the corresponding weight. We then sum those up and multiply by the derivative there. That is:

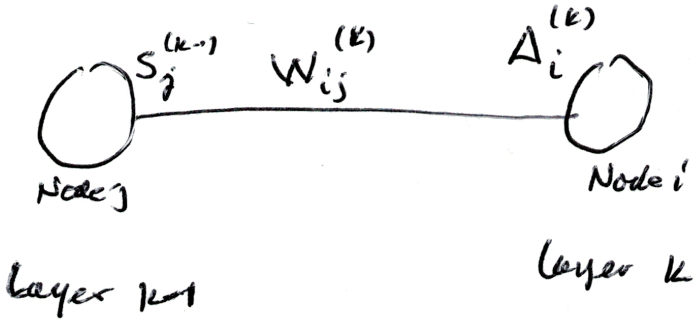
$$\Delta_k^{(i)} = {S_k^{(i)}}' \sum_{j=1}^{N_{i+1}} \Delta_j^{(i+1)} W_{jk}^{(i+1)}$$

This may look complicated, but think of it in terms of the reverse affine map from $\mathbb{R}^{N_{i+1}}$ to $\mathbb{R}^{N_i}$. Then we can compute all of the $\Delta' s$ in layer i all at once:

$$\Delta^{(i)} = ((W^{(i+1)})^T \Delta^{(i+1)}) .* {S^{(i)}}'$$

Somewhere a linear algebra teacher is crying. OK, so just to be crystal clear, our multiplication is not a dot product, we are performing the multiplication (using the Matlab symbol `.*`) between two vectors *element-wise* so the result is also a vector of the same size. Also to be very clear, let's figure out the dimensions of everything there: $W^{(i+1)}$ is $N_{i+1} \times N_i$ (so the transpose reverses those), $\Delta^{(i)}$ has N_{i+1} elements, and ${S^{(i)}}'$ has N_i elements.

Once we reach the input layer, we're done. Now what do we do with these Δ values? I like to use the following memory device to help me remember how this goes. Below we see an edge, and we want to use the **state on the left** and the Δ **on the right** with the edge and weight connecting the two.



Then here is the big conclusion:

Backpropagation of Error

$$\Delta W_{ij}^{(k)} = S_j^{(k-1)} \Delta_i^{(k)} \tag{13.1}$$

The update rule for the weights is now (using gradient descent):

$$\text{new } W_{ij}^{(k)} = \text{old } W_{ij}^{(k)} + \alpha \Delta W_{ij}^{(k)}$$

where the plus sign is not a typo, and α is the learning rate.

Thinking back, this kind of update rule was a lot like Widrow-Hoff, and in that case, we were able to write the update using linear algebra so we could update weights all at once.

Update Rule Using Linear Algebra

Remember that matrix $W^{(i)}$ connects layer $i-1$ to layer i, so that $W^{(i)}$ is $N_i \times N_{i-1}$. In our update rule, we are taking states from layer $i-1$ (so we have a vector with N_{i-1} entries) and the deltas from layer i (so that is a vector with N_i entries. To give away the punch line, if we take the outer product between these

two vectors (in the right order), we'll have a matrix the same size as $W^{(i)}$. You should verify this works by looking at the dimensions of the three objects below.

$$\Delta W^{(i)} = \Delta^{(i)}(S^{(i-1)})^T$$

And finally,

$$W^{(i)}_{\text{new}} = W^{(i)} + \alpha \Delta W^{(i)}$$

You might think we're done, but there are still three big questions we have to answer:

1. These computations were for a single data point. What do I do with p data points?
2. Is this really gradient descent? Prove it!
3. Where did the bias terms go??

These questions will be answered in the next section, where we do actually show that our computations do result in gradient descent.

13.6 Backprop Proved

The answer to the **first question** goes pretty quickly. With p data points, if we use the sum of squares as the error, then each of the 4 numbers we computed per node should be summed over all the input (that's before the backward phase). Some people use the average error, and in that case, you would average each of the 4 numbers over all the input.

For the **second question**, we'll need to break out our Calculus. The function we want to minimize is the error function. We'll put a 1/2 in the front so we don't have to deal with putting 2 in front of everything when we differentiate.

$$E = \frac{1}{2} \sum_{i=1}^{p} \|\boldsymbol{t}^{(i)} - \boldsymbol{y}^{(i)}\|^2$$

We'll show that our rules do indeed produce the gradient descent. Recall that W^l_{mn} connects node n in the layer to the left to node m in the layer to the right. Therefore, S^l_m is the state of node m in layer l (to the right of the edge labeled W^l_{mn}). Using these relationships, we can write:

$$\frac{\partial E}{\partial W^l_{mn}} = \frac{\partial E}{\partial S^l_m} \frac{\partial S^l_m}{\partial P^l_m} \frac{\partial P^l_m}{\partial W^l_{mn}} \tag{13.2}$$

The two values on the right can readily be computed:

$$\frac{\partial S^l_m}{\partial P^l_m} = \sigma'\left(P^l_m\right) \qquad \frac{\partial P^l_m}{\partial W^l_{mn}} = S^{l-1}_n \tag{13.3}$$

This leaves the first term which can be evaluated on the output layer L:

$$\frac{\partial E}{\partial S^L_m} = \frac{\partial E}{\partial y_m} = (t_m - y_m)(-1)$$

