HW2-Deep Learning

Baktash Ansari

99521082

Q1)

a)

Overfitting occurs when a model learns the training data too well, to the point where it starts to memorize the noise or random fluctuations in the data rather than learning the underlying patterns. As a result, the model performs very well on the training data but poorly on unseen, new data (validation or test data).

Signs of overfitting include:

The model has very low training error (i.e., it fits the training data extremely well).
The model's performance on a separate validation or test set is significantly worse than its performance on the training set.
Causes of overfitting can include having too many parameters in the model relative to the amount of training data or using a very complex model architecture that can capture noise in the data.

Underfitting, on the other hand, occurs when a model is too simple to capture the underlying patterns in the data. In this case, the model performs poorly not only on the training data but also on new, unseen data.

Signs of underfitting include:

The model has high training error (i.e., it doesn't fit the training data well).
The model's performance on a separate validation or test set is also poor.
Underfitting can happen when the model is not complex enough to capture the underlying relationships in the data, or when there isn't enough training data to learn from.

b)

If the loss value of the training dataset is much smaller than the loss value of the test or validation dataset we can understand that our model fits very well on training data but the model couldn't generalize the learning and the loss value is high in the test. Therefore, our model becomes overfit on our data. In addition, we can If we use the K-fold method for our evaluation we can analyze the variance of our model's predictions and if it has a high variance and low bias on our folds, we can find that our model becomes overfit on our data.

c)

1.6	0	0	1.9
0	2.5	2.5	0
0	3.2	3.7	0
1.3	0	0	1.2

The model's output after the dropout layer will be the values above. But I should mention that we should divide the values with the dropout probability in the training step or multiply the values with the dropout probability in the testing step. It is important to note that the matrix operation above just applies in the training step, not in the testing step.

For the train, we have:
1.6 + 1.9 + 2.5 + 2.5 + 3.2 + 3.7 + 1.3 + 1.2 = 17.9
For the test, we have:
1.6 - 0.7 - 0.2 + 1.9 - 2.3 + 2.5 +2.5 - 0.9 - 0.5 + 3.2 + 3.7 - 0.4 + 1.3 -0.4 -2.6 +1.2 = 9.9

Q2)

a)
In this question, we should consider the problem in general. As k increases (meaning more neighbors are considered), the model tends to become simpler. It averages over a larger number of neighbors, which tends to smooth out the decision boundaries. This can lead to an increase in bias because the model may not capture fine patterns in the data as well. It might make more generalizations, potentially missing out on important details. Conversely, as k decreases (considering fewer neighbors), the model becomes more complex and can potentially capture finer details in the data. This can lead to lower bias.

For variance:

As k increases the model tends to become simpler, so it will become smoother and always predict the model with more data and always predicts the same value, so the variance decreases. On the other hand, if k decreases, the model becomes more complex, and for each data point, it can predict whether to label one or two by the distances, so the variance increases.

b)

1) True, if the rate of regularization becomes more than enough it can cause high changes in the model; 's parameters. For instance, in weight decay, if the addition's value becomes bigger than we need, regularization not only doesn't help the model but leads to bad impacts.
2) False, if the chosen feature won't be a good representation of the dataset, it can lead to overfitting. Because it will learn the unnecessary features that are just represented in test data.
3) False, The goal of regularization is to prevent the overfitting process. If we highly increase the rate of regularization it will lead to underfitting.

c)
For exp1: L2 because the weights have a high uniform rate

For exp2: L1 because most of the weights change to zero

For exp3: None of them, because L2 helps the model prevent large weights and L1 wants to make them zero but these values are so big.

For exp4: None of them, because It isn't uniform so the L2 is not applied and also it is not L1 because it is not trying to reduce the weights to reach zero because we have zero and also 8.5

Q3)

a)

Knowledge distillation is a technique in machine learning where knowledge from a larger, more complex model is transferred to a smaller, simpler model. The larger model, which has a higher capacity for knowledge, is trained on the data first, and its learned knowledge representations are then distilled into the smaller model. This process allows the smaller model to benefit from the larger model's knowledge without the computational expense of the larger model. It's a powerful technique for making machine learning models more efficient and suitable for deployment on less powerful hardware, without sacrificing their performance.

b)

The architecture in the image you provided is a typical setup for **knowledge distillation**, a process where a smaller model (the student) is trained to mimic the behavior of a larger, more complex model (the teacher). Here's how the training process works:

1. Train the Teacher Model: The teacher model, which is typically a large and complex model, is trained on the dataset. This model learns to predict the labels of the dataset with high accuracy.

