työn on enemmänkin tutkielma neuroverkkoihin erittäin vähäisillä pohjatiedoilla. Tarkkoja tavoitteita työlle ei ollut mutta yksi idea oli opettaa neuroverkko pelaamaan jotain peliä, joka myös toteutui.

Avainsanat Neuroverkko, koneoppiminen, tekoäly, peli

Sivut 27 sivua, joista liitteitä 19 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Nika Korpi **Year** 2018

Subject Artificial neural networks

Supervisor Petri Kuittinen

ABSTRACT

Goal of the thesis is to take a closer look at how artificial neural networks (ANN) work. ANNs is a challenging subject and at a quick glance even a mystery, which was also the motivation for the subject.

In this thesis the focus is on how the ANNs work mathematically and in the thesis project a few programs utilizing ANNs were built. The first program is an introduction to the basic structure of ANNs. The second program utilizes gradient descent in ANN training. The third program uses gradient descent to train a deep neural network. In the fourth program ANN is trained to play a breakout game using an evolutionary algorithm

The thesis is more of an exploration to ANNs and it was realized with very minimal earlier understanding of the subject. The end goal was not clear but teaching an ANN to play a game was an idea which was also realized.

Keywords Artificial neural network, machine learning, artificial intelligence, game

Pages 27 pages including appendices 19 pages

CONTENTS

1 INTRODUCTION

Artificial intelligence (AI) is currently in a hype cycle mostly because of artificial neural networks (ANN). There is a lot of discussion on what AIs can do in the future and the effects it might have on society, such as self driving cars, automation of middle management and service jobs. Obviously, AI is not going to change the world over night but will slowly develop over the upcoming decades. (Humans Need Not Apply, 2014)

The goal of this work was to take a closer look at ANNs and how they work. ANNs seem simple and complex at the same time. It is common to see simplified diagrams of ANNs and to get a rough sense of how they work but there are lots of things going on behind the surface.

To find out how ANNs work a few simple ANN programs are made and no libraries for ANNs are used. By building the ANNs from start to finish helps understand how they really work and gives a more solid understanding of ANNs. ANN programs are relatively simple starting form the simplest ANN to a breakout game AI.

2 ARTIFICIAL INTELLIGENCE

Artificial intelligence is a form of non-natural intelligence demonstrated by computers for example instead of humans or animals. Simple forms of AI would be game AIs which most commonly are pre-programmed to do predetermined set of actions at a given game state. This type of AI may not be considered as “true AI” because of its deterministic and limited nature. The goal for AI is to build something akin to human intelligence, being able to learn, reason, plan etc. (Artificial intelligence in video games, n.d.) (Artificial intelligence, n.d.)

2.1 Machine learning

Machine learning is a series of algorithms that allows a computer to “learn” to solve a problem by improving at a given task without being explicitly programmed. There are two methods of machine learning, supervised learning and unsupervised learning. Let us say we want to make a computer program to tell us if there is a dog or a cat in a given picture.

In supervised learning the program would learn by looking at pictures of dogs and cats and by being told that the given picture is either of a

dog or a cat. With enough examples the program would learn to distinguish between a cat and a dog in pictures.

In unsupervised learning the program tries to find similarities between pictures and categorize the pictures. With enough time the program should be able to distinguish between cats and dogs. Still the program will not know if they are either a cat or a dog but is able to tell that they belong to different categories. Unsupervised learning is more unpredictable as the program might find categories that are completely arbitrary to a given task, for example pose, background and even image noise, but in some cases, this is useful as some structure might be found in previously completely unstructured data or new structures to existing datasets.
(Machine learning, n.d.)

2.2 **Artificial neural networks**

Artificial neural networks (ANN) are inspired by natural neural networks like the brain but are greatly simplified. ANN is one form of machine learning and is used for such tasks as computer vision, speech recognition, playing games and so on.

The first ideas behind ANNs were developed in 1940s but didn't see any significant use until the past decade. ANN require lost of data and computational power which was and is limiting factor for ANN use. Today ANNs are actively researched and development, so new techniques and uses are found relatively rapidly. (Artificial neural network, n.d.)

The basic parts of ANNs are building the network and training it but there are several steps behind those. In the following chapters there are few example programs using ANNs to see how they function.

3 SIMPLE ARTIFICIAL NEURAL NETWORK

Here is an example of building a simple artificial neural network and an introduction to the different elements that build up artificial neural networks. The problem for the network to solve is simple and imaginary and works only as an example and platform to explain the different elements and the math behind artificial neural networks.

3.1 **Problem and data**

Creating a problem to be solved. Let us say you want to buy a house and you ask your friend's opinion on different houses. You collect all the opinions of your friend but notice that you forgot to mark down his/her opinion on one of the houses

Size (m^2)	Price (€)	Worth buying
105	95 000	Yes
110	150 000	No
95	85 000	Yes
40	55 000	No
75	65 000	Yes
195	200 000	No
150	120 000	Yes
180	180 000	No
150	100 000	Yes
130	140 000	No
230	190 000	Yes
130	160 000	?

Table 1 Friend's opinions on buying a house

As table 1 shows only house size and price are used as factors or at least documented to form the friend's opinion. Obviously more datapoints could be used like age of the house, it's condition, location and so on, but for simplicity only size and price were used here to form an opinion.

3.1.1 Graphing the data

To visualize the data better the data can be graphed. Now that only two datapoint are used the data can be easily graphed on a x, y scatter.

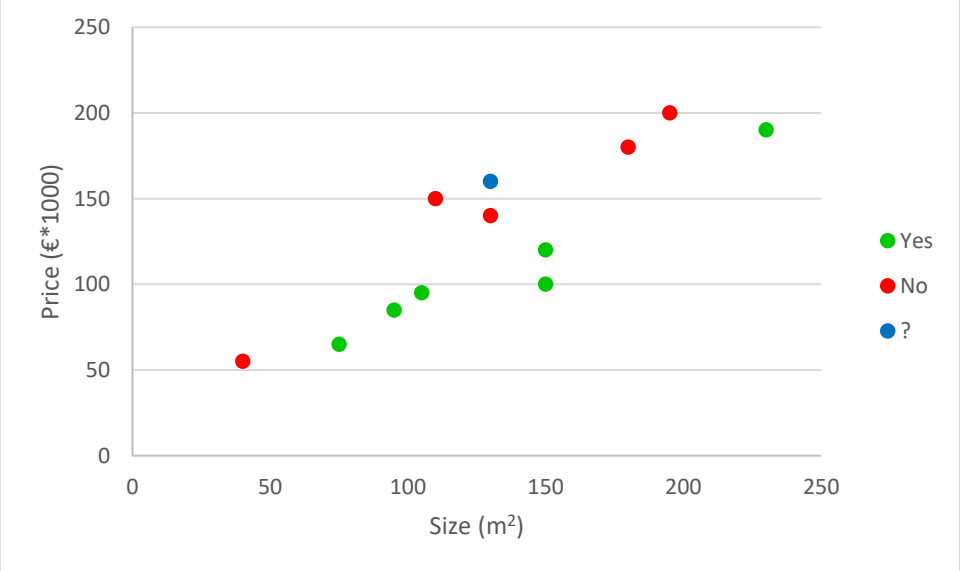


Figure 1 friend's opinion graphed

In figure 1 the data is graphed, and it shows that the unknown house (blue dot) is closely clustered with the not worth buying houses (red dots). By the clustering it is easy to deduce that the house is not worth buying but the goal is to make an ANN to solve this for us.

3.2 Forward flow

The house problem can be solved by a very simple artificial neural network which hopefully becomes apparent later, but first let us have look at the neural network and its components.

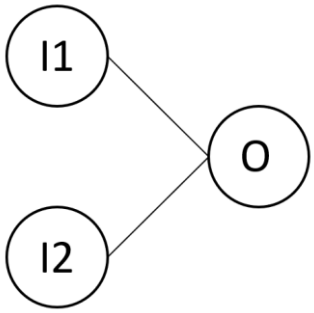


Figure 2 Artificial neural network for solving the house problem.

Figure 2 illustrates the simplest neural network that can be built that is still useful. This type of ANN is sometimes referred as perceptron. The perceptron has only two layers, input and output and here we have two input neurons I1 and I2 and one output neuron O. Synapses or connections are depicted as lines between the neurons. (Neural network zoo, n.d.)

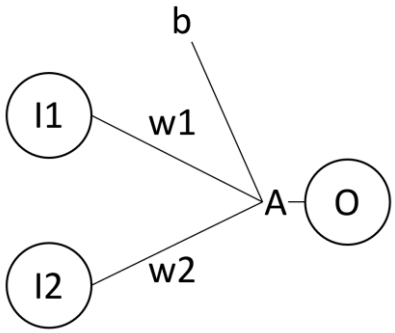


Figure 3 Artificial neural network expanded.

Figure 3 represents an ANN with all its parts that are used to calculate the output or prediction of the network.

$I1 = Input\ 1 = House\ size$
$I2 = Input\ 2 = House\ price$
$w1 = weight\ 1$
$w2 = weight\ 2$
$b = bias$
$A = Activation\ function$
$O = Output = Worth\ buying$

First let's see the result using identity function $f(x) = x$ which is a mathematical way of saying that no activation function is used. (Activation function, n.d.)

$$O = I1 * w1 + I2 * w2 + b$$

Now let's graph the function by giving w1, w2 and b random values.

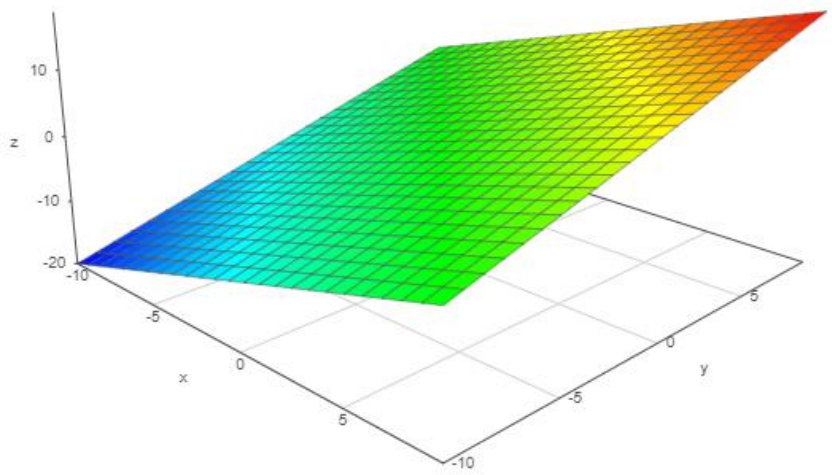


Figure 4 I1=x w1=1 I2=y w2=1 b=0 O=z (3D surface plotter, n.d.)

Figure 4 is a 3D result of the function with different inputs. The function creates a straight or linear plane that can give any value, but the output should be yes or no (0 for no and 1 for yes). To compress the result between 0 and 1 sigmoid function is used as the activation function.

$$y = \frac{1}{1 + e^{-x}}$$

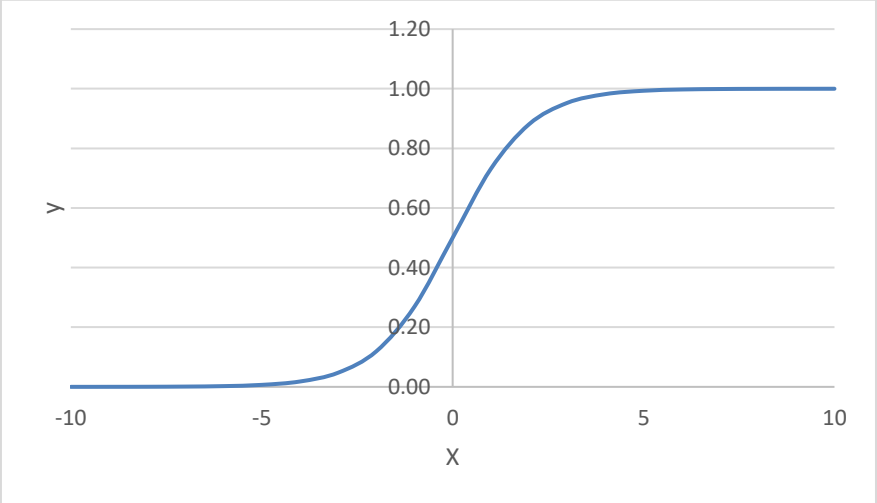


Figure 5 Sigmoid function.

Figure 5 is a graph of the sigmoid function. As x value get bigger the result gets closer to 1 and as the value of x gets smaller the result gets closer to 0. Now to write down the entire function and graphing it.

$$O = \frac{1}{1 - e^{-(I1*w1+I2*w2+b)}}$$

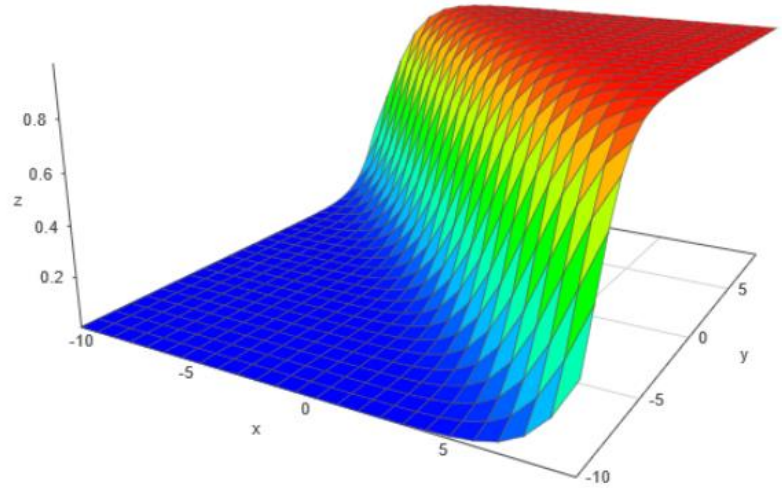


Figure 6 I1=x w1=1 I2=y w2=1 b=0 O=z (3D surface plotter, n.d.)

