COSC525: Project 1

Harshvardhan, Yu Jiang[*]

February 13, 2023

1 Introduction to the Problem

Our goal in this project is to build a basic artificial neural network (ANN) library using only `numpy` library. We apply this model to forecast the logic `AND` and `XOR`. We also examine the impact of the loss function, activation, and hyperparameter (learning rate) on the model.

2 Assumptions and Choices Made

2.1 Activation Functions

In this project, our library can support both the logistic and linear functions. We can have a method called `activationderivative` method to compute the activation function's derivative with respect to a neuron's input net.

2.1.1 Logistic Activation Function

The logistic activation function, also known as the sigmoid activation function, is a common nonlinear activation function. It can be defined as follows:

$$\varphi(x) = \frac{1}{1+e^{-x}}$$

The logistic function maps any real-valued input to a value between 0 and 1, making it useful for binary classification problems. The objective is to predict a binary output, 0 or 1.

2.1.2 Linear Activation Function

The linear activation function is a simple and linear function of form defined below:

$$\varphi(x) = x$$

The linear activation function simply passes its input value to its output unchanged. It is often utilized in the output layer of a neural network for regression problems. The goal is to predict a continuous output value.

[*]Business Analytics and Statistics, Haslam College of Business, University of Tennessee, Knoxville.

2.2 Loss Functions

In this project, our library can support two loss functions: square error and binary cross entropy loss.

2.2.1 Square Error Loss Function

The square error loss function, also known as mean squared error (MSE) loss, is a popular loss function for regression problems. It is defined as the average squared difference between the predicted output of a model and the actual output.

$$E = \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2$$

2.2.2 Binary Cross Entropy Loss Function

The binary cross entropy loss function is widely used in binary classification problems. It measures the difference between the true label (either 0 or 1) and the predicted probability of the positive class (i.e., the class with label 1). It is defined as:

$$E = \frac{1}{N}\sum_i -[y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})]$$

2.3 Weight Update

The equations of backpropagation for weight update can be defined as follows.

$$\delta^{L_j} = \frac{\partial E_{total}}{\partial out_{L_j}} \times \frac{\partial out_{L_j}}{\partial net_{L_j}}$$

$$\delta^{l_j} = \sum_k w_{l_j l+1_k} \times \delta^{l+1_k} \times \frac{\partial out_{l_j}}{\partial net_{l_j}}$$

$$\frac{\partial E_{total}}{\partial w_{l_j l+1_k}} = \delta^{l+1_k} \times out_{l_j}$$

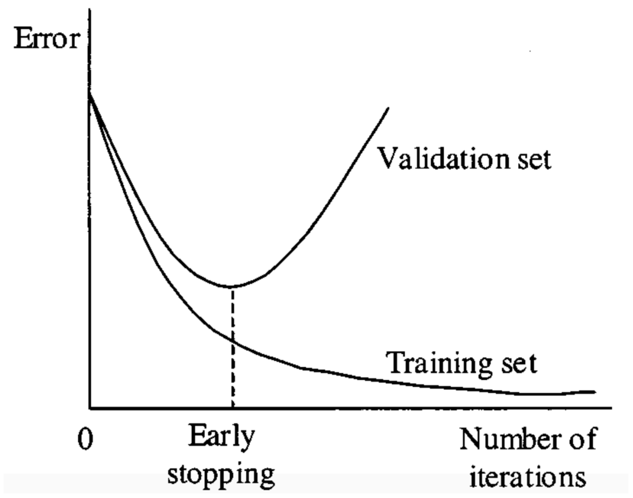
$$w_i = w_i - \alpha \frac{\partial E}{\partial w_i}$$

To update the weights in a neural network, each neuron computes its own δ by multiplying the derivative of the activation function with the error signal. The neuron then sends the vector of $w\delta$ to the `FullyConnectedLayer`. The layer collects these vectors from all the neurons, sums them up, and returns the resulting vector to the `NeuralNetwork`. The `NeuralNetwork` uses this vector to update the weights of the network

2.4 Early Stopping

Early stopping is a technique used to prevent the model from being overfit to the training data and to enhance its generalization performance. It can be explained by the Fig.1 elaborately.

The training stops when parameter updates cannot improve the outcome on a validation set. In our project, we define that if the change in the loss function is less than 10^{-8}, the iteration process can be stopped.



Stopping.png

Figure 1: Early Stopping

3 Problems and Limitations

We found that our results depended on the starting points for finding the best weights. If we provided a good initial points, it worked well. However, if the quality of initial weights weren't close to the optimal solution, our solution depended on the value of learning rate. If the learning rate was closer to the optimal, then everything worked perfectly. Else, we would have suboptimal results.

