Name: Hazem Taha ID: 400550393

Assignment 4 Report

I. Feedforward Neural Network (FNN) Implementation

For modularity and adaptability for future expansion. I have developed my FNN to be fully parametrizable, giving me the below key futures:

- Support for generic FNN architectures (any number of layers and nodes).
- Support for encapsulating different activation functions for hidden and output layers (ReLU, Sigmoid, Tanh and Softmax). More can be easily integrated without changing the training backbone code (feed-forward, back propagation, and network update).
- Support for encapsulating different loss functions without changing the training backbone code. Currently added Binary Cross-Entropy Loss but others can be easily integrated.
- Support for encapsulating different types of weight initialization. Currently added "He Initialization" but other methods can be easily integrated.
- Support for different types of optimizers like Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent (MB-GD), and Batch Gradient Descent (B-GD). With a data loader that generates batches of data on the fly during training.
- Support for L2 Regularization and for early stopping.

Note that all coded equations were taken from the lecture slides.

II. Experimentation:

The dataset used in this assignment is the same one as in Assignment 3 "spambase". I first loaded the dataset, dropped duplicate rows, and divided it into training, validation, and testing sets with ratios (60%, 20%, 20%) respectively. I then standardized the 3 sets based on the training set data.

1. Training Neural Networks without Regularization:

Target: Find the least number of hidden nodes that achieves a training loss of zero for 1, 2, and 3 hidden layers.
Solution method: Binary Search (Bisection Search).

First of all, I have to define the threshold under which I can say that I have achieved a training loss of 0, because clearly we cannot obtain an absolute 0 BCE loss as we have to add a small value inside the log() to avoid having log(0), and the dataset is not so trivial. Hence, by convention, I will choose my threshold to be 1e-3.

Now, I have to define my search space (upper and lower search bounds) for the binary search algorithm. The lower bound can be trivially set to a small number like 8 nodes. However, for the upper bound, I will experiment manually a very high number of hidden nodes (ex: 1024) and see if they will be able to achieve the desired training loss (< 1e-3) and hence can be a viable choice.

Name: Hazem Taha ID: 400550393

Trial 1:

#Hidden Layers	#Nodes	Learning rate	Number of Epochs	Optimizer Type	Batch Size
1	1024	1e-3	1000	SGD	1 (Due to SGD)

As shown in figure 1, the training loss converged at around 0.009, a little bit above the threshold. Although I am certain that 1024 nodes should be more than enough to achieve the desired loss (if it's ever achievable). I should decrease the learning rate to get more fine-grained learning steps.

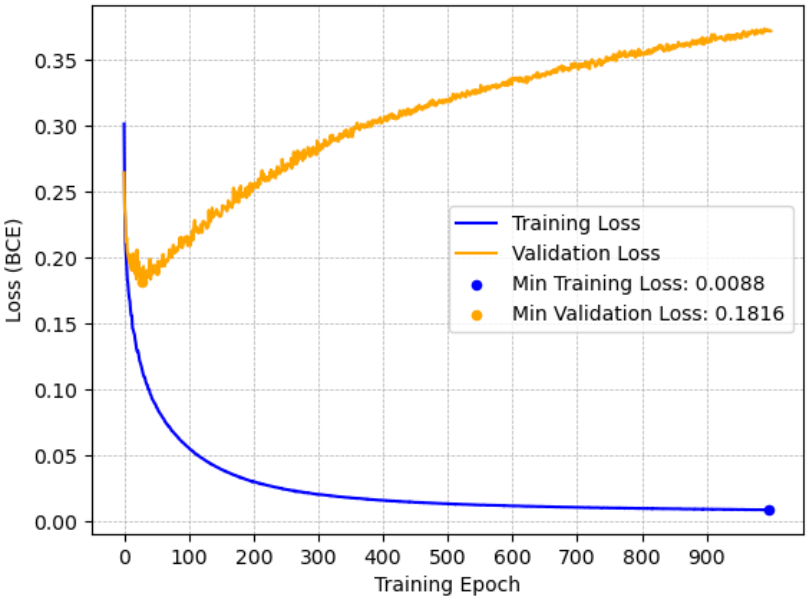

Figure 1: (Trial 1) Training and Validation BCE Loss over Epochs

Trial 2:

#Hidden Layers	#Nodes	Learning rate	Number of Epochs	Optimizer Type	Batch Size
1	1024	1e-4	1000	SGD	1 (Due to SGD)

Figure2 shows that the loss is still decreasing and hasn't converged yet. This must be due to the smaller learning rate we choose in this trial that needs more time in training. I need to increase the number of epochs.

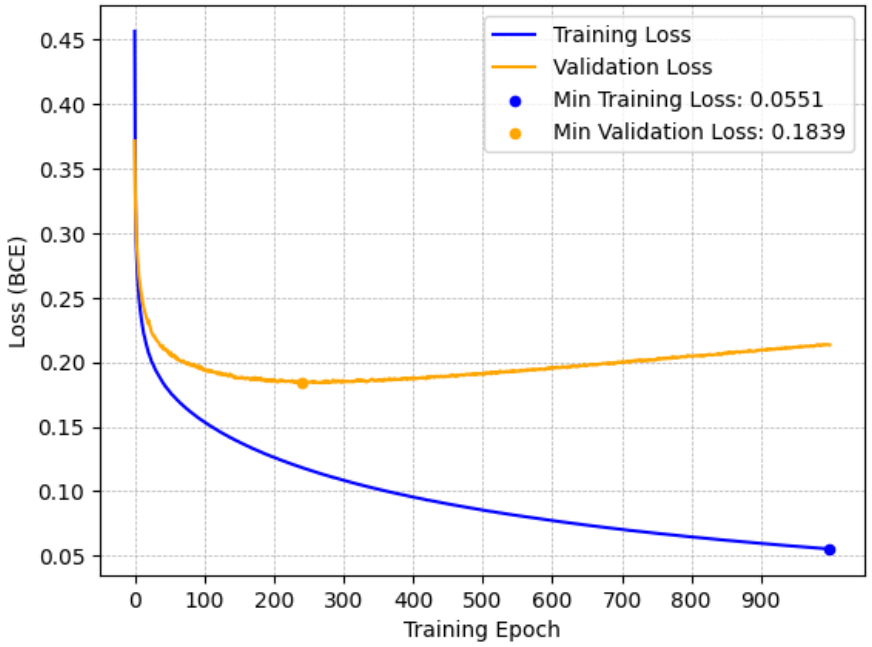

Figure 2: (Trial 2) Training and Validation BCE Loss over Epochs.

Trial 3:

#Hidden Layers	#Nodes	Learning rate	Number of Epochs	Optimizer Type	Batch Size
1	1024	1e-4	2000	SGD	1 (Due to SGD)

As seen in figure 3, even after doubling the number of epochs, the loss is still decreasing and didn't converge. This suggests that the learning rate might be too low and trail 1 was more promising. However, this means that the best training loss we can get from 1 hidden layer with very high number of nodes (1024) nodes cannot get below a threshold of 1e-3. Hence, the threshold must be adjusted to 1e-2. The 1024 nodes can then be taken as an upper bound for the binary search as it achieves a training loss of 0.0088 less than the threshold.

New target: Find the least number of hidden nodes that achieve a training loss less than 1e-2 for 1, 2, and 3 hidden layers.

Solution method: Binary Search (Bisection Search) between 8 and 1024 hidden nodes.

