Neural Nets

There are a large number of different types of networks, but they all are characterized by the following components: a set of nodes, and connections between nodes.

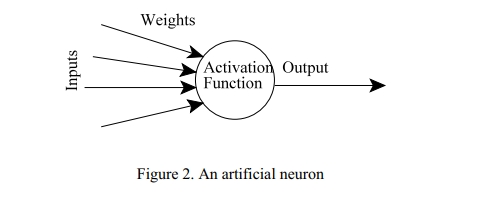
The nodes can be seen as computational units. They receive inputs, and process them to obtain an output. This processing might be very simple (such as summing the inputs), or quite complex (a node might contain another network...)

The connections determine the information flow between nodes. They can be unidirectional, when the information flows only in one sense, and bidirectional, when the information flows in either sense.

The interactions of nodes though the connections lead to a global behaviour of the network, which cannot be observed in the elements of the network. This global behaviour is said to be emergent. This means that the abilities of the network supercede the ones of its elements, making networks a very powerful tool.

One type of network sees the nodes as ‘artificial neurons’. These are called artificial neural networks (ANNs). An artificial neuron is a computational model inspired in the natural neurons. Natural neuronsreceive signals through synapses located on the dendrites or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal though the axon. This signal might be sent to another synapse, and might activate other neurons.

The complexity of real neurons is highly abstracted when modelling artificial neurons. These basically consist of inputs (like synapses), which are multiplied by weights (strength of the respective signals), and then computed by a mathematical function which determines the activation of the neuron. Another function (which may be the identity) computes the output of the artificial neuron (sometimes in dependance of a certain threshold). ANNs combine artificial neurons in order to process information.

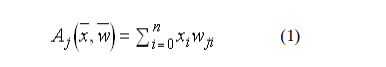


The higher a weight of an artificial neuron is, the stronger the input which is multiplied by it will be. Depending on the weights, the computation of the neuron will be different. By adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. But when we have an ANN of hundreds or thousands of neurons, it would be quite complicated to find by hand all the necessary weights. But we can find algorithms which can adjust the weights of the ANN in order to obtain the desired output from the network. This process of adjusting the weights is called ***learning or training.***

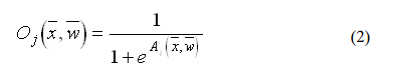
The Backpropagation Algorithm

The backpropagation algorithm (Rumelhart and McClelland, 1986) is used in layered feed-forward ANNs. This means that the artificial neurons are organized in layers, and send their signals “forward”, and then the errors are propagated backwards. The network receives inputs by neurons in the input layer, and the output of the network is given by the neurons on an output layer. There may be one or more intermediate hidden layers. The backpropagation algorithm uses supervised learning, which means that we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. The idea of the backpropagation algorithm is to reduce this error, until the ANN learns the training data. The training begins with random weights, and the goal is to adjust them so that the error will be minimal.

The activation function of the artificial neurons in ANNs implementing the backpropaga i tion algorithm is a weighted sum (the sum of the inputs x multiplied by their ji respective weights w ):



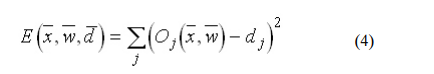
We can see that the activation depends only on the inputs and the weights. If the output function would be the identity (output=activation), then the neuron would be called linear. But these have severe limitations. The most common output function is the sigmoidal function:



The sigmoidal function is very close to one for large positive numbers, 0.5 at zero, and very close to zero for large negative numbers. This allows a smooth transition between the low and high output of the neuron (close to zero or close to one). We can see that the output depends only in the activation, which in turn depends on the values of the inputs and their respective weights. Now, the goal of the training process is to obtain a desired output when certain inputs are given. Since the error is the difference between the actual and the desired output, the error depends on the weights, and we need to adjust the weights in order to minimize the error. We can define the error function for the output of each neuron:



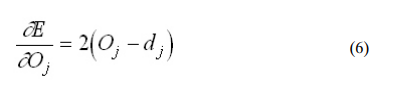
We take the square of the difference between the output and the desired target because it will be always positive, and because it will be greater if the difference is big, and lesser if the difference is small. The error of the network will simply be the sum of the errors of all the neurons in the output layer:



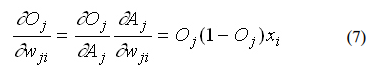
The backpropagation algorithm now calculates how the error depends on the output, inputs, and weights. After we find this, we can adjust the weights using the method of gradient descendent:



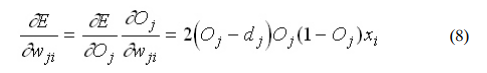
This formula can be interpreted in the following way: the adjustment of each weight (Δwij) will be the negative of a constant eta (η) multiplied by the dependance of the pre i vious weight on the error of the network, which is the derivative of E in respect to w . The size of the adjustment will depend on 0, and on the contribution of the weight to the error of the function. This is, if the weight contributes a lot to the error, the adjustment will be greater than if it contributes in a smaller amount. (5) is used until we find appropriate weights (the error is minimal). If you do not know derivatives, don’t worry, you can see them now as functions that we will replace right away with algebraic expressions. If you understand derivatives, derive the expressions yourself and compare your results with the ones presented here. If you are searching for a mathematical proof of the backpropagation algorithm, you are advised to check it in the suggested reading, since this is out of the scope of this material. So , we “only” need to find the derivative of E in respect to wji . This is the goal of the backpropagation algorithm, since we need to achieve this backwards. First, we need to calculate how much the error depends on the output, which isthe derivative of E in respect to Oj (from (3)).



And then, how much the output depends on the activation, which in turn depends on the weights (from (1) and (2)):



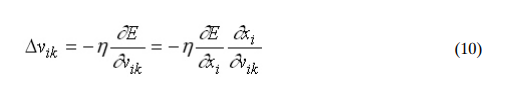
And we can see that (from (6) and (7)):



And so, the adjustment to each weight will be (from (5) and (8)):



We can use (9) as it is for training an ANN with two layers. Now, for training the network with one more layer we need to make some considerations. If we want to adjust the weights (let’s call them vjk ) of a previous layer, we need first to calculate how the error depends not on the weight, but in the input from the previous layer. This is easy, we would i ji just need to change x with w in (7), (8), and (9). But we also need to see how the error of the network depends on the adjustment of vjk . So:



Where:



And, assuming that there are inputs uk into the neuron with vjk (from (7)):



If we want to add yet another layer, we can do the same, calculating how the error depends on the inputs and weights of the first layer. We should just be careful with the indexes, since each layer can have a different number of neurons, and we should not confuse them. For practical reasons, ANNs implementing the backpropagation algorithm do not have too many layers, since the time for training the networks grows exponentially. Also, there are refinements to the backpropagation algorithm which allow a faster learning.