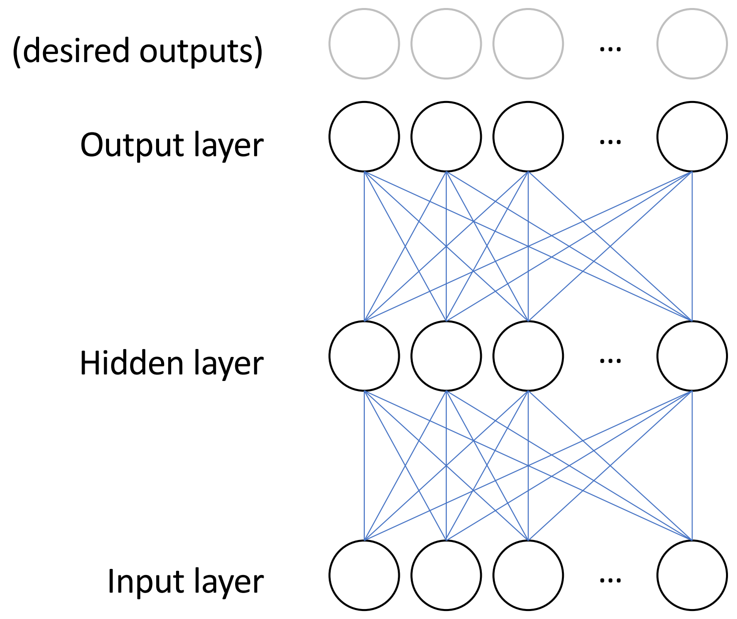
Implementing Backprop

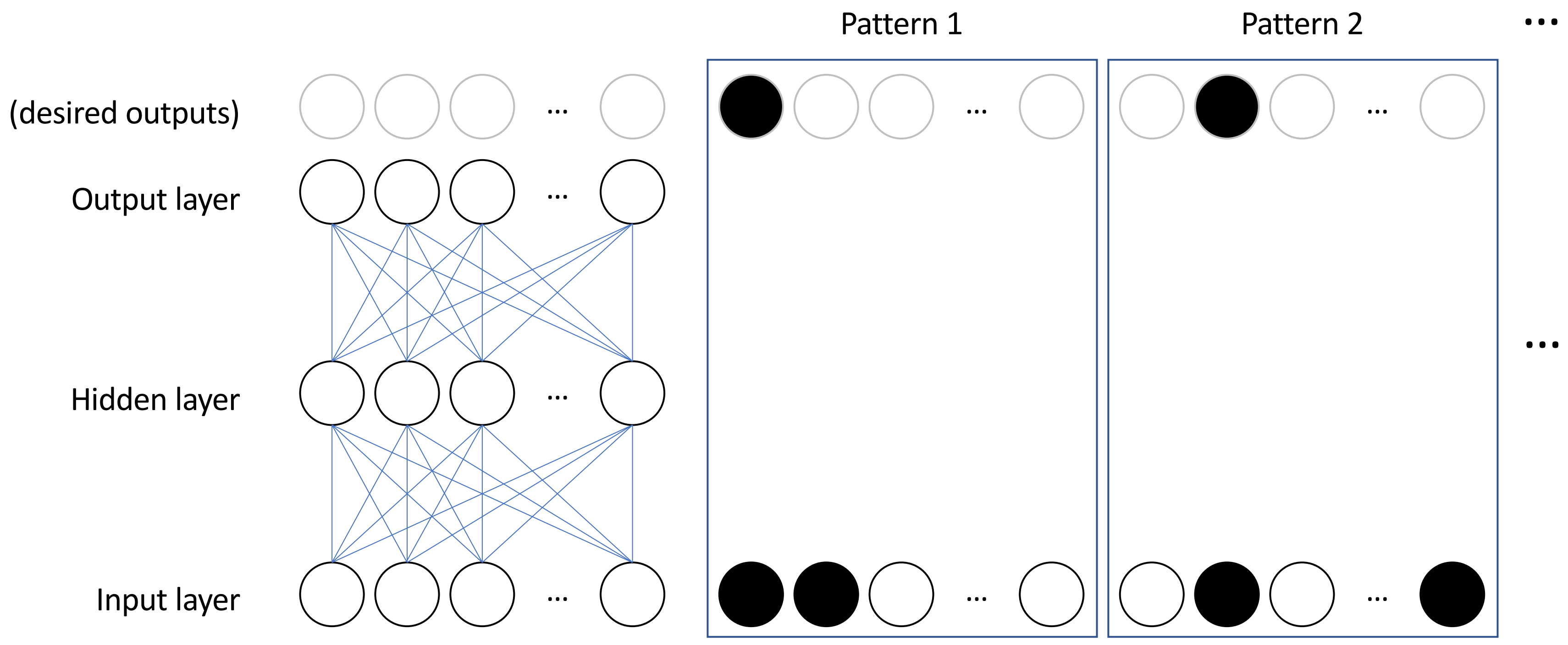
1)Create a Network

- Instantiate an object of the BackPropNet class with the required number of layers, and units per layer.
- The example below has two layers (above the input layer)



