# Deriving SGD for Neural Networks

Swarthmore College CS63 Spring 2018

A neural network $NN$ computes some function mapping input vectors $\vec{x}$ to output vectors $\vec{y}$:

$$NN(\vec{x}) = \vec{y}$$

But if the weights of the network change, the output will also change, so we can think of the output as a function of the input and the vector of all weights in the network $\vec{w}$:

$$NN(\vec{x}, \vec{w}) = \vec{y}$$

The loss $\epsilon$ of the network is a function of the output (itself a function of weights and inputs) and the target $\vec{t}$:

$$\epsilon(NN) = \epsilon(\vec{y}, \vec{t}) = \epsilon\left(\vec{w}, \vec{x}, \vec{t}\right)$$

The gradient of the loss function with respect to the weights $\nabla_{\vec{w}}(\epsilon)$ points in the direction of steepest increase in the loss. In stochastic gradient descent, our goal is to update weights in a way that reduces loss, so we take a step of size $\alpha$ in the direction opposite the gradient:

$$\vec{w}' = \vec{w} - \alpha \nabla_{\vec{w}}(\epsilon)$$

$$\begin{bmatrix} w_1' \\ w_2' \\ \vdots \\ w_W' \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_W \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial \epsilon}{\partial w_1} \\ \frac{\partial \epsilon}{\partial w_2} \\ \vdots \\ \frac{\partial \epsilon}{\partial w_W} \end{bmatrix} = \begin{bmatrix} w_1 - \alpha \frac{\partial \epsilon}{\partial w_1} \\ w_2 - \alpha \frac{\partial \epsilon}{\partial w_2} \\ \vdots \\ w_W - \alpha \frac{\partial \epsilon}{\partial w_W} \end{bmatrix}$$

Where $W$ is the total number of connection weights in the network. Therefore, to take a gradient descent step, we need to update every weight in the network using the partial derivative of loss with respect to that weight:

$$w_i' = w_i - \alpha \frac{\partial \epsilon}{\partial w_i}$$

We will now derive formulas for these partial derivatives for some of the weights in a neural network with *sigmoid activation functions* and *sum of squared errors* loss function. Other activation functions and other loss functions are possible, but would require re-deriving the partial derivatives used in the stochastic gradient descent update. Recall that the sigmoid activation function, for weighted sum of inputs $z$ computes:

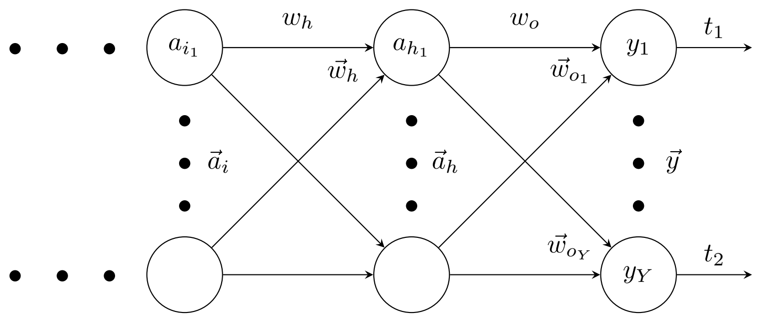
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

and the sum of squared errors loss function, for targets $\vec{t}$ and output activations $\vec{y}$ computes:

$$SSE = \sum_{i=1}^{Y} (t_i - y_i)^2$$

Where $Y$ is the number of output nodes (the and therefore dimension of the target vector).

Consider the following partially-specified neural network. We will find the partial derivative of the loss function with respect to one output-layer weight $w_o$ and one hidden layer weight $w_h$. It should then be clear how these derivations extrapolate to the updates in the backpropagation algorithm we are implementing.



First consider $w_o$, the weight of an incoming edge for an output layer node. We want to compute the partial derivative of the loss function with respect to this weight:

$$\begin{aligned}\frac{\partial \epsilon}{\partial w_o} &= \frac{\partial}{\partial w_o}\left[\sum_{i=1}^{Y}(t_i - y_i)^2\right] \\ &= \sum_{i=1}^{Y}\frac{\partial}{\partial w_o}(t_i - y_i)^2\end{aligned}$$

Since the only term in this sum that depends on $w_o$ is $y_1$ (the activation of the destination node for the edge with weight $w_o$), this derivative simplifies to:

$$\frac{\partial \epsilon}{\partial w_o} = \frac{\partial}{\partial w_o}(t_1 - y_1)^2$$

Here we can apply the chain rule $[f(g(x))]' = f'(g(x))g'(x)$ to get

$$\begin{aligned}\frac{\partial \epsilon}{\partial w_o} &= 2(t_1 - y_1)\frac{\partial}{\partial w_o}(t_1 - y_1) \\ &= -2(t_1 - y_1)\frac{\partial}{\partial w_o}(y_1) \\ &= -2(t_1 - y_1)\frac{\partial}{\partial w_o}\sigma(\vec{w}_{o_1} \cdot \vec{a}_h)\end{aligned}$$

