# Homework 1

NE 795-001: Fall 2023

Arjun Earthperson

October 3rd, 2023

## Problem 1 - Backpropagation

The four fundamental equations of backpropagation are derived from the chain rule of calculus and are used to update the weights and bias in the network.

Let's denote:

- $a_j^l$ as the activation of the $j$-th neuron in the $l$-th layer
- $w_{jk}^l$ as the weight for the connection from the $k$-th neuron in the $(l-1)$-th layer to the $j$-th neuron in the $l$-th layer
- $b_j^l$ as the bias of the $j$-th neuron in the $l$-th layer
- $z_j^l$ as the weighted input of the $j$-th neuron in the $l$-th layer
- $\sigma$ as the activation function
- $C$ as the cost function
- $\delta_j^l$ as the error of the $j$-th neuron in the $l$-th layer

### Equation 1: Error of the Output Layer - $\delta^L$

The first equation of backpropagation is derived from the chain rule of calculus. It calculates the error of the output layer. We start with the definition of the error $\delta_j^L$ in the output layer $L$:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1}$$

The chain rule states that the derivative of a composite function is the derivative of the outer function times the derivative of the inner function. The cost function $C$ is a function of the activations $a_j^L$ of the output layer. Therefore, the partial derivative of the cost function with respect to the activations $\frac{\partial C}{\partial a_j^L}$ measures how much the cost changes when the activations change. The activation of a neuron is a function of the weighted input $z_j^L$, i.e., $a_j^L = \sigma(z_j^L)$. Therefore, the derivative of the activation function with respect to the weighted input $\sigma'(z_j^L)$ measures how much the activation changes when the weighted input changes. By the chain rule, the change in the cost with respect to the weighted input is the product of these two quantities:

$$\frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{2}$$

But this is exactly the definition of the error $\delta_j^L$ in the output layer:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} \tag{3}$$

Therefore, we have:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{4}$$

**Equation 2:**

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l)$$

The second fundamental equation of backpropagation is used to compute the error for any neuron in the network. It is given by:

1. The definition of the error $\delta^l$ in the lth layer:

$$\delta^l = \frac{\partial C}{\partial z^l} \tag{5}$$

2. The cost function C in terms of the weighted inputs $z^{l+1}$ of the next layer:

$$C = \frac{1}{2}\sum_j (a_j^{l+1} - y_j)^2 \tag{6}$$

3. Differentiating C with respect to $z^l$ using the chain rule:

$$\frac{\partial C}{\partial z^l} = \sum_j \frac{\partial C}{\partial a_j^{l+1}}\frac{\partial a_j^{l+1}}{\partial z^l} \tag{7}$$

4. The derivative of the activation $a_j^{l+1}$ with respect to the weighted input $z^l$:

$$\frac{\partial a_j^{l+1}}{\partial z^l} = \frac{\partial \sigma(z_j^{l+1})}{\partial z^l} = w_j^{l+1}\sigma'(z^l) \tag{8}$$

5. Substituting this into the previous equation:

$$\frac{\partial C}{\partial z^l} = \sum_j \frac{\partial C}{\partial a_j^{l+1}} w_j^{l+1}\sigma'(z^l) \tag{9}$$

6. The error $\delta_j^{l+1}$ of the jth neuron in the l+1 layer:

$$\delta_j^{l+1} = \frac{\partial C}{\partial a_j^{l+1}} \tag{10}$$

7. Substituting this into the previous equation to get the third fundamental equation of backpropagation:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{11}$$

**Equation 3:**

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

1. The cost function C is defined as:

$$C = \frac{1}{2} \sum (y - a)^2 \tag{12}$$

2. The error $\delta$ in the output layer is defined as:

$$\delta_j^l = \frac{\partial C}{\partial a_j^l} \cdot \sigma'(z_j^l) \tag{13}$$

3. The derivative of the cost function with respect to the actual output a is:

$$\frac{\partial C}{\partial a} = -(y - a) \tag{14}$$

4. Substituting this into the definition of $\delta$ gives:

$$\delta = -(y - a) \cdot \sigma'(z) \tag{15}$$

5. The bias b affects the cost function C through the actual output a, which in turn is a function of the weighted input z, which is a function of the bias b. Therefore, we can write:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} \tag{16}$$

