Neural networks and Back Propagation

Hans Mathias Mamen Vege

December 4, 2018

1 Introduction

A short presentation on back propagation is given, in the derivation of the back propagation algorithm is given. This is done as an exercise as well as an future reference for the author. For a note on the notation in this short paper, consult the appendix A.

We begin with an short presentation of the layout of a neural network, then we will go through and present different cost functions and different output activations. A presentation of middle layer activations is also given. Then, we will begin deriving the back-propagation and the gradients which will be used to change the weights and biases in a neural network.

2 A neural network

A neural network, more specifically the *Multilayer Perceptron*, is a machine learning method which is takes some data as input and outputs some desired output based on a cost function and labeled training data. A typical would be some sort of classification, e.g. image classification.

A basic neural network consists of an input layer, and $L-1$ hidden layers, where L is the total number of layers in addition to an output layer. For each layer we have a number of neurons, which will take as input the previous layer output, multiply it by some weights and add some biases. Then, it will be send through some activation function and taken as input for the next layer. For a simple neural network with just an input layer one hidden layer we will have two weights and two biases. One for the hidden layer and one for the output layer. To illustrate,

$$\begin{aligned} \boldsymbol{a}^1 &= \sigma(\boldsymbol{z}^1) = \sigma(W^1\boldsymbol{x} + \boldsymbol{b}^1) && (1) \\ \hat{\boldsymbol{y}} &= \sigma_f(\boldsymbol{z}^2) = \sigma_f(W^2\boldsymbol{a}^1 + \boldsymbol{b}^2) && (2) \end{aligned}$$

The first equation (1) takes x which is the input data, and multiplies it with the weights matrix for the hidden layer W_1, and adds biases b_1. Then this is passed through some activation function σ. The output, $\hat{\boldsymbol{y}}$ will be the output of the neural network, and is equivalent to $\boldsymbol{a}^L$(or $\boldsymbol{a}^2$ for this case). From now on, $\boldsymbol{a}^L$ will be used for the output layer. The z_1 is simply a collective term for the input to the activation function, and is generalized as

$$\boldsymbol{z} = W\boldsymbol{x} + \boldsymbol{b} \tag{3}$$

To generalize this notation to any layer l, the output of a layer will be given as

$$\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l) = \sigma(W^l\boldsymbol{a}^{l-1} + \boldsymbol{b}^l) \tag{4}$$

This will be called the *forward pass* of a single layer.

If we extend this model a a full neural network, the final layer output will be,

$$\boldsymbol{a}^L = \sigma(W^L\boldsymbol{a}^{L-1} + \boldsymbol{b}^L) \tag{5}$$

Inserting the previous layer outputs, the full forward pass will be,

$$\boldsymbol{a}^L = \sigma(W^L\sigma(W^{L-1}\sigma(\cdots\sigma(W^1\boldsymbol{x} + \boldsymbol{b}^1)\cdots) + \boldsymbol{b}^{L-1}) + \boldsymbol{b}^L) \tag{6}$$

In order to make the network *learn*, we will have to update these weights and biases somehow. The way that is done is by first having a measure of how good the network is performing. This is done using a *cost function*, $\mathcal{C}$. We will end up taking the gradient of this function with respect to either the bias $\boldsymbol{b}^l$ or W^l,

$$\nabla^l_{j,W}\mathcal{C} = \frac{\partial \mathcal{C}}{\partial W^l_j}$$

and

$$\nabla^l_{j,\boldsymbol{b}}\mathcal{C} = \frac{\partial \mathcal{C}}{\partial b^l_j}$$

Note that we are taking the derivative component-wise. Let us start by investigating two cost functions which is widely used.

3 Cost functions

The cost function is a function which measures how good our network predicts an outcome based on some labeled outcome. We which wish to minimize, and in doing so will use stochastic gradient descent and update our weights based on the gradient descent. We will be looking at two main cost functions, the *quadratic cost*(or MSE cost function), or the *cross entropy.*

3.1 Quadratic cost

The quadratic cost, or mean square error (MSE) cost function is given by

$$\mathcal{C}_{\mathrm{MSE}} = \frac{1}{2N}\sum_{i=0}^{N}||\boldsymbol{a}_i - \boldsymbol{y}_i||_2^2, \tag{7}$$

where we take Euclidean(or L^2 norm) between the final layer output $\boldsymbol{a}$ minus the labeled training data $\boldsymbol{y}$. The sum i is over all of the training data, N. Note, that we will usually ignore the full cost function, since we are mostly dealing with one sample at a time. We then get

$$\mathcal{C}_{\mathrm{MSE}} = \frac{1}{2}||\boldsymbol{a}_i - \boldsymbol{y}_i||_2^2, \tag{8}$$

with no notational differences being given.