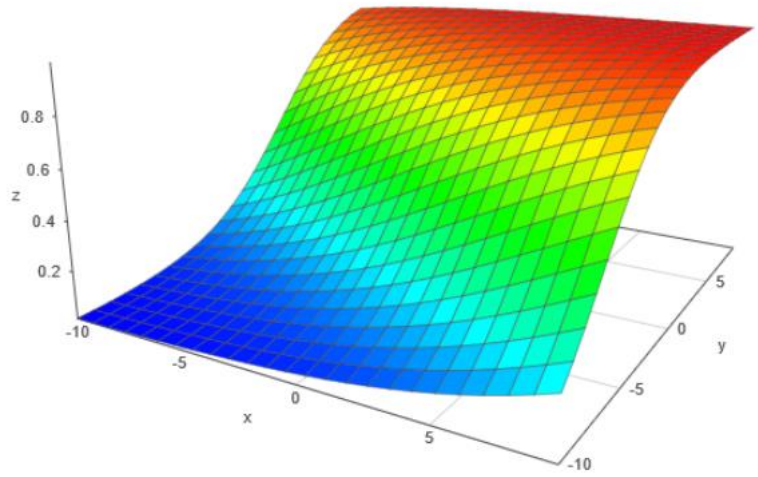


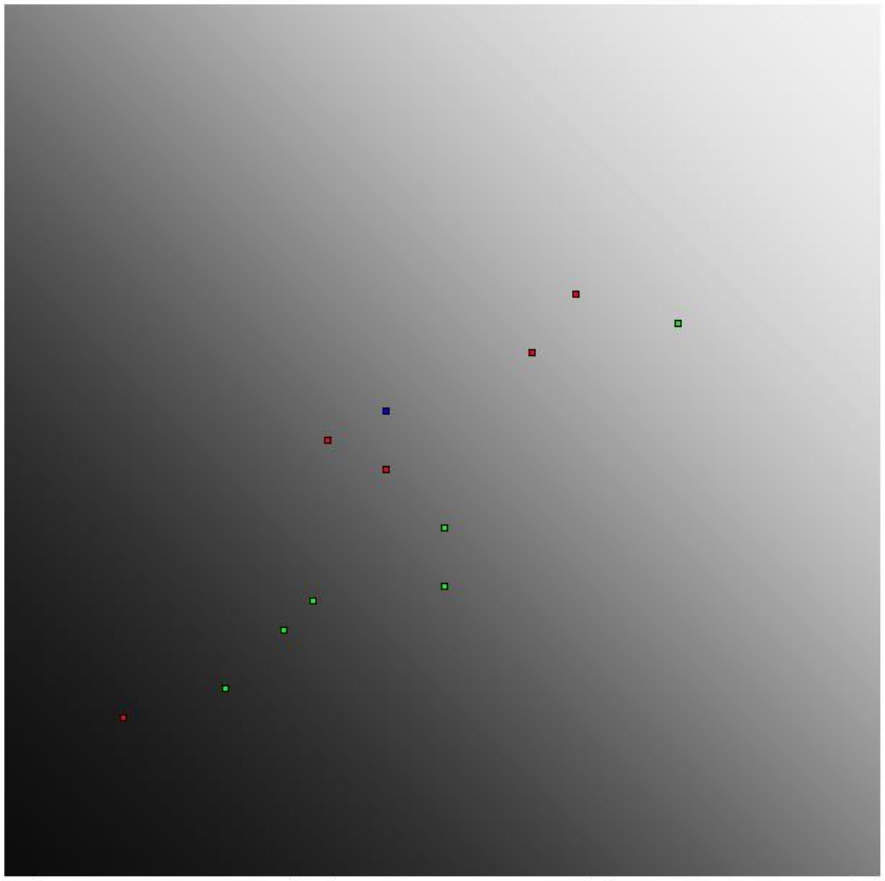
Figure 7 I1=x w1=0.2 I2=y w2=0.4 b=1 O=z (3D surface plotter, n.d.)

Figures 6 and 7 show something of the behavior of the sigmoid function. The slope gets steeper higher w1 and w2 get. The angle changes according to difference between w1 and w2. The position can be shifted by changing b. (Beginner Intro to Neural Networks, n.d.) (Activation function, n.d.)

3.2.1 Programming feed forward

This chapter contains as description of a simple JavaScript program to test and visualize the feed forward functionality. The whole source code is in appendix 1.

First the input data is changed so that it is closer to 1. The house sizes are divided by 100 and the prices are divided by 100 000. Now 100m^2 is 1 and 100 000€ is 1. The reason for doing this is when using sigmoid activation function, it returns values that are so close to 1 or 0 that double precision (64bit) floating-point numbers run out of decimal precision. Sigmoid for -700 for example is $9.9*10^{-305}$ and the closest values to 0 with double precision floating point are $2.2251*10^{-308}$ and $-2.2251*10^{-308}$. (Realmin, n.d.)



Size (I1)	Price (I2)	Expectation	Output
1.05	0.95	1	0.2689414213699951
1.1	1.5	0	0.401312339887548
0.95	0.85	1	0.23147521650098232
0.4	0.55	0	0.11405238127979088
0.75	0.65	1	0.16798161486607552
1.95	2	0	0.7211151780228631
1.5	1.2	1	0.4255574831883411
1.8	1.8	0	0.6456563062257954
1.5	1	1	0.3775406687981454
1.3	1.4	0	0.4255574831883411
2.3	1.9	1	0.7685247834990175
1.3	1.6	?	0.4750208125210601

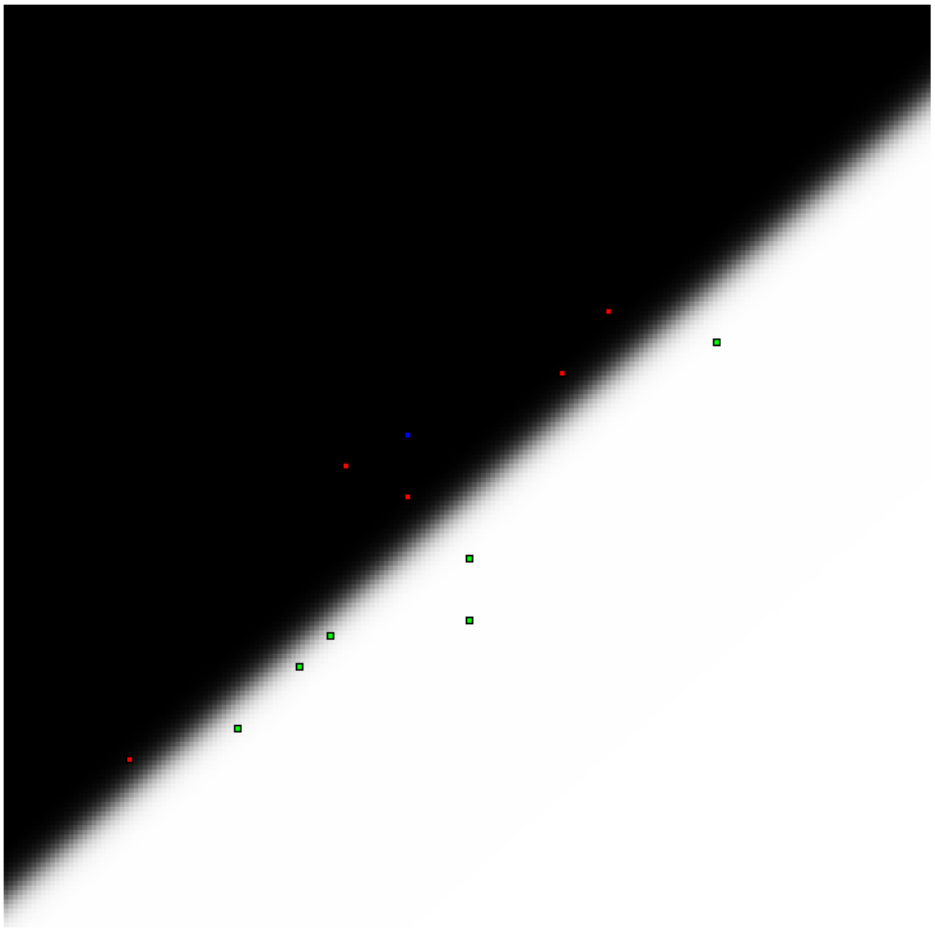
Figure 8 Screen capture of the feed forward program

Figure 9 illustrates is a screen capture of the software. With this tool it is possible to try different input weights and bias values and see the results. X axis (left to right) is the house size and Y axis (up and down) is house price. Bottom left is X=0, Y=0 and top right is X = 3 = 300 000€, Y = 3 = 300m². Red dots are the houses not worth buying, green ones are worth buying and the blue one is the unknown house. The background of the graph is representative of the output value the neural network. White means that the output of the network is 1 (yes). Black means that the output is 0 (no), shades of grey show

values between 1 and 0 as the value gets closer to 1 brighter the shade gets and as the values gets closer to 0 darker the shade becomes.

below the graph there are input fields for the network weights and bias. Pushing the draw button, the software reads the input fields and redraws the graph background according to the inputs and updates the outputs of the table at the end.

In the previous figure w1=1 w2=1 and b=-3 and it is clear form the graph and the output values in the table are off the target.



Size (I1)	Price (I2)	Expectation	Output
1.05	0.95	1	0.8581489350995123
1.1	1.5	0	0.0000015151418164850494
0.95	0.85	1	0.9002495108803148
0.4	0.55	0	0.04310725494108614
0.75	0.65	1	0.9525741268224334
1.95	2	0	0.0018329389424927972
1.5	1.2	1	0.9975273768433653
1.8	1.8	0	0.01477403169327312
1.5	1	1	0.9999938558253978
1.3	1.4	0	0.005486298899450429
2.3	1.9	1	0.9969815836752917
1.3	1.6	?	0.000013674009084599785

Figure 2 Screen capture of the feed forward program

Now the weights and biases have been adjusted so that the network output is very close to what is should be. The next step is to make the network adjust the weights and biases by itself.

3.3 Network training

Network training means that the networks adjusts its weights and biases so that the network output or prediction gets closer to the desired result. One common method of doing this is by backpropagation which was also used here. (Backpropagation, n.d.)

The first step is to calculate the network's error or cost and for that a squared error cost function is used.

$$Cost = (prediction - target)^2$$

The reason for using a squared error function is to have the error always as a positive number. Always positive cost value makes it possible to calculate the sum of the cost for multiple datapoints.

$$Cost = \sum_{i=1}^{n} (prediction_i - target_i)^2$$

$$Cost = (prediction_1 - target_1)^2 + (prediction_2 - target_2)^2 + \cdots (prediction_n - target_n)^2$$

Here the same function is seen represented in two different ways. It is just a sum of the cost for all the datapoints in the dataset to get the total cost of the network predictions. Calculating the average cost can be also useful in which the resulting cost is divided by the number of datapoints.

$$Cost = \frac{1}{n}(\sum_{i=1}^{n} (prediction_i - target_i)^2)$$

Graphing the cost function results in a parabola as seen in figure 10.

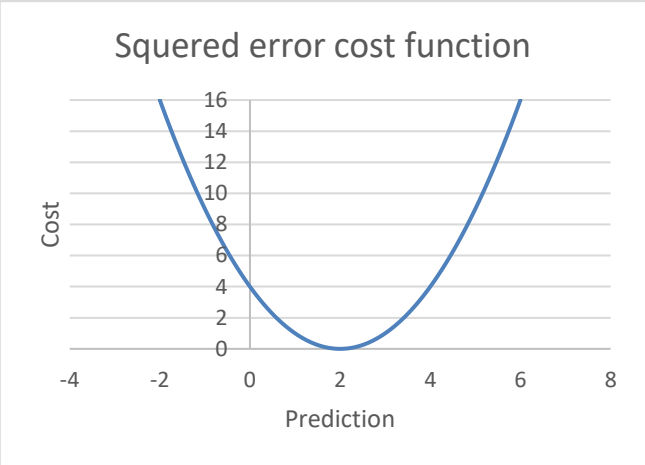


Figure 10 Squared error cost function. Target = 2, Prediction = x and Cost = y.

The goal is to minimize the cost and in figure 10 the cost is 0 when the prediction is 2 as the target is also 2. The next step is to make the function minimize the cost, for this the slope of the cost is used and that can be calculated by deriving the cost function.

$$dCost = 2(prediction - target)$$

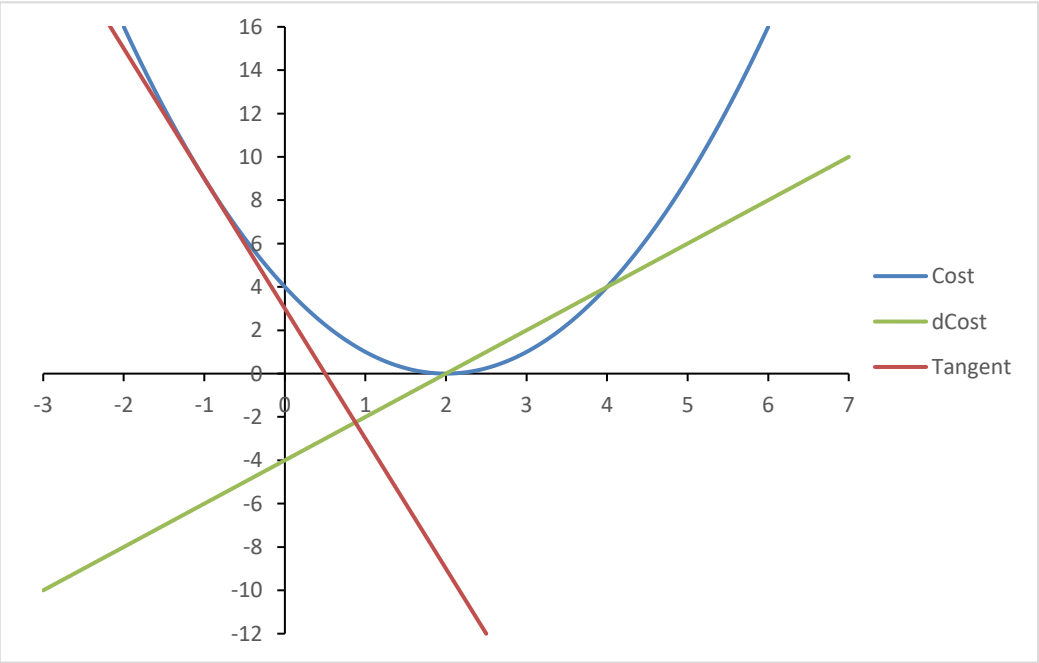


Figure 11 Cost function, derivative of the cost and tangent of the cost at -1.

In figure 11 the red tangent line is tangent to the cost when the prediction (x axis) is -1. The slope of the tangent line is -6 meaning that when the value of x is increased by 1 the value of y changes by -6. The dCost line shows that y is -6 when x is -1 so the dCost is the slope of the cost line at a given point.

