The neural network This section is based on information from the book AI techniques for game programming[?] chapter 7, 8 & 9. Also from the book Neural networks: a systematic introduction[?] chapter 7.

Neural networks are models that imitate the brains behaviour. They have been created as an option to model artificial intelligence and analyse machine learning. The human brain is made up of billions of neurons that are interconnected in a big grid. They communicate by firing electrical shocks through the network of neurons. The human brain is extremely complex and can calculate vast amounts of data in no time. This is why scientist and mathematicians have been trying to emulate this behaviour to create artificial intelligence. Artificial neural networks (ANN) are artificial neurons (nodes) that are connected in a network. The network consists of an arbitrary number of layers that are interconnected. The most common structures in these networks are a feed forward structure. This kind of structure has the characteristic that it only flows data from the input layer through the layers to the output layer. There are no loops in the network thus making it unable to reiterate any information. Normally all of the nodes in the input layer connect to all of the nodes in the second layer. The same connections are done in the next layers until we hit the output layer. This will give us he sum of all the previous nodes(x_i) and their weights(w_i)($\sum_{i=0}^{i=n+1} w_i x_i$) in every node in the next layer:

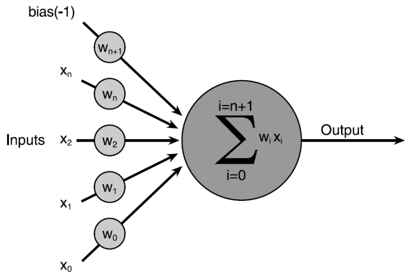

Figure 1: How the weight is calculated from one layer to the next

All of these connections carry a weight that dictates how data flows through the network and reflects the relations between the inputs and the outputs of the network. The inputs of the network should be all the factors that has an influence on the output we want from the network. In our example we would want to input; weather data, temperature, demand, availability [?] to get the price as an output from the network. Every node in our network contains an activation function. This function, when calculated, tells us whether the artificial neuron should fire or not. That is if the neuron should transmit the data from the current layer to the next layer. There are a lot of different activation functions. The simplest form is the binary step function which means that it either fires or it does not. This depends on the input and gives us a low threshold of complexity which is good in simple neural networks as the relations between the nodes does not need to be that fine grained. In more complex systems we want activation functions with a broader output range than binary. In many cases a sigmoid function is used as the activation function. This is because of the S-shape which enables it to compute outputs in a non-linear way. The non-linear nature of the sigmoid functions is what makes the neural network able to compute non-trivial problems in reasonable sized networks. To be able to calculate a non-trivial problem in these kinds of networks we need what is called a training algorithm. This algorithm depicts how the network evolves over time also known as learning. There are two kinds of learning; supervised learning and unsupervised learning.

0.1 Unsupervised learning

Der skal skrives noget her.

0.2 Supervised learning

Supervised learning is an algorithm that uses a dataset that contains both the inputs and what the output is expected to be. This dataset is used to train the neural network to make it able to do calculations on data and predict the outcome. An example of an algorithm used for supervised learning is the back-propagation algorithm. It starts out by randomly assigning all the weights on the connections between the neurons. It then calculates the output of the network and compares it to the expected output. From that it calculates the error margin between the expected and the calculated output and adjusts the weights accordingly. This is done for all the hidden layers as well until we hit the input layer. All of these steps are called an epoch. We will repeat as many epochs as we need until the sum of all the errors are within a given threshold. The name of the algorithm originated from this approach where it propagates the error backwards in the network.

Supervised learning can be thought of as learning with a teacher. As an example we can use the XOR table:

Input #1	Input #2	Output
0	0	0
1	0	1
0	1	1
1	1	0

Table 1: This training set is very simple yet it illustrates a training set for a supervised learning algorithm very well.

[FORKLAR MATEMATIKKEN EVT MED EKSEMPEL s. 332 i bogen]

0.3 Problems to be avoided

When we are trying to fit out algorithm and make it recognize patterns we will encounter several possible pitfalls. First of all there is the chance of ending up in a local minima. This is when the the backpropagation algorithm attempts to find the global minimum of the error curve, thus having reduce the error as much as possible. The algorithm works by trying to reduce this error margin a little step at a time. If it encounters a local minima on the curve and thinks it has reached the global minima it gets stuck and we will get inaccurate results. [BILLEDE AF LOCAL/GLOBAL MINIMUM] To avoid the backpropagation algorithm to falsely accept a local minima as the global minima we can give the algorithm momentum. This is done by adding a bit of the last error correction from the earlier layer to the next layers error correction. This way the algorithm, so to say, will scoot right by any small deviations in the error correction face.
Another pitfall when working with neural networks is over fitting the algorithm. This is when the algorithm instead of finding a generalized pattern in the inputs it will find an over-fit pattern that will fit exactly that input. This is better shown in figure 7. This can be avoided by some simple techniques. First of all we

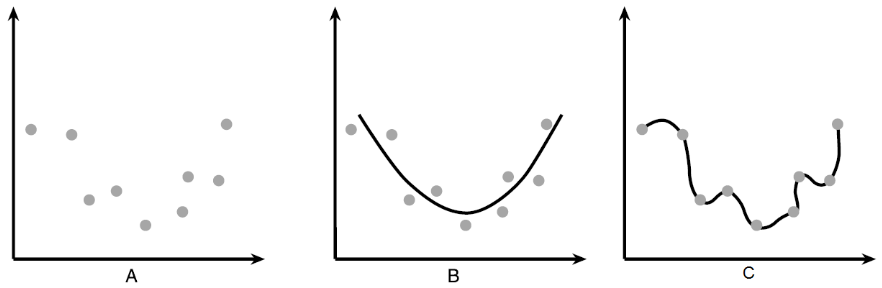

Figure 2: A. The plot graph of the input B. The generalized function C. An over-fit function

want to reduced the neurons as much as possible as long as it does not interfere with our performance of the system. This is a trial and error problem and has to be tweaked along with evolving your neural network. We can add noise to avoid this problem. By adding noise(random data values) we prevent the algorithm from fitting the function to closely to the given data. Thus giving us a more generalized function where it hopefully will be able to fit new data presented to it better. Early stopping is another method to avoid over-fitting. This is only doable with large datasets where you can split it into two equal datasets. The first will work as a training set and the second will work as a validation set. We will keep training the dataset and checking with the validation set until the difference between those two start to increase.