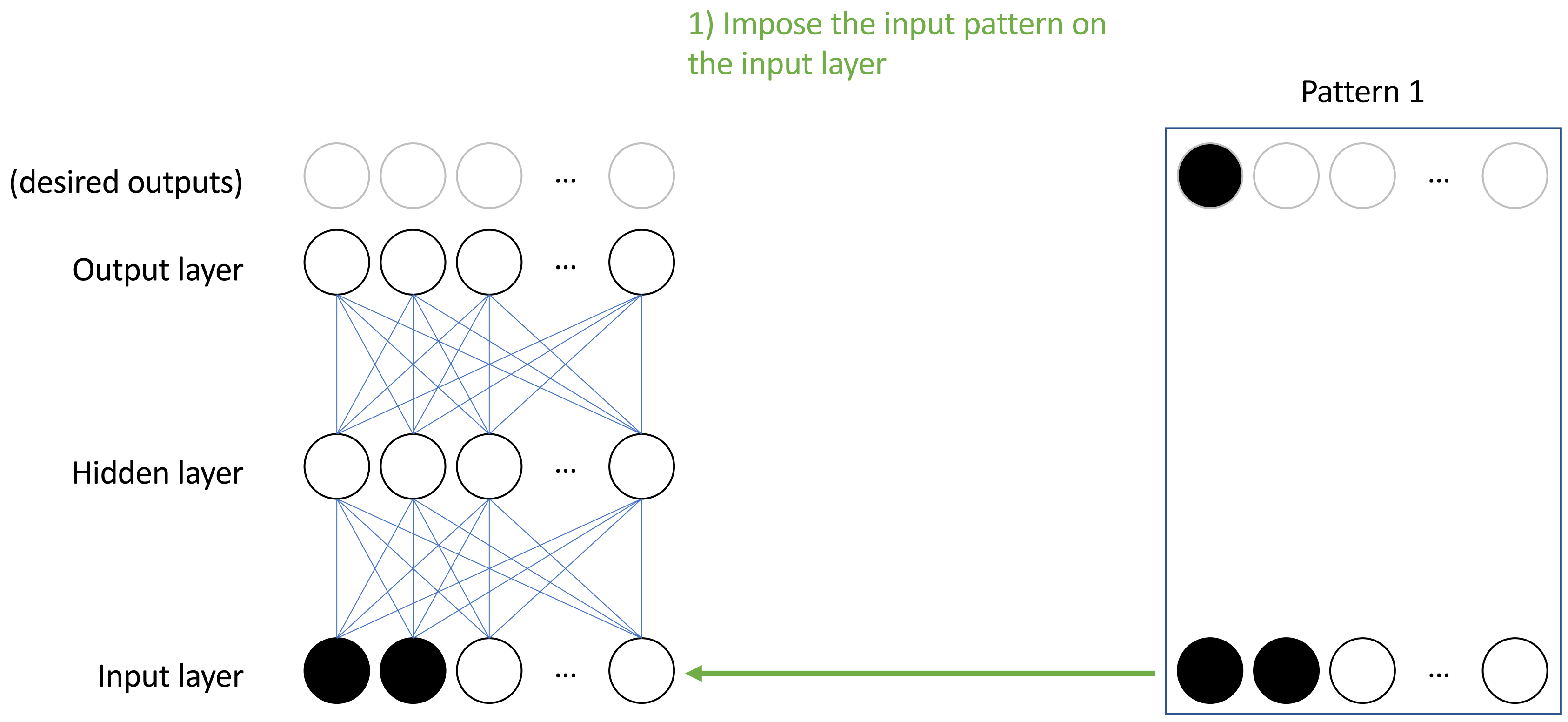
2) Create a Training Set

- A collection of patterns (input-output mappings) to be trained, where each training pattern
- Consists of (1) a pattern of activation on the input units and (2) the *desired* output for that input (in the figure below, black circles are activation (or desired activation) of 1.0 and white circles are 0.0.