Where the second step eliminated $t_1$ because it doesn't depend on $w_o$ and thus has derivative 0, and the third step expanded $y_1$ to show the sigmoid activation function applied to the weighted sum of previous-layer inputs. We should now take a moment to find the derivative of a sigmoid function $\sigma'(z)$, using the reciprocal rule $[1/f]' = -f'/f^2$.

$$\begin{aligned}
\sigma(z) &= \frac{1}{1+e^{-z}} \\
\sigma'(z) &= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}}\left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) \\
&= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) \\
&= \sigma(z)(1-\sigma(z))
\end{aligned}$$

Now we can return to our partial derivative calculation and apply the chain rule in equation 1 to the sigmoid activation function:

$$\begin{aligned}
\frac{\partial \epsilon}{\partial w_o} &= -2(t_1 - y_1)\frac{\partial}{\partial w_o}\sigma(\vec{w}_{o_1} \cdot \vec{a}_h) \\
&= -2(t_1 - y_1)\sigma(\vec{w}_{o_1} \cdot \vec{a}_h)(1-\sigma(\vec{w}_{o_1} \cdot \vec{a}_h))\left[\frac{\partial}{\partial w_o}(\vec{w}_{o_1} \cdot \vec{a}_h)\right] && (1) \\
&= -2(t_1 - y_1)y_1(1-y_1)\left[\frac{\partial}{\partial w_o}(\vec{w}_{o_1} \cdot \vec{a}_h)\right] && (2) \\
&= -2(t_1 - y_1)y_1(1-y_1)\left[\frac{\partial}{\partial w_o}\sum_{i=1}^{|\vec{a}_h|} w_i a_i\right] && (3) \\
&= -2(t_1 - y_1)y_1(1-y_1)\left[\frac{\partial}{\partial w_o} w_o a_{h_1}\right] && (4) \\
&= -2(t_1 - y_1)y_1(1-y_1)a_{h_1} && (5)
\end{aligned}$$

Where equation 2 follows from simplifying the sigmoid functions using the stored node activation $y_1$, equation 3 re-writes the dot product as a weighted sum, equation 4 eliminates the elements of the sum that don't depend on $w_o$, and equation 5 finalizes the partial derivative. Note that most of equation 5 depends only on the target and the output, and is therefore the same for all incoming edges to output node $y_1$. We define $\delta_o$ for output node $o$ accordingly:

$$\delta_1 = y_1(1-y_o)(t_o - y_o)$$

and the gradient descent update for weights $w_i$ from node $i$ into an output node is then:

$$\begin{aligned}
w_i' &= w_i - \alpha(-2)\delta_o a_i \\
&= w_i + \alpha\delta_o a_i
\end{aligned}$$

Where in the second equation, we've absorbed the constant 2 into the learning rate $\alpha$. Next, consider $w_h$, the weight of an incoming edge for a hidden layer node. Again, we want to compute the partial derivative of the loss function with respect to this weight:

$$\begin{aligned}\frac{\partial \epsilon}{\partial w_h} &= \frac{\partial}{\partial w_h}\left[\sum_{i=1}^{Y}(t_i - y_i)^2\right] \\ &= \sum_{i=1}^{Y} \frac{\partial}{\partial w_o}(t_i - y_i)^2\end{aligned}$$

This time, we have to consider each term in the sum, since the output of the hidden layer node can contribute to errors at every output node, so the weight of an edge into the hidden node can also affect every term. Luckily, each term in the sum is independent, so we will focus on the derivate of one representative term and reconstruct the full sum afterwards. Taking $y_1$ as our representative, we want to find:

$$\begin{aligned}\frac{\partial}{\partial w_h}(t_1 - y_1)^2 &= 2(t_1 - y_1)\frac{\partial}{\partial w_h}(t_1 - y_1) \\ &= -2(t_1 - y_1)\frac{\partial}{\partial w_h}(y_1) \\ &= -2(t_1 - y_1)\frac{\partial}{\partial w_h}\sigma(\vec{w}_{o_1} \cdot \vec{a}_h) \\ &= -2(t_1 - y_1)\sigma(\vec{w}_{o_1} \cdot \vec{a}_h)(1 - \sigma(\vec{w}_{o_1} \cdot \vec{a}_h))\left[\frac{\partial}{\partial w_h}(\vec{w}_{o_1} \cdot \vec{a}_h)\right] \\ &= -2(t_1 - y_1)y_1(1 - y_1)\left[\frac{\partial}{\partial w_h}(\vec{w}_{o_1} \cdot \vec{a}_h)\right] \\ &= -2(t_1 - y_1)y_1(1 - y_1)\left[\frac{\partial}{\partial w_h}\sum_{i=1}^{|\vec{a}_h|} w_i a_i\right]\end{aligned}$$

