Neural Networks Videos

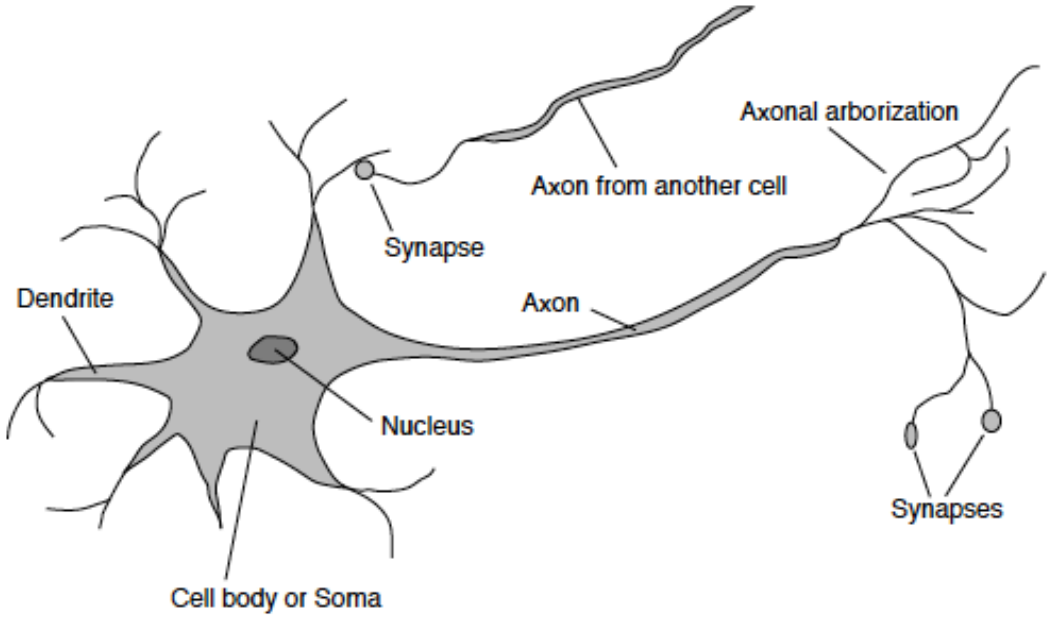
Brief Review

The Next Generation Neural Networks - Geoff Hinton

Neural Networks – Brief Review

Brains

10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential

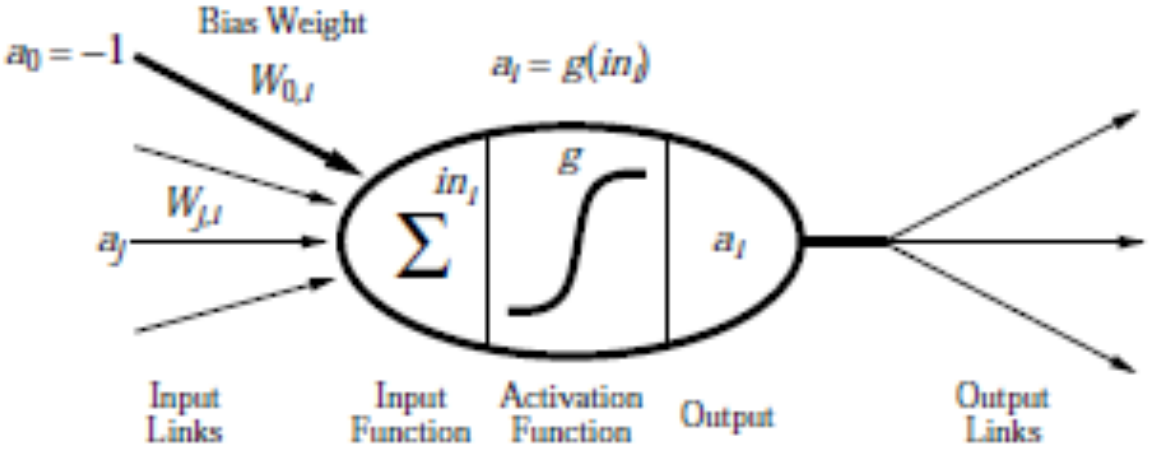


Neural Networks – Brief Review

McCulloch–Pitts "unit"