The slope of the cost indicates which way to adjust the prediction, if the slope is negative, the prediction should be increased and if the

slope is positive, the prediction should be decreased. To adjust the prediction a fraction of the slope, called learning rate, is subtracted from the prediction.

$$prediction = prediction - learning\ rate * dCost$$

Running this function in a loop, the prediction gets closer and closer to the minimum cost value.

$$prediction = pred = -1$$
$$target = 2$$
$$learning\ rate = 0.3$$

Loop:
$$dCost = 2(pred - target) = 2(-1 - 2) = -6$$
$$pred = pred - learning\ rate * dCost = -1 - 0.3 * -6 = 0.8$$

The new prediction can be fed back to the loop. Calculating the loop again with the prediction value of 0.8 results in a new prediction value of 1.52 and using that for the next loop results in a prediction value of 1.808 and so on

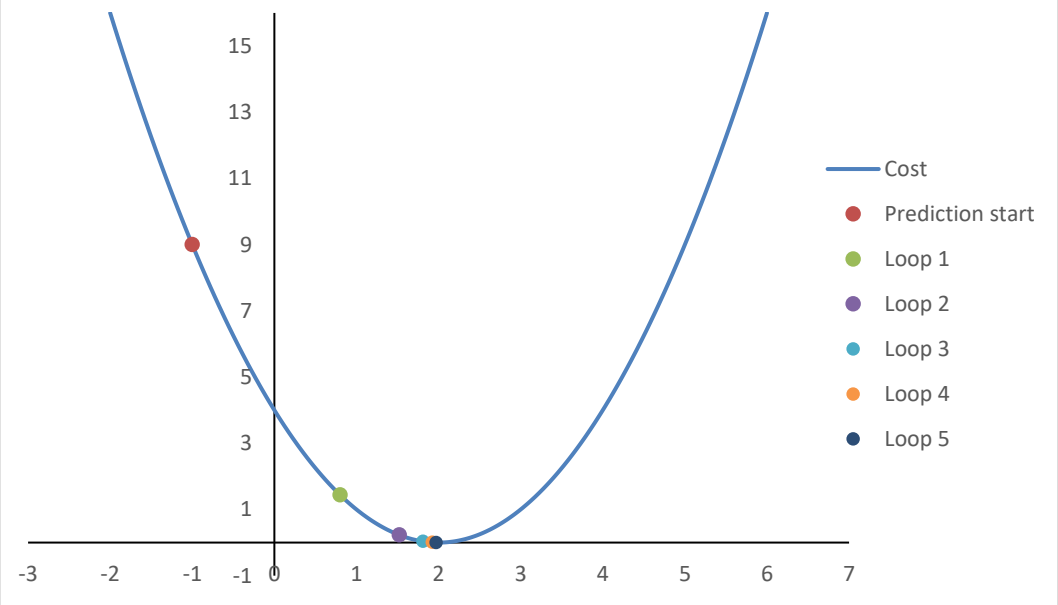


Figure 12 How the prediction gets closer to the target value.

Figure 13 shows how the prediction gets closer and closer to the target value of 2. Using too large learning rate will cause the correction to overshoot the target. This might not be a problem if the resulting cost is still lower than before but if the overshoot is so large that the cost increases then the cost will increase every subsequent loop also. This method of optimization is called gradient decent (Gradient descent, n.d.).

On neural networks the prediction is not a single number, it is the feedforward function of the entire network and what can be adjusted are the weights and the biases of the network.

Using the previous house example, the slope for w1, w2 and b can be calculated using the chain rule (Chain rule, n.d.). Finding the slope for the weights and bias can be done in steps.

$$z = I1 * w1 + I2 * w2 + b$$

First the z function calculates the input for the activation function.

$$prediction = Sigmoid(z) = \frac{1}{1 + e^{-z}}$$

Prediction is the sigmoid function for z. Now the derivatives of these functions.

$$dSigmoid(z) = predicton(1 - prediction)$$
$$dz(w1) = I1$$
$$dz(w2) = I2$$
$$dz(d) = 1$$

Now the derivative of the cost from before

$$dCost = 2(prediction - target)$$

Using the chain rule, it is possible to create function to calculate the cost slope for w1, w2 and b.

$$dCost(w1) = dCost * dSigmoid * dz(w1)$$
$$dCost(w2) = dCost * dSigmoid * dz(w2)$$
$$dCost(b) = dCost * dSigmoid * dz(b)$$

The derivative for b is 1 so the function can be simplified.

$$dCost(b) = dCost * dSigmoid$$

Calculating the new values for w1, w2 and b.

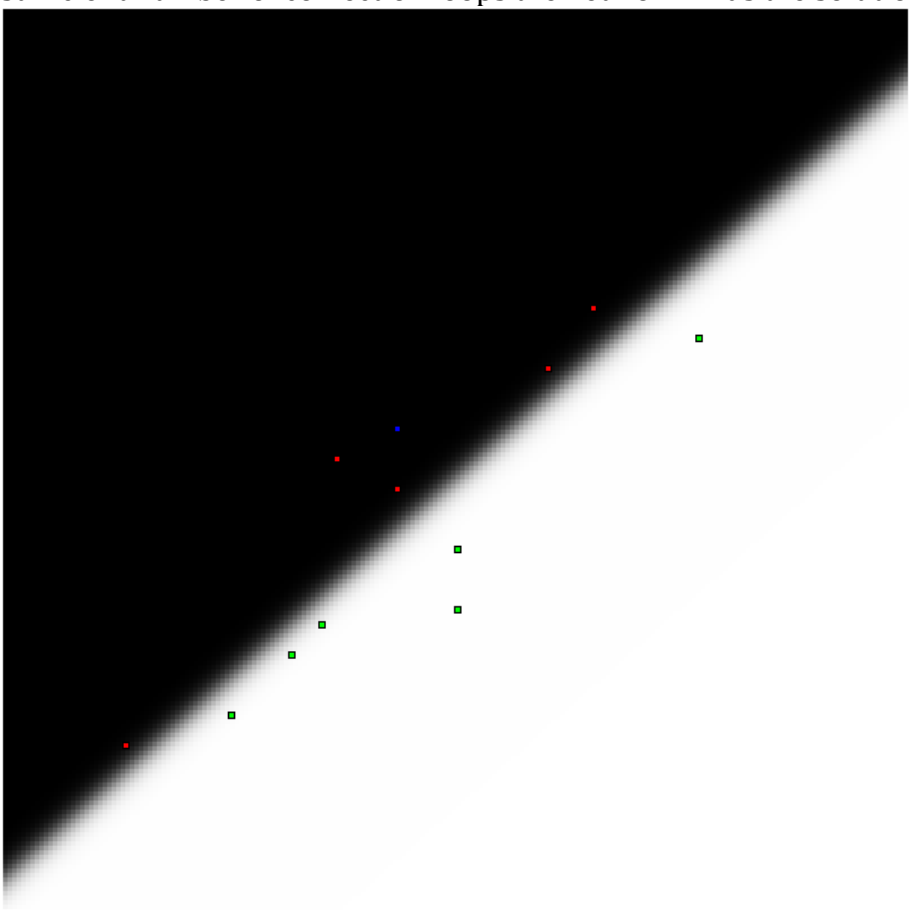
$$w1 = w1 - learning\ rate * dCost(w1)$$
$$w2 = w2 - learning\ rate * dCost(w2)$$
$$b = b - learning\ rate * dCost(d)$$

Now that the weights and bias are updated a new datapoint can be chosen and for that the prediction, slope, and update can be calculated creating a loop which will eventually find the solution for the problem.
(Beginner Intro to Neural Networks, n.d.)

3.3.1 Programming artificial neural network

The feed forward program is modified to train the artifician neural network in the house example. The whole code is in the appendix 2.

In this program only one random datapoint is taken and then the network's weights and bias are adjusted. It is be possible to first calculate the slope for all the datapoints and adjust the network after that, this way the network would always correct to the optimal direction. Using only one datapoint the network decreases the error for that datapoint but might increase the error for others, but after sufficient number of correction loops the network finds the solution.



Start! Stop!

w1: 30.00532748091059 w2: -33.83412448477286 b: 3.6274425830252115

Error: 0.007420750171203508

Training loop: 214970

Size (I1)	Price (I2)	Expectation	Output
1.05	0.95	1	0.9521484832695268
1.1	1.5	0	7.390615330347834e-7
0.95	0.85	1	0.9668659330573618
0.4	0.55	0	0.04837467808112358
0.75	0.65	1	0.9843152493757702
1.95	2	0	0.003948678575312102
1.5	1.2	1	0.999676013419662
1.8	1.8	0	0.03681315992198771
1.5	1	1	0.9999996268590821
1.3	1.4	0	0.008719756493587368
2.3	1.9	1	0.9997648047390053
1.3	1.6	?	0.000010127652589539598

Figure 13 Screenshot of the artificial neural network software.

First the program gives random values for w1, w2 and b. By pressing “start” the program starts to train the network and pressing “stop” will stop the training. “Error” tells the sum of the errors of the entire network. In the screenshot in figure 13 the program ran for 214970 training loops and the result was quite close what the goal was. The training loops slowed down so that in the beginning the program ran 1 loop per cycle (30ms) and increases the loops by 1% per cycle slowly speeding up the training. Slowing down the training provides a possibility to show how the network behaves while it works its way to the solution.

4 DEEP NEURAL NETWORK

A deep neural network or short for DNN is an artificial neural network with multiple layers between the input and the output. (Deep neural networks, n.d.) The goal was to make a network to predict house prices but nothing accurate just as a practice and an introduction to deep neural networks. The result was a simple tool that can adjust some of the parameters of the network like the number of layers and nodes per layer. With this tool it is possible to test if and how networks of different size can solve the problem

The training data was created by an algorithm so that as many datapoints could be created as needed. The data did not correlate with reality and only works as an example. There were four pieces of data for every datapoint house size, condition, location and price. Size, condition and location were the inputs for the network and the network output was the price. Backpropagation was used to train the network.

4.1 Backpropagation for deep neural network

First a closer look at how backpropagation works in larger networks as the previous example had no hidden layers. There are couple of small details that might not be self-evident form the previous example. First step in backpropagation is to calculate the networks prediction for given datapoint so that the slope of the cost can be calculated.

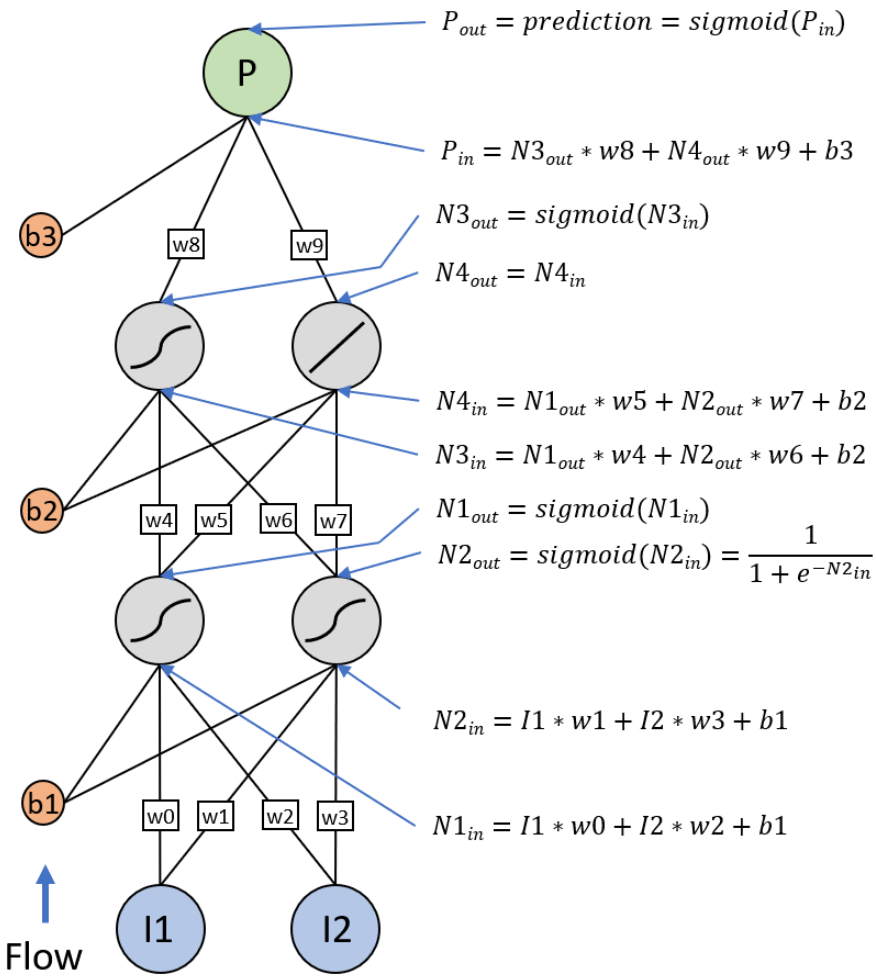


Figure 14 DNN forward flow

In figure 14 are the functions that are used to calculate the networks prediction. The calculation starts flowing form the inputs I1 and I2 through the network until arriving at the prediction or P node.

In figure 14 the blue nodes are the input neurons. Grey nodes are the hidden layer neurons and the s shape on the node means that sigmoid activation function is used, and the straight line means that identity activation function is used. Orange nodes are the bias nodes. Green node is the prediction node.

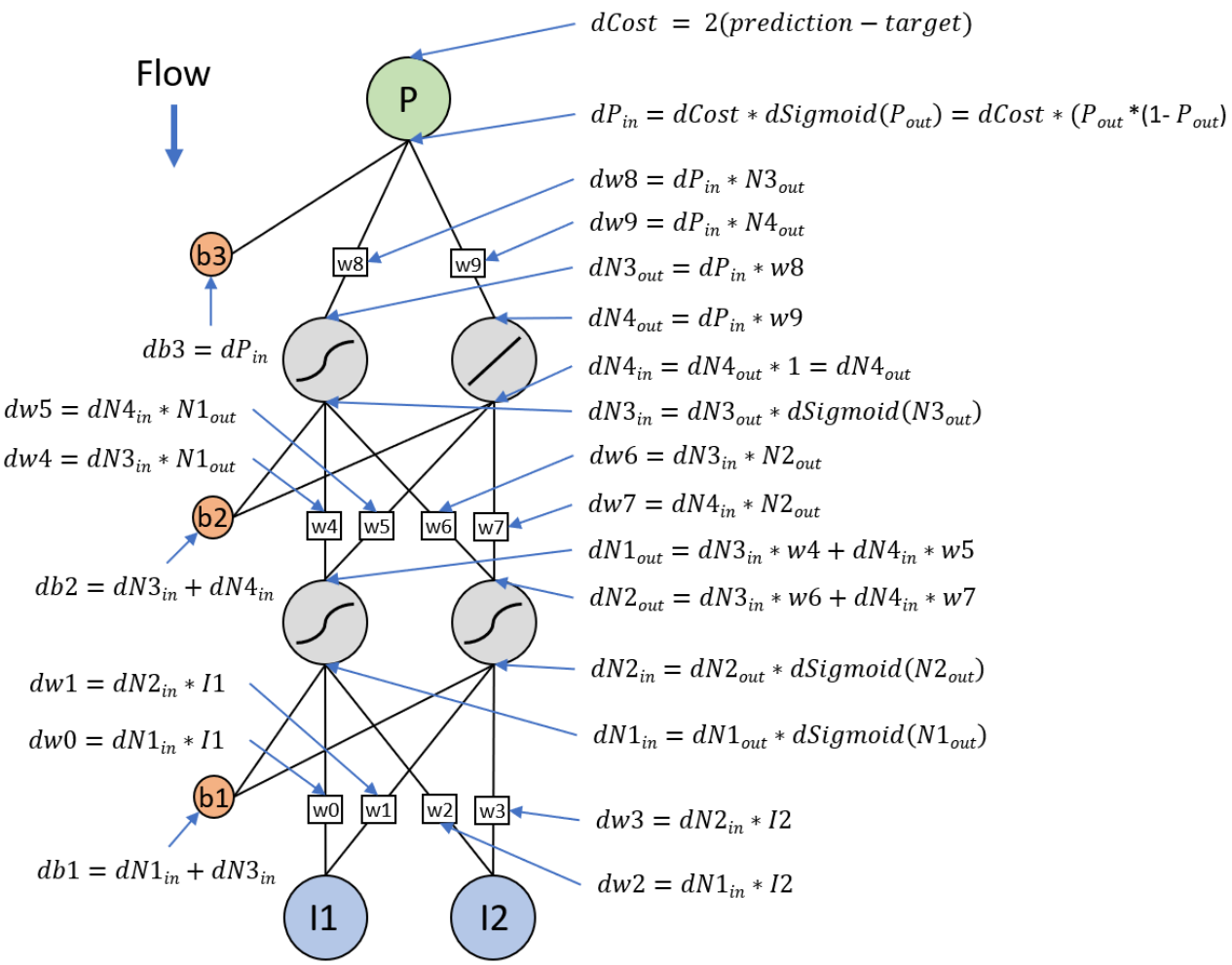


Figure 15 backpropagation

Figure 15 shows how the slope or gradient is calculated in every point of the network. The calculation starts from the prediction of the network and flows backwards thus the name backpropagation.