6. The derivative of the weighted input z with respect to the bias b is 1, because z = wx + b, where w is the weight and x is the input. Therefore, we have:

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \tag{17}$$

7. Substituting this and the definition of $\delta$ into the above equation gives:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{18}$$

**Equation 4:**

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

The fourth fundamental equation of backpropagation is given by:

$$\frac{\partial C}{\partial w^{(l)}_{jk}} = a^{(l-1)}_j \delta^{(l)}_k \tag{19}$$

where $\delta^{(l)}_k$ is the error of the k-th neuron in the l-th layer, defined as:

$$\delta^{(l)}_k = \frac{\partial C}{\partial z^{(l)}_k} \tag{20}$$

The error $\delta^{(l)}_k$ can be computed using the first fundamental theorem of backpropagation, which states:

$$\delta^{(l)}_k = \frac{\partial C}{\partial a^{(l)}_k} \sigma'(z^{(l)}_k) \tag{21}$$

where $\sigma'(z^{(l)}_k)$ is the derivative of the activation function.
Substituting equation (3) into equation (1), we get:

$$\frac{\partial C}{\partial w^{(l)}_{jk}} = a^{(l-1)}_j \frac{\partial C}{\partial a^{(l)}_k} \sigma'(z^{(l)}_k) \tag{22}$$

This equation gives the rate of change of the cost function with respect to the weights in terms of the activations of the neurons and the derivative of the activation function.

## Problem 2 - Cross-entropy Cost Function

### Part (a)

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n}\sum_x a_k^{L-1}(a_j^L - y_j)$$

The cross-entropy cost function for a general many-layer multi-output-neuron artificial neural network (ANN) is given by:

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)]$$

where: - $C$ is the cost function, - $n$ is the total number of training examples, - $x$ is a single training example, - $y$ is the actual output, - $a$ is the predicted output (activation of the output layer), - $w_{jk}^L$ is the weight of the connection from the $k^{th}$ neuron in the $(L-1)^{th}$ layer to the $j^{th}$ neuron in the $L^{th}$ layer.

We want to compute the partial derivative of $C$ with respect to $w_{jk}^L$, denoted as $\frac{\partial C}{\partial w_{jk}^L}$.

First, we need to compute the derivative of $C$ with respect to $a$, denoted as $\frac{\partial C}{\partial a}$:

$$\frac{\partial C}{\partial a} = -\frac{1}{n}\sum_x \left[\frac{y}{a} - \frac{1-y}{1-a}\right]$$

Next, we need to compute the derivative of $a$ with respect to the weighted input $z_j^L$, denoted as $\frac{\partial a}{\partial z_j^L}$. The activation function of the output layer is usually the sigmoid function, denoted as $\sigma(z)$, where $a = \sigma(z_j^L)$. The derivative of the sigmoid function is $\sigma'(z) = \sigma(z)(1-\sigma(z))$, so:

$$\frac{\partial a}{\partial z_j^L} = \sigma'(z_j^L) = a(1-a)$$

Then, we need to compute the derivative of $z_j^L$ with respect to $w_{jk}^L$, denoted as $\frac{\partial z_j^L}{\partial w_{jk}^L}$. The weighted input is given by $z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$, so:

$$\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$$

Finally, we can compute $\frac{\partial C}{\partial w_{jk}^L}$ using the chain rule:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial C}{\partial a} \cdot \frac{\partial a}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{jk}^L}$$

Substituting the above results, we get:

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\sum_x \left[\frac{y}{a} - \frac{1-y}{1-a}\right] \cdot a(1-a) \cdot a_k^{L-1}$$

This is the derivative of the cost function with respect to the weight $w_{jk}^L$. It tells us how much the cost changes when we change this weight. We can use this derivative to update the weight during the training of the ANN using gradient descent.