3.2 Cross entropy

The cross entropy cost function is given by

$$\mathcal{C}_{\mathrm{CE}} = -\frac{1}{N}\sum_{i=0}^{N}\sum_{k=1}^{K} y_{ik}\ln a_{ik} \tag{9}$$

This is the full cross entropy cost function, which is the average over all the samples N. The K-sum is a sum over all of the different labels. We will mostly simple use the expression without the explicit sum over all samples,

$$\mathcal{C}_{\mathrm{CE}} = -\sum_{k=1}^{K} y_k \ln a_k \tag{10}$$

This cost function has a nice expression in the binary classification case. We have that the sum of the output vector $\boldsymbol{y}$ has be equal to one when summed up(since it is a probability, the maximum can be one),

$$\sum_{k=0}^{K} y_k = 1,$$

we will in the binary classification case with $K = 2$, have

$$\mathcal{C}_{\mathrm{CE}}^{2} = -\sum_{k=1}^{K=2} y_k \ln a_k.$$

Writing this out, we get

$$\mathcal{C}_{\mathrm{CE}}^{2} = -y_1 \ln a_1 - y_2 \ln a_2$$

We can now rewrite y_2 given that the total probability is 1, as $y_2 = 1 - y_1$. We can then relabel $y_1 \to y$. The same can be done for the output.

$$\mathcal{C}_{\mathrm{CE}}^{2} = -y \ln a - (1-y)\ln(1-a). \tag{11}$$
$$\tag{12}$$

With these cost functions, we can begin at looking at the activation functions.

4 Activation functions

So far, we have not touched on activation functions. We will different between the output activation function, which will in the classification case be forced to be between 0 and 1, and the hidden layer activation functions.

4.1 Output layer activation functions

4.1.1 Sigmoidal activation function

The **sigmoid** activation function is given as,

$$\sigma_{\mathrm{sig}}(z) = \frac{1}{1+\mathrm{e}^{-z}} \tag{13}$$

The sigmoidal activation function can also be used as a hidden layer activation function. The derivative with respect to z of the sigmoidal activation function is easy,

$$\begin{aligned}
\frac{\partial}{\partial z}(\sigma_{\mathrm{sig}}(z)) &= \frac{\partial}{\partial z}\left(\frac{1}{1+\mathrm{e}^{-z}}\right) \\
&= (-\mathrm{e}^{-z})\frac{-1}{(1+\mathrm{e}^{-z})^2} \\
&= \sigma_{\mathrm{sig}}(z)\left(\frac{1-1+\mathrm{e}^{-z}}{1+\mathrm{e}^{-z}}\right) \\
&= \sigma_{\mathrm{sig}}(z)\left(1-\frac{1}{1+\mathrm{e}^{-z}}\right) \\
&= \sigma_{\mathrm{sig}}(z)\left(1-\sigma_{\mathrm{sig}}(z)\right)
\end{aligned}$$

And this, summed up we have

$$\sigma'_{\mathrm{sig}}(z) = \sigma_{\mathrm{sig}}(z)(1-\sigma_{\mathrm{sig}}(z)) \tag{14}$$

4.1.2 Softmax activation function

The **softmax** activation function is defined component-wise, and is given as

$$\sigma_{\mathrm{sm}}(z)_i = \frac{\mathrm{e}^{z_i}}{\sum_{k=1}^{K} \mathrm{e}^{z_k}} \tag{15}$$

The softmax activation function can only be used in the output layer[1]. Let us now find the derivative of the softmax function with respect to z.

