Fundamentals of Neural Networks

Atharva Tambat

31 August 2022

Contents

1 Introduction

This is the report of Assignment-1 of the course GNR 638 offered in the autumn semester of '22 in IIT Bombay, by Prof. Biplab Banerjee. In the coding part, we train a neural network consisting of three layers each having 2 nodes. Here, we give the derivations of both the forward and backward propagation for all the parameters. So without further ado, let's start exploring.

2 Setting up the Neural Network

In this section, we give an overview of the model and list the various parameters involved in making the same.

After importing the libraries, we start sampling the input data from the given two Gaussian Distributions as instructed. We then add labels to this sampled data and club the two into a single dataset namely 'dataset'. To generate the same dataset every time, we have set the seed to 10. We have also provided a scatter plot to better visualise the data.

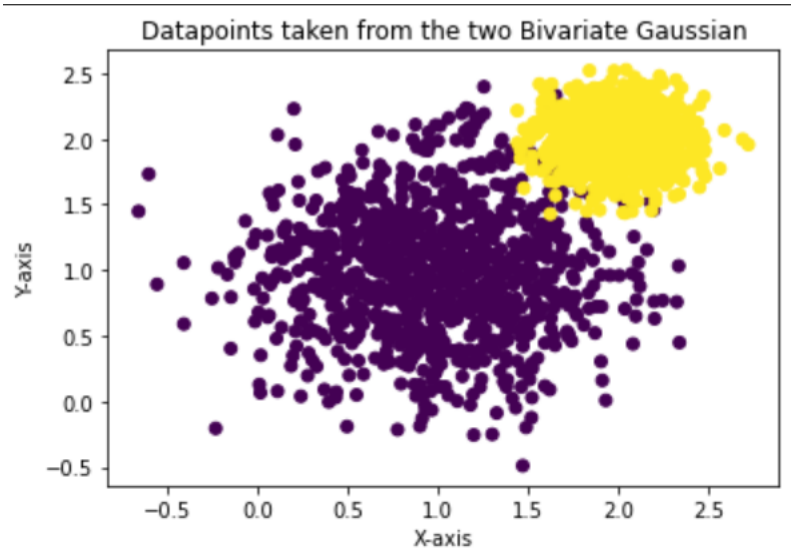

Figure 1: Scatter Plot

Next, we create the weight matrices and bias vectors namely, $W1$, $W2$, $B1$ and $B2$ whose dimensions are given by the node count in the input, hidden and the output layers.

3 Forward Propagation

We start by defining the tanh and softmax functions which are given by,

$$tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1}$$

$$softmax(x) := \frac{e^x}{\sum_{i=0}^{size(X)-1} e^{X[i]}} \tag{2}$$

Now, we state the forward propagation for the hidden layer(M) followed by the output layer(Y),

$$M := tanh(W1.X + B1) \tag{3}$$

$$Y := softmax(W2.M + B2) \tag{4}$$

With the help of forward propagation, we get the output for a given input. But, to train the neural network we need to implement something known as 'Gradient Descent' which is done via Backward Propagation. Hence, we move to the next section.

4 Backward Propagation

The loss function for the model is the sum of Mean Squared Error and Cross Entropy Loss, the formula for the two are given as follows

$$MSE\_Loss = \frac{\sum_{i=1}^{N}(y - \hat{y})^2}{2N} \tag{5}$$

$$CE\_Loss = -\sum_{i=1}^{N} y.ln(\hat{y}) \tag{6}$$

where, N is the node count, y is the true output, $\hat{y}$ is the predicted output, MSE_Loss stands for Mean Squared Error and CE_Loss stands for Cross Entropy Loss. Hence, putting all this together, we get the overall Loss Function for the model as

$$LossFunction(L) = MSE\_Loss + CE\_Loss \tag{7}$$

In Gradient Descent, we update the weights and bias in every iteration of training given by the following set of equations

$$
\begin{aligned}
W1 &= W1 - \alpha \times \frac{\partial L}{\partial W1} \\
W2 &= W2 - \alpha \times \frac{\partial L}{\partial W2} \\
B1 &= B1 - \alpha \times \frac{\partial L}{\partial B1} \\
B2 &= B2 - \alpha \times \frac{\partial L}{\partial B2}
\end{aligned} \tag{8}
$$

where, L is the loss function for the model and α being the learning rate, which in our case is initialized to 10^{-6}.

So, all we need is to calculate the partial derivative of the loss function with respect to the weights and bias. This is exactly where we need Backward Propagation.

We start by taking only the MSE_Loss for the time being.

$$
\begin{aligned}
\frac{\partial L}{\partial W2} &= \frac{\partial L}{\partial \hat{Y}} . \frac{\partial \hat{Y}}{\partial W2} \\
\frac{\partial L}{\partial \hat{Y}} &= \frac{\hat{Y} - Y}{N} \\
\frac{\partial \hat{Y}}{\partial W2} &= \frac{\partial (softmax(M^T.W2 + B2))}{\partial W2} \\
\implies \frac{\partial \hat{Y}}{\partial W2} &= (\hat{Y} - \hat{Y}^2).M^T \\
\implies \frac{\partial L}{\partial W2} &= \frac{(\hat{Y} - Y)(\hat{Y} - \hat{Y}^2).M^T}{N}
\end{aligned} \tag{9}
$$