On the rest of the layers, the term can be defined recursively,

$$\frac{\partial E}{\partial S^l_m} = \sum_{j \in \text{nextlayer}} \frac{\partial E}{\partial S^{l+1}_j} \frac{\partial S^{l+1}_j}{\partial S^l_m} \tag{13.4}$$

Since $S_j^{l+1} = \sigma(P_j^{l+1}) = \sigma(W_{jm}^{l+1} S_m^l + \text{ other terms})$, the derivative will be

$$\frac{\partial S_j^{l+1}}{\partial S_m^l} = \sigma'(P_m^{l+1}) W_{jm}^{l+1}$$

Now we'll connect up the two sets of notation:

Definition: We'll define Δ as the product of the first two terms of Equation (13.2):

$$\Delta_m^l = -\frac{\partial E}{\partial S_m^l} \frac{\partial S_m^l}{\partial P_m^l} = -\frac{\partial E}{\partial S_m^l} \sigma'(P_m^l)$$

Therefore, on the output layer,

$$\Delta_m^L = \frac{\partial E}{\partial S_m^L} \frac{\partial S_m^L}{\partial P_m^L} = -(t_m - y_m)(-1)\sigma'(P_m^L)$$

which matches Equation (**??**). Now, using Equation (13.4), the nodes on the previous layer are computed:

$$\Delta_m^l = -\frac{\partial E}{\partial S_m^l} \frac{\partial S_m^l}{\partial P_m^l} = \left(\sum_{j \in \text{layer} l+1} -\frac{\partial E}{\partial S_j^{l+1}} \frac{\partial S_j^{l+1}}{\partial S_m^l} \right) \sigma'(P_m^l) =$$

$$\sigma'(P_m^l) \left(\sum_{j \in \text{layer} l+1} -\frac{\partial E}{\partial S_j^{l+1}} \sigma'(P_m^{l+1}) W_{jm}^{l+1} \right) =$$

$$\sigma'(P_m^l) \left(\sum_{j \in \text{ layer} l+1} \Delta_j^{l+1} W_{jm}^{l+1} \right)$$

And this is Equation (**??**). Finally, by Equation (13.3), we can write:

$$-\frac{\partial E}{\partial W_{mn}^l} = \left(-\frac{\partial E}{\partial S_m^l} \frac{\partial S_m^l}{\partial P_m^l} \right) \frac{\partial P_m^l}{\partial W_{mn}^l} = \Delta_m^l S_n^{l-1}$$

which proves that the update rule in Equation (13.1) is indeed gradient descent. Now for the second question: How should we deal with bias terms?

For the bias term, we will add a node to each layer, but for these nodes, the state is the constant function, $S = \sigma(x) = 1$, and the weight connecting this node to node k of the next layer could be labeled b_k^l. That also means that Δ for a bias node is 0, but now Equation (**??**) becomes:

$$b_k^l = b_k^l + \epsilon \Delta_k^l$$

where the Δ_k^l is the value of Δ to the node to which b_k^l is connected.

13.7 Simple Construction of a Feed Forward Neural Net

The neural net is simple to construct if we view it in terms of its layers.

- Initialize the $n - k - m$ network:

  Build the weights and biases with random numbers. Be sure to pay attention to dimensions.

$$W^{(1)} \text{ is } k \times n \qquad \boldsymbol{b}^{(1)} \text{ is } k \times 1$$

$$W^{(2)} \text{ is } m \times k \qquad \boldsymbol{b}^{(2)} \text{ is } m \times 1$$

- Compute the output of a neural net given the weights, biases. Here we'll assume that the activation function is $\sigma(x)$, and has been determined. The computation proceeds in layers. For the three layer network (layers 0, 1, and 2), we'll have the series of computations. This is a forward pass:
  - $P^{(1)} = W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$
  - $S^{(1)} = \sigma(P^{(1)})$. Compute $S(1)'$ as well.
  - $P^{(2)} = W^{(2)}S^{(1)} + \boldsymbol{b}^{(2)}$
  - $S^{(2)} = P^{(1)}$. Then $S(1)'$ is a vector of ones. The output is $S^{(2)}$.
- A backwards pass for backpropagation of error:
  - $\Delta^{(2)} = (\mathbf{t} - \mathbf{y})$ (This is a vector in $\mathbb{R}^m$).
  - $\Delta^{(1)} = (W^{(2)})^T\Delta^{(2)}$ `.*` $S^{(i)'}$ (We use Matlab's `.*` to denote elementwise mutliplication)
- Compute the changes to the weights and biases:
  - $\Delta W^{(1)} = \Delta^{(1)}\mathbf{x}^T$
  - $\Delta W^{(2)} = \Delta^{(2)}(S^{(1)})^T$
  - $\Delta\mathbf{b}^{(1)} = \Delta^{(1)}$
  - $\Delta\mathbf{b}^{(2)} = \Delta^{(2)}$
- Apply the changes:

$$W^{(1)} = W^{(1)} + \alpha\Delta W^{(1)}, \quad W^{(2)} = W^{(2)} + \alpha\Delta W^{(2)}, \quad \mathbf{b}^{(1)} = \mathbf{b}^{(1)} + \alpha\Delta\mathbf{b}^{(1)} \quad \mathbf{b}^{(2)} = \mathbf{b}^{(2)} + \alpha\Delta\mathbf{b}^{(2)}$$