$$\begin{aligned}
\frac{\partial}{\partial z_j}\left(\sigma_{\mathrm{sm}}(z)_i\right) &= \frac{\partial}{\partial z_j}\left(\frac{\mathrm{e}^{z_i}}{\sum_{k=1}^{K}\mathrm{e}^{z_k}}\right) \\
&= \begin{cases} \frac{\partial}{\partial z_j}\left(\frac{\mathrm{e}^{z_i}}{\sum_{k=1}^{K}\mathrm{e}^{z_k}}\right) & \text{if } j \neq i \\ \frac{\partial}{\partial z_j}\left(\frac{\mathrm{e}^{z_i}}{\sum_{k=1}^{K}\mathrm{e}^{z_k}}\right) & \text{if } j = i \end{cases} \\
&= \begin{cases} \frac{-\mathrm{e}^{z_j}\mathrm{e}^{z_i}}{\left(\sum_{k=1}^{K}\mathrm{e}^{z_k}\right)^2} & \text{if } j \neq i \\ \frac{\mathrm{e}^{z_j}}{\sum_{k=1}^{K}\mathrm{e}^{z_k}} - \frac{\mathrm{e}^{z_j}\mathrm{e}^{z_i}}{\left(\sum_{k=1}^{K}\mathrm{e}^{z_k}\right)^2} & \text{if } j = i \end{cases} \\
&= \begin{cases} -\sigma_{\mathrm{sm}}(z)_j\sigma_{\mathrm{sm}}(z)_i & \text{if } j \neq i \\ \sigma_{\mathrm{sm}}(z)_j - \sigma_{\mathrm{sm}}(z)_j\sigma_{\mathrm{sm}}(z)_i & \text{if } j = i \end{cases} \\
&= \sigma_{\mathrm{sm}}(z)_j\left(\delta_{ji} - \sigma_{\mathrm{sm}}(z)_i\right)
\end{aligned}$$

Which is, summing

$$\frac{\partial}{\partial z_j}\left(\sigma_{\mathrm{sm}}(z)_i\right) = \sigma_{\mathrm{sm}}(z)_j\left(\delta_{ji} - \sigma_{\mathrm{sm}}(z)_i\right) \tag{16}$$

4.2 Hidden layer activation functions

For hidden layer activation functions, we have several options. We have already looked at the sigmoidal function as an output layer activation function, and it can be used as a hidden layer activation function as well.

4.2.1 Hyperbolic tangens activation function

The **hyperbolic tangens** activation function is given as

$$\sigma_{\tanh}(z) = \tanh(z) \tag{17}$$

with its derivative

$$\sigma'_{\tanh}(z) = 1 - \tanh^2(z) \tag{18}$$

4.2.2 Relu activation function

The **relu** or rectifier activation is given as,

$$\sigma_{\mathrm{relu}}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{19}$$

with its derivative

$$\sigma'_{\mathrm{relu}}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{20}$$

[1]See this page for why softmax is commonly not used.

4.2.3 Heaviside activation function

The **Heaviside** activation function is given as

$$\sigma_{\text{Heaviside}}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{21}$$

with its derivative

$$\sigma'_{\text{Heaviside}}(z) = 0 \tag{22}$$

5 Training a network

As mentioned, we will train the network using *Stochastic Gradient Descent*(SGD) and *mini-batches*. The idea is to shuffle the data set, and randomly select a subset of elements in the dataset. We will then perform a forward-pass of the network, and using the output $\boldsymbol{a}^L$, perform back propagation, and update each weight and bias in the network. The main algorithm for training a network can be seen below,

Algorithm 1 Training a MLP

```
1: Initialize learning rate $\eta$.
2: Initialize with input data $\{\boldsymbol{x}_i\}$ and labels $\{\boldsymbol{y}_i\}$ of size $N$.
3: for $i_{\text{epoch}} = 1 : N_{\text{epochs}}$ do
4:     Shuffle data.
5:     Split data into $K_{\text{mb}}$ mini batches, $\{\boldsymbol{x}_{i_{\text{mb}}}\}$ and $\{\boldsymbol{y}_{i_{\text{mb}}}\}$.
6:     for $i_{\text{mb}} = 1 : N/K_{\text{mb}}$ do
7:         Perform a forward pass.
8:         Perform a back propagation, retrieving $\nabla W^l_{i_{\text{mb}}}$ and $\nabla b^l_{i_{\text{mb}}}$ for each $i_{\text{mb}}$.
9:     end for
10:    Average the $\nabla W^l_{i_{\text{mb}}}$ and $\nabla b^l_{i_{\text{mb}}}$, and update weights $W^l$ and biases $b^l$ with these averages.
11:    Update learning rate $\eta$ if needed.
12: end for
```

5.1 Learning rate

When updating the weights and biases with SGD, we did so by a learning rate parameter η. There are several way to define η, with the simplest one having $\eta =$ constant. Another option is one that is inversely decreasing as a function of the epochs. That is, for a given epoch i_{epoch} out a total N_{epochs}, we set the learning rate as

$$\eta(i_{\text{epoch}}) = \eta_0 \left(1 - \frac{i_{\text{epoch}}}{1 + N_{\text{epochs}}}\right) \tag{23}$$

This will force the step size to decrease toward 0 as we close in on the maximum number of epochs N_{epochs}.