It is possible to calculate the slope for multiple datapoints before updating the networks weights and biases by calculating the average of the slope for given datapoints / inputs.

Example of adjusting w0 using multiple datapoints. First calculating the average slope for w0.

$$dw0_{avg} = \frac{1}{n}(\sum_{i=1}^{n} dw0_i) = \frac{dw0_1 + dw0_2 + \cdots + dw0_n}{n}$$

To adjust the weight.

$$w0 = w0 - learning\ rate * dw0_{avg}$$

This method works for weights and the biases.

(A step by step backpropagation example, n.d.) (Backpropagation, n.d.) (Deep neural networks, n.d.) (Beginner Intro to Neural Networks, n.d.)

4.2 Datapoint algorithm

The algorithm has 3 inputs house size, condition and location. Bigger the house is more expensive it is, and the house size is limited to 20m^2 – 300m^2 and the size is divided by 100, like in the previous example so that the data is between 0.2 – 3. Condition is a number between 0-3 and 3 being like a brand-new house and 0 being something barely livable. Location is also a number between 0-3 and 3 would be like say in the middle of Helsinki and 0 would be in the middle of nowhere.

Testing few different algorithms, the result was the following.

$s = House\ size$
$c = House\ condition$
$l = House\ location$

$$price = \left(\sqrt{s} + 0.7s\right) * \left(\frac{c}{10} + 0.7\right) * \left(\frac{l}{8} + 0.625\right)$$

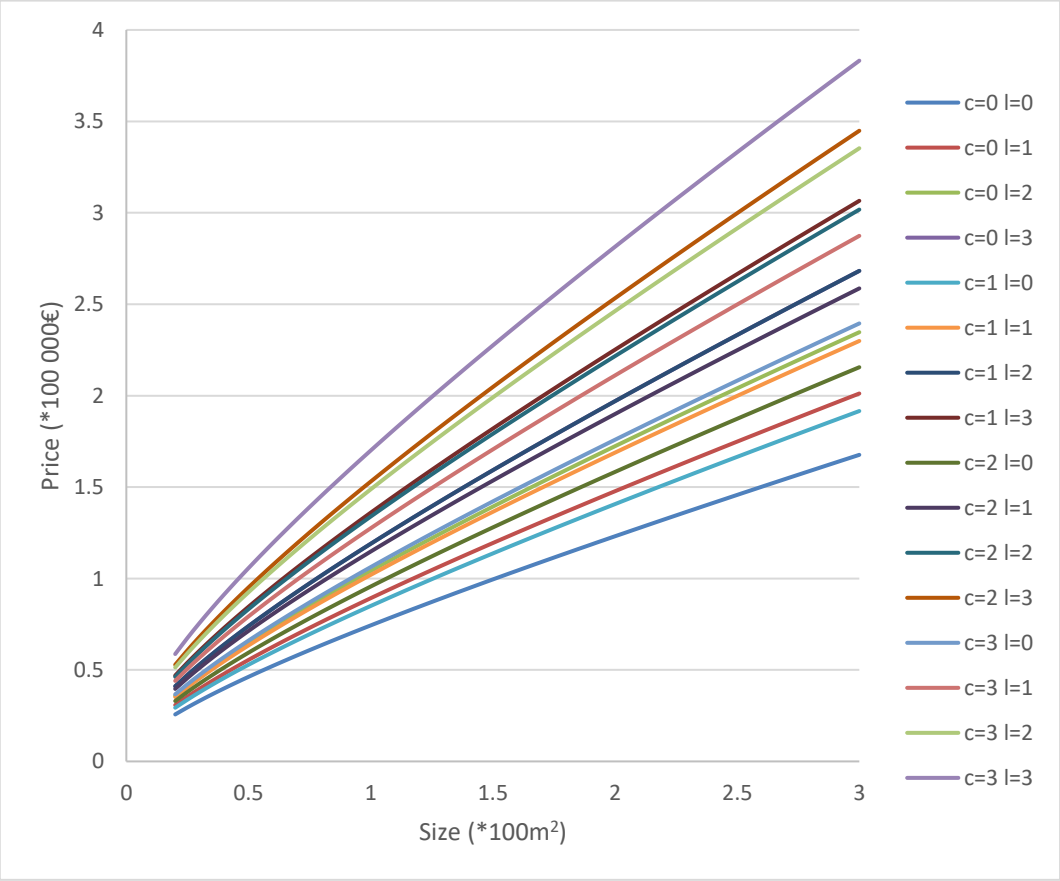


Figure 16 Price algorithm graphed

The algorithm creates a curve that makes smaller houses more expensive for the size and creates some separation between the houses in different condition and location.

To add some randomness to the data a possibility for that was also added. To create some randomness the result is multiplied by a random number between 0.9 and 1.1 creating variance of ±10% from the actual algorithm result.

4.3 Network structure

The network went through several iterations during the process. First version was with two hidden layers and four nodes or neurons per layer. The prediction node used the identity activation function as the output should be the house price limiting the result with the sigmoid between 0 and 1 would not work.

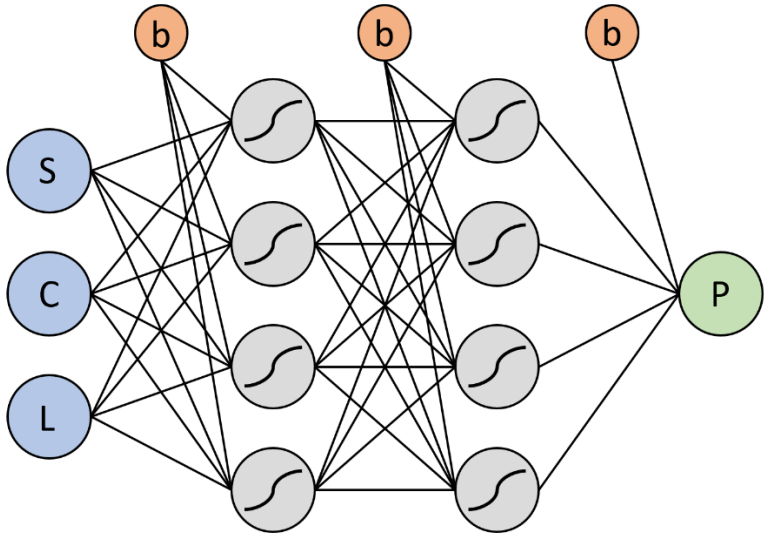


Figure 17 Version 1 of the network.

The first version was not able to solve the house price problem sufficiently. Using the same basic structure bigger networks were built like one with three hidden layers with five nodes each and this worked a bit better but did not give satisfactory results, so the structure of the network was changed to see if it would help.

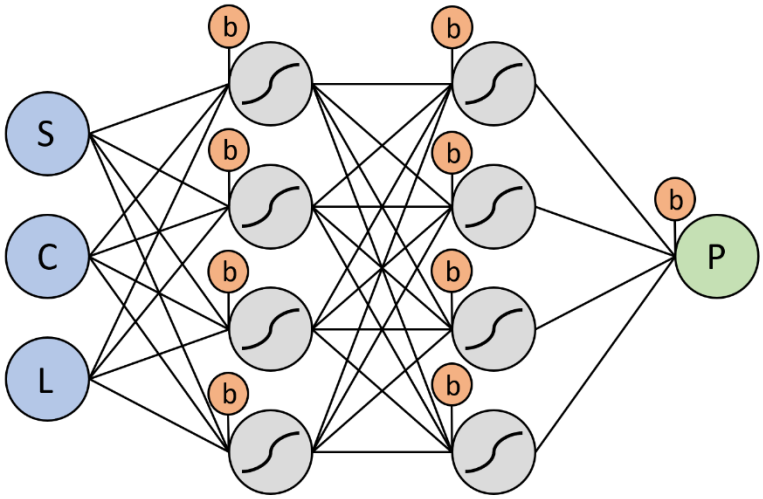


Figure 18 Version 2 of the network.

On the second version the bias was changed so that every node had its own bias. This change did not have any major effect on the results possibly none. It is hard to say as there was no measured testing done only subjective observation.

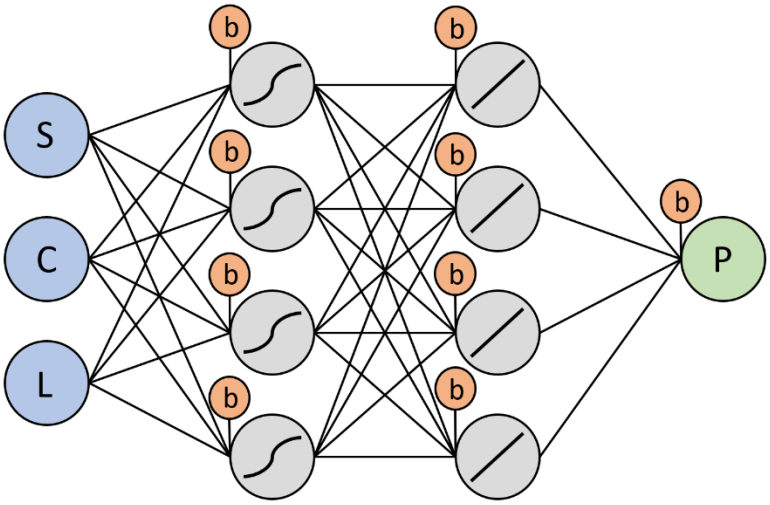


Figure 19 Version 3 of the network

On the third version the network was changed so that every layer could use ether identity or sigmoid activation function. Identity function is represented as the straight line on the hidden layer nodes. This had some effect on the shape of the resulting output but did not make the result any more correct.

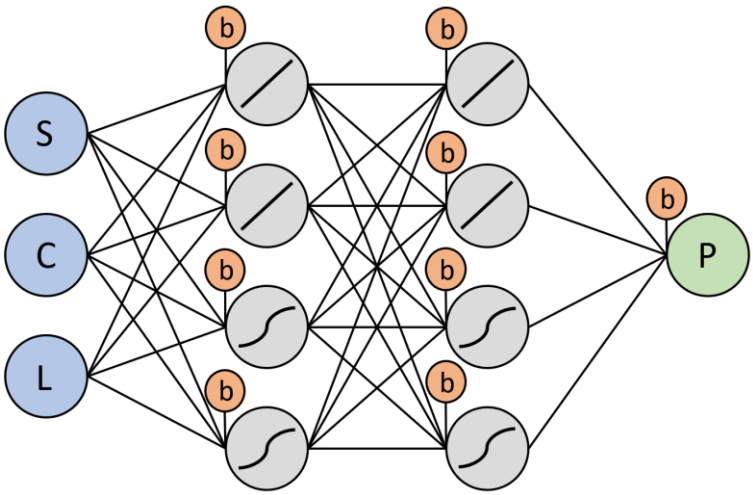


Figure 20 Version 4 of the network

The fourth version had the possibility to change every node individually to use either identity or sigmoid activation function. This had some effect on the result but still could not solve the problem perfectly.

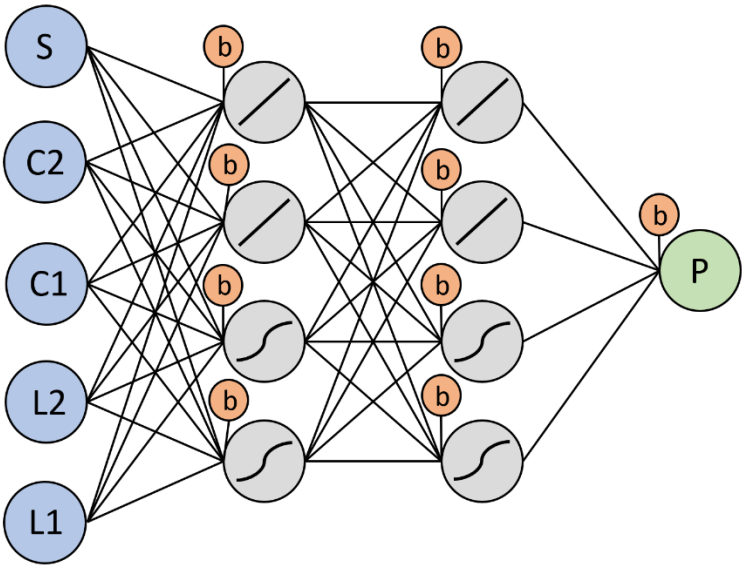
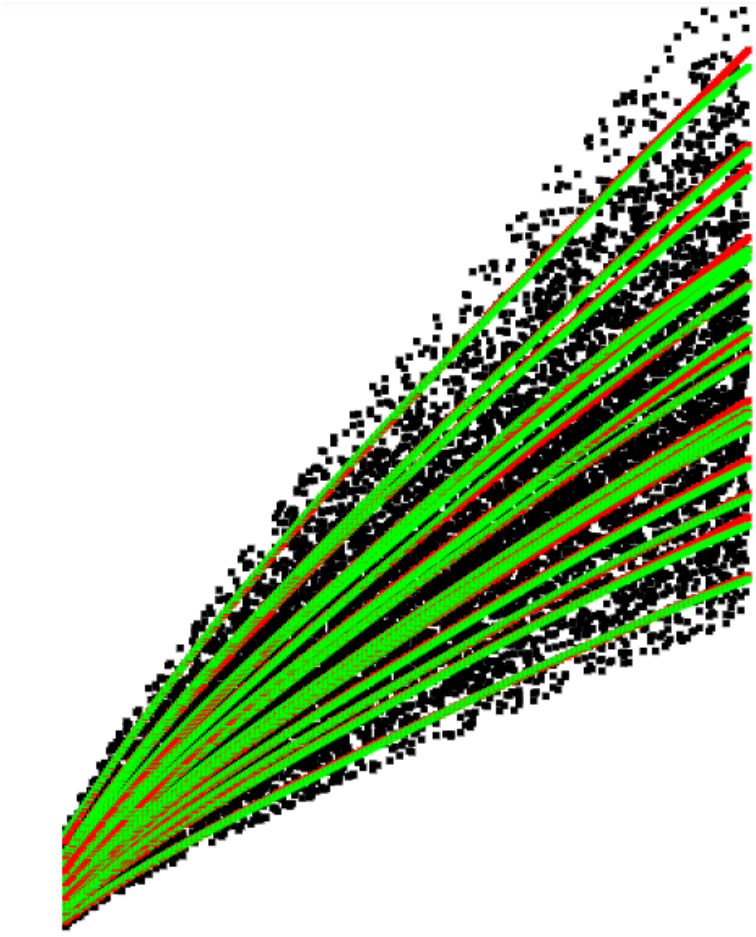


Figure 21 Version 5 of the network

On the fifth version the inputs were changed so that the condition and location were in binary format. Value 2 for example would be 10 in binary and so if the condition was 2 C2 would be 1 and C1 would be 0. This didn't either have that big of an effect on the result.