We can simplify the term inside the square brackets using the fact that $a(1-a) = a - a^2$:

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\sum_x [y(1-a) - a(1-y)] \cdot a_k^{L-1}$$

This simplifies to:

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\sum_x [y - ya - a + ay] \cdot a_k^{L-1}$$

Further simplifying, we get:

$$\frac{\partial C}{\partial w_{jk}^L} = -\frac{1}{n}\sum_x [y - a] \cdot a_k^{L-1}$$

Finally, we can remove the negative sign by swapping the terms inside the square brackets:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n}\sum_x [a - y] \cdot a_k^{L-1}$$

This is equivalent to the starting equation:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{1}{n}\sum_x a_k^{L-1}(a_j^L - y_j)$$

**Part (b)**

$$\frac{\partial C}{\partial b_j^l} = \frac{1}{n}\sum_x [a_j^L - y_j]$$

We want to compute the partial derivative of $C$ with respect to $b_j^L$, denoted as $\frac{\partial C}{\partial b_j^L}$.

First, we need to understand that the activation $a$ of the output layer is a function of the weighted inputs $z_j^L$ of the neurons in the last layer, which in turn are functions of the biases $b_j^L$. Therefore, we can apply the chain rule of calculus to compute the derivative:

$$\frac{\partial C}{\partial b_j^L} = \frac{\partial C}{\partial a} \cdot \frac{\partial a}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial b_j^L}$$

Let's compute each of these derivatives separately:

1. $\frac{\partial C}{\partial a}$:

$$\frac{\partial C}{\partial a} = -\frac{1}{n}\sum_x \left[\frac{y}{a} - \frac{1-y}{1-a}\right]$$

2. $\frac{\partial a}{\partial z_j^L}$:

This derivative depends on the activation function used in the output layer. If we denote the activation function as $f$, then $a = f(z_j^L)$, and the derivative is the derivative of this function. For example, if the sigmoid function is used, then $\frac{\partial a}{\partial z_j^L} = a(1-a)$.

3. $\frac{\partial z_j^L}{\partial b_j^L}$:

The weighted input $z_j^L$ is given by $z_j^L = \sum_i w_{ji}^L a_i^{L-1} + b_j^L$, where $w_{ji}^L$ are the weights and $a_i^{L-1}$ are the activations of the previous layer. Therefore, the derivative of $z_j^L$ with respect to $b_j^L$ is 1.

Finally, substituting these results back into the chain rule gives:

$$\frac{\partial C}{\partial b_j^L} = -\frac{1}{n}\sum_x \left[\frac{y}{a} - \frac{1-y}{1-a}\right] \cdot a(1-a) \cdot 1$$

This is the derivative of the cost function with respect to the bias of the $j^{th}$ neuron in the last layer for a general many-layer multi-output-neuron ANN using the cross-entropy cost function. The result is a simplified version of the derivative of the cost function with respect to the bias of the $j^{th}$ neuron in the $l^{th}$ layer. Here, $l = L$ (the last layer), and $\delta_{lj}$ is the Kronecker delta, which is 1 when $l = j$ and 0 otherwise.

$$\frac{\partial C}{\partial b_j^l} = \frac{1}{n}\sum_x [a_j^L - y_j]\delta_{lj}$$

This expression can be derived from the previous result by stating that for the cross-entropy cost function and sigmoid activation function, the term

$$-\frac{1}{n}\sum_x \left[\frac{y}{a} - \frac{1-y}{1-a}\right] \cdot a(1-a)$$

simplifies to

$$\frac{1}{n}\sum_x [a_j^L - y_j]$$

This simplification occurs because the derivative of the sigmoid function $\sigma(z)$ is

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

and for the output layer, $a_j^L = \sigma(z_j^L)$. Therefore,

$$a_j^L(1-a_j^L) = \sigma(z_j^L)(1-\sigma(z_j^L)) = \sigma'(z_j^L)$$

Substituting this into the derivative of the cost function gives

$$\frac{\partial C}{\partial b_j^L} = \frac{1}{n}\sum_x [a_j^L - y_j]$$

The $\delta_{lj}$ term ensures that the derivative is computed only for the bias of the $j^{th}$ neuron in the last layer, as the derivative of the cost function with respect to the biases of the neurons in the other layers is 0.