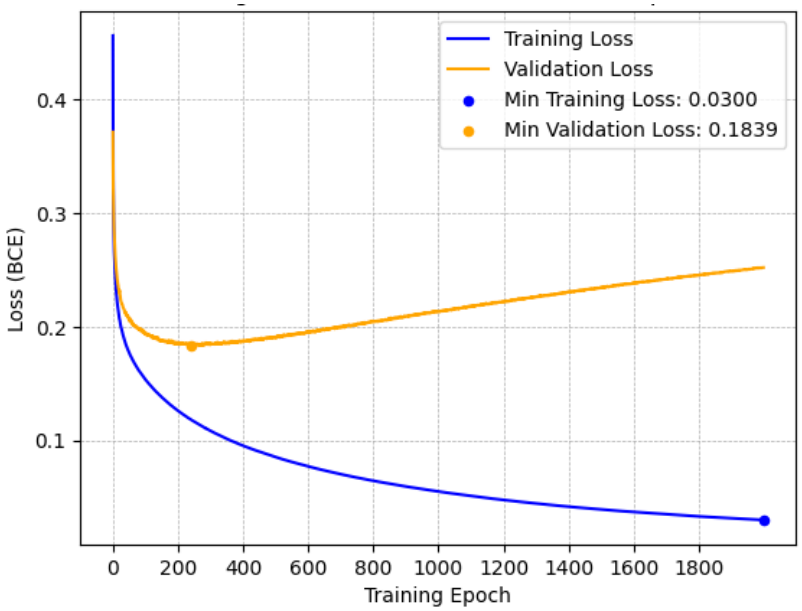

Figure 3: (Trial 3) Training and Validation BCE Loss over Epochs.

Binary Search Algorithm:

After defining the lower and upper bounds of the binary search space to be 8 to 1024 respectively. I developed and conducted the search algorithm to find the minimum number of nodes per hidden layer that achieves a training error of less than 1e-2 using 1, 2, and 3 hidden layers and a learning rate of 1e-3 (best learning rate as proven before). The results were as follows:

Using **ReLU** activation at hidden layers:

- For 1 hidden layer/s: 770 hidden nodes with a training loss of 0.0095
- For 2 hidden layer/s: 72 hidden nodes with a training loss of 0.0097
- For 3 hidden layer/s: 55 hidden nodes with a training loss of 0.0071

Using **Tanh** activation at hidden layers:

- For 1 hidden layer/s: hidden nodes with a training loss of (Still running)
- For 2 hidden layer/s: 149 hidden nodes with a training loss of 0.0093
- For 3 hidden layer/s: 18 hidden nodes with a training loss of 0.0082

1. Training Neural Networks with Regularization and Early Stopping:

Given that the relationship between the regularization coefficient λ and the loss function is not monotonic “the loss does not consistently increase or decrease as λ changes”. a binary search approach will be unlikely to succeed in finding the optimal λ that gives the lowest validation loss. Consequently, a grid search strategy has been followed by trying 100 distinct logarithmic steps in range from 1e-10 to 1e-1 as seen in the screenshot in figure 4.

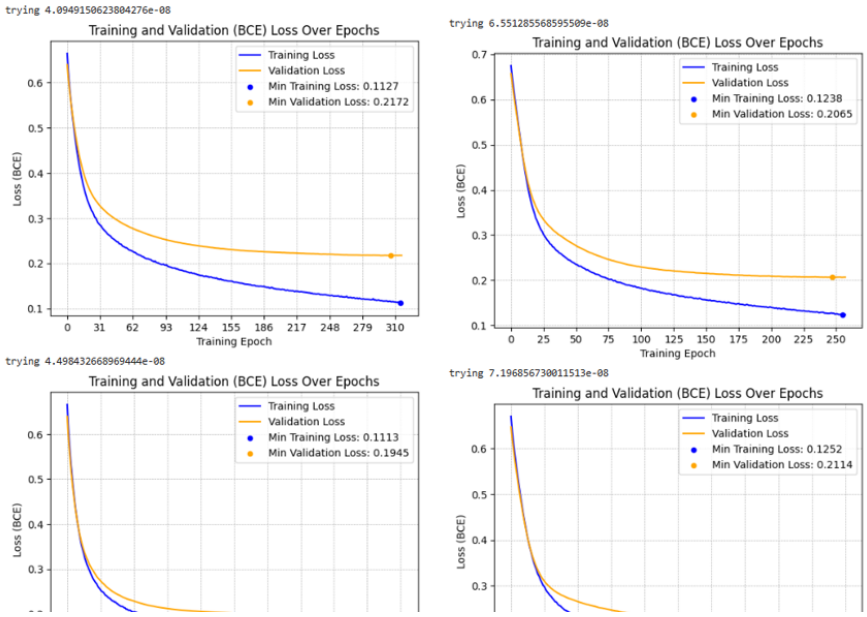

Figure 4: Screenshot of the Grid Search process for the optimal λ

Results:

The below results were collected using ReLU activation in the hidden layers and a learning rate of 1e-3:

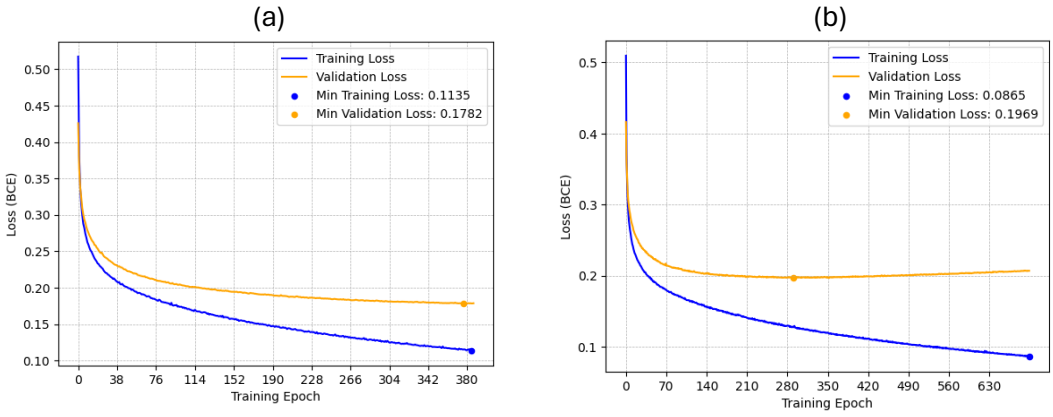

Figure 5: Showing training a neural network (of 1 hidden layer) with early stopping. (a) shows the training with regularization. (b) Shows the training without

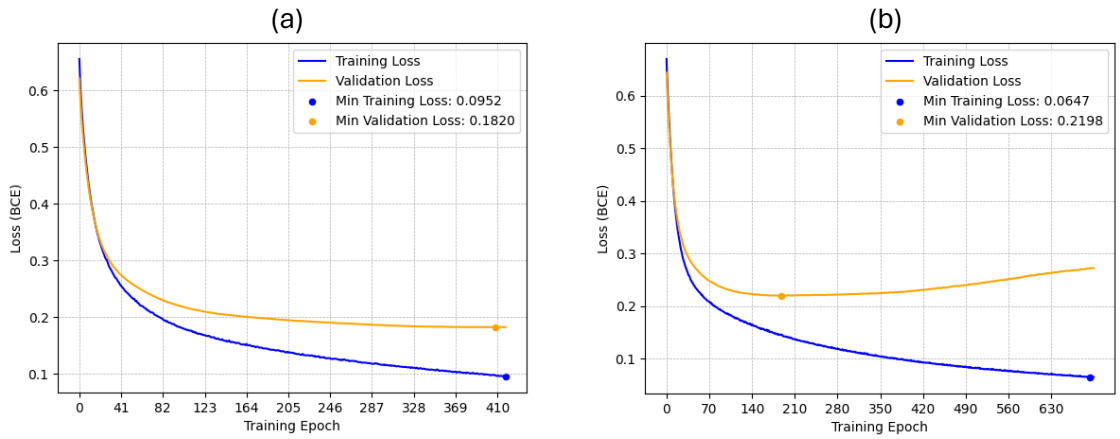

Figure 6: Showing training a neural network (of 2 hidden layers) with early stopping. (a) shows the training with regularization. (b) Shows the training without regularization.

