$$E = \frac{1}{2m}\sum_{\mu}(y^{\mu} - \hat{y}^{\mu})^2$$

Here's the general algorithm for updating the weights with gradient descent:

- Set the weight step to zero: $\Delta w_i = 0$
- For each record in the training data:
  - Make a forward pass through the network, calculating the output $\hat{y} = f(\sum_i w_i x_i)$
  - Calculate the error term for the output unit, $\delta = (y - \hat{y}) * f'(\sum_i w_i x_i)$
  - Update the weight step $\Delta w_i = \Delta w_i + \delta x_i$
- Update the weights $w_i = w_i + \eta \Delta w_i / m$ where η is the learning rate and m is the number of records. Here we're averaging the weight steps to help reduce any large variations in the training data.
- Repeat for e epochs.

You can also update the weights on each record instead of averaging the weight steps after going through all the records.

Remember that we're using the sigmoid for the activation function, $f(h) = 1/(1 + e^{-h})$

And the gradient of the sigmoid is $f'(h) = f(h)(1 - f(h))$

where h is the input to the output unit,

$h = \sum_i w_i x_i$

Implementing with NumPy

For the most part, this is pretty straightforward with NumPy.

First, you'll need to initialize the weights. We want these to be small such that the input to the sigmoid is in the linear region near 0 and not squashed at the high and low ends. It's also important to initialize them randomly so that they all have different starting values and diverge, breaking symmetry. So, we'll initialize the weights from a normal distribution centered at 0. A good value for the scale is $1 / \sqrt{n}$ where n is the