## Problem 3 - Softmax Activation Function

The softmax activation function is defined as:

$$a_j^L = \frac{e^{z_j^L}}{\sum_{k=1}^K e^{z_k^L}}$$

where $a_j^L$ is the activation of the j-th neuron in the output layer, $z_j^L$ is the input to the j-th neuron in the output layer, and K is the total number of neurons in the output layer.

We want to derive $\frac{\partial a_j^L}{\partial z_k^L}$, which is the derivative of the activation of the j-th neuron with respect to the input of the k-th neuron.

There are two cases to consider:

1. When $j = k$, we have:

$$\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial}{\partial z_j^L}\left(\frac{e^{z_j^L}}{\sum_{k=1}^K e^{z_k^L}}\right)$$

Using the quotient rule for differentiation, we get:

$$\frac{\partial a_j^L}{\partial z_j^L} = \frac{e^{z_j^L}\sum_{k=1}^K e^{z_k^L} - e^{z_j^L}e^{z_j^L}}{(\sum_{k=1}^K e^{z_k^L})^2}$$

Simplifying, we get:

$$\frac{\partial a_j^L}{\partial z_j^L} = a_j^L(1 - a_j^L)$$

2. When $j \neq k$, we have:

$$\frac{\partial a_j^L}{\partial z_k^L} = \frac{\partial}{\partial z_k^L}\left(\frac{e^{z_j^L}}{\sum_{k=1}^K e^{z_k^L}}\right)$$

Using the quotient rule for differentiation, we get:

$$\frac{\partial a_j^L}{\partial z_k^L} = \frac{-e^{z_j^L}e^{z_k^L}}{(\sum_{k=1}^K e^{z_k^L})^2}$$

Simplifying, we get:

$$\frac{\partial a_j^L}{\partial z_k^L} = -a_j^L a_k^L$$

From these two cases, we can see that $\frac{\partial a_j^L}{\partial z_k^L}$ is always non-negative when $j = k$ and always non-positive when $j \neq k$. Therefore, $a_j^L$ increases monotonically with $z_j^L$.

## Problem 4 - Negative Log-likelihood Cost Function

The softmax function is used in the output layer of a neural network, converting raw output scores into probabilities that sum up to 1. The softmax function $S$ for a vector $z$ is defined as:

$$S(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

where $z_j$ is the $j$-th element of the vector $z$, and $K$ is the total number of classes.

The NLL cost function is used to measure the error between the predicted probabilities (from softmax) and the true labels. For a single data point, the NLL cost $C$ is defined as:

$$C = -\log(S(z)_y)$$

where $y$ is the true class label.

To understand how the combination of softmax and NLL can avoid learning slowdown, we need to compute the derivative of the NLL cost with respect to the raw scores $z$. Using the chain rule, we have:

$$\frac{\partial C}{\partial z_j} = \frac{\partial C}{\partial S(z)_j} \cdot \frac{\partial S(z)_j}{\partial z_j}$$

The derivative of the NLL cost with respect to $S(z)_j$ is:

$$\frac{\partial C}{\partial S(z)_j} = -\frac{1}{S(z)_j}$$

The derivative of the softmax function with respect to $z_j$ is:

$$\frac{\partial S(z)_j}{\partial z_j} = S(z)_j \cdot (1 - S(z)_j)$$

Therefore, the derivative of the NLL cost with respect to the raw scores is:

$$\frac{\partial C}{\partial z_j} = -\frac{1}{S(z)_j} \cdot S(z)_j \cdot (1 - S(z)_j) = -(1 - S(z)_j)$$

The derivative of the cost with respect to the raw scores is used to update the weights in the neural network during training. The derivative is not dependent on the cost itself, but only on the output of the softmax function. This means that even if the cost is very small (which usually happens as the training progresses), the derivative will not become very small. Therefore, the weights can still be updated with a reasonable step size, avoiding the learning slowdown.

