CENG499
INTRODUCTION TO MACHINE LEARNING
THE1
PART3 REPORT

Erdem
Şamlıoğlu
2448843

Part1

Training Setting

Firstly, I created my model, which is Multi Layer Perceptron model. The input layer is already defined in the pdf as 784 inputs. The hyperparameters are given as number of hidden layers, number of neurons in the hidden layers, learning rate, number of iterations, activation functions.
After creating my model, I connected the input layer to the first hidden layer by using linear transformation(nn.Linear). Since the hidden layer configuration is changable, I used nn.ModuleList to contain a sequence of nn.Linear layers. With this way, each layer maps the output of the previous layer to next one. For the activation function, I used only Tanh and Sigmoid as stated in the pdf. I applied activation function for all layers. The output layer is given in the pd as 10 output nodes. For the loss function I have used nn.CrossEntropyLoss to calculate the prediction error during training. I have chosen Adam optimizer, and used weight decay in the optimizer with value 0.01 to prevent overfitting. (I havent used mini-batch)

Procedures and Decisions

For the part3 of this assignment, I implemented grid search to check different hyperparameter configurations, as stated in the pdf. In this process, I have defined different values for each of the hyperparameter, and then tested each combination. Here are the values I tried, which I have taken from my code

```
hidden_layer_options = [[8, 8], [8, 16], [16, 16], [32,64], [64,64],
[128], [128, 64], [256, 128, 64]]
learning_rate_options = [0.01, 0.001, 0.0001]
```

```
epoch_options = [10, 50, 100, 150, 200]
activation_functions = [nn.Sigmoid, nn.Tanh]
```

I have tested up to 3 hidden layers(multiple versions), 3 learning rates, multiple values of epochs to find the balance between underfitting and overfitting, 2 activation functions. In total it makes 240 different hyperparameter configurations.

For each configuration, I ran the model 10 times to compute the mean validation accuracy and its 95% confidence interval.

After getting the output, I found the best configuration with the best mean validation accuracy. (I save both the best accuracy in a single run and best average accuracy configuration. I saved the single run to experiment it myself rather than the average one and it gave worse results.)

As it is stated in the pdf, after finding the best hyperparameter configuration, I combined validation dataset and training sets to a larger dataset. I trained the model in this combined dataset and calculated the mean accuracy and the confidence interval:

```
Mean Test Accuracy Over 10 Runs: 83.26%
95% Confidence Interval for Test Accuracy: [83.04%, 83.47%]
```

Finally, I have included the SRM plots in my code to visualize the performance. The plots show the training and average validation losses for each configuration for 10 runs.

Results

The configuration with "[32, 64] hidden neurons, learning rate of 0.01, 200 epochs, Tanh activation function" gave the highest mean validation accuracy with 84.64, which I used it in the next step of the assignment.Here is the output I got:

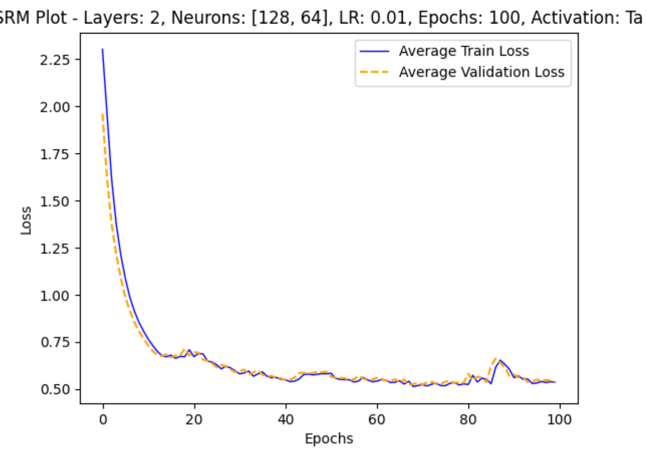
```
Hidden Layers: 2, Neurons: [32, 64], Learning Rate: 0.01, Epochs:200,
Activation: Tanh, Mean Validation Accuracy: 84.64%, Confidence Interval: [84.52%, 84.75%]
```