In the end user interface was built for the network so that different configurations and network sizes can be tested easily. And after further testing it seemed that the networks just needed to be bigger and train longer for satisfactory results. The program still runs on the version 5 of the network.



Start! Stop!
Random
200 Datapoints per training loop
100 Training loop per cycle
0.5 learning rate for sigmoid
0.01 learning rate for identity
Update!

Change networks hidden layer size.
L1 12 L2 12 L3 12 L4 0 L5 0 L6 0

Activation function. checked = sigmoid function. not checked = identity function.
L1
L2
L3
Update network

Test the network:
Size 150 m^2
Condition 2 0-3 higher better
Location 3 0-3 higher better
calculate

Network prediction:204004€
Goal:204000€

Figure 22 Screenshot form the user interface for network testing

In the program first there is a canvas which shows the goal for the network as red lines, the black dots are the datapoints and the green lines are the network prediction. In the program 10000 datapoints are created.

Checked random means that some randomness is added to the data if not checked the data would perfectly correlate with the datapoint algorithm. Datapoints per training loop adjusts how many datapoints are used to calculate the slope of the cost before adjusting the networks weights and biases. Adjusting training loops changes the count of training loops calculated between every canvas update loop which is 300ms or more if the computer can't keep up. The learning rates are also adjustable and are different for identity and sigmoid activation function. There is possibility to use up to 6 hidden layers and up to 99 nodes per layer. Every node can either use sigmoid or identity activation function. Also, it is possible to test and compare the networks result to the datapoint algorithm.

5 BREAKOUT GAME AI WITH ANN

Building a game AI with artificial neural network requires different training method for the network. Backpropagation won't work because backpropagation requires data with correct answers to calculate the error and gradient for the network. To use backpropagation in game AI training would require a perfect AI for the game already or lots of example games which to compare. Neither of those is an option here so evolutionary training is used instead. (Evolutionary algorithm, n.d.)

Breakout was chosen as the game to build the artificial neural network AI. The game is modified version of the phaser.io example of breakout. In addition to the neural network AI the game was modified so that multiple iterations of the game can be run simultaneously. (Breakout, n.d.)

The whole game code is in the appendix 4.

5.1 Evolutionary training

As backpropagation won't work other learning method is required. And one solution is to mimic evolution and implement a version of the survival of the fittest in the network training. This works so that a set of ANNs are created. The ANNs in this case play breakout and the fittest meaning the best players are selected to reproduce or mutate. In reproduction couple of the fittest networks are selected and a child is created which is basically just a mixture of the parents. Mutated version is the same network with small variations. Then the mutated

and or reproduced networks play the game again and by playing several rounds of the game the networks should slowly learn to play the game. (Evolutionary algorithm, n.d.) (Fitness function, n.d.)

5.2 Breakout

The used version of the breakout game is a phaser.io example. The game was modified so that every game instance is an object, with that multiple games can be run on top of each other. This allows to have multiple versions of the artificial neural network playing the game simultaneously. (Breakout, n.d.)

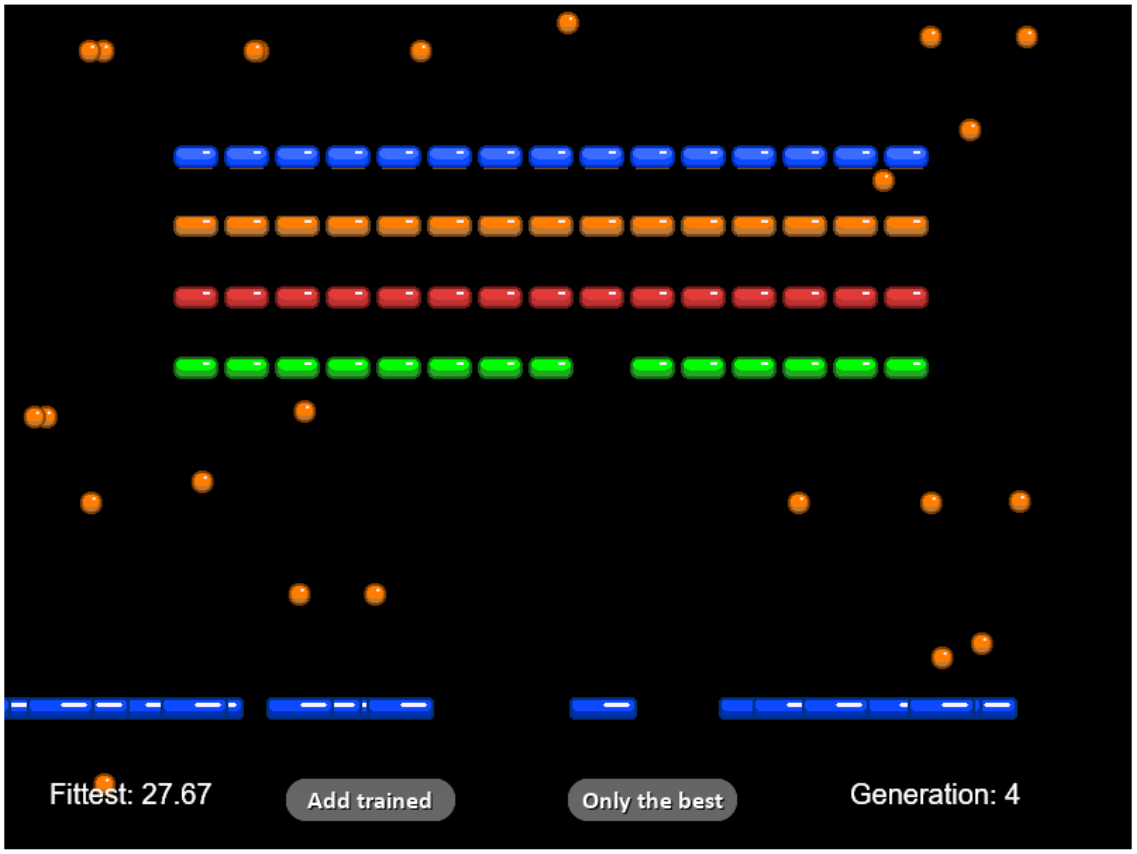


Figure 23 Screenshot of several artificial neural networks playing.

In breakout the player controls a paddle at the bottom of the screen. The paddle only moves left to right and the goal is to break all the bricks with the ball. If the ball falls off the bottom of the screen the game is lost.

In the game screen bottom the highest fitness score so far is shown and the current generation. The button “add trained” adds a previously trained network in the game to speed up training and to help demo what the result can be quickly. “Only the best” button shows only one game on the screen at once. Either the previous generations best is shown or if the previous generations best happens to lose before other ANNs haven’t then one of those is followed. After pressing the “only the best” button the button changes to “show all” and by pressing that all the players are shown.

5.3 Building the ANN

The basic ANN feed forward structure was picked form the house price example and modified slightly. The network size can be changed easily in code and in the current version the network has two hidden layers with 6 nodes each.

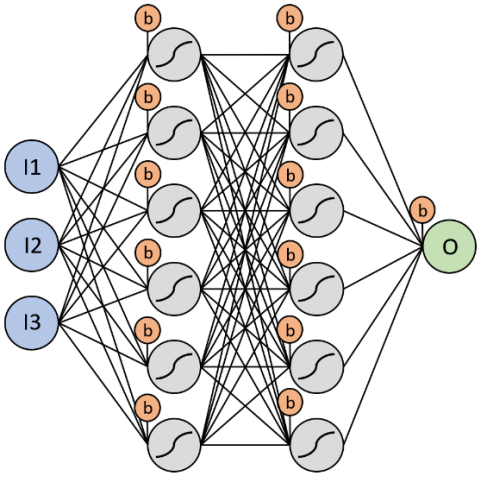


Figure 24 Breakout ANN

TanH was used as the activation function instead of sigmoid. TanH is a similar function to sigmoid but limits the result between -1 and 1.

$$TanH(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

(Activation function, n. d.)

At first the network inputs were paddle x position, ball x and y position and ball x and y velocity. Having 5 inputs for the network resulted in slow learning. The inputs were changed to ball x position subtracted from the paddle x position, ball y position and paddle x position. This resulted in faster learning. The inputs are also changed so that values are smaller to avoid running out of floating point precision.

$PaddleX = Paddle\ x\ positon$
$BallX = Ball\ x\ positon$
$BallY = Ball\ y\ positon$
$I1, I2, I3 = The\ network\ inputs$

$$I1 = \frac{PaddleX - BallX}{200}$$

$$I2 = \frac{BallY * -1 + 500}{400}$$

$$I3 = \frac{PaddleX}{300}$$

Output of the networks was at first two neurons one for moving left and one for moving right and if either of the outputs was over 0 the network would move in the direction of the output with the highest value. Later the network was change so that there was only one output and if the output was below -0.25 the paddle would move right and if the output was above 0.25 the paddle would move left.

5.4 ANN training

For the network training only, mutations of the evolutionary method were used. To calculate the fitness, it is more important in the beginning to prioritize on the paddle hitting the ball rather than the game score. The idea is to first learn to bounce the ball around and after that trying to maximize the game score.

$hits = Times\ ball\ has\ hit\ the\ paddle$
$score = Game\ score$
$fitness = ANN\ fitness\ rating$

If hits is smaller than 100

$$fitness = \frac{score}{100 - hits} + hits;$$

if hits is equal or larger than 100

$$fitness = score + hits;$$

Using the functions above will result in fitness rating that starts to prioritize game score longer the game goes on and after the ball has hit the paddle 100 times the game scores are calculated fully. In the game one point is added for every brick the ball hits and 50 points are added every time all the tiles have been cleared.

In this case 5 of the fittest networks are taken to the next round. From the 5 fittest 5 mutated versions are created for each and 10 completely random networks are also added. There are 35 networks playing every generation, 5 of the previous generations fittest 25 mutated networks and 10 random networks. The game starts a new generation after all the players have lost or after 3 minutes of playing. By adding a time limit the game won't get stuck if some network learns to bounce the ball in one corner forever.

Mutated versions were created so that the ANN weights and biases are adjusted random amount between $\pm 10\%$. If the weight or bias value is above -1 or below 1 the value is adjusted by random amount between -0.05 and 0.05. By not using percent adjustment near the zero value gives the possibility for the network to flip the value form positive value to negative or vice versa.

6 CHALLENGES AND PROBLEMS WITH ANNS

The holy grail of AI research is to build a human level general intelligence and as of today there is no clear path to this. ANNs are a powerful tool for creating AI but at least in their current form they fall short in any kind of general intelligence compared to humans and have their own shortcomings.

In programming examples ANNs learn slowly and require many training iterations. With more advanced ANNs the problem persists. Humans learn to identify objects with just a couple of examples, but for ANNs this requires thousands of examples and still they are not perfect. ANNs require lots of data and computing power. The goal is to do more with less but at this point ANN training requires basically infinite amount of data and computing power. (Marcus, 2018)

6.1 Fooling ANNs

In image recognition adding a small amount of a specific type of noise to an image can fool the ANNs into thinking a picture of a dog is a toaster with very high confidence. Even changing one pixel on an image can fool an ANN. In ANN training the ANN should learn to find everything relevant in an image and disregard the rest, but the ANN might find patterns that just happen to work great with the training data, with images outside of the training set it can fail in unpredictable ways. These types of attacks are a concern for the safety and security of AI. (Jiawei Su, 2018) (Anh Nguyen, 2015)

Images with added noise that are built to fool ANNs can also fool time limited humans meaning that the image is shown quickly (in less than 100ms). When viewing the image longer it can be seen what really is in the image. Images with noise that fool humans even with longer viewing can be created in a similar way, but with this type of cases you can argue that the image does not represent the original category anymore, but it shows that humans are not infallible either. (Gamaleldin F. Elsayed, 2018) (This Fools Your Vision, 2018)

7 EVOLUTION AND AI

As a thought experiment we can state that nature has had hundreds of millions of years of evolution to produce us humans and our intellect, while we have only been seriously trying to build AI since the 1940s which is for just 80 years. Even if it takes us 100 or 1000 or even 10000 years to build AI comparable to human intelligence it is still like a blink of an eye compared to evolutionary timescales.

With games we can see how AI is starting to beat us at everything like Deep Blue beat Garry Kasparov in chess in 1996 (Deep Blue, n.d.), Watson beat Ken Jennings and Brad Rutter in Jeopardy! in 2011 (Watson, n.d.), AlphaGo beat Lee Sedol in go in 2016 (AlphaGo, n.d.) and in 2018 OpenAI Five beat five player pro team in Dota 2 albeit in a limited game (OpenAI Five benchmark results, n.d.). AI is constantly improving and at least I cant see any fundamental limit that prevent us from creating AI that is smarter than us, it is just a question of time.

8 SUMMARY

The goal of the thesis was to take a closer look at artificial neural networks and to make some simple programs to learn at least the basics behind ANNs. As ANNs were only familiar by name to me at the start it was not clear if I could even make this thesis, but here it is.

