The Backpropogated Delta Rule.

Although the Perceptron Learning Rule and the Delta Rule work, and can learn associations (PLR) or minimise mean squared error (DR), both are limited in what they can achieve by the single layer architecture they work in.

The PLR can learn linearly seperable classifications.

The DR can produce a mapping which minimises the error:
but the error is almost certain to remain non-zero
because of the limited range of functions possible

It has long been known that general networks of units provided a much richer computation capacity: but the absence of applicable learning rules had made these networks little more than an intellectual curiosity. This changed with the discovery of some learning rules applicable to more complex networks. We will discuss the Backpropogated Delta rule (BP) here: we will discuss others later (specifically the Boltzmann machine, and some extensions to BP).

The Backpropagated Delta rule (BP)

BP is an extension to the Delta rule.

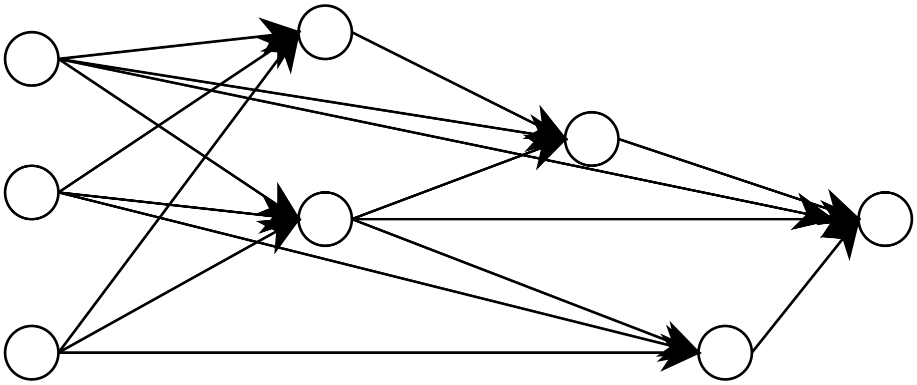
The use of differentiable output functions is crucial to its application.

(This was perhaps one reason that Minsky and Papert failed to find an extension to the PLR: no-one has yet managed to extend the (very general) PLR to multiple layers).

Though the term backpropagation was coined about 1986, the algorithm was disovered by Werbos in 1974.
But its importance was not realised, and it was rediscovered in 1985:
by Parker, Le Cun, and Rumelhart, Hinton, and Williams.

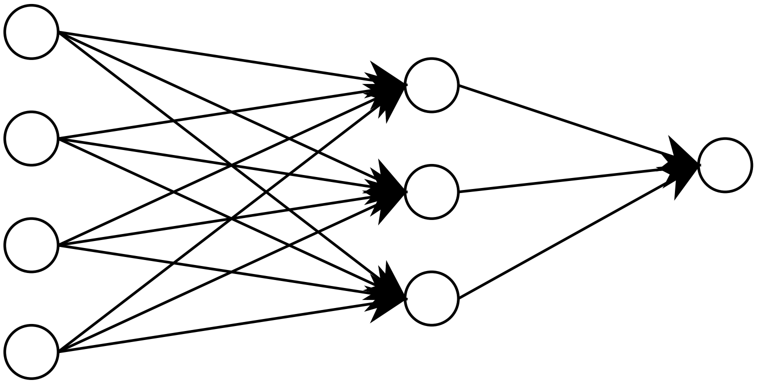
Architecture:

BP is not applicable to a general network: it requires a feedforward network.



General Feedforward Network.

In fact it is nearly always applied to a layered feedforward network.



Simple layered feedforward network.

Usually, the architecture is that of a *totally connected layered feedforward network*:

That is, each node in layer X (where X=0 means inputs, X=1 is first hidden layer of units, etc.) is connected to all the units in layer X+1.