We have not chosen to use the command line to execute scripts and programs on this project. Our programming expertise wasn't deep enough to work with command line programs and we believe Jupyter Notebooks provided a better alternative. Jupyter Notebook allows you to create a document that contains both code and rich text elements, such as paragraphs, equations, images, and visualizations — like loss function plots in our case. This makes it easier to share your work with others and to keep track of the steps you have taken during your analysis.

Since the results do not depend on the interface, it is not a major or breaking change.

4 Process of Running Code

Our code can be executed via the function: `execute_neural_net`. This function takes two arguments: learning rate as `learnin_rate` and problem type as `problem_type`. The learning rate is the initial learning rate to start from. The problem type could be one of `example`, `xor` or `and`.

The implementation of classes in this context is differentiated solely by the execution of appropriate codes, contingent upon the inputs provided. We do not implement the main function, as our notebook code has to be executed using a Notebook interface.

5 Impact of Learning Rate

In this section, we depict the relationship between loss and learning rate. The purpose of these graphs is to demonstrate how the loss changes as the learning rates are varied. This information can help us to select the optimal learning rate for our machine-learning model.

vs Learning Rate for Example Problem.png

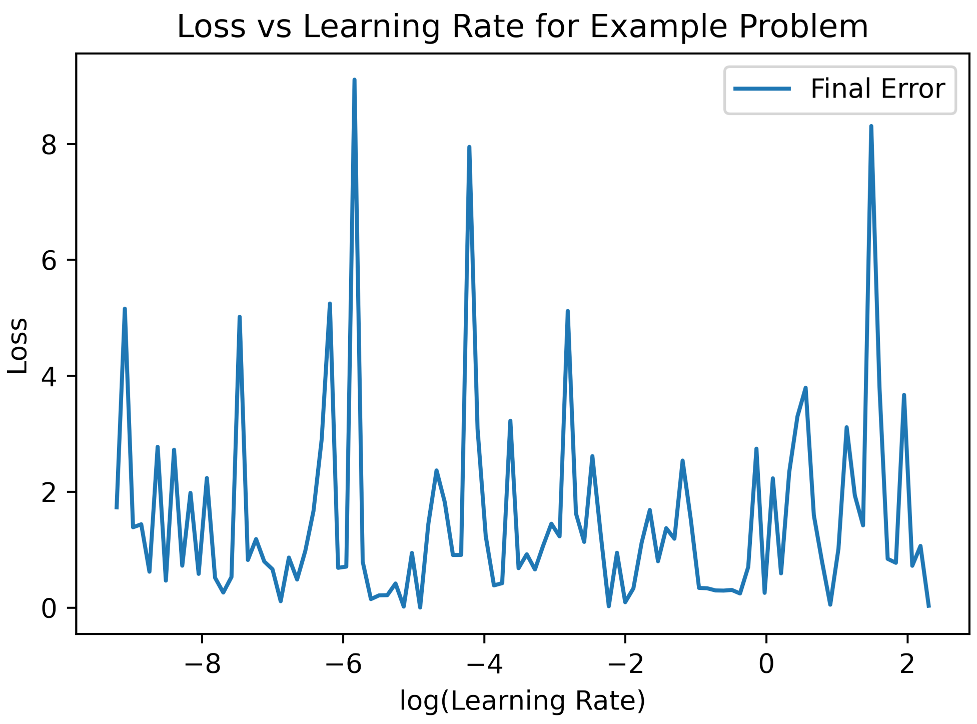


Figure 2: Graph of loss vs the log of the learning rate for the example problem. The loss is generally decreasing, although it shows signs of stochasticity. We chose to plot log of learning rate as we wanted to test various orders of learning rates.

5.1 Example Problem

In order to better understand the relationship between the learning rate and the loss, it's often helpful to plot the loss as a function of the learning rate. In this case, we plot the loss against the logarithm of the learning rate instead of the learning rate itself (see Figure 2). This is because the range of possible learning rates can be very large, and plotting on a logarithmic scale helps us better visualize the changes in the loss for different orders of magnitude of the learning rate.

The plot of loss vs log of the learning rate for the example problem reveals that the loss generally decreases as the learning rate increases. However, there is some degree of stochasticity in the loss values. The presence of stochasticity suggests that the neural network is not converging to a global minimum of the loss function and that the loss values may fluctuate over time even when the network is trained with the same learning rate and other parameters.