The firs step was to try to build the simplest possible ANN and to start form there. The next step was to expand the ANN to solve something more complex and, in this case, to predict house prices with deep neural networks. Programming the forward pass of ANNs was relatively easy to understand, but backpropagation was confusing and hard to understand. After lots of trial, error and digging around the internet backpropagation became manageable even though it is still not easy.

There are many ways of training ANNs backpropagation being just one of them, so evolutionary training for breakout game AI was used here. The evolutionary training method was a lot easier to understand than backpropagation, but it is computationally more expensive. The benefit of the evolutionary training is that no prior knowledge of how to do a given task is required only the measurement of fitness, meaning which ANN is the best one at a given task.

The thesis could be easily expanded to for example, convolutional neural networks which are used in image recognition. Image recognition would require tens of thousands of images for training and testing. There might be some image sets with indexing of what there is in the images floating around the internet for free, but no serious searching was done in this project.

The goals of the thesis were met much to my surprise. The subject was so vast, and it is still developing rapidly so that there is near infinite amount still to learn. The basic idea behind the thesis was to get a rough understanding of ANNs and to get a sense what can be done with them and in that sense, it was a success. The future it is still unknown whether I will do something with ANNs or if I will just be an observer of future developments.

References

3D surface plotter. (n.d.). Retrieved 6 11, 2018, from Academo: https://academo.org/demos/3d-surface-plotter/

A step by step backpropagation example. (n.d.). Retrieved 8 4, 2018, from Mattmazur: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Activation function. (n.d.). Retrieved 6 11, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Activation_function

AlphaGo. (n.d.). Retrieved 9 22, 2018, from Wikipedia: https://en.wikipedia.org/wiki/AlphaGo

Anh Nguyen, J. Y. (2015). *Deep Neural Networks are Easily Fooled.* Retrieved from Evolving Artificial Intellgence Laboratory: http://www.evolvingai.org/files/DNNsEasilyFooled_cvpr15.pdf

Artificial intelligence in video games. (n.d.). Retrieved 9 10, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games

Artificial intelligence. (n.d.). Retrieved 9 10, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Artificial_intelligence

Artificial neural network. (n.d.). Retrieved 9 20, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Artificial_neural_network

Backpropagation. (n.d.). Retrieved 7 18, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Backpropagation

Beginner Intro to Neural Networks. (n.d.). Retrieved 7 9, 2018, from YouTube: https://www.youtube.com/watch?v=ZzWaow1Rvho&list=PLxt59R_fWVzT9bDxA76AHm3ig0Gg9S3So

Breakout. (n.d.). Retrieved 8 23, 2018, from Phaser.io: https://phaser.io/examples/v2/games/breakout

Chain rule. (n.d.). Retrieved 7 18, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Chain_rule

Deep Blue. (n.d.). Retrieved 9 22, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

Deep neural networks. (n.d.). Retrieved 8 4, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks

Evolutionary algorithm. (n.d.). Retrieved 8 24, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Evolutionary_algorithm

Fitness function. (n.d.). Retrieved 8 24, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Fitness_function

Gamaleldin F. Elsayed, S. S.-D. (2018, 5 22). *Adversarial Examples that Fool both Computer*. Retrieved from Cornell University Library: https://arxiv.org/pdf/1802.08195.pdf

Gradient descent. (n.d.). Retrieved 7 29, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Gradient_descent

Humans Need Not Apply. (2014, 8 13). Retrieved from YouTube: https://www.youtube.com/watch?v=7Pq-S557XQU

Jiawei Su, D. V. (2018, 2 22). *One pixel attack for fooling deep neural networks.* Retrieved from Cornell University Library: https://arxiv.org/pdf/1710.08864.pdf

Machine learning. (n.d.). Retrieved 9 20, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Machine_learning

Marcus, G. (2018, 1 2). *Deep Learning: A Critical Appraisal.* Retrieved from Cornell University Library: https://arxiv.org/ftp/arxiv/papers/1801/1801.00631.pdf

Neural network zoo. (n.d.). Retrieved 9 1, 2018, from Asimov instititute: http://www.asimovinstitute.org/neural-network-zoo/

OpenAI Five benchmark results. (n.d.). Retrieved 9 22, 2018, from OpenAI: https://blog.openai.com/openai-five-benchmark-results/

Realmin. (n.d.). Retrieved 6 20, 2018, from Mathworks: https://www.mathworks.com/help/matlab/ref/realmin.html

This Fools Your Vision. (2018, 4 5). Retrieved from YouTube: https://www.youtube.com/watch?v=AbxPbfODGcs

Watson. (n.d.). Retrieved 9 22, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Watson_%28computer%29

Appendix 1

Feed forward

```
<!DOCTYPE html>
<html>
  <head>
  <title>Feed forward</title>
</head>
  <body>
    <canvas id="myCanvas" width="600" height="600"></canvas>
    <br>
    w1:<input type="number" id="w1">
    w2:<input type="number" id="w2">
    b:<input type="number" id="b">
    <br><button onclick="draw()">draw!</button><br>
    <p id="points"></p>

  </body>
  <script>
    //data is shifted so that the input for the network is closer to 0
    //house size is divided by 100 so 100m^2 = 1
    //house price is divided by 100 000 so 100 000e = 1
    var data = [[1.05,0.95,1],
                [1.10,1.50,0],
                [.95,.85,1],
                [.40,.55,0],
                [.75,.65,1],
                [1.95,2.00,0],
                [1.50,1.20,1],
                [1.80,1.80,0],
                [1.50,1.00,1],
                [1.30,1.40,0],
                [2.30,1.90,1],];
    var unknown = [1.30,1.60];
    //randomise network values and load them to the input fields and run draw
function
    var w1 = Math.random();
    document.getElementById("w1").value = w1;
    var w2 = Math.random();
    document.getElementById("w2").value = w2;
    var b = Math.random();
    document.getElementById("b").value = b;
    draw();
    //outputs sigmoid for input x
    function sigmoid(x){
        return 1/(1+Math.exp(-x));
    }
    //outputs network output for input I1 and I2
    function forwardFeed(I1,I2){
        return sigmoid(I1 * w1 + I2 * w2 + b);
    }
    //draw's canvas
    function draw(){
        //Load w1, w2 and b values from the inputs
        w1 = parseInt(document.getElementById("w1").value);
        w2 = parseInt(document.getElementById("w2").value);
        b = parseInt(document.getElementById("b").value);
        //initialize canvas
        var c = document.getElementById("myCanvas");
        var ctx = c.getContext("2d");
        //variables for scaling data to the canvas
        var pixels = 200;
        var c_size = c.width;
        var pixel_size =  c_size / pixels ;
        var data_size =  pixels / 3;
```

```
        //Loop for drawing background to the canvas
        //The canvas works as a xy scatter where x axis = house size and y axis =
house price.
        //background gets it color from the network output
        //1 is white and shift to gray and becomes black at output 0
        for (x=0; x < pixels; x++){
            for (y=0; y < pixels; y++){
                var z = forwardFeed(x/data_size, y/data_size);
                z = (Math.floor(z*255)).toString(16);
                if (z.length==1){
                        z ="0"+z;
                }
                ctx.fillStyle = "#"+z+""+z+""+z;
                ctx.fillRect(x*pixel_size,(pixels-
y)*pixel_size,pixel_size,pixel_size);
            }
        }
        //loop to draw datapoints to the canvas
        //red dots are houses not worth buying (0)
        //green dot are houses worth buying (1)
        //blue dot is the mystery house
        //Creates table which shows all data points and the network output on those
points.
        var text = "<table><tr><th>Size (I1)<th>Price (I2)<th>Expectation<th>Output";
        for (d=0; d<data.length; d++){
            var p = data[d];
            ctx.fillStyle = "#000000"
            ctx.fillRect(
                p[0]*data_size*pixel_size-1,
                (pixels-p[1]*data_size)*pixel_size-1,
                pixel_size+2,pixel_size+2);
            if (p[2]==0){
                ctx.fillStyle = "#ff0000"
            }
            else{
                ctx.fillStyle = "#00ff00"
            }
            ctx.fillRect(
                p[0]*data_size*pixel_size,
                (pixels-p[1]*data_size)*pixel_size,
                pixel_size,pixel_size);
            ctx.fillStyle = "#000000"
            text
+="<tr><td>"+p[0]+"<td>"+p[1]+"<td>"+p[2]+"<td>"+forwardFeed(p[0],p[1])+"</tr>";
        }
        ctx.fillStyle = "#000000"
        ctx.fillRect(
            unknown[0]*data_size*pixel_size-1,
            (pixels-unknown[1]*data_size)*pixel_size-1,
            pixel_size+2,pixel_size+2);
        ctx.fillStyle = "#0000ff"
        ctx.fillRect(
            unknown[0]*data_size*pixel_size,
            (pixels-unknown[1]*data_size)*pixel_size,
            pixel_size,pixel_size);
        text +=
"<tr><td>"+unknown[0]+"<td>"+unknown[1]+"<td>?<td>"+forwardFeed(unknown[0],unknown[1]
)+"</tr></table>";
        document.getElementById("points").innerHTML = text;
    }
  </script>
</html>
```

Appendix 2

Simple artificial neural network

```
<!DOCTYPE html>
<html>
  <head>
  <title>Simple artificial neural network</title>
</head>
  <body>
    <canvas id="myCanvas" width="600" height="600"></canvas>
    <br>
    <br><button  onclick="interval()">Start!</button>
    <button  onclick="stopInterval()">Stop!</button>
    <p id="values"></p>
    <p id="error"></p>
    <p id="loop"></p>
    <p id="points"></p>
  </body>
  <script>
    //data is shifted so that the input for the network is closer to 0
    //house size is divided by 100 so 100m^2 = 1
    //house price is divided by 100 000 so 100 000e = 1
    var data = [[1.05,0.95,1],
              [1.10,1.50,0],
              [.95,.85,1],
              [.40,.55,0],
              [.75,.65,1],
              [1.95,2.00,0],
              [1.50,1.20,1],
              [1.80,1.80,0],
              [1.50,1.00,1],
              [1.30,1.40,0],
              [2.30,1.90,1],];
    var unknown = [1.30,1.60];
    //randomize network values
    var w1 = Math.random();
    var w2 = Math.random();
    var b = Math.random();

    var learning_rate = 0.4;

    var iterations = 1;
    var iteration_count = 0;
    var intervalID;

    draw();
    //creating program loop pressing start begins the program
    //The program loop runs loop program every 30ms to slowdown network training
    //Slowing the training gives the possibility to see what happens in the network.
    function interval(){
        intervalID = setInterval(loop,30);
    }
    function stopInterval(){
        clearInterval(intervalID);
    }
    //main program loop
    function loop(){
        train();
        draw();
        //iterations or training loops are increased by 1% every round to slowly
speed up the network training
        iterations = iterations * 1.01
    }

    //outputs sigmoid for input x
    function sigmoid(x){
        return 1/(1+Math.exp(-x));
```

```
    }

    //outputs derivitive of sigmoid
    function sigmoid_d(x){
        return sigmoid(x)*(1-sigmoid(x));
    }

    //outputs network output for input I1 and I2
    function forwardFeed(I1,I2){
        return sigmoid(I1 * w1 + I2 * w2 + b);
    }
    //training loop
    function train(){
        for (i=0;i<iterations;i++){
            //first to pic random house from the data
            var ri = Math.floor(Math.random()*data.length);
            var point = data[ri];
            //calculate the network prediction for the house
            var z = point[0] * w1 + point[1] * w2 + b;
            var pred = sigmoid(z)
            //load the target for the house (yes = 1 or no = 0)
            var target = point[2];
            //calculate the error for the prediction
            var cost =  Math.pow(pred - target, 2);
            //calculate slope of the cost
            var dcost_pred = 2 * (pred - target);
            //calculate derivatives for all the points in the network
            var dpred_dz = sigmoid_d(z);
            var dz_dw1 = point[0];
            var dz_dw2 = point[1];
            var dz_db = 1;
            //calculate slope for the sigmoid function(which way to correct the
parameters)
            var dcost_dz = dcost_pred * dpred_dz;
            //Calculating slope for w1 w2 and b
            var dcost_dw1 = dcost_dz * dz_dw1
            var dcost_dw2 = dcost_dz * dz_dw2
            var dcost_db = dcost_dz * dz_db
            //Adjusting the network parameters
            w1 = w1 - learning_rate * dcost_dw1
            w2 = w2 - learning_rate * dcost_dw2
            b = b - learning_rate * dcost_db
            //Add one to loop counter
            iteration_count++;
        }
    }

    //Updates canvas and text fields.
    function draw(){
        //calculate the sum of all errors.
        var cost_sum = 0;
        for (j=0; j<data.length; j++){
            point = data[j];
            pred = forwardFeed(point[0], point[1]);
            target = point[2];
            cost_sum += Math.pow(pred - target, 2);
        }
        //Updates text fields for network values, network error and training loop
count.
        document.getElementById("values").innerHTML = "w1: "+w1+" w2: "+w2+" b: "+b;
        document.getElementById("error").innerHTML = "Error: "+cost_sum;
        document.getElementById("loop").innerHTML = "Training loop:
"+iteration_count;
        //initalize canvas
        var c = document.getElementById("myCanvas");
        var ctx = c.getContext("2d");
        //variables for scaleing data to the canvas
```

```
        var pixels = 200;
        var c_size = c.width;
        var pixel_size =  c_size / pixels ;
        var data_size =  pixels / 3;
        //Loop for drawing background to the canvas
        //The canvas works as a xy scatter where x axis = house size and y axis =
house price.
        //background gets it color from the network output
        //1 is white and shift to gray and becomes black at output 0
        for (x=0; x < pixels; x++){
            for (y=0; y < pixels; y++){
                var z = forwardFeed(x/data_size, y/data_size);
                z = (Math.floor(z*255)).toString(16);
                if (z.length==1){
                        z ="0"+z;
                }
                ctx.fillStyle = "#"+z+""+z+""+z;
                ctx.fillRect(x*pixel_size,
                            (pixels-y)*pixel_size,
                            pixel_size,pixel_size);
            }
        }
        //loop to draw datapoints to the canvas
        //red dots are houses not worht bying (0)
        //green dot are houses worth bying (1)
        //blue dot is the mystery house
        //Creates table whitch shows all of the input and the network output on those
points.
        var text = "<table><tr><th>Size (I1)<th>Price (I2)<th>Expectation<th>Output";
        for (d=0; d<data.length; d++){
            var p = data[d];
            ctx.fillStyle = "#000000"
            ctx.fillRect(
                p[0]*data_size*pixel_size-1,
                (pixels-p[1]*data_size)*pixel_size-1,
                pixel_size+2,pixel_size+2);
            if (p[2]==0){
                ctx.fillStyle = "#ff0000"
            }
            else{
                ctx.fillStyle = "#00ff00"
            }
            ctx.fillRect(
                p[0]*data_size*pixel_size,
                (pixels-p[1]*data_size)*pixel_size,
                pixel_size,pixel_size);
            ctx.fillStyle = "#000000"
            text
+="<tr><td>"+p[0]+"<td>"+p[1]+"<td>"+p[2]+"<td>"+forwardFeed(p[0],p[1])+"</tr>";
        }
        ctx.fillStyle = "#000000"
        ctx.fillRect(
            unknown[0]*data_size*pixel_size-1,
            (pixels-unknown[1]*data_size)*pixel_size-1,
            pixel_size+2,pixel_size+2);
        ctx.fillStyle = "#0000ff"
        ctx.fillRect(
            unknown[0]*data_size*pixel_size,
            (pixels-unknown[1]*data_size)*pixel_size,
            pixel_size,pixel_size);
        text +=
"<tr><td>"+unknown[0]+"<td>"+unknown[1]+"<td>?<td>"+forwardFeed(unknown[0],unknown[1]
)+"</tr></table>";
        document.getElementById("points").innerHTML = text;
    }
  </script>
</html>
```