Output is a "squashed" linear function of the inputs:

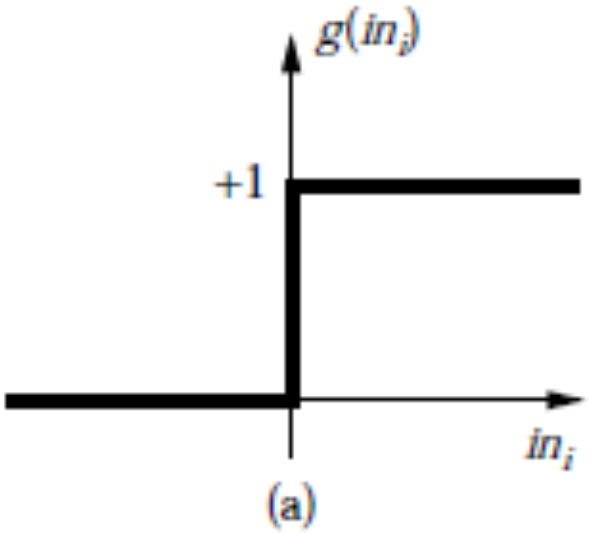
$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$

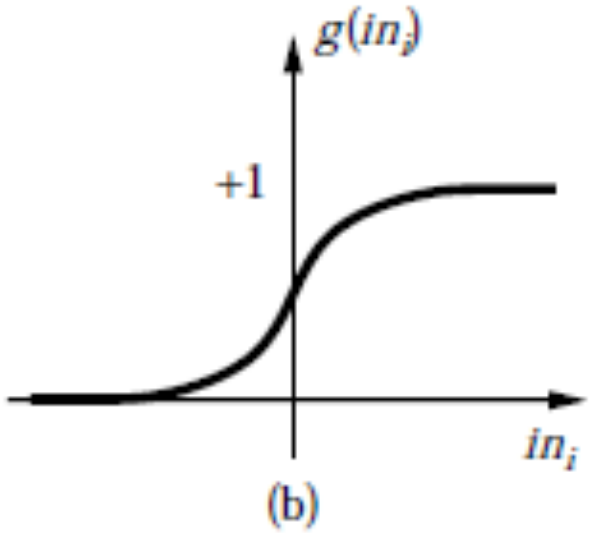


A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Neural Networks – Brief Review

Activation functions





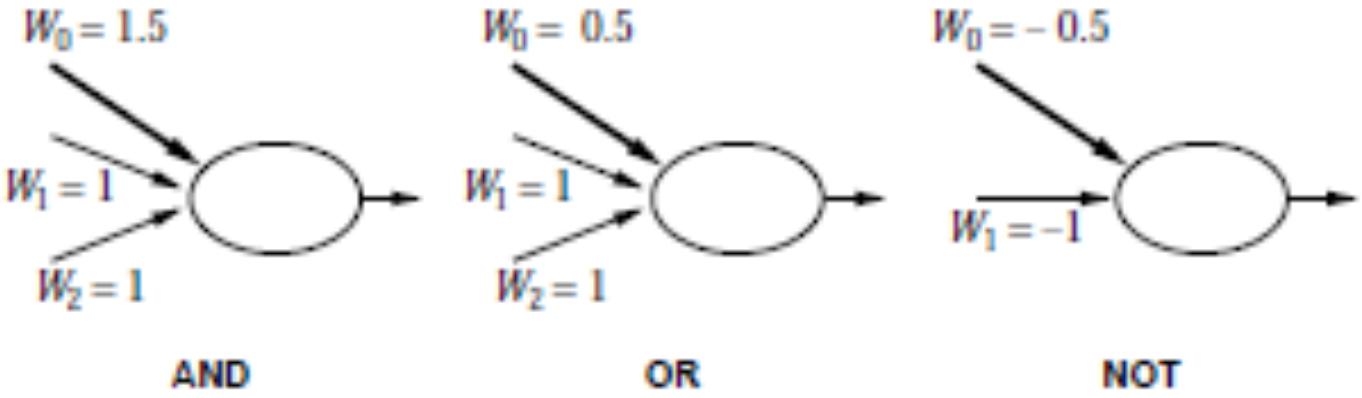
(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Neural Networks – Brief Review

