# Introduction

In large neural networks simulating more complex functions, finding the weights of each synapse is too complex to be done manually, instead the back propagation algorithm is commonly used to find weights which will approximate the desired behaviour.

Back propagation works by calculating the difference between each output of the network and the expected output for the inputs to the network, then propagating this error backwards through the network to adjust the weights and reduce the error in the future. By repeating this train and test loop over a large range of valid inputs, the neural network will “learn” the expected behaviour and the errors will decrease. Back propagation can be run for any number of valid inputs, however the network will eventually reach a state where back propagation can no longer reduce the average error.

In some cases, the inputs used to train the network can be picked randomly and the expected output can be calculated directly from this, e.g. a network trained to add numbers or apply the sin function to its input. Other cases require the training examples to be pre computed, for example, in training a network to recognise the character in an image, the character in each image will have to be manually given as the expected output.

This graph shows how the average total error of a network decreases as back propagation teaches the network using 50,000 examples. The network trained is a two input and two output network with 40 neurons in a hidden layer used to convert Cartesian coordinates to polar coordinates.

# The Algorithm

## Initialisation

Before the back propagation is run, the weights all synapses in the network are initialised to small random values, usually between -1 and 1. The range of the initial values is not important so long as the weights are not all equal, in this case, back propagation will change each weight by the same amount and the behaviour of the network will never change significantly.

## Train Test Loop

Once all weights are initialised, apply an input in the training examples to the network and measure the difference between each of the actual outputs and the expected outputs of the network. Then for each synapse between the last hidden layer and the outputs, calculate the product of the learning rate, the output of the hidden neuron and the error for the output and add the result to the weight of the synapse. Next

backpropagation()

for o = 0 to length(outputs) do:

for m = 0 to length(medial\_neurons) do:

synTwo[m, o] := synTwo[m, o] + learning\_rate \* medial\_out[m] \* errors[o]

endfor

endfor

# Limitations

Searches for local, not global minimums.

Momentum

# Improving Results

Increasing number of neurons in a hidden layer does not necessarily reduce the error at which the network settles, the network can in fact perform worse if there are too many neurons in a hidden layer.

Use the implementation to create graphs of the average error for some function with varying numbers of hidden layers and neurons in each hidden layer.

# References