The derivation so far has followed exactly the same procedure as equations 1–3 above, only that the partial derivative is with respect to $w_h$ instead of $w_o$. This sum over $\vec{a}_h$ includes all of the activations in the last hidden layer, but only one of these activations depends on $w_h$, so again we can simplify to

$$\frac{\partial}{\partial w_h}(t_1 - y_1)^2 = -2(t_1 - y_1)y_1(1 - y_1)\left[\frac{\partial}{\partial w_h}(w_o a_{h_1})\right]$$

but now $a_{h_1}$ depends on $w_h$, so we need to break it down further:

$$\begin{aligned}\frac{\partial}{\partial w_h}(t_1 - y_1)^2 &= -2(t_1 - y_1)y_1(1 - y_1)w_o\left[\frac{\partial}{\partial w_h}(a_{h_1})\right] && (6) \\ &= -2(t_1 - y_1)y_1(1 - y_1)w_o\left[\frac{\partial}{\partial w_h}\sigma(\vec{a}_i \cdot \vec{w}_h)\right] && (7) \\ &= -2(t_1 - y_1)y_1(1 - y_1)w_o\sigma(\vec{a}_i \cdot \vec{w}_h)(1 - \sigma(\vec{a}_i \cdot \vec{w}_h))\left[\frac{\partial}{\partial w_h}(\vec{a}_i \cdot \vec{w}_h)\right] \\ &= -2(t_1 - y_1)y_1(1 - y_1)w_o a_{h_1}(1 - a_{h_1})\left[\frac{\partial}{\partial w_h}(\vec{a}_i \cdot \vec{w}_h)\right]\end{aligned}$$

Equation 6 pulls out the $w_o$ term which does not depend on $w_h$. Equation 7 breaks apart the activation of the hidden node, showing its dependence on weights and activations from the previous layer. The remaining equations follow similar steps to those from before, but this time applied to the activation function of the hidden node (as opposed to the output node previously). Once again, only one element of the dot product depends on $w_h$, so we can simplify to:

$$\begin{aligned}\frac{\partial}{\partial w_h}(t_1 - y_1)^2 &= -2(t_1 - y_1)y_1(1 - y_1)w_o a_{h_1}(1 - a_{h_1})\left[\frac{\partial}{\partial w_h}(a_{i_1} w_h)\right] \\ &= -2(t_1 - y_1)y_1(1 - y_1)w_o a_{h_1}(1 - a_{h_1})a_{i_1}\end{aligned}$$

Recall that this was only one element of the sum of squared errors. Each other output node gives us a similar term, resulting in the following partial derivative of loss:

$$\begin{aligned}\frac{\partial \epsilon}{\partial w_h} &= \sum_{i=1}^{Y} \frac{\partial}{\partial w_o}(t_i - y_i)^2 \\ &= \sum_{o=1}^{Y} -2(t_o - y_o)y_o(1 - y_o)w_{(h1 \to o)} a_{h_1}(1 - a_{h_1})a_{i_1}\end{aligned}$$

Where $w_{(h1 \to o)}$ is the weight from hidden node $h_1$ to output node $o$. Substituting the already-computed output deltas gives

$$\frac{\partial \epsilon}{\partial w_h} = -2\sum_{o=1}^{Y} \delta_o w_{(h1 \to o)} a_{h_1}(1 - a_{h_1})a_{i_1}$$

Noting that only $a_{i_1}$ depends on our choice of $w_h$, we can gather the terms that will be the same for all other edges into the same hidden node into a hidden layer delta term, $\delta_h$:

$$\delta_h = a_{h_1}(1 - a_{h_1})\left(\sum_{o=1}^{Y} \delta_o w_{(h1 \to o)}\right)$$

So our weight update for any edge $i$ into this hidden layer becomes:

$$\begin{aligned}w_i' &= w_i - \alpha(-2)\delta_h a_i \\ &= w_i + \alpha \delta_h a_i\end{aligned}$$

We have now derived the stochastic gradient descent weight updates for edges into the output layer and the final hidden layer of the network. Note that the updates for the hidden layer depend on the final error of the network, but that the error terms do not appear in the update we have to perform. This dependence is captured entirely by the dependence on the output deltas $\delta_o$. While we haven't derived it, the same will be true of earlier layers in the network, and we will get the same formula for other layers' hidden deltas $\delta_h$, where the sum will be over the deltas from the next layer. Hopefully this has convinced you that the backpropagation update we are implementing is a correct stochastic gradient descent step for a network of densely-connected sigmoid nodes with sum of squared errors for the loss function.