

HW4 Algorithm:
 Learning in Perceptrons:
 Back Propagation:

1 layer neuron network is a perceptron network. 1 neuron is a perceptron

Learning for perceptrons

Let y be the output that we want a perceptron to give on input X .
 i.e. (X, y) is an example pair.

Denote by $h_w(X)$ what the perceptron actually outputs.
 The square of the error we make in using this over y is.

$$E = \frac{1}{2} \text{Err}^2 = \frac{1}{2} (y - h_w(X))^2$$

We want to choose \mathbf{W} so as to make this error as small as possible.

Components of the gradient of this function are:

$$\begin{aligned} \text{Derivative } E / \text{Derivative } W_j &= \text{Err} \times \text{Derivative Err} / \text{Derivative } W_j \\ &= \text{Err} * -g'(\sum (w_j X_j \text{ from } h = 0 \text{ to } n)) * X_j \end{aligned}$$

g' for sigmoid is well defined.
 g' for step function -> oh no!

At a local minimum, all these partials will be 0.

So choose new W_j 's to be

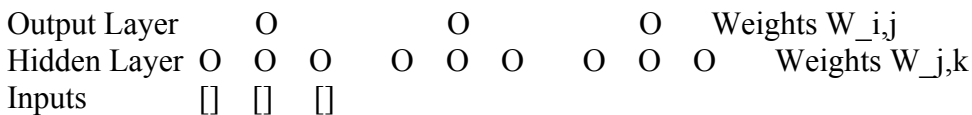
$$\begin{aligned} W_j &\leftarrow W_j + \text{sigma} (\text{Err} * g' (\sum(w_j x_j)) * X_j) \\ \text{sigma} &= \text{called learning rate (is some constant, say } -1) \end{aligned}$$

Learning algorithm

Repeat till error reasonable small or non-changing
 for each (X, y) training pair compute update to weights using our update rule

Two more More Layers

(For HW if use probably want just 2 layers)



Each neuron is connected to every neuron in the level above.

Training rule for this layer same as for one layer case:

$$W_{j,i} \leftarrow W_{j,i} + \text{Sigma} * a_j * \text{delta}_i \quad (\text{Err}_i * g'(\sum W_{j,i}, x_i)) = \text{delta}_i$$

What about updating the $W_{k,j}$'s? (called back propagation)

Want to compute how much error in final layer is caused by a given $\text{delta}_j = g'(in_j) \sum w_{j,i} \text{delta}_i$

$W_{j,k} \leftarrow W_{j,k} + \alpha \cdot a_k \cdot \delta_j$