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

Neural Networks – Brief Review

Network structures

Feed-forward networks:

- single-layer perceptrons
- multi-layer perceptrons

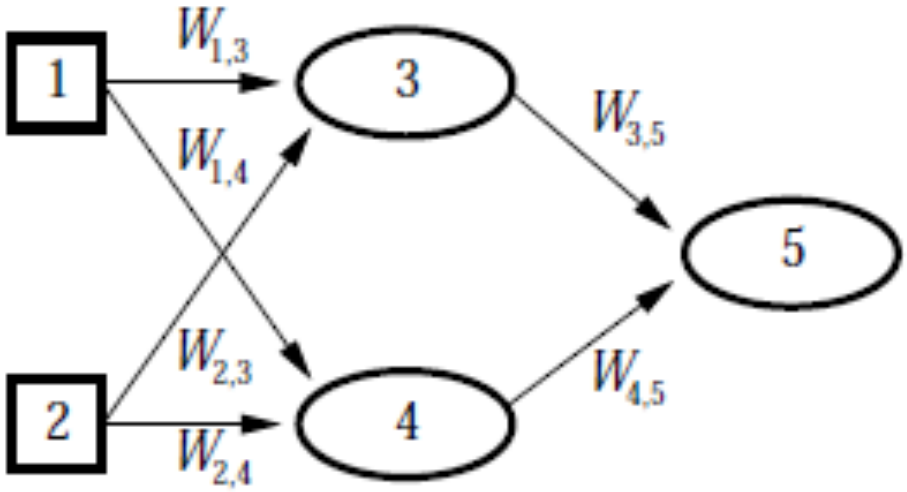
Feed-forward networks implement functions, have no internal state

Recurrent networks:

- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  $g(x) = \text{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**
- Boltzmann machines use stochastic activation functions,
  $\approx$ MCMC in Bayes nets
- recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

Neural Networks – Brief Review

Feed-forward example



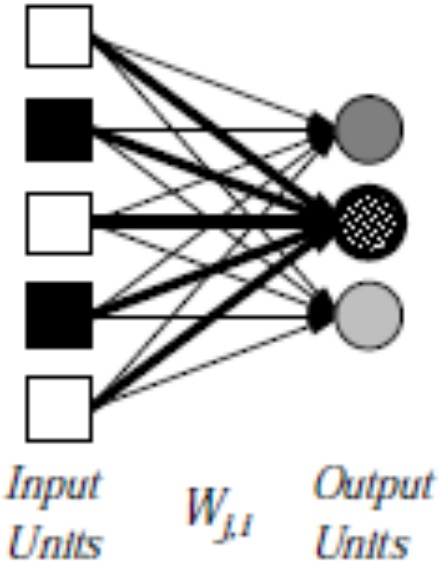
Feed-forward network = a parameterized family of nonlinear functions:

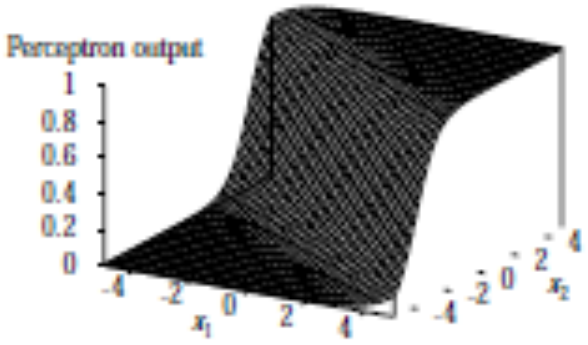
$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Neural Networks – Brief Review

