IA Solutions 7: Neural Networks

Exercise 1: Simple Perceptron

1. For example, $w_0 = 3$, $w_1 = w_2 = 2$ works.
2. For example, $w_0 = 1$, $w_1 = w_2 = 2$.
3. For a single perceptron to compute `XOR` it needs to return 0 on inputs 1 and 1, thus $w_1 + w_2 \leq w_0$. At the same time, if exactly one of the inputs is 1 it needs to return 1, thus $w_1 > w_0$ and $w_2 > w_0$. Finally, if both inputs are 0 it must return 0, so $0 \leq w_0$. This last inequality implies that $2w_0 \geq w_0$, thus we get

   $$w_1 + w_2 > 2w_0 \geq w_0 \geq w_1 + w_2$$

   which is a contradiction.

   Note we need all of the inequalities above: consider $w_1 = w_2 = -2$ and $w_0 = -3$.
4. One method here is to construct this network out of smaller functions such as `OR` and `NEGATION`.

Exercise 2: Update rule for multilayer neural networks

1. The derivative of the sigmoid function can be calculated as follows:

   $$\begin{aligned}
   \frac{d}{dx}\sigma(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} \\
   &= \frac{d}{dx}(1+e^{-x})^{-1} \\
   &= (-e^{-x})(-(1+e^{-x})^{-2}) \\
   &= \frac{e^{-x}}{(1+e^{-x})^2} \\
   &= \frac{1}{(1+e^{-x})} \cdot \frac{e^{-x}}{1+e^{-x}} \\
   &= \sigma(x)t\frac{1+e^{-x}-1}{1+e^{-x}} \\
   &= \sigma(x)\left(1-\frac{1}{1+e^{-x}}\right) \\
   &= \sigma(x)(1-\sigma(x))
   \end{aligned}$$

2. We want to know how to adjust the weights given some training example for which we know what the output should be. We consider the error as a function of the weights. With our current assignment of weights, this function outputs our current error. Using this function, we want to work

out which "direction" will reduce the error the most, i.e. the gradient of steepest descent. We can then adjust the weights by some small amount in this direction. (The method we use here is sometimes called *stochastic gradient descent*; it perfoms gradient descent on single training examples rather than the full set of training examples.)

Thus, we want to work out a function which describes how changing each of the weights will change the error function E_d. That is, for each weight w_{ji}, we want to work out

$$\frac{\partial E_d}{\partial w_{ji}}$$

We proceed by working out how changes to the weight will propogate into the network. First note that changing the weight that leads into some node first changes the weighted sum of inputs for this node. Thus we consider how changes to the weight change the weighted sum, and how changes to the weighted sum change the error; we use the chain rule:

$$\begin{aligned}
\frac{\partial E_d}{\partial w_{ji}} &= \frac{\partial E_d}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} \\
&= \frac{\partial E_d}{\partial net_j} x_{ji} && \text{as } net_j = \sum_i w_{ji} x_{ji}
\end{aligned}$$

Now we want to work out $\frac{\partial E_d}{\partial net_j}$.

Only now do we divide into the two cases suggested by the question. First, we consider weights that are directly attached to output units. Output units take the weighted sum, apply the sigmoid function, then give the result as the output. It is this output that we wish to make close to the expected value. Once again, we use the chain rule: changing the weighted sum will change the output (through the sigmoid function), and changing the output will change the error:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

Let us first break down the left hand side of this. This expresses the rate of change of the error according to how the output changes.

$$\begin{aligned}
\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in out} (t_k - o_k)^2 \\
&= \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 && \text{as only the output } o_j \text{ changes} \\
&= \frac{1}{2} \cdot 2 \cdot (t_j - o_j) \cdot \frac{\partial}{\partial o_j} (t_j - o_j) \\
&= -(t_j - o_j)
\end{aligned}$$

It is perhaps easier to think about what this says in words: it says that if the output is smaller than the expected value, then increasing the output will decrease the error, and inversely if the output is larger decreasing it will decrease the error.

Now let us look at the right hand side. Here, the output is really just the sigmoid function applied to the weighted sum, so we use the result to part 1. above:

$$\begin{aligned}\frac{\partial o_j}{\partial net_j} &= \frac{\partial}{\partial net_j}\sigma(net_j)\\ &= \sigma(net_j)(1-\sigma(net_j))\\ &= o_j(1-o_j)\end{aligned}$$

Putting this all together, we see:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1-o_j)$$

Finally, if we suppose that we should take a "step" of size η in the direction of maximum descent, the change to the weight is:

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)x_{ji}$$

What about the hidden layers? Here, the output cannot be directly compared to the expected output, but instead relies on how these outputs affect downstream nodes. The general techniques are similar, we just need to propogate the changes further into the network using repeated applications of the chain rule. Here we only consider a hidden layer one level back from the output layer.

$$\begin{aligned}\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream j} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j} & \\ &= \sum_{k \in downstream j} -\delta_k \frac{\partial net_k}{\partial net_j} & \text{writing } \delta_k = -\frac{\partial E_d}{\partial net_k}\\ &= \sum_{k \in downstream j} -\delta_k \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j} & \text{chain rule}\\ &= \sum_{k \in downstream j} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} & net_k \text{ is affected by the weight of its input}\\ &= \sum_{k \in downstream j} -\delta_k w_{kj} o_j(1-o_j) & \text{derivative of sigmoid}\end{aligned}$$

If desired, you can substitute this value into

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{net_j} \cdot \frac{net_j}{\partial w_{ji}}$$

and multiply by some $-\eta$ to get Δw_{ji}.