5.2 Weight initialization

When initializing weights and biases, we will look at two ways of how this can be done. The first is through a Gaussian distribution, $(0, 1)$ which we will call *large*, as the biases will have large, spread-out distribution.

Then, we will use a Gaussian distribution but divided by the number of training samples, $(0, 1/N_{\text{train}})$, dubbing that one to be called *default*, as this is the one we will by default use in our neural network.

The effect of these two is essentially shrinking in the initial search space, and we should expect them to converge at large epoch times.

5.3 Measuring the performance

The performance of a neural network(or any classifier), can in its simplest form be measured by the accuracy, which is defined as

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n}, \tag{24}$$

where n is the number of samples we are testing against, I is the indicator function, which returns 1 if the prediction t_i equals the true values y_i.

6 Deriving the back propagation

We have now collected a bunch of tools, and are now ready to apply them to the back-propagation. We begin by imagining us to update the weights based on the gradient of the cost function. We begin with updating the output weights, W^L. Labeling the vectors and matrices, we get

$$\frac{\partial \mathcal{C}}{\partial W^L_{jk}} = \frac{\partial \mathcal{C}}{\partial a^L_j} \frac{\partial a^L_j}{\partial z^L_j} \frac{\partial z^L_j}{\partial W^L_{jk}} \tag{25}$$

We see that two of the products are component-wise, such that we actually are multiplying by the Hadamard product.

$$\frac{\partial \mathcal{C}}{\partial W^L} = \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{a}^L} \odot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \right) \frac{\partial \boldsymbol{z}^L}{\partial W^L} \tag{26}$$

We see that we need three derivatives, one for the cost function, one for the output activation layer and one for the z^L.

To sum up, we have to find three derivatives, the first being the of the cost function

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{a}^L}.$$

We then need to find the output activation function derivative.

$$\frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} = \frac{\partial}{\partial \boldsymbol{z}^L} \left(\sigma(\boldsymbol{z}^L) \right) \tag{27}$$

The final derivative, is given as

$$\frac{\partial \boldsymbol{z}^L}{\partial W^L} = \boldsymbol{a}^{L-1} \tag{28}$$

Let us no find the exact derivatives of the cost functions.

6.1 Quadratic cost derivative

We start with finding the derivative of the quadratic cost,

$$\begin{aligned} \frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{a}^L} &= \frac{\partial}{\partial \boldsymbol{a}^L} \left(\frac{1}{2} (\boldsymbol{a}^L - \boldsymbol{y})^2 \right) \\ &= \boldsymbol{a}^L - \boldsymbol{y}, \end{aligned}$$

which is summed up as,

$$\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{a}^L} = \boldsymbol{a}^L - \boldsymbol{y}. \tag{29}$$

6.2 Cross entropy derivative

The derivative of the cross entropy is given as,

$$\begin{aligned}\frac{\partial \mathcal{C}_{\mathrm{CE}}}{\partial \boldsymbol{a}^L} &= \frac{\partial}{\partial \boldsymbol{a}^L}\left(-\boldsymbol{y}\ln \boldsymbol{a}^L\right) \\ &= -\frac{\boldsymbol{y}}{\boldsymbol{a}^L}.\end{aligned}$$

Which, summed up is,

$$\frac{\partial \mathcal{C}_{\mathrm{CE}}}{\partial \boldsymbol{a}^L} = -\frac{\boldsymbol{y}}{\boldsymbol{a}^L}. \tag{30}$$

If we have a binary classification($K = 2$ classes), we can rewrite the expression as,

$$\begin{aligned}\frac{\partial \mathcal{C}^2_{\mathrm{CE}}}{\partial a^L} &= -\frac{\partial}{\partial a^L}\left(y\ln a^L - (1-y)\ln(1-a^L)\right) \\ &= \frac{1-y}{1-a^L} - \frac{y}{a^L} \\ &= \frac{a^L - y}{a^L(1-a^L)}\end{aligned}$$

6.3 Calculating the full back propagation

We have four different output(in reality only three[2]), which can be seen in 1.

Table 1: Different cost function activations. MSE is the same as quadratic cost.

Cost function	Output activation
MSE	Sigmoid
MSE	Softmax
CE	Softmax

For the derivations, we will mostly ignore that we are dealing with vectors and matrices, and instead bring it back in at the very end.