Single-layer perceptrons





Output units all operate separately—no shared weights

Adjusting weights moves the location, orientation, and steepness of cliff

Neural Networks – Brief Review

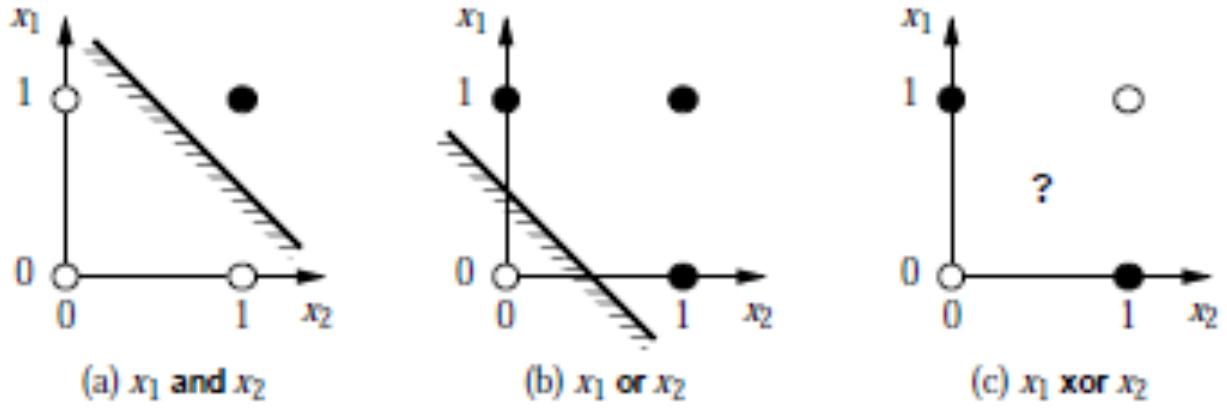
Expressiveness of perceptrons

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$\sum_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Minsky & Papert (1969) pricked the neural network balloon

Neural Networks – Brief Review

Perceptron learning

Learn by adjusting weights to reduce error on training set

The squared error for an example with input $\mathbf{x}$ and true output y is

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

Perform optimization search by gradient descent:

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\Sigma_{j=0}^{n} W_j x_j))$$
$$= -Err \times g'(in) \times x_j$$

Simple weight update rule:

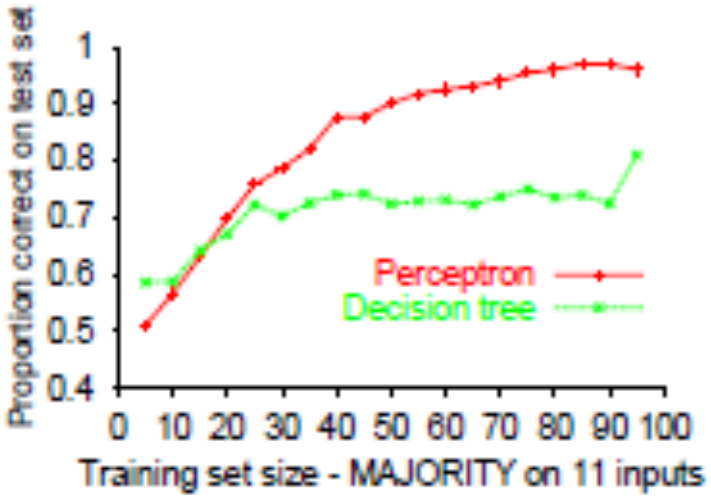
$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

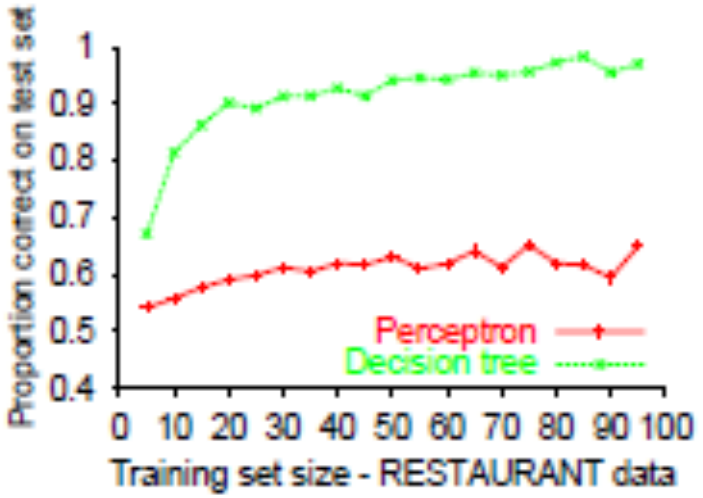
E.g., +ve error $\Rightarrow$ increase network output
$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs

Neural Networks – Brief Review

Perceptron learning contd.

