# 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 accomplishes this by calculating the difference between each output in the network and the expected output for a given input, then propagating this error backwards through the network while adjusting the weights to reduce the error in the future. By applying back propagation many times over the range of valid inputs, the neural network will “learn” the expected behaviour and the errors will decrease. This train and test loop can be run for any length of time, however the network will eventually reach a state where back propagation can no longer reduce the average error.

# The Algorithm

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