To create this plot, we used the Matplotlib library in Python. We first run the `execute_neural_net` function for a range of learning rates and collect the loss values for each one. Then, we plot these values using the Matplotlib's plot function, where the x-axis represents the logarithm of the learning rate, and the y-axis represents the loss.

vs Learning Rate for AND Gate.png

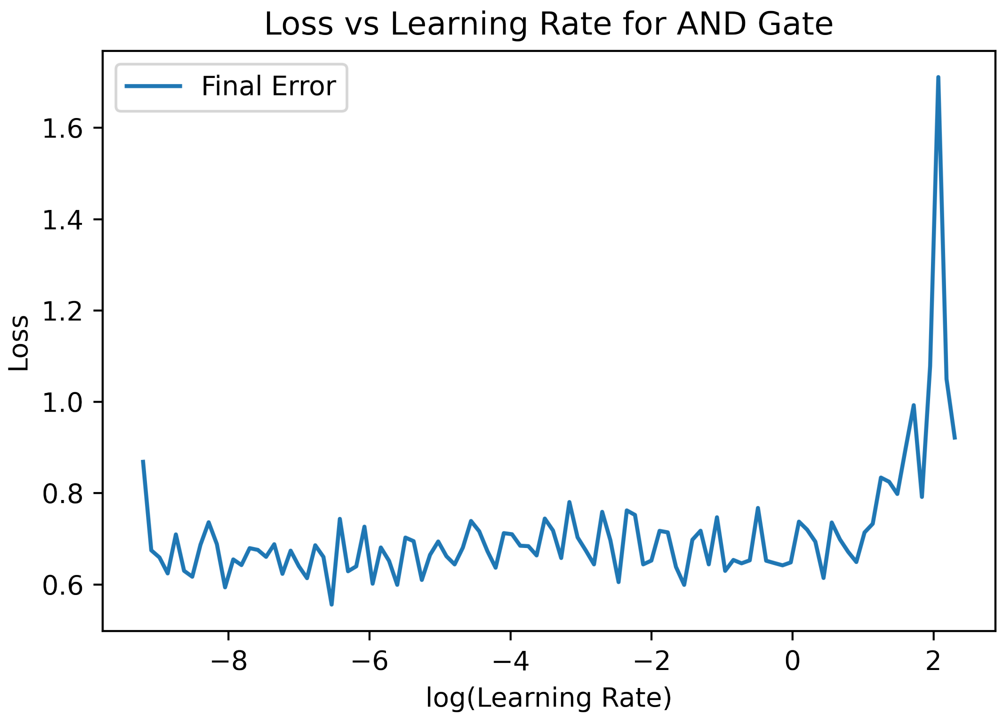


Figure 3: Graph of loss vs the log of the learning rate for the AND gate. The loss is largely the same except around zero, where it is low and eventually increases with high learning rates. It is unclear if we're in the right set of learning rates. We chose to plot the log of learning rates as we wanted to test various orders of learning rates.