6.3.1 MSE and sigmoidal output activation

Let us then begin with looking at MSE and sigmoidal output activation. The change in the output layer L is given as,

$$\begin{aligned}\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial W^L} &= \left(\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial \boldsymbol{a}^L_{\mathrm{sig}}} \odot \frac{\partial \boldsymbol{a}^L_{\mathrm{sig}}}{\partial \boldsymbol{z}^L}\right)\frac{\partial \boldsymbol{z}^L}{\partial W^L} \\ &= \big(\underbrace{(\boldsymbol{a}^L - \boldsymbol{y})}_{\text{MSE}} \odot \underbrace{\boldsymbol{a}^L(1-\boldsymbol{a}^L)}_{\text{Sigmoid}}\big)\boldsymbol{a}^{L-1}.\end{aligned}$$

For the change in bias b^L, we get

$$\begin{aligned}\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial \boldsymbol{b}^L} &= \left(\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial \boldsymbol{a}^L_{\mathrm{sig}}} \odot \frac{\partial \boldsymbol{a}^L_{\mathrm{sig}}}{\partial \boldsymbol{z}^L}\right)\frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{b}^L} \\ &= \big(\underbrace{(\boldsymbol{a}^L - \boldsymbol{y})}_{\text{MSE}} \odot \underbrace{\boldsymbol{a}^L(1-\boldsymbol{a}^L)}_{\text{Sigmoid}}\big) \times 1.\end{aligned}$$

[2]For multiclass cross entropy output, sigmoidal activation as final layer is simply as special case of the softmax function. Thus, we will only use softmax activation for final layer output for the cross entropy.

We observe that the only change in output between the derivatives is $\boldsymbol{a}^{L-1}$.

If we now wish to find the change in the $L-1$ layer with respect to W^{L-1}, we get

$$\begin{aligned}
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^{L-1}} &= \frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{a}^L_{\text{sig}}} \frac{\partial \boldsymbol{a}^L_{\text{sig}}}{\partial \boldsymbol{z}^L} \frac{\partial \boldsymbol{z}^L}{\partial W^{L-1}} \\
&= \frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{a}^L_{\text{sig}}} \frac{\partial \boldsymbol{a}^L_{\text{sig}}}{\partial \boldsymbol{z}^L} \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \frac{\partial \boldsymbol{z}^{L-1}}{\partial W^{L-1}} \\
&= (\boldsymbol{a}^L - \boldsymbol{y})\boldsymbol{a}^L(1-\boldsymbol{a}^L)W^L\sigma'_{\text{sig}}(\boldsymbol{z}^{L-1})\boldsymbol{a}^{L-2}
\end{aligned}$$

Here, we recognize the first second as what we had for the first layer, so we can call that,

$$\delta^L = (\boldsymbol{a}^L - \boldsymbol{y})\boldsymbol{a}^L(1-\boldsymbol{a}^L) \tag{31}$$

Using this, for the $L-1$ layer we get

$$\begin{aligned}
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^L} &= \delta^L \boldsymbol{a}^{L-1} \\
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{b}^L} &= \delta^L
\end{aligned}$$

and for the $L-2$ layer, we get

$$\begin{aligned}
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^{L-1}} &= \delta^L W^L \sigma'_{\text{sig}}(\boldsymbol{z}^{L-1})\boldsymbol{a}^{L-2} \\
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{b}^{L-1}} &= \delta^L W^L \sigma'_{\text{sig}}(\boldsymbol{z}^{L-1}).
\end{aligned}$$

Looking at the derivative for layer $L-2$ with respect to W^{L-2} we get,

$$\begin{aligned}
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^{L-2}} &= \frac{\partial \mathcal{C}_{\text{MSE}}}{\partial \boldsymbol{a}^L_{\text{sig}}} \frac{\partial \boldsymbol{a}^L_{\text{sig}}}{\partial \boldsymbol{z}^L} \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \frac{\partial \boldsymbol{z}^{L-1}}{\partial \boldsymbol{a}^{L-2}} \frac{\partial \boldsymbol{a}^{L-2}}{\partial \boldsymbol{z}^{L-2}} \frac{\partial \boldsymbol{z}^{L-2}}{\partial W^{L-2}} \\
&= \delta^L W^L \sigma'_{\text{sig}}(\boldsymbol{z}^{L-1}) W^{L-1} \sigma'_{\text{sig}}(\boldsymbol{z}^{L-2})\boldsymbol{a}^{L-3}
\end{aligned}$$