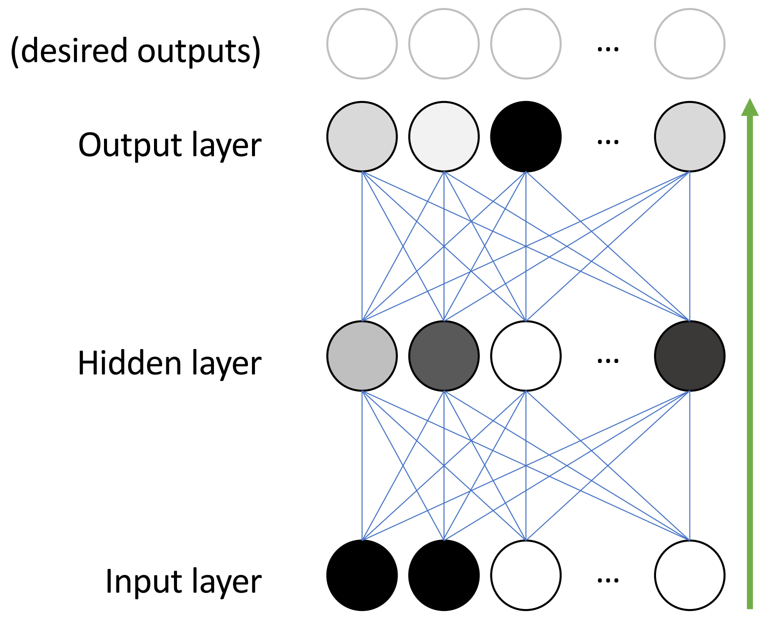
3) Train the Network on the Training Set

- Present each training pattern one at a time. For each pattern…



3) Train the Network on the Training Set

- Present each training pattern one at a time. For each pattern…



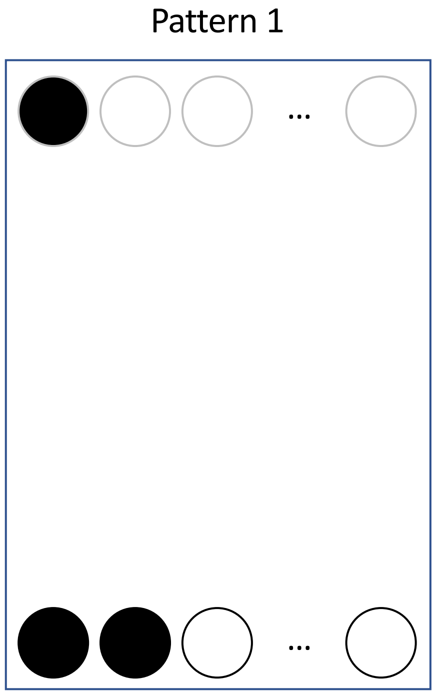
1) Impose the input pattern on the input layer