Perceptron learning rule converges to a consistent function
for any linearly separable data set





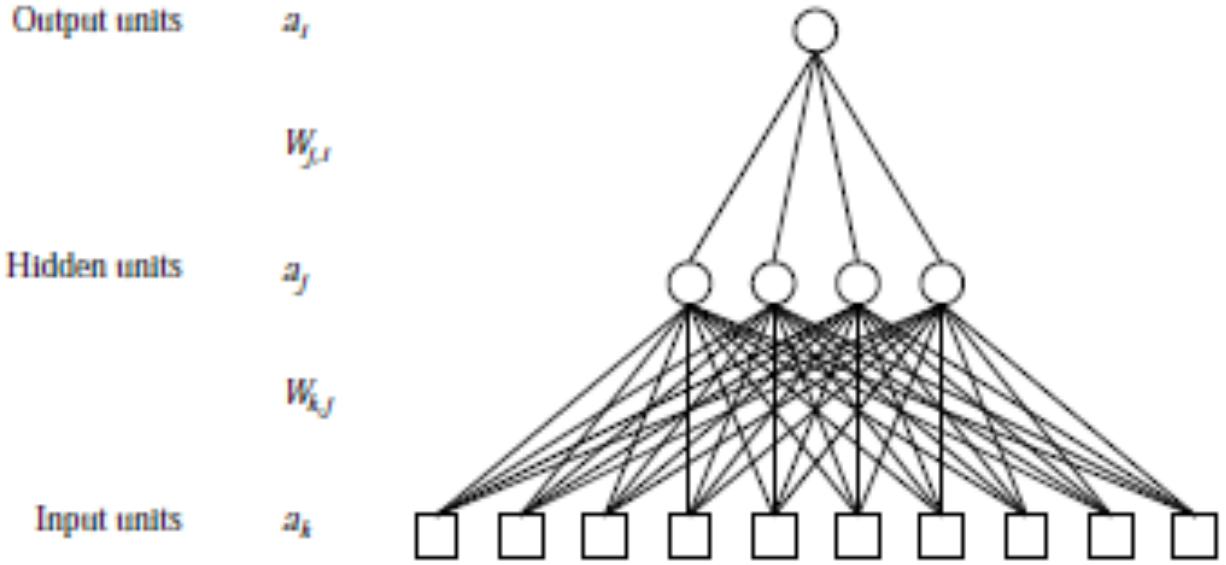
Perceptron learns majority function easily, DTL is hopeless

DTL learns restaurant function easily, perceptron cannot represent it

Neural Networks – Brief Review

Multilayer perceptrons

Layers are usually fully connected;
numbers of hidden units typically chosen by hand



Neural Networks – Back Propagation Algorithm

Most people would consider the Back Propagation network to be the quintessential Neural Net. Actually, Back Propagation is the training or learning algorithm rather than the network itself. Let's consider what Back Propagation is and how to use it.

A Back Propagation network learns by example. You give the algorithm examples of what you want the network to do and it changes the network's weights so that, when training is finished, it will give you the required output for a particular input. Back Propagation networks are ideal for simple Pattern Recognition and Mapping Tasks. To train the network you need to give it examples of what you want – the output you want (called the *Target*) for a particular input.

Neural Networks – Back Propagation Algorithm

Once the network is trained, it will provide the desired output for any of the input patterns. Let's now look at how the training works.

The network is first initialised by setting up all its weights to be small random numbers − say between −1 and +1. Next, the input pattern is applied and the output calculated (this is called the *forward pass*). The calculation gives an output which is completely different to what you want (the Target), since all the weights are random. We then calculate the *Error* of each neuron, which is essentially: *Target - Actual Output* (i.e., What you want − What you actually get). This error is then used mathematically to change the weights in such a way that the error will get smaller. In other words, the Output of each neuron will get closer to its Target (this part is called the *reverse pass*). The process is repeated again and again until the error is minimal.

Neural Networks – Back Propagation Algorithm

Let's do an example with an actual network to see how the process works. We'll just look at one connection initially, between a neuron in the output layer and one in the hidden layer.

Single connection learning in
a Back Propagation network.