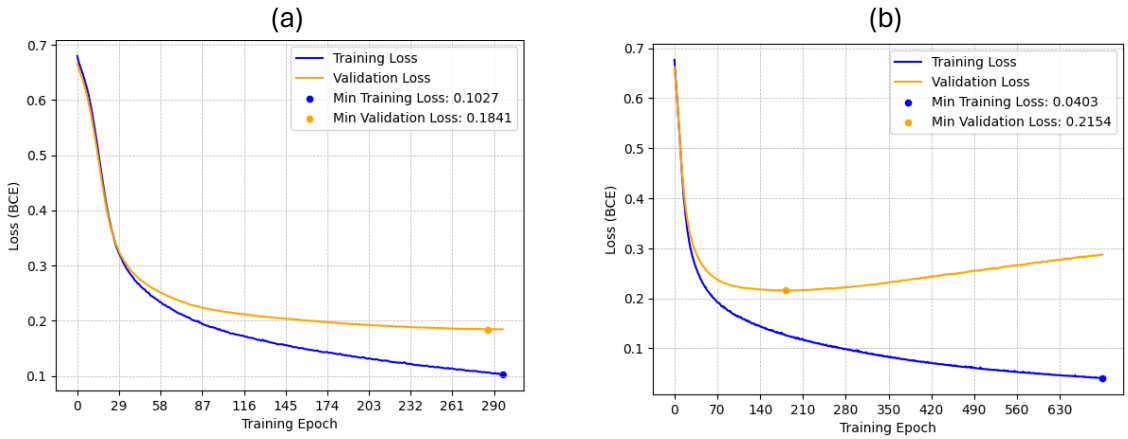

Figure 7: Showing training a neural network (of 3 hidden layers) with early stopping. (a) shows the training with regularization. (b) Shows the training without regularization.

Table 1: Summary of the results

Number of hidden layers	Best found λ	Final Training BCE Loss	Final BCE Validation Loss	Final Training BCE Loss (with regularization and ES)	Final BCE Validation Loss (with regularization and ES)
1	1.207e-08	0.0865	0.207	0.1135	0.1782
2	3.556e-07	0.0652	0.272	0.0952	0.1820
3	4.498e-10	0.0397	0.308	0.1027	0.1841

Table 1 summarizes the information shown in figures 5, 6 and 7. It can be seen that using regularization and early stopping decreased the final achieved validation loss in all three neural networks. I also want to point out the effect of the regularization, you may notice the small drop in y-axis that happened to the yellow point "minimum validation loss" from plot (b) to plot (a) in all the 3 figures. This drop is not due to early stopping but mainly due to the regularization.

III. Final testing

At the last stage, I used the testing set to evaluate the performance of the three trained neural networks. The results were as follows (notice that I will use misclassification rate):

- The first model (1 hidden layer of 770 nodes), the misclassification rate is 0.062
- The second model (2 hidden layers of 72 nodes each), the misclassification rate is 0.074
- The third model (3 hidden layers of 55 nodes each), the misclassification rate is 0.070

IV. Other possible contributions

There are many other possible configurations for the neural network that could be tested out, but I didn't do them due to time limitations and the fact that each experiment trial takes quite a while to finish.

1. One thing I thought about doing is to see the performance of the neural networks using other activation functions like the ones already implemented (Tanh and Sigmoid). I started the experiments of Tanh, but it will take some time to finish.
2. Another thing to test is to train using Minibatch Gradient Decent (MB-GD) and Batch Gradient Decent (B-GD) which are also already implemented. I believe they will not only give better results, but they will also train faster. SGD takes a long time because it doesn't utilize the vectorization and parallelization capabilities of NumPy because we train with one sample at a time.
3. I believe that there are better methods to find and train an optimal neural network than to restrict ourselves with a specific architecture (that gets training loss of 0 and then add regularization). I think that a lower number of nodes that don't initially cause overfitting can be used, and it can achieve higher performance. I wanted to test and document that, but the experiments above took days, so I don't have additional time.