And propagate activation forward, from each layer, *j*, up to the next layer, *i*, where:

$$n_i = \sum w_{ij} a_j$$

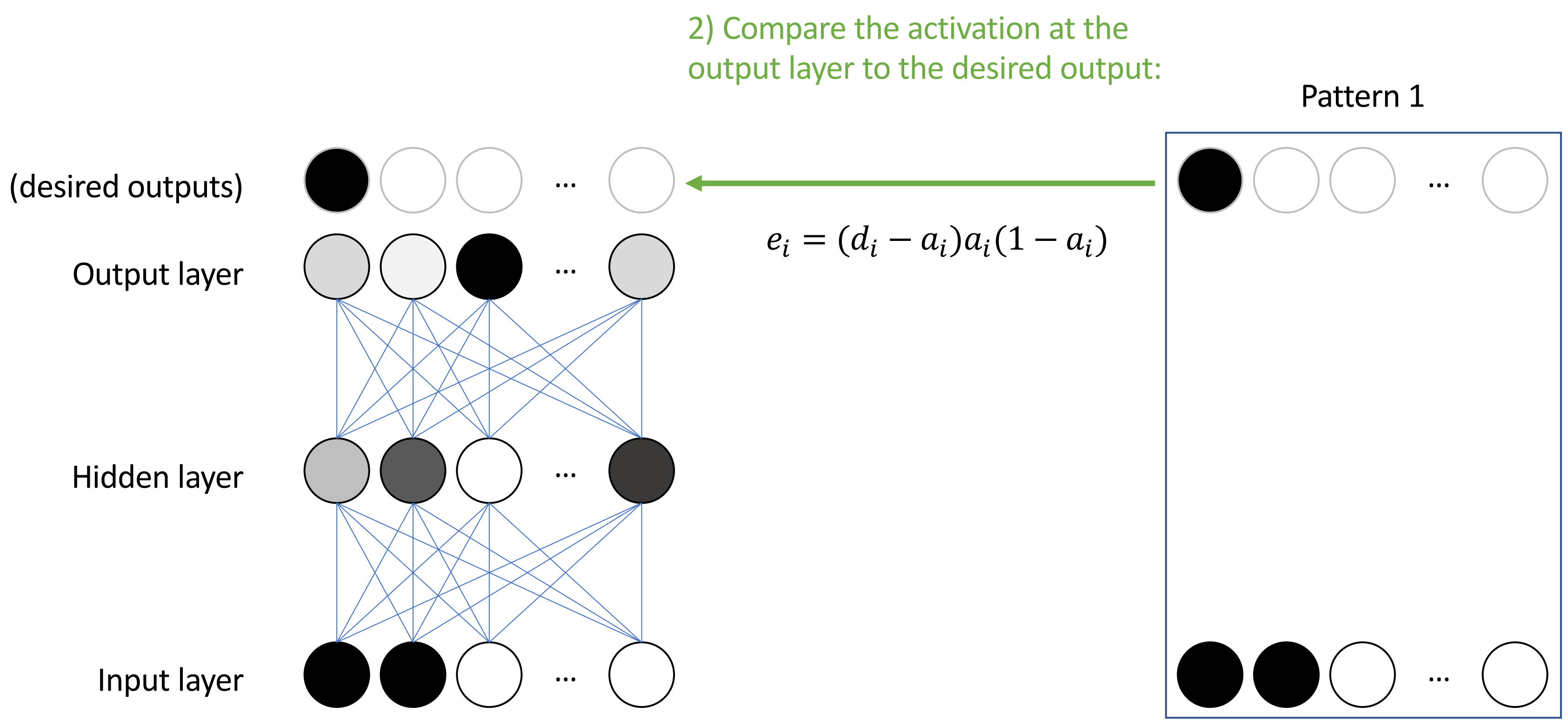
and

$$a_i = \frac{1}{1 + e^{-n_i}}$$



3) Train the Network on the Training Set

- Present each training pattern one at a time. For each pattern…



2) Compare the activation at the output layer to the desired output:
Pattern 1
(desired outputs)
Output layer
Hidden layer
Input layer
…

