Neural networks and Back Propagation

Hans Mathias Mamen Vege

December 2, 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,

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

$$\hat{\boldsymbol{y}} = \sigma_f(\boldsymbol{z}^2) = \sigma_f(W^2\boldsymbol{a}^1 + \boldsymbol{b}^2) \tag{2}$$

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,

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

With these basics in place, we can now start to consider how one measures how good the network is at prediction.

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

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

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

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

with no notational differences being given.

3.2 Cross entropy

The cross entropy cost function is given by

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

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,

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

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

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

Writing this out, we get

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

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.

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

$$\tag{12}$$

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,

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

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 will be given by,

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

4.1.2 Softmax activation function

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

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

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.

$$\frac{\partial}{\partial z_j}\sigma_{\text{sm}}(z)_i = \frac{\partial}{\partial z_j}\left(\frac{\text{e}^{z_i}}{\sum_{k=1}^{K}\text{e}^{z_k}}\right)$$

4.2 Hidden layer activation functions

For hidden layer activation functions, we have several options. We have already looked at the

4.2.1 Hyperbolic tangens activation function

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

$$\sigma_{\tanh}(z) = \tanh(z) \tag{16}$$

with its derivative

$$\sigma'_{\tanh}(z) = 1 - \tanh^2(z) \tag{17}$$

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

4.2.2 Relu activation function

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

$$\sigma_{\text{relu}}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{18}$$

with its derivative

$$\sigma'_{\text{relu}}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{19}$$

4.2.3 Heaviside activation function

The **Heaviside** activation function is given as

$$\sigma_{\text{Heaviside}}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{20}$$

with its derivative

$$\sigma'_{\text{Heaviside}}(z) = 0 \tag{21}$$

5 Deriving the back propagation

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.