The connection we're interested in is between neuron A (a hidden layer neuron) and neuron B (an output neuron) and has the weight W_{AB}. The diagram also shows another connection, between neuron A and C, but we'll return to that later. The algorithm works like this:

Neural Networks – Back Propagation Algorithm

1. First apply the inputs to the network and work out the output – remember this initial output could be anything, as the initial weights were random numbers.
2. Next work out the error for neuron B. The error is *What you want – What you actually get*, in other words:

$$Error_B = Output_B (1-Output_B)(Target_B - Output_B)$$

The "*Output(1-Output)*" term is necessary in the equation because of the Sigmoid Function – if we were only using a threshold neuron it would just be *(Target – Output)*.

3. Change the weight. Let $W+_{AB}$ be the new (trained) weight and W_{AB} be the initial weight.

$$W+_{AB} = W_{AB} + (Error_B \times Output_A)$$

Notice that it is the output of the connecting neuron (neuron A) we use (not B). We update all the weights in the output layer in this way.

Neural Networks – Back Propagation Algorithm

4. Calculate the Errors for the hidden layer neurons. Unlike the output layer we can't calculate these directly (because we don't have a Target), so we *Back Propagate* them from the output layer (hence the name of the algorithm). This is done by taking the Errors from the output neurons and running them back through the weights to get the hidden layer errors. For example if neuron A is connected as shown to B and C then we take the errors from B and C to generate an error for A.

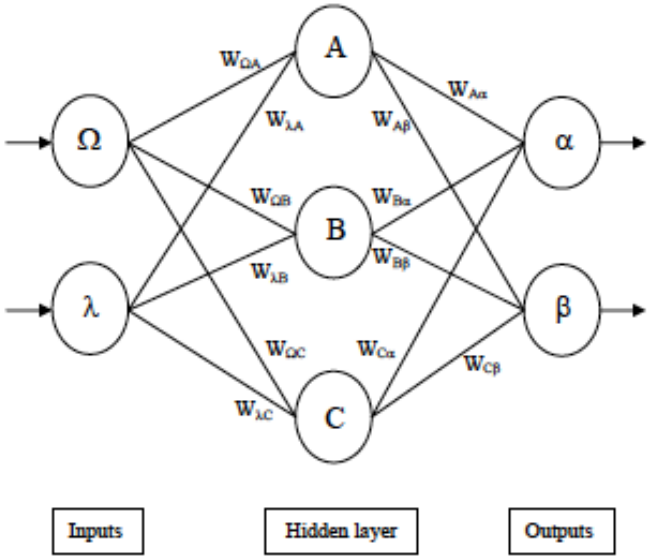
$$Error_A = Output_A (1 - Output_A)(Error_B W_{AB} + ErrorC W_{AC})$$

Again, the factor "*Output (1 - Output)*" is present because of the sigmoid squashing function.

5. Having obtained the Error for the hidden layer neurons now proceed as in stage 3 to change the hidden layer weights. By repeating this method we can train a network of any number of layers.

Neural Networks – Back Propagation Algorithm

This may well have left some doubt in your mind about the operation, so let's clear that up by explicitly showing *all* the calculations for a full sized network with 2 inputs, 3 hidden layer neurons and 2 output neurons as shown below. W+ represents the new, recalculated, weight, whereas W represents the old weight.



Neural Networks – Back Propagation Algorithm

1. Calculate errors of output neurons

$$\delta_\alpha = out_\alpha\,(1 - out_\alpha)\,(Target_\alpha - out_\alpha)$$
$$\delta_\beta = out_\beta\,(1 - out_\beta)\,(Target_\beta - out_\beta)$$

2. Change output layer weights

$$W^+_{A\alpha} = W_{A\alpha} + \eta\delta_\alpha\,out_A \qquad W^+_{A\beta} = W_{A\beta} + \eta\delta_\beta\,out_A$$
$$W^+_{B\alpha} = W_{B\alpha} + \eta\delta_\alpha\,out_B \qquad W^+_{B\beta} = W_{B\beta} + \eta\delta_\beta\,out_B$$
$$W^+_{C\alpha} = W_{C\alpha} + \eta\delta_\alpha\,out_C \qquad W^+_{C\beta} = W_{C\beta} + \eta\delta_\beta\,out_C$$

3. Calculate (back-propagate) hidden layer errors

$$\delta_A = out_A\,(1 - out_A)\,(\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$$
$$\delta_B = out_B\,(1 - out_B)\,(\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$$
$$\delta_C = out_C\,(1 - out_C)\,(\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$$