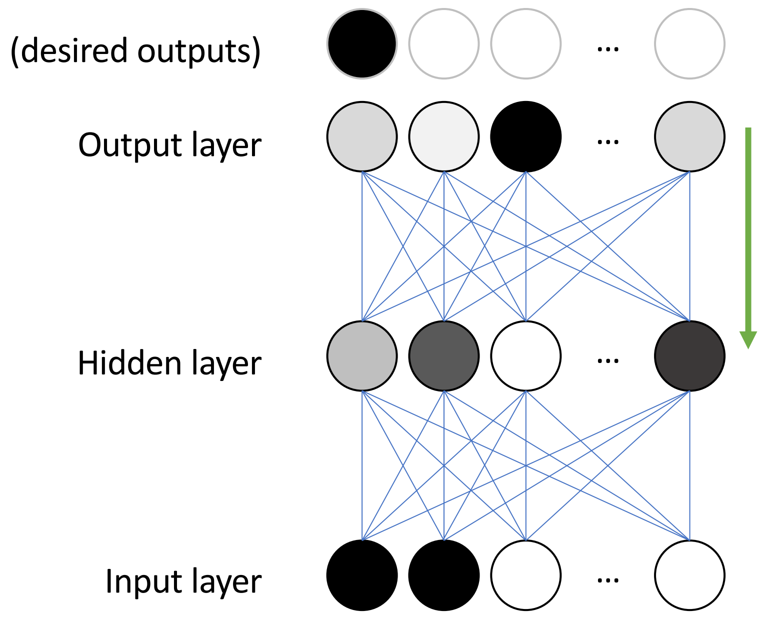
$$e_i = (d_i - a_i)a_i(1 - a_i)$$

3) Train the Network on the Training Set

- Present each training pattern one at a time. For each pattern…

2) Compare the activation at the output layer to the desired output

And propagate the error backward:



$$e_i = (d_i - a_i)a_i(1 - a_i)$$

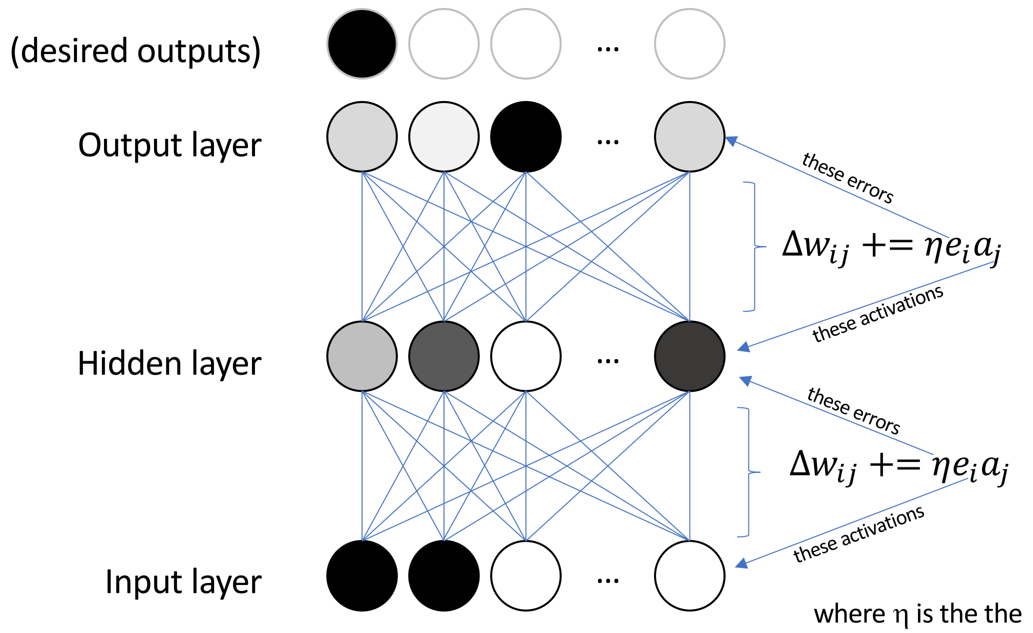
$$e_j = \sum_i w_{ij} e_i \, a_j(1 - a_j)$$

Pay *very* careful attention to the subscripts here!