Then I used this configuration with the combined training and validation dataset, I got this as output:

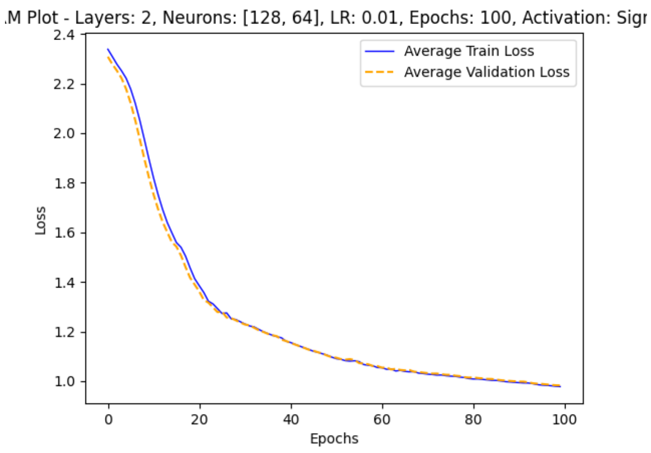
```
Mean Test Accuracy Over 10 Runs: 83.26%
95% Confidence Interval for Test Accuracy: [83.04%, 83.47%]
```

Which has a slightly lower mean accuracy than the validation accuracy, possibly by the differences between validation and test data or overfitting on the combined dataset.

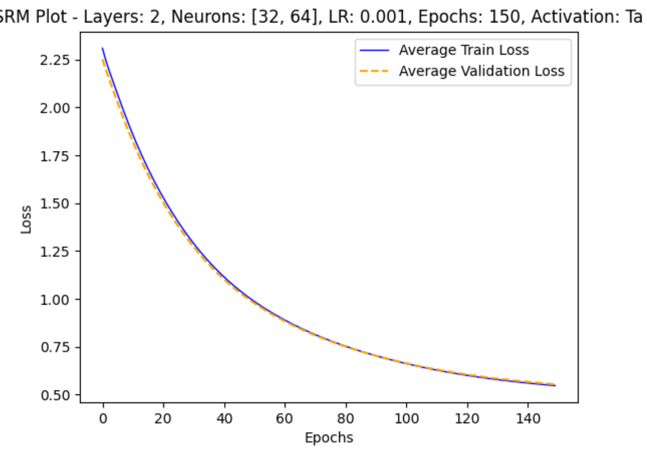
I plotted the SRM plot for every configuration I have used. For configurations with lower learning rates ”0.001 or 0.0001” and smaller models(hidden layers), tended to show slower but more stable convergence, and the configurations with higher learning rates sometimes exhibited faster but less stable training.



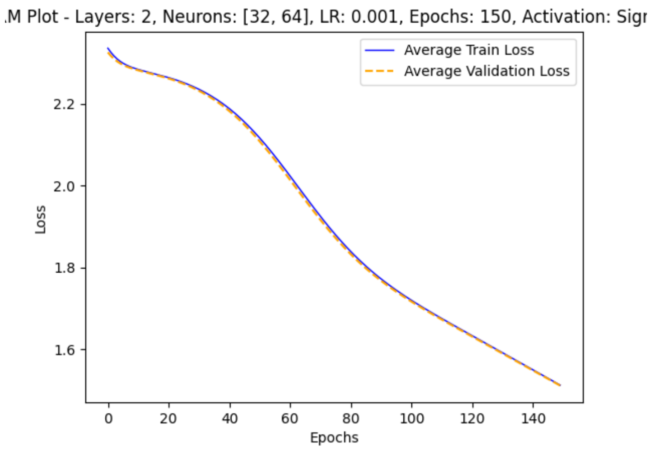
(a) Stable - Tanh, Epochs=100



(b) Stable - Sigmoid, Epochs=100



(c) Less Stable - Tanh, Epochs=150



(d) Less Stable - Sigmoid, Epochs=150

Figure 1: SRM plots for Tanh and Sigmoid

From the plots, for all of the cases, the training and validation losses are similar, which shows that the models are not overfitting, which also shows they are generalizing well.