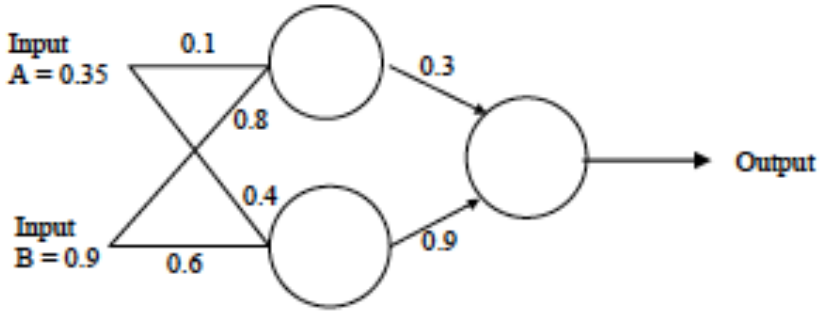
4. Change hidden layer weights

$$W^+_{\lambda A} = W_{\lambda A} + \eta\delta_A\,in_\lambda \qquad W^+_{\Omega A} = W^+_{\Omega A} + \eta\delta_A\,in_\Omega$$
$$W^+_{\lambda B} = W_{\lambda B} + \eta\delta_B\,in_\lambda \qquad W^+_{\Omega B} = W^+_{\Omega B} + \eta\delta_B\,in_\Omega$$
$$W^+_{\lambda C} = W_{\lambda C} + \eta\delta_C\,in_\lambda \qquad W^+_{\Omega C} = W^+_{\Omega C} + \eta\delta_C\,in_\Omega$$

The constant η (called the learning rate, and nominally equal to one) is put in to speed up or slow down the learning if required.

Neural Networks – Back Propagation Algorithm - worked example

Consider the simple network below:



Assume that the neurons have a Sigmoid activation function and

(i) Perform a forward pass on the network.
(ii) Perform a reverse pass (training) once (target = 0.5).
(iii) Perform a further forward pass and comment on the result.

Answer:
(i)
Input to top neuron = (0.35x0.1)+(0.9x0.8)=0.755. Out = 0.68.
Input to bottom neuron = (0.9x0.6)+(0.35x0.4) = 0.68. Out = 0.6637.
Input to final neuron = (0.3x0.68)+(0.9x0.6637) = 0.80133. Out = 0.69.

(ii)
Output error δ=(t-o)(1-o)o = (0.5-0.69)(1-0.69)0.69 = -0.0406.

Neural Networks – Back Propagation Algorithm - worked example

New weights for output layer

$w1^{+} = w1+(\delta \text{ x input}) = 0.3 + (-0.0406\text{x}0.68) = 0.272392.$

$w2^{+} = w2+(\delta \text{ x input}) = 0.9 + (-0.0406\text{x}0.6637) = 0.87305.$

Errors for hidden layers:

$\delta 1 = \delta \text{ x } w1 = -0.0406 \text{ x } 0.272392 \text{ x } (1\text{-}o)o = -2.406\text{x}10^{-3}$

$\delta 2 = \delta \text{ x } w2 = -0.0406 \text{ x } 0.87305 \text{ x } (1\text{-}o)o = -7.916\text{x}10^{-3}$

New hidden layer weights:

$w3^{+} = 0.1 + (-2.406 \text{ x } 10^{-3} \text{ x } 0.35) = 0.09916.$

$w4^{+} = 0.8 + (-2.406 \text{ x } 10^{-3} \text{ x } 0.9) = 0.7978.$

$w5^{+} = 0.4 + (-7.916 \text{ x } 10^{-3} \text{ x } 0.35) = 0.3972.$

$w6^{+} = 0.6 + (-7.916 \text{ x } 10^{-3} \text{ x } 0.9) = 0.5928.$

(iii)

Old error was -0.19. New error is -0.18205. Therefore error has reduced.

Neural Networks – Back Propagation Algorithm

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Neural Networks – Geoff Hinton's Lecture

Please view the 59 minute video at

http://www.youtube.com/watch?v=AyzOUbkUf3M

Concluding Remarks

T. T. T.
Put up in a place
where it's easy to see
the cryptic admonishment
T. T. T.
When you feel how
Depressingly slowly you climb,
it's well to remember that
Things Take Time.