3) Train the Network on the Training Set

- Present each training pattern one at a time. For each pattern…

3) Update the changes in the weights, Δw_{ij}, based on the errors of the receiving units, *i*, and the activations of the sending units, *j*:



$$\Delta w_{ij} \mathrel{+}= \eta e_i a_j$$

$$\Delta w_{ij} \mathrel{+}= \eta e_i a_j$$

where η is the the *learning rate*

Note that the equations on the left express the *change* in Δw_{ij} (the change in the change in the weights). Hence the "+=" rather than the "=" in the equations.

This is because you're not going to change the weights, themselves, just yet.

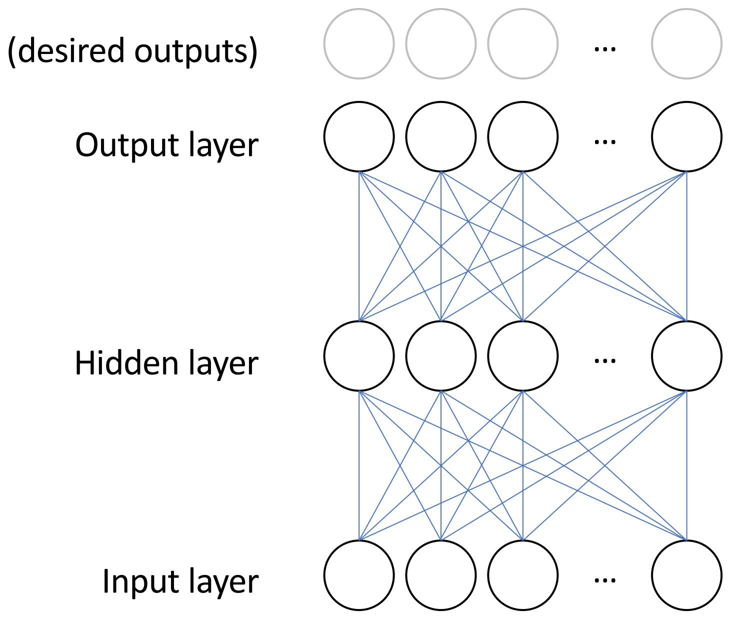
Instead, you're going to *accumulate* (add up) all the changes due to each training pattern. (This is "batch updating".)

For this reason, we are *incrementing* the value of Δw_{ij} ("+=") rather than setting it ("=").

3) Train the Network on the Training Set

- Present each training pattern one at a time.
- After you have presented (trained) every training pattern *exactly once*, and incremented the changes in the weights due to each pattern, then…

(1) update the weights themselves and (2) initialize the delta weights:



$$w_{ij} \mathrel{+}= \Delta w_{ij}$$

Update the weight, w_{ij}, by adding the change in the weight, Δw_{ij}, to it

$$\Delta w_{ij} \mathrel{*}= \mu$$

μ is the *momentum* term: The proportion of the change in the weights your algorithm carries over between one complete training *epoch* (i.e., one pass through the whole training set) and the next.

3) Train the Network on the Training Set: Summary

On each *epoch* of training…

- Impose a pattern of activation (an input pattern) on the input units
- Propagate activation forward, from the units, *j*, in each layer, *J*, to the units, *i*, in layer *I* = *J*+1
- Compare the activation on the output layer to the desired outputs to compute the error, e_i on output unit *i*
- Propagate that error backward from units *i* in layer *I* to units *j* in lower layers, *J* = *I* – 1
- Use each unit's error*,* e_i, and the activation of unit *j* to update the change in the weight, Δw_{ij}, on the connection from *j* to *i*
- At the end of the epoch, change the weights, w_{ij}, and initialize the weight changes, Δw_{ij}
- Be sure to keep track of the network's *global error, G,* (the mean square error over all output units, *i*, over all patterns, *p*):

$$G = \frac{\sum_p \frac{\sum_i (d_i - a_i)^2}{n_i}}{n_p}$$

Repeat the procedure above over and over until *G* is sufficiently low. (It may take hundreds or thousands of epochs of training to accomplish this. And your network may never accomplish it.) You will have to figure out what "sufficiently low" means numerically. Remember that G is the mean squared error (MSE) of all the output units. How low do you want this MSE to be? Also you may find that some other settling criterion proves more satisfactory that MSE.