For the stable case (a and b) it is visible that for Tanh, the curve is smooth, and for the sigmoid it is not. Sigmoid converges slower than Tanh. So, Tanh is better in stable and faster convergence in low learning rate for this dataset.

For the less stable case (c and d), with the higher learning rate case (0.01), it is visible that Tanh converges way faster but has some instability at the end, Sigmoid has less instability with the curve at the end but Tanh reaches better accuracy compared to Sigmoid.

As a result, Tanh shows faster convergence and smoother lines overall, and has lower final loss compared to Sigmoid, which makes Tanh better in both cases.

Answers to the Questions asked in PDF

Q: What type of measure or measures have you considered to prevent overfitting?
A: To prevent overfitting, I used weight decay as Adam optimizer's parameter, which penalizes large weights, and in result helps prevent overfitting by making the model find simpler patterns in data.

Q: How could one understand that a model being trained starts to overfit?
A: By comparing the validation loss and training loss. If the training loss decrease while validation loss is increasing after a certain point, which means overfitting.

Q:Could we get rid of the search over the number of iterations (epochs) hyperparameter by setting it to a relatively high value and doing some additional work? What may this additional work be? (Hint: You can think of this question together with the first one.)
A:Yes, setting epoch a high number could get rid of the iterations of the epoch hyperparameter. We can implement early stopping in the case validation loss does not improve for a specificed number of epochs.

Q:– Is there a "best" learning rate value that outperforms the other tested learning values in all hyperparameter configurations? (e.g. it may always produce the smallest loss value and highest accuracy score among all of the tested hyperparameter configurations.).
A:No, even though higher learning rates gave better results, but sometimes resulted with unstability, so there is not a "best" learning rate value. [1em] Q:Is there a "best" activation function that outperforms the other tested activation functions in all hyperparameter configurations? (e.g. it may always produce

the smallest loss value and highest accuracy score among all of the tested hyperparameter configurations.).
A:Even though Tanh gave the best results in almost every case, for few cases sigmoid was better, especially with smaller learning rates, so we cant say there is a "best" activation function.

Q:What are the advantages and disadvantages of using a small learning rate?
A:Advantage:Smaller learning rates give more stable training and better chance at avoiding the minimum loss for the gradient, and convergence is not fast, which is more precise to find the optimal solution. Disadvantage:Needs more epochs, there is a risk of getting stuck in local minimum.

Q:What are the advantages and disadvantages of using a big learning rate?
A:Advantages: Less epochs needed, converges faster. Disadvantages:Risk of overshooting the optimal point, due to the larger rate, it might miss steps, and return suboptimal solution.

Q:Is it a good idea to use stochastic gradient descent learning with a very large dataset? What kind of problem or problems do you think could emerge?
A:I have used SGD as well, but it might lead to some problems. Since each update is based on a single instance, it can have high variance and instability. And it has higher computational cost.

Q:In the given source code, the instance features are divided by 255 (Please recall that in a gray scale-image pixel values range between 0 and 255). Why may such an operation be necessary? What would happen if we did not perform this operation? (Hint: These values are indirectly fed into the activation functions (e.g. sigmoid, tanh) of the neuron units. What happens to the gradient values when these functions are fed with large values?)
A:It is normalization from 0 to 255, to, 0 to 1. This helps with decreasing the input value, and if it wasnt, after the activation function, it could result in a near zero gradient. Also normalized input is better for the model to converge faster, because the gradients will be more balanced.