So, now we have the partial derivative of the MSE_Loss wrt W2. Calculating the same for W1, we have

$$
\begin{aligned}
\frac{\partial L}{\partial W1} &= \frac{\partial L}{\partial M}.\frac{\partial M}{\partial W1} \\
\frac{\partial L}{\partial M} &= \frac{\partial L}{\partial \hat{Y}}.\frac{\partial \hat{Y}}{\partial M} \\
\frac{\partial L}{\partial \hat{Y}} &= \frac{\hat{Y}-Y}{N} \\
\frac{\partial \hat{Y}}{\partial M} &= (\hat{Y}-\hat{Y}^2).W2^T \\
\implies \frac{\partial L}{\partial M} &= \frac{(\hat{Y}-Y)(\hat{Y}-\hat{Y}^2).W2^T}{N} \\
\frac{\partial M}{\partial W1} &= \frac{\partial(tanh(X^T.W1+B1))}{\partial W1} \\
\implies \frac{\partial M}{\partial W1} &= (1-M^2).X^T \\
\implies \frac{\partial L}{\partial W1} &= \frac{(\hat{Y}-Y)(\hat{Y}-\hat{Y}^2).W2^T(1-M^2).X^T}{N}
\end{aligned}
\tag{10}
$$

So, now we have the derivative of the MSE_Loss wrt both the weight matrices. We do the same for the CE_Loss.

The categorical cross entropy loss is:

$$J = -\sum_m y_m \cdot ln(a_m)$$

The activation of the n^{th} neuron in the last layer being:

$$a_n = softmax(z_n) = \frac{e^{z_n}}{\sum_m e^{z_m}}$$

By chain rule, we write:

$$\delta_n = \frac{\partial J}{\partial z_n} = \sum_m \frac{\partial J}{\partial a_m} \cdot \frac{\partial a_m}{\partial z_n}$$

Simplifying things a bit:

$$\delta_n = \frac{\partial J}{\partial z_n} = \sum_{m \neq n} \frac{\partial J}{\partial a_m} \cdot \frac{\partial a_m}{\partial z_n} + \frac{\partial J}{\partial a_n} \cdot \frac{\partial a_n}{\partial z_n}$$

$$\frac{\partial J}{\partial a_n} = -\frac{y_n}{a_n}$$

For the case m = n:

$$\frac{a_n}{z_n} = a_n(1-a_n)$$

For the case $m \neq n$:

$$\frac{a_m}{z_n} = -a_m \cdot a_n$$

Multiplying both the results:

$$\delta_n = \sum_{m \neq n} y_n a_n + -y_n(1 - a_n)$$

Which can be simplified to:

$$\delta_n = a_n \sum_m y_n + -y_n$$

Since,

$$\sum_m y_m = 1$$

We have,

$$\delta_n = a_n - y_n \tag{11}$$

This means that each component of the δ vector is the n^{th} component of the $\hat{y}$ vector - n^{th} component of the y vector.

Since we have to calculate $\frac{\partial L}{\partial w_2}$, and, since $Y = W2^T M + B2$ (where $W2$ and $B2$ are the weights and biases between hidden and output layer, only a factor of $M^T (= \frac{\partial Y}{\partial W2})$ will be multiplied with the above calculated derivative (M is the output of the hidden layer).

Therefore,

$$\frac{\partial L}{\partial W2} = M^T \cdot (\hat{Y} - Y) \tag{12}$$

where, $\hat{Y}$ is the output of the model and Y is the one hot label.

Thus, we get the derivative of the CE_Loss wrt W2. For calculating the derivative of the loss function wrt to the weights in hidden layer, notice the following results:

$$\frac{\partial L}{\partial z_n} = a_n - y_n$$

where, a_n and y_n are as defined above.

$$\frac{\partial z_n}{\partial M} = W_2^T$$

where M is the output of the hidden layer.

Also,

$$\frac{\partial M}{\partial W1} = X^T \cdot (derivative\_of\_tanh(M))$$

$$\frac{\partial M}{\partial W1} = X^T \cdot (1 - M^2)$$

where X is the input to the neural network.

Multiplying the three derivatives together, we get:

$$\frac{\partial L}{\partial W1} = X \cdot (\hat{Y} - Y) \cdot W2^T \cdot (1 - M^2) \tag{13}$$

Finally, we have the derivative of the CE_Loss wrt both the weight matrices. For biases, observe that the calculation of derivative wrt biases is similar to the derivation for the weights of the respective layers, except multiplying at the end X^T or M^T for the respective layers. Since the function associated in Forward Propagation is $X^T W + B$, by chain rule it follows that the derivatives of L wrt W and B should only differ by X^T or M^T.

5 Plot

After all the gradient calculations are done, we are all set to implement Gradient Descent. In every iteration of training, the weights and biases tend to change in such a way that leads to the decrease of the overall loss function, which is as evident from the plot provided.

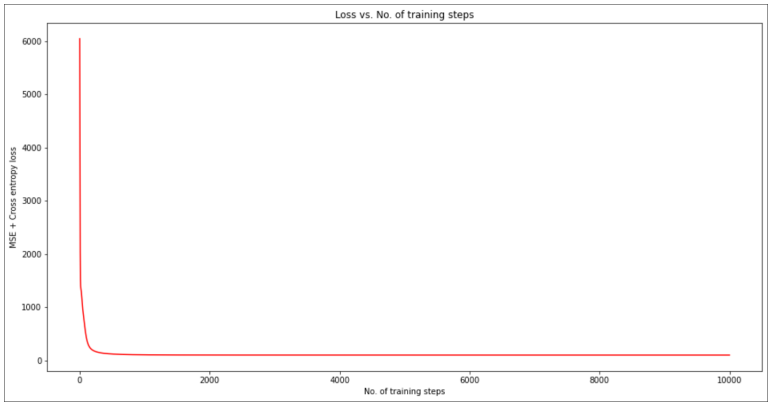

Figure 2: Loss Function

Also, we have plotted the weights and biases in each iteration to give an idea of how these variables are changing.

6 Google Colab Link

The Google Colab on which this Neural Network has been implemented can be found **here**.