vs Learning Rate for XOR Gate no hidden layer.png

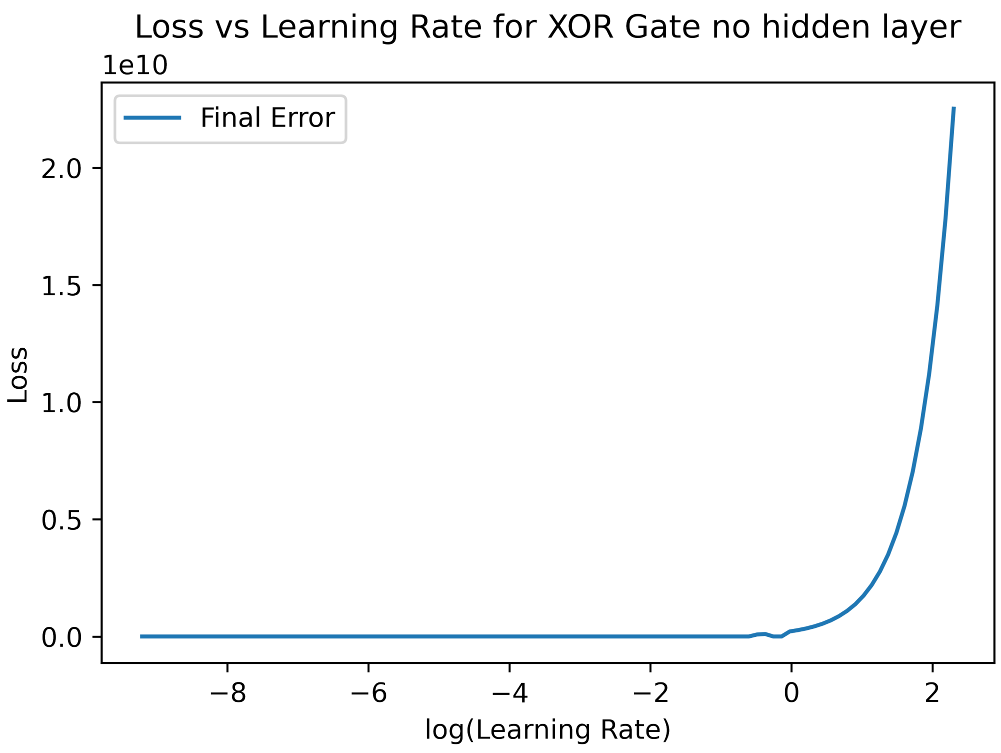


Figure 4: This figure shows loss vs learning rate for the XOR gate with no hidden layers. We can see that the loss blows up with the increase in the order of the learning rate. The scale of errors shows how this problem cannot be solved with a Perceptron with no hidden layers.

5.2 AND Gate

In Figure 3, the x-axis represents the log of the learning rate, while the y-axis represents the loss, which is a measure of the difference between the predicted outputs of the model and the true outputs. In the case of the AND gate, the plot shows that the loss is largely the same across different learning rates, except around zero, where the loss is low. As the learning rate increases, the loss eventually increases as well.

The logarithmic scale allows us to see the impact of both small and large learning rates on the loss, which can be useful in determining the optimal learning rate for a given model and problem. It doesn't look like we are using a powerful enough model from Figure 3. We should use a model with more complete networks.

5.3 XOR Gate

XOR Gate with No Hidden Layer Perceptron The XOR gate is a non-linearly separable problem, meaning that a simple linear model like the Perceptron cannot solve it. The Perceptron can only make decisions based on a straight line, but the XOR function requires a more complex decision boundary. In Figure 4, we can see that as the learning rate increases, the error (loss) of the model also increases, indicating that the model cannot find a good solution.

The blow-up of the error at high learning rates is a common issue in deep learning and is often referred to as overshooting or exploding gradients. This can be mitigated using techniques such as regularization or using more appropriate activation functions.

vs Learning Rate for XOR Gate single hidden layer.png

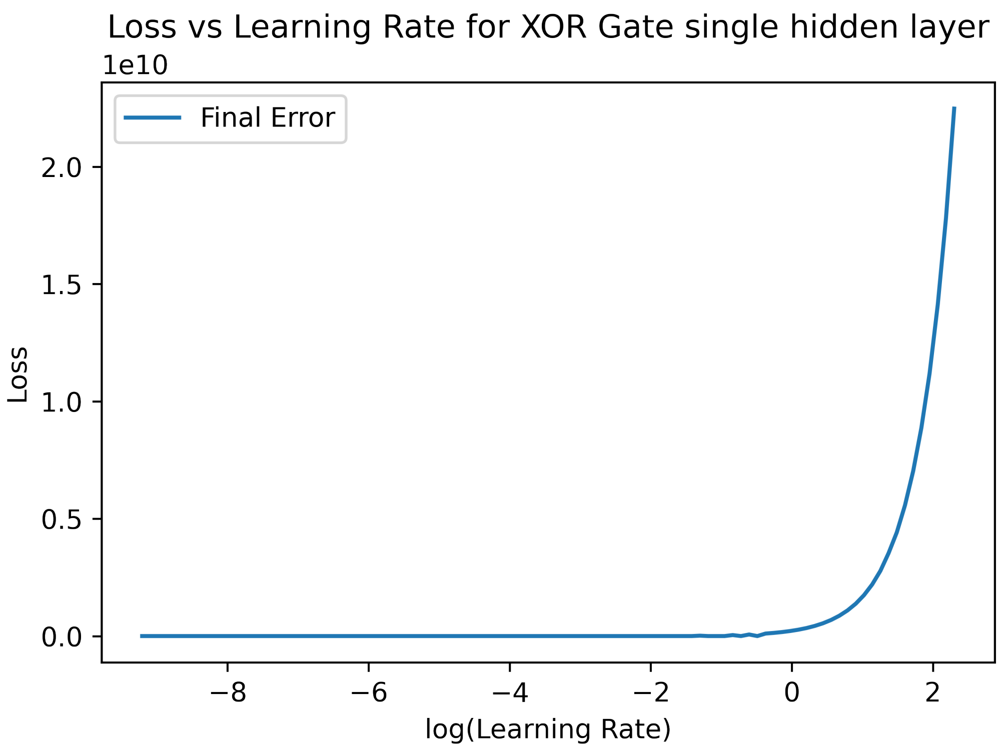


Figure 5: This figure is very similar to a perception with no hidden layers. The errors blow up when the learning rate's order increases.

XOR Gate with Single Hidden Layer Perceptron In Figure 5, we can again see that the loss increases dramatically when the learning rate's order increases, which means that the model cannot learn the relationship between the input and output data effectively.