HW4 Algorithm:
Learning in Perceptrons:
Back Propogation:

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

<u>Learning for perceptrons</u>

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.

$$E = ½\ Err^2 = ½\ (y - h_W\ (X))^2$$

We want to choose **W** so as to make this error as small as possible.

Components of the gradient of this function are:

Deriaitive E / Derivative Wj = Err X Derivitive Err / Derivitive Wj
= Err * -g'(Σ (wjXj) from h = 0 to n) * Xj

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 Wj's to be

Wj <- Wj + sigma (Err * g' (Σ(wjxj))*Xj
sigma = called learning rate (is some constant, say –1)

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

<u>Two more More Layers</u>

(For HW if use probably want just 2 layers)

Output Layer		O			O			O		Weights W_i,j
Hidden Layer	O	O	O	O	O	O	O	O	O	Weights W_j,k
Inputs	[]	[]	[]							

Each neuron is connected to every neuron in the level above.
Training rule for this layer same as for one layer case:

$W_{j,\,i}$ <- $W_{j,\,i}$ + Sigma * a_j * $delta_i$ (Err_i * g'($\Sigma W_{j,\,i}$, x_i)) = $delta_i$

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

Want to compute how much error in final layer is caused by a given $delta_j = g'(in_j)\Sigma w_{j,\,i}\ delta_i$

W_j,k <- W_j,k + alpha*a_k*delta_j