Appendix 3

Deep neural network

```
<!DOCTYPE html>
<html>
  <head>
  <title>House price</title>
</head>
  <body>
    <canvas id="myCanvas" width="600" height="600"></canvas>
    <br>
    <br><button  onclick="interval()">Start!</button>
    <button  onclick="stopInterval()">Stop!</button><br>
    <input type="checkbox" id="random" checked>Random<br>
    <input type="number" id="points" min="1" max="10000" value="100">Datapoints per
training loop<br>
    <input type="number" id="loops" min="1" max="100000" value="100">Training loop
per cycle<br>
    <input type="number" id="sigmoid" value="0.5">learning rate for sigmoid<br>
    <input type="number" id="identity" value="0.01">learning rate for identity<br>
    <button onclick="update()">Update!</button><br><br>
    Change networks hidden layer size.<br>
    L1
    <input type="number" id="layer1" min="0" max="99" value="8">L2
    <input type="number" id="layer2" min="0" max="99" value="8">L3
    <input type="number" id="layer3" min="0" max="99" value="0">L4
    <input type="number" id="layer4" min="0" max="99" value="0">L5
    <input type="number" id="layer5" min="0" max="99" value="0">L6
    <input type="number" id="layer6" min="0" max="99" value="0"><br><br>
    Activation function. checked = sigmoid function. not checked = identity function.
    <table id="sig"></table>
    <button onclick="cns()">Update network</button><br><br>
    Test the network:<br>
    <table>
        <tr><td>Size</td><td><input type="number" id="size" min="20" max="300"
value="150">m^2</td></tr>
        <tr><td>Condition</td><td><input type="number" id="con" min="0" max="3"
value="3">0-3 higher better</td></tr>
        <tr><td>Location</td><td><input type="number" id="loc" min="0" max="3"
value="3">0-3 higher better</td></tr>
        <tr><td><button onclick="calculate()">calculate</button></td></tr>
    </table>
    <p id="result"></p>
  </body>
  <script>

    var ann = [];
    var weights = [];
    var bias = [];
    var data = [];
    var intervalID;
    var ann_size = [5,8,8,1];
    var sig = create_sig();
    var learning_rate=[0.01,0.5];
    var random=true;
    var points=100;
    var loops=100;

    DataPoint();
    Network();
    Draw();

    //Pressing start button will start the training loop
    function interval(){
        intervalID = setInterval(loop,100);
    }
```

```
    //Pressing stop will stop the training
    function stopInterval(){
        clearInterval(intervalID);
    }

    //calculater the prediction of the network an datapoint algorithm for givien
datapoint in the UI
    function calculate(){
        s = num_input(document.getElementById("size").value,false,20,300);
        document.getElementById("size").value = s;
        s /= 100;

        c = num_input(document.getElementById("con").value,false,0,3);
        document.getElementById("con").value = c;

        l = num_input(document.getElementById("loc").value,false,0,3);
        document.getElementById("loc").value = l;

        ForwardFeed([s,c,l]);
        p_pred = Math.floor(ann[ann.length-1][0]*100000);
        p_goal = Math.floor(Price(s,c,l)*100000);

        document.getElementById("result").innerHTML = "Network
prediction:"+p_pred+"&#8364<br>Goal:"+p_goal+"&#8364";
    }

    //Updates the network according to the UI inputs.
    function cns(){
        var l=[];
        for(z=0; z < 6; z++){
            l[z] =
num_input(document.getElementById("layer"+(z+1)).value,false,0,99);
        }
        for(z = l.length-1; z >= 0; z--){
            if (l[z] == 0){
                l.splice(z,1);
            }
        }
        for(z=0; z < 6; z++){
            if(isNaN(l[z])){
                document.getElementById("layer"+(z+1)).value = 0;
            }
            else{
                document.getElementById("layer"+(z+1)).value = l[z]
            }
        }
        ann_size = [5]
        for(z = 0; z < l.length; z++){
            ann_size[z+1]=l[z];
        }
        ann_size.push(1);
        sig = create_sig();
        stopInterval();
        Network();
        Draw();
    }

    //Checks number inputs if they are valid.
    //Changes the string input to either float (float=true) or int (float=false)
    //Return minimum value if the input is not valid.
    //Corrects values over the max to max and values below min to min.
    function num_input(num,float,min,max){
        if (float){
            num = parseFloat(num);
        }
        else{
```

```
            num = parseInt(num);
        }
        if (num <= min || isNaN(num)){
            num = min;
        }
        else if (num >= max){
            num = max;
        }
        return num;
    }

    //Updates the training variables form the UI
    function update(){
        random = document.getElementById("random").checked;

        points = num_input(document.getElementById("points").value,false,1,10000);
        document.getElementById("points").value = points;

        loops = num_input(document.getElementById("loops").value,false,1,100000);
        document.getElementById("loops").value = loops;

        learning_rate[1] =
num_input(document.getElementById("sigmoid").value,true,0,1);
        document.getElementById("sigmoid").value = learning_rate[1];

        learning_rate[0] =
num_input(document.getElementById("identity").value,true,0,1);
        document.getElementById("identity").value = learning_rate[0];

        stopInterval();
        DataPoint();
        Network();
        Draw();
    }

    //Training loop controlled by interval (start and stop buttons)
    function loop(){
        for (i=0;i<loops;i++){
            Train();
        }
        Draw();
    }

    //calculates price for given house size condition and location
    function Price(size,condition,location){
        condition = condition/10+0.7;
        location = location/8+0.625;
        var price = (Math.pow(size, 0.5)+0.7*size)*condition*location;
        price = Math.floor(price*100)/100;
        return price
    }

    //creates datapoints at with random size condition and location and if used adds
+-10% randomness to the data.
    //creates 10000 datapoints.
    function DataPoint(){
        var size;
        var condition;
        var location;
        for(var i = 0; i<10000;i++){
            size = (Math.floor(Math.random()*280)+20)/100;
            condition = Math.floor(Math.random()*4);
            location = Math.floor(Math.random()*4);
            randomness = Math.random()*0.2+0.9;
            if (random){
                data[i] =
[size,condition,location,Price(size,condition,location)*randomness];
```

```
            }
            else{
                data[i] = [size,condition,location,Price(size,condition,location)];
            }
        }
    }

    //returns random number between -0.5 and 0.5
    function R(){
        return Math.random()-0.5;
    }

    //outputs sigmoid for input x
    function sigmoid(x){
        return 1/(1+Math.exp(-x));
    }

    //outputs derivative of sigmoid
    function dsigmoid(x){
        return x*(1-x);
    }

    //Generates 2D array which has the data if the given node uses sigmoid or
identity function.
    //Reads and updates se activation function table on the UI
    function create_sig(){
        var text = "";
        s = [[]];
        for(node = 0; node < ann_size[0]; node++){
            s[0][node]=0;
        }
        for (layer = 1; layer < ann_size.length-1; layer++){
            s[layer]=[];
            text +="<tr><td>L"+layer+"</td>";
            for (node = 0; node < ann_size[layer]; node++){
                text += "<td><input type=\"checkbox\" id=\"node"+layer+node+"\"";
                if(document.getElementById("node"+layer+node)==null ||
document.getElementById("node"+layer+node).checked){
                    s[layer][node]=1;
                    text +=" checked></td>";
                }
                else {
                    s[layer][node]=0;
                    text +="></td>"
                }
            }
            text += "</tr>"
        }
        document.getElementById("sig").innerHTML = text;
        s.push([0]);
        return s;
    }

    //creates an returns array which has the weights of the network also used to
generate array to store the slope of the weights.
    function create_weights(random){
        var w = [];
        for (layer = 0; layer < ann_size.length-1; layer++){
            w[layer] = [];
            for(node = 0; node < ann_size[layer+1]; node++){
                w[layer][node] = [];
                for(synapse = 0; synapse < ann_size[layer]; synapse++){
                    if (random){
                        w[layer][node][synapse] = R();
                    }
                    else{
                        w[layer][node][synapse] = 0;
```

```
                    }
                }
            }
        }
        return w;
    }

    //creates and returns array to store the network bias values also used to
generate array to store slope of the bias values.
    function create_bias(random){
        var b = [];
        for (layer = 0; layer < ann_size.length-1; layer++){
            b[layer] = [];
            for(node = 0; node < ann_size[layer+1]; node++){
                if (random){
                    b[layer][node] = R();
                }
                else{
                    b[layer][node] = 0;
                }
            }
        }
        return b;
    }

    //Creates and retunrs array which contains the node data.
    function create_neurons(for_slope){
        var n = [];
        for (layer = 0; layer < ann_size.length; layer++){
            n[layer] = [];
            for(node = 0; node < ann_size[layer]; node++){
                n[layer][node] = 0;
            }
        }
        if (for_slope){
            n.splice(0, 1);
        }
        return n;
    }

    //Creates or resets the network.
    function Network(){
        ann = create_neurons(false);
        weights = create_weights(true);
        bias = create_bias(true);
    }

    //Calculates the forward pass of the network
    function ForwardFeed(p){
        z=[p[0],Math.floor(p[1]/2),p[1]%2,Math.floor(p[2]/2),p[2]%2];
        for (var i=0; i < ann[0].length; i++){
            ann[0][i] = z[i];
        }
        for (layer = 0; layer < weights.length; layer++){
            for (node = 0; node < weights[layer].length; node++){
                ann[layer+1][node]=0;
                for (synapse = 0; synapse < weights[layer][node].length; synapse++){
                    ann[layer+1][node] +=
ann[layer][synapse]*weights[layer][node][synapse];
                }
                ann[layer+1][node] += bias[layer][node];
                if(sig[layer+1][node]==1){
                    ann[layer+1][node] = sigmoid(ann[layer+1][node]);
                }
            }
        }
    }
```

```
    //Training loop.
    function Train(){
        var ri = Math.floor(Math.random()*data.length);
        var slope_w = create_weights(false);
        var slope_b = create_bias(false);
        var ds;
        for (var i=0; i < points; i++){
            var slope_nn = create_neurons(true);
            if (ri+i>data.length-1){
                ri = -i
            }
            p = data[ri+i];
            ForwardFeed(p);
            pred = ann[ann.length-1][0];
            target = p[3]
            dcost = 2*(pred-target)
            for (x = slope_nn.length-1; x >= 0; x--){
                for(z = 0 ;  z < slope_nn[x].length; z++){
                    if (slope_nn.length-1 == x){
                        slope_nn[x][z] = dcost;
                    }
                    else{
                        for (y = 0; y < slope_nn[x+1].length; y++){
                            slope_nn[x][z] += slope_nn[x+1][y]*weights[x+1][y][z];
                        }
                    }
                    if(sig[x+1][z]==1){
                        slope_nn[x][z] *= dsigmoid(ann[x+1][z])
                    }
                }
            }
            for(x = 0; x < slope_nn; x++){
                for(z = 0; z < slope_nn[x]; z++){
                    slope_b[x][b] += slope_nn[x][b];
                }
            }

            for (x = weights.length-1; x >= 0; x--){
                for (z = 0; z < weights[x].length; z++){
                    for (y = 0; y < weights[x][z].length ; y++){
                        slope_w[x][z][y] += slope_nn[x][z]*ann[x][y];
                    }
                }
            }
        }
        for (x = weights.length-1; x >= 0; x--){
            for (z = 0; z < weights[x].length; z++){
                for (y = 0; y < weights[x][z].length ; y++){
                    weights[x][z][y] = weights[x][z][y] -
(learning_rate[sig[x+1][z]]*(slope_w[x][z][y]/points));
                }
                bias[x][z] = bias[x][z]-
(learning_rate[sig[x+1][z]]*(slope_b[x][z]/points));
            }
        }
    }

    //calculates the sum of the cost for every datapoint (currently not used)
    function Cost(){
        cost_sum=0;
        for (i=0; i<data.length-1; i++){
            ForwardFeed(data[i]);
            pred = ann[4][0];
            target = data[i][3];
            cost_sum += Math.pow(pred-target,2);
        }
```

```
        return cost_sum;
    }


    //Updates the canvas.
    function Draw(){
        //initialize canvas
        var c = document.getElementById("myCanvas");
        var ctx = c.getContext("2d");
        ctx.clearRect(0, 0, c.width, c.height);
        //variables for scaling data to the canvas
        var pixels = 250;
        var c_size = c.width;
        var pixel_size =  c_size / pixels ;
        var data_size =  pixels / 4;

        for (d=0; d<data.length; d++){
            var p = data[d];
            ctx.fillStyle = "#000000"
            ctx.fillRect(
                p[0]*data_size*pixel_size-1,
                (pixels-p[3]*data_size)*pixel_size-1,
                pixel_size+2,pixel_size+2);
        }
        for (l = 0; l < 4; l++){
            for(c = 0; c < 4; c++){
                for (s=0.2; s <= 3; s += 0.01 ){
                    ctx.fillStyle = "#ff0000"
                    ctx.fillRect(
                            s*data_size*pixel_size-1,
                            (pixels-Price(s,c,l)*data_size)*pixel_size-1,
                            pixel_size+2,pixel_size+2);

                    ForwardFeed([s,c,l]);

                    ctx.fillStyle = "#00ff00"
                    ctx.fillRect(
                            s*data_size*pixel_size-1,
                            (pixels-ann[ann.length-1][0]*data_size)*pixel_size-1,
                            pixel_size+2,pixel_size+2);
                }
            }
        }
    }

  </script>
</html>
```