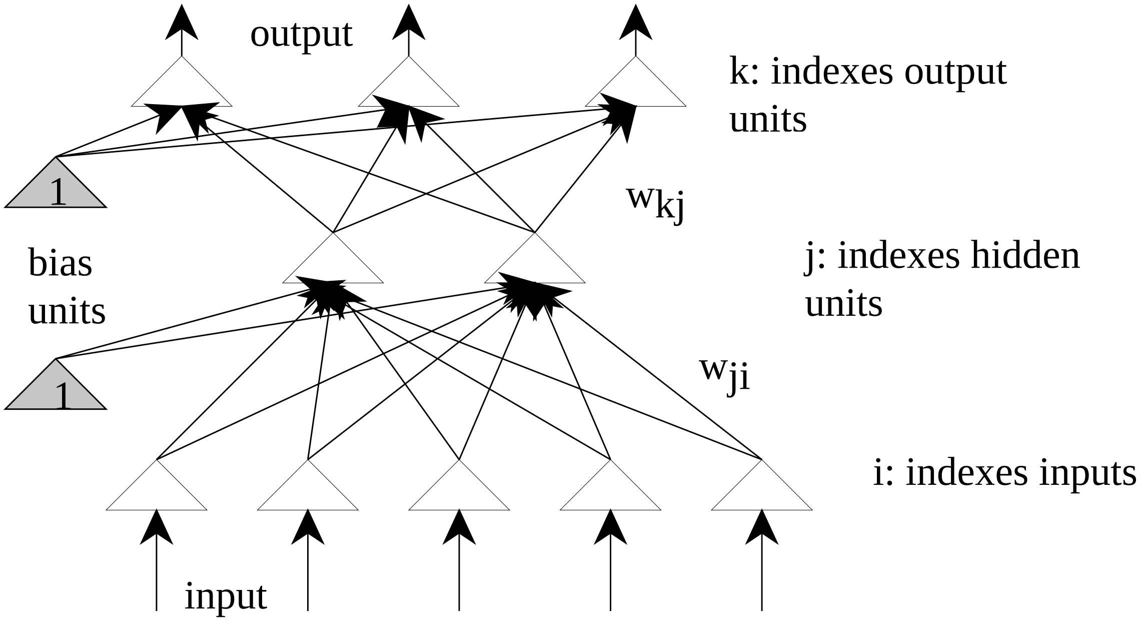
In fact, the proof simply requires the network to be loop free.

The units used cannot simply be linear because the combination of a set of feedforward linear units is equivalent to a different linear unit.

As discussed in the Delta rule, they need to be differentiable and strictly monotonic. Usually either *tanh* or *logistic* functions are used.

The Algorithm.

Consider the network:



This layered feedforward network has 1 layer of hidden units. We have added a bias unit to the inputs, and to the hidden layer.

The question is how should the weights be adjusted to achieve gradient descent.

As before, we take the error measure to be

$$E = \sum_p \sum_k (D_k^p - Y_k^p)^2$$

where p indexes the patterns, and k the output units.

For weights on each output unit, we can simply apply the Delta rule:

$$\Delta w_{kj} = \eta . \sum_p (D_k^p - Y_k^p) . \frac{dY_k}{dA_k} . Y_j^p$$

to find the weight change on one weight on an output unit (k).

But how should we alter weights to the hidden units?

If we had a value for the error at a hidden unit, E_j, say, then we could apply the Delta rule there too:

$$\Delta w_{ji} = \eta . E_j . \frac{dY_j}{dA_j} . Y_i$$

and we could continue this to any number of layers of hidden units.

Another way of asking this question is "How should we allocate the error that occurs at the output units amongst the hidden units?". This is a credit (perhaps blame in this case!) assignment problem.

The solution taken is to "funnel" the errors at the output units back through the weights which connect the hidden units to the output units. That is, the error at a single hidden unit is the sum of the errors at all the output units to which it is connected, multiplied by the weight from the hidden unit to each output unit.

Perhaps it is clearer as an equation. The error at hidden unit j, E_j, is

$$E_j = \sum_{k=1}^{s} E_k \frac{dY_k}{dA_k} w_{kj}$$

where *k* indexes the *s* output units.

If we use the logistic function for the output unit, then

$$Y_k = \frac{1}{(1 + e^{(-\beta . A_k)})}$$

where β is the slope of the logistic, and A_k is the activation of unit k (including weighted bias input). Then

$$\frac{dY_k}{dA_k} = \beta Y_k (1 - Y_k)$$

So that we can find an expression for the error at hidden unit j, E_j,

$$E_j = \sum_{k=1}^{s} E_k \beta Y_k (1 - Y_k) w_{kj}$$

where s is the number of output units. From this we can produce an expression for the weight change at a hidden unit, by applying the Delta rule:

$$\Delta w_{ji} = \sum_{k=1}^{s} E_k \beta_k Y_k (1 - Y_k) w_{kj} \beta_j Y_j (1 - Y_j) Y_i$$

and this can be applied directly to update the weights.

It is clear that this argument can be applied to any number of hidden layers.