Next, the derivative of the cost function C with respect to the weight $w_{jk}^L$ in the last layer L of the neural network can be computed using the chain rule. First, let's define the net input $z_j^L$ to the neuron j in the last layer L as:

$$z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$$

where $a_k^{L-1}$ is the activation of neuron k in the layer L-1 and $b_j^L$ is the bias of neuron j in the layer L. The derivative of the cost function C with respect to the weight $w_{jk}^L$ is then:

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial C}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{jk}^L}$$

We have already computed the derivative of the cost function C with respect to the net input $z_j^L$ in the previous part:

$$\frac{\partial C}{\partial z_j^L} = -(1 - S(z^L)_j)$$

The derivative of the net input $z_j^L$ with respect to the weight $w_{jk}^L$ is simply the activation of the neuron k in the layer L-1:

$$\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$$

Therefore, the derivative of the cost function C with respect to the weight $w_{jk}^L$ is:

$$\frac{\partial C}{\partial w_{jk}^L} = -(1 - S(z^L)_j) \cdot a_k^{L-1}$$

This derivative is used to update the weight $w_{jk}^L$ during the backpropagation step of the training process.

**Part (a) $\frac{\partial C}{\partial w_{jk}^L}$**

We know that $S(z^L)_j$ is the softmax activation function, which is equivalent to $a_j^L$ (the activation of the j-th neuron in the last layer). So, we can rewrite the above equation as:

$$\frac{\partial C}{\partial w_{jk}^L} = -(1 - a_j^L) \cdot a_k^{L-1}$$

In the case of Negative Log-Likelihood (NLL) cost function for classification problems, the target output $y_j$ for the correct class is 1, and 0 for all other classes. So, $(1 - a_j^L)$ is actually equivalent to $(a_j^L - y_j)$, because when $j$ is the correct class, $y_j = 1$ and when $j$ is not the correct class, $y_j = 0$.

Therefore, we can further simplify the equation to:

$$\frac{\partial C}{\partial w_{jk}^L} = (a_j^L - y_j) \cdot a_k^{L-1}$$

## Part (b) $\frac{\partial C}{\partial b_j^L}$

Finally, the derivative of the cost function C with respect to the bias $b_j^L$ is then:

$$\frac{\partial C}{\partial b_j^L} = \frac{\partial C}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial b_j^L}$$

We have already computed the derivative of the cost function C with respect to the net input $z_j^L$ in the previous part:

$$\frac{\partial C}{\partial z_j^L} = -(1 - S(z^L)_j)$$

The derivative of the net input $z_j^L$ with respect to the bias $b_j^L$ is simply 1, because the bias term is added directly to the net input and its coefficient is 1:

$$\frac{\partial z_j^L}{\partial b_j^L} = 1$$

Therefore, the derivative of the cost function C with respect to the bias $b_j^L$ is:

$$\frac{\partial C}{\partial b_j^L} = -(1 - S(z^L)_j)$$

Again, we know that $S(z^L)_j$ is the softmax activation function, which is equivalent to $a_j^L$ (the activation of the j-th neuron in the last layer). So, we can rewrite the above equation as:

$$\frac{\partial C}{\partial b_j^L} = -(1 - a_j^L)$$

In the case of Negative Log-Likelihood (NLL) cost function for classification problems, the target output $y_j$ for the correct class is 1, and 0 for all other classes. So, $(1 - a_j^L)$ is actually equivalent to $(a_j^L - y_j)$, because when $j$ is the correct class, $y_j = 1$ and when $j$ is not the correct class, $y_j = 0$.

Therefore, we can further simplify the equation to:

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j$$

This derivative is used to update the bias $b_j^L$ during the backpropagation step of the training process.