Appendix 4

Breakout

```
<!DOCTYPE html>
<html>
<head>
<title>Breakout</title>
</head>
<body>
</body>
<script src="../phaser-master/v2/build/phaser.js"></script>
<script>
var game = new Phaser.Game(800, 600, Phaser.AUTO, 'phaser-example', { preload:
preload, create: create, update: update });

function preload() {
    game.load.spritesheet("button", "button.png", 120, 30);
    game.load.atlas('breakout', 'breakout.png', 'breakout.json');
}

//nexGen is how many of the best players are put to the next round
var nextGen = 5
//mutations is how many mutated versions of the best players are made
var mutations = 5
//how many random ANN players there are in every generation
var random = 10

//ANN size. Firs value is the number of inputs.
//Last value is the number of outputs
//numbers in the middle are the amount of nodes per hidden layer
var ann_size = [3,6,6,1];

//Global variables used for game control.
var showButton;
var showAll = true;
var addTrained;
var alive = 0;
var games = [];
var players = nextGen*mutations+nextGen+random;
var gameCycles = 0;
var generationText, fitnessText;
var gen = 0;

//pre-trained ANN
var trainedANN = {"node":[[0.029902748020109157,0.37,1.155],[0.831291905996752,-
0.9983800581432521,-0.9984428811409688,-
0.9999617744520164,0.9983698453878356,0.998460645474632],[0.9999999999999998,-
0.8803360268327359,-0.9998766436730712,-0.5924845163672232,-
0.9936970360448035,0.6054761370894707],[0.05520514998156249]],"bias":[[2.411007024398
5944,-0.7640663609458792,-1.0646188678353972,-
3.6570762420422778,4.0891932900071115,0.2678108954712675],[2.0933311221411985,-
4.297903619782945,-1.9472069471429005,5.65258598609732,-2.3675971830475784,-
2.7132076733281925],[-2.145548743316961]],"weight":[[[2.5045772338361703,-
0.9713768215367399,-0.8088206473839363],[-2.4851219577823866,-0.9088554982846978,-
2.064239030596589],[-1.6754918635328413,-2.0162249677482067,-1.4873759890189375],[-
0.33313466726293556,-1.0002639676249123,-1.2081659887305773],[-3.6347493346842574,-
1.4158790482730634,0.08578099052874952],[2.9145545889264084,2.1136396585455346,2.1189
38430970524]],[[3.8747100114655493,-1.7830131544637493,-5.546335551363569,-
3.7486397506611273,3.8566448240561355,-1.576734815618401],[-0.25329597467672393,-
1.368903229314551,3.3964444406856806,-3.2967480847436215,-
1.3910395986147253,3.2528346975819686],[-
3.077551663347056,4.268418407881842,3.7474377985898557,-
3.5054994784567604,3.6825901192301367,0.48068522196890184],[-4.568351242953575,-
0.14941008075413856,3.968290404024162,-4.889553837266608,-1.0354266322058425,-
2.583099693855981],[2.7440787705071235,0.867409056374516,2.028255664680055,-
3.36252282841232,0.8452251848557879,-4.1133685230024],[3.6635494501783823,-
4.64208967156704,-2.441377059156036,-1.1142827951538252,-3.883391068519417,-
```

```
3.9459224555704533]],[[0.18560165392243222,3.3563181242095514,-
0.9058757347361631,0.25094195161039223,-2.0867025213015347,3.533188470470847]]]}

//Phaser creates the game instance
function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);

    game.physics.arcade.checkCollision.down = false;

    for(i=0; i<players;i++){
        games.push(new createGame());
    }
    time = new Date();
    fitnessText = game.add.text(32, 550, 'Fittest: 0', { font: "20px Arial", fill:
"#ffffff", align: "left" });
    generationText = game.add.text(600, 550, 'Generation: 0', { font: "20px Arial",
fill: "#ffffff", align: "left" });

    showButton = game.add.button(game.world.centerX, 550,"button", show, this, 2, 1,
0);
    addTrained = game.add.button(200,550,"button",trained,this,8,7,6);

}

//Phaser game loop
function update() {
    //gameCycles works as a timer
    gameCycles++
    //Firs two game cyles is skipped
    if (gameCycles > 2){
        //update the game
        for (i in games){
            games[i].update();
        }
        //checks if there are still players alive
        var go = true;
        for (i in games){
            if (!games[i].dead){
                go = false;
                break;
            }
        }
        //if no player are alive or the game has ran 3min the game is reset and new
generation is created
        if (go || gameCycles > 10800){
            newGeneration();
            gen++
            generationText.text = "Generation: "+ gen;
        }
    }
    //if gameCycles is below 2 the game is reset
    //reset is done twice to make sure that the players start at the same position
    else{
        for (i in games){
            games[i].restart();
        }
        show();
        alive = 0;
    }
    //if only the best is shown and the first game is dead then the next game alive
is shown
    if (!showAll && games[alive].dead){
        for (i in games){
            if (!games[i].dead){
                alive = i;
                games[i].paddle.alpha = 1;
                games[i].ball.alpha = 1;
```

```
                games[i].bricks.alpha = 1;
                break;
            }
        }
    }
}

//Controlling the fuction of the only the best button.
function show(){
    if (showAll){
        showButton.setFrames(5,4,3);
        showAll = false;
        for (i in games){
            games[i].paddle.alpha = 0;
            games[i].ball.alpha = 0;
            games[i].bricks.alpha = 0;
        }
        games[alive].paddle.alpha = 1;
        games[alive].ball.alpha = 1;
        games[alive].bricks.alpha = 1;

    }
    else{
        showButton.setFrames(2,1,0);
        showAll = true;
        for (i in games){
            if (!games[i].dead){
                games[i].paddle.alpha = 1;
                games[i].ball.alpha = 1;
                games[i].bricks.alpha = 1;
            }
        }
    }
}

//overwrites the first players ANN with the pre trained network
//Setting the gameCycles to 0 restart the game
function trained(){
    gameCycles = 0;
    games[0].ann = JSON.parse(JSON.stringify(trainedANN));
}

//Creates a game instance
function createGame(){
    this.bricks = game.add.group();
    this.bricks.enableBody = true;
    this.bricks.physicsBodyType = Phaser.Physics.ARCADE;
    this.ballOnPaddle = true;
    this.score = 0;
    this.hit = 0;
    this.fitness = 0;
    this.ann = {node: createNeurons(), bias: createBias(), weight: createWeights()};
    this.dead = false;

    var brick;

    //create the bricks
    for (var y = 0; y < 4; y++)
    {
        for (var x = 0; x < 15; x++)
        {
            brick = this.bricks.create(120 + (x * 36), 100 + (y * 50), 'breakout',
'brick_' + (y+1) + '_1.png');
            brick.body.bounce.set(1);
            brick.body.immovable = true;
        }
    }
```

```
    //create the paddle
    this.paddle = game.add.sprite(game.world.centerX, 500, 'breakout',
'paddle_big.png');
    this.paddle.anchor.setTo(0.5, 0.5);

    game.physics.enable(this.paddle, Phaser.Physics.ARCADE);

    this.paddle.body.bounce.set(1);
    this.paddle.body.immovable = true;

    //create the ball
    this.ball = game.add.sprite(game.world.centerX, this.paddle.y - 16, 'breakout',
'ball_1.png');
    this.ball.anchor.set(0.5);
    this.ball.checkWorldBounds = true;

    game.physics.enable(this.ball, Phaser.Physics.ARCADE);

    this.ball.body.collideWorldBounds = true;
    this.ball.body.bounce.set(1);

    //if the ball hits a brick
    this.ballHitBrick = function(_ball, _brick) {
        //removes the brick that was hit
        _brick.kill();
        //adds one to the game score
        this.score++;
        //If no bricks are left the bricks are revived
        if (this.bricks.countLiving() == 0)
        {
            this.score += 50;
            this.ballOnPaddle = true;
            this.ball.body.velocity.set(0);
            this.releaseBall();
            this.bricks.callAll('revive');
        }
    }

    //Paddle movement to the left
    this.left = function(){
        this.paddle.body.velocity.x = -300;
    }

    //Paddle movement to the right
    this.right = function(){
        this.paddle.body.velocity.x = 300;
    }

    //Updates the gamestate and runs the ANN ai
    this.update = function(){
        this.paddle.body.velocity.x = 0; //stops the paddle motion

        //if this game is dead skip the update
        if (!this.dead){
            //Lets the paddle travel just ouside of the screen but no further
            if (this.paddle.x < -24)
            {
                this.paddle.x = -24;
            }
            else if (this.paddle.x > game.width + 24)
            {
                this.paddle.x = game.width + 24;
            }
            //If the ball is on the paddle reset the paddle postion and release the
ball
            if (this.ballOnPaddle)
```

```
            {
                this.ball.body.x = this.paddle.x;
                this.paddle.x = game.world.centerX;
                this.releaseBall();
            }
            //if the game is going calculate the physics
            else
            {
                game.physics.arcade.collide(this.ball, this.paddle,
this.ballHitPaddle, null, this);
                game.physics.arcade.collide(this.ball, this.bricks,
this.ballHitBrick, null, this);
            }

            //runs the ANN
            this.forwardFeed();

            //checks if the player should move or stay still
            if (this.ann.node[this.ann.node.length-1][0] > 0.25 ){
                this.left();
            }
            else if (this.ann.node[this.ann.node.length-1][0] < -0.25 ){
                this.right();
            }
        }
    }

    //if the ball is on the paddle the ball is released and the bricks are revived.
    this.releaseBall = function() {
        if (this.ballOnPaddle)
        {
            this.bricks.callAll('revive');
            this.ballOnPaddle = false;
            this.ball.reset(this.paddle.body.x + 16, this.paddle.y - 16);
            this.ball.body.velocity.y = -300;
            this.ball.body.velocity.x = 75;
        }
    }

    //If the ball is lost (drops off the screen)
    //Also hides the the game
    this.ballLost = function() {
        this.ballOnPaddle = true;
        this.ball.body.velocity.set(0);
        this.bricks.callAll('kill');
        this.paddle.alpha = 0;
        this.ball.alpha = 0;
        this.dead = true;
    }

    //when the ball hits the paddle.
    this.ballHitPaddle = function(){
        var diff = 0;

        //calcutates which way to bounce the ball.
        if (this.ball.x < this.paddle.x)
        {
            //Ball is on the left-hand side of the paddle
            diff = this.paddle.x - this.ball.x;
            this.ball.body.velocity.x = (-10 * diff);
        }
        else if (this.ball.x > this.paddle.x)
        {
            //Ball is on the right-hand side of the paddle
            diff = this.ball.x - this.paddle.x;
            this.ball.body.velocity.x = (10 * diff);
        }
```

```
        else
        {
            //Ball is perfectly in the middle
            this.ball.body.velocity.x = 0;
        }
        //adds one to the paddle and ball hit counter
        this.hit++
    }

    //Calculates the forward pass of the network
    this.forwardFeed = function(){
        this.ann.node[0][0] = (this.paddle.x-this.ball.x)/200;
        this.ann.node[0][2] = (this.ball.y*-1+500)/400;
        this.ann.node[0][1] = this.paddle.x/300;
        for (layer = 0; layer < this.ann.weight.length; layer++){
            for (node = 0; node < this.ann.weight[layer].length; node++){
                this.ann.node[layer+1][node]=0;
                for (synapse = 0; synapse < this.ann.weight[layer][node].length;
synapse++){
                    this.ann.node[layer+1][node] +=
this.ann.node[layer][synapse]*this.ann.weight[layer][node][synapse];
                }
                this.ann.node[layer+1][node] += this.ann.bias[layer][node];
                this.ann.node[layer+1][node] = TanH(this.ann.node[layer+1][node]);
            }
        }
    }

    //Calculates the fitness of the network
    this.fit = function(){
        if (this.hit < 100){
            this.fitness = this.score/(100-this.hit)+this.hit;
        }
        else{
            this.fitness = this.score+this.hit;
        }
    }

    //Restarts the game
    this.restart = function(){
        this.dead = false;
        this.ballOnPaddle = true;
        this.hit = 0;
        this.score = 0;
        this.fitness = 0;
        this.paddle.alpha=1;
        this.ball.alpha=1;
        this.paddle.body.velocity.set(0);
        this.ball.body.velocity.set(0);
        this.paddle.x = game.world.centerX;
    }

    this.ball.events.onOutOfBounds.add(this.ballLost, this);
}


//returns random number between -5 and 5
function R(){
    return Math.random()*10-5;
}

//Returns TanH for input x
function TanH(x){
    return (Math.exp(x)-Math.exp(-x))/(Math.exp(x)+Math.exp(-x))
}

//creates and returns network weight array with random values.
```

```
function createWeights(){
    var w = [];
    for (layer = 0; layer < ann_size.length-1; layer++){
        w[layer] = [];
        for(node = 0; node < ann_size[layer+1]; node++){
            w[layer][node] = [];
            for(synapse = 0; synapse < ann_size[layer]; synapse++){
                    w[layer][node][synapse] = R();
            }
        }
    }
    return w;
}

//creates and returns network bias array with random values.
function createBias(){
    var b = [];
    for (layer = 0; layer < ann_size.length-1; layer++){
        b[layer] = [];
        for(node = 0; node < ann_size[layer+1]; node++){
                b[layer][node] = R();
        }
    }
    return b;
}

//Creates and returns array for network nodes.
function createNeurons(){
    var n = [];
    for (layer = 0; layer < ann_size.length; layer++){
        n[layer] = [];
        for(node = 0; node < ann_size[layer]; node++){
            n[layer][node] = 0;
        }
    }
    return n;
}

//Makes a new generation of ANNs
function newGeneration(){
    //calculate the fitness of the networks
    for (i in games){
        games[i].fit();
    }
    gameCycles=0;

    //Sorts the games from best to worse
    games.sort(function(a, b){return b.fitness - a.fitness});

    //Creates mutated players
    for (y = 0; y < nextGen;y++){
        for (x = 0; x < mutations; x++){
            var i = nextGen+y*mutations+x
            games[i].ann = JSON.parse(JSON.stringify(games[y].ann));
            games[i].ann = createChild(games[i].ann);
        }
    }

    //adds players with random ANNs
    for (i = 0; i < random; i++){
        i = parseInt(i);
        games[(i+players-random)].ann = {node: createNeurons(), bias: createBias(),
weight: createWeights()};
    }
    //updates the fitness text
    fitnessText.text = "Fittest: " + games[0].fitness.toFixed(2);
}
```

```
//returns a mutated version of the input ANN (object)
function createChild(ann){
    for(layer in ann.weight){
        for (node in ann.weight[layer]){
            for (synapse in ann.weight[layer][node]){
                if (ann.weight[layer][node][synapse] < 1 &&
ann.weight[layer][node][synapse] > -1){
                    ann.weight[layer][node][synapse] += Math.random()*0.1-0.05
                }
                else{
                    ann.weight[layer][node][synapse] *= Math.random()*0.2+0.90
                }
            }
            if (ann.bias[layer][node] < 1 && ann.bias[layer][node] > -1){
                ann.bias[layer][node] += Math.random()*0.1-0.05
            }
            else{
                ann.bias[layer][node] *= Math.random()*0.2+0.90
            }
        }
    }
    return ann;
}
</script>
</html>
```