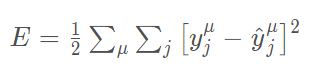
Learning Weights

In order for your neural network to learn, it needs the weights to adjust from example data, then use those weights to make predictions.

We want to minimize our errors so tha our model for predictions are as close as possible to the real values.

We use the sum of the squared errors (SSE):



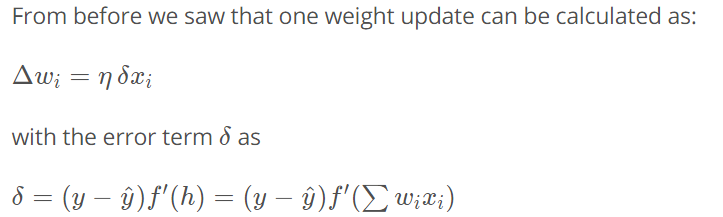
Squared error ensures the error is always positive and larger errors are penalized more than smaller errors.

This is the error between all the output units J of difference between the true output and the predicted output. And then sum that over all data points u(mew).



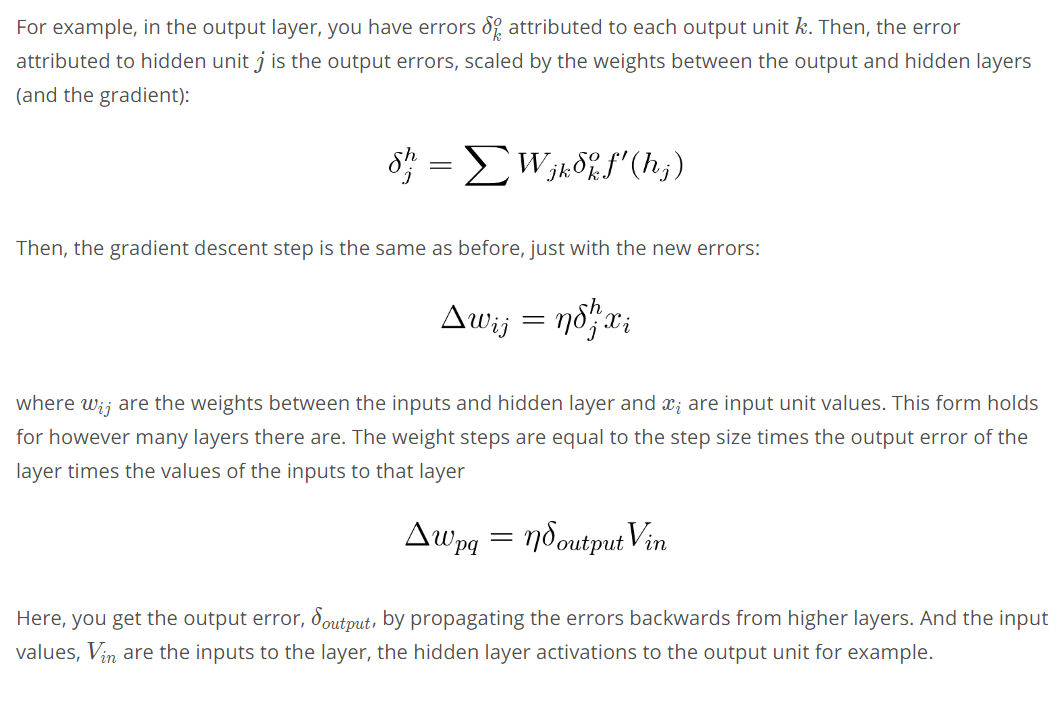
We want to minimize the error by changing the values of w. The algorithm you would use is gradient descent.

Change the value of w at each step of the gradient decent process if the next steps error is smaller than before.



To ensure that the weights chosen are not the lowest point, but the local minimum, there is a method to avoid this called momentum algorithm for gradient descent.

In order to determine the change in weights for hidden layers as well as their outputs:



From this example, you can see one of the effects of using the sigmoid function for the activations. The maximum derivative of the sigmoid function is 0.25, so the errors in the output layer get reduced by at least 75%, and errors in the hidden layer are scaled down by at least 93.75%! You can see that if you have a lot of layers, using a sigmoid activation function will quickly reduce the weight steps to tiny values in layers near the input. This is known as the **vanishing gradient** problem. Later in the course you'll learn about other activation functions that perform better in this regard and are more commonly used in modern network architectures.