Updating δ^L for the subsequent layers gives us

$$\begin{aligned}
\delta^{L-1} &= \delta^L W^L \sigma'_{\text{sig}}(\boldsymbol{z}^{L-1}) \\
\delta^{L-2} &= \delta^{L-1} W^{L-1} \sigma'_{\text{sig}}(\boldsymbol{z}^{L-2}).
\end{aligned}$$

Using these expressions, we can organize our previous expressions to

$$\begin{aligned}
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^{L-1}} &= \delta^{L-1} \boldsymbol{a}^{L-2} \\
\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^{L-2}} &= \delta^{L-2} \boldsymbol{a}^{L-3}.
\end{aligned}$$

We see a pattern, and can summarize our equations. Note that so far, we have mostly assumed that much of the matrix and vector multiplications are component wise, but will now include the full expression. Without specifying the sigmoidal function as the layer activation and writing the derivative of the cost function with respect to **a** as $\nabla_{\boldsymbol{a}^L}\mathcal{C}$, we get

$$\delta^L = \nabla^L_{\boldsymbol{a}}\mathcal{C} \odot \sigma'(\boldsymbol{a}^L) \tag{32}$$

$$\frac{\partial \mathcal{C}_{\text{MSE}}}{\partial W^L} = \delta^L (\boldsymbol{a}^{L-1})^T \tag{33}$$

$$\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial W^l} = \delta^L \tag{34}$$

For the subsequent layers, we get

$$\delta^{l-1} = \left(\left(W^l \right)^T \delta^l \right) \odot \sigma'(\boldsymbol{z}^{l-1}) \tag{35}$$

$$\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial W^l} = \delta^l \boldsymbol{a}^{l-1} \tag{36}$$

$$\frac{\partial \mathcal{C}_{\mathrm{MSE}}}{\partial W^l} = \delta^l \tag{37}$$

6.3.2 MSE and softmax output activation

For MSE and softmax as the output activation, we need only change the MSE the first two derivatives in equation (25). We use what we found in equation

$$\frac{\partial \mathcal{C}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = (a_j^L - y_j)\sigma_{\mathrm{sm}}(z)_j \left(\delta_{ji} - \sigma_{\mathrm{sm}}(z)_i\right)$$

To simplify, this becomes

$$\delta_j^L = (a_j^L - y_j)\sigma_{\mathrm{sm}}(z)_j \left(\delta_{ji} - \sigma_{\mathrm{sm}}(z)_i\right) \tag{38}$$

6.3.3 Cross entropy and softmax

We start by taking the derivative with respect to component j[3],

$$\frac{\partial \mathcal{C}_{\mathrm{CE}}}{\partial a_{j,\mathrm{SM}}^L} \frac{\partial a_{j,\mathrm{SM}}^L}{\partial z_j^L} = \sum_k \left(-\frac{y_k}{a_k^L} \right) a_i^L \left(\delta_{ik} - a_k^L \right)$$

Splitting the delta function into the case of $i \neq k$ and $i = k$,

$$\begin{aligned} \sum_k \left(-\frac{y_k}{a_k^L} \right) a_i^L \left(\delta_{ik} - a_k^L \right) &= -\frac{y_i}{a_i^L} a_i^L \left(1 - a_i^L \right) - \sum_{k \neq i} \frac{y_k}{a_k^L} \left(-a_i^L a_k^L \right) \\ &= -y_i + y_i a_i^L - \sum_{k \neq i} y_k (-a_i^L) = -y_i + \left(\sum_k y_k \right) a_i^L \\ &= a_i^L - y_i \end{aligned}$$

Summing up, we have

$$\frac{\partial \mathcal{C}_{\mathrm{CE}}}{\partial a_{j,\mathrm{SM}}^L} \frac{\partial a_{j,\mathrm{SM}}^L}{\partial z_j^L} = a_i^L - y_i \tag{39}$$

Appendices

A Notation

For notation, we will use bold italic as vectors, e.g. $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$. For matrices, we will use capital letters, W, B, X.

[3]An alternative derivation of the cross entropy with softmax as output layer can be found here

B Resources

- A derivation of Softmax for Cross Entropy.
- Another short derivation of Softmax for Cross Entropy on math stackexchange.
- Yet another derivation of Cross Entropy with softmax output.
- A good guide to neural networks.
- Wikipedia on Cross Entropy.
- CNN introduction.