2. Generate Soft Labels: Once the teacher model is trained, it is used to make predictions on the same dataset. These predictions, also known as soft labels, are probabilities that reflect the teacher model's confidence about each class.

3. Train the Student Model: The student model, which is a smaller model, is then trained on the same dataset. However, instead of using the original labels (hard labels), the student model uses the soft labels generated by the teacher model. This allows the student model to learn from the teacher model's knowledge.

4. Combine Soft Labels and Hard Labels: To ensure that the student model also learns from the ground truth labels, a combination of the teacher model's predictions (soft labels) and the ground truth labels (hard labels) is used during the training of the student model.

The goal of this process is to create a smaller model (the student) that performs as closely as possible to the larger model (the teacher) but with reduced complexity and computational

requirements. This is particularly useful in scenarios where resources are limited, such as on mobile devices or embedded systems.

c)

In the architecture of knowledge distillation, the loss function typically used is a combination of the distillation loss and the original loss.

1. Distillation Loss: This is the loss between the soft labels (teacher model's predictions) and the student model's predictions. It measures how well the student model is mimicking the teacher model. The most commonly used distillation loss is the Kullback-Leibler (KL) Divergence.

2. Original Loss: This is the loss between the hard labels (original labels) and the student model's predictions. It measures how well the student model is predicting the actual labels. This could be any standard loss function used for classification tasks, such as Cross-Entropy loss.

The final loss function is a weighted sum of these two losses. The weights control the balance between the distillation loss and the original loss.

The weights of the student model are updated with respect to both the distillation loss and the original loss.

During the backpropagation process, the gradients are calculated for both these losses. The final gradient used to update the weights of the student model is a combination of the gradients from the distillation loss and the original loss. The exact combination depends on the weights assigned to these losses in the final loss function.

Q4)

For Stochastic gradient descent, As we know the model parameters are updated after each individual data point or a small batch of data points so the convergence happens after so many epochs(As we can see in the plots). In the SGD method if we increase the learning rate we can make the model converge sooner but if we increase lr more than enough it can lead to escaping from the local minimum. As we can see for lr = 0.001 the model converges in epoch 1000, but for lr = 0.1 and 0.01, the model converges sooner.

For momentum method, helps accelerate the convergence of the optimization process by giving momentum to the movement toward the minimum. This means that it can help the model reach the optimal solution faster compared to stochastic gradient descent. However, it's important to note that the model may overshoot the minimum several times and oscillate within the curve, but ultimately it reaches convergence. Also here as we increase the lr the convergence happens sooner and passes from the local minima fewer times.

Q5)

In this question, my model has a Linear layer with 784 neurons and an output of 10 neurons with ReLU activation function. I don’t use the 2dl library; Instead, I write the same function that gives us the test accuracy of the model and also gives us the 10 images and the corresponding true and predicted labels. Finally, This model gives me an accuracy of 82 percent.

Q6)

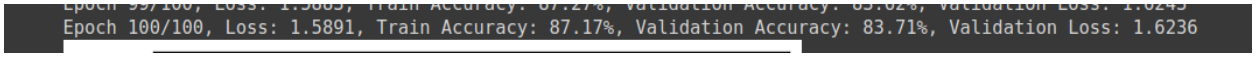
a)
For overfitting, I increase the model's complexity. So I added 4 layers with high neurons.
784 → 600 → 300 → 150 → 10
With the ReLU activation function and Softmax for the last layer.
To reach the goal of overfitting, I increased the epochs to 100 times.
So as we can see in the plot, the model is overfitted.
Finally, the reasons for overfitting were the complexity and the training time of the model.

b)
For Data augmentation, I use V2 Pytorch transformation and I change the train images in this way:

- With the probability of 1 flip all the images
- Make a rotation with degrees of -30 to +30

As we can see the accuracy of the model on the validation set increases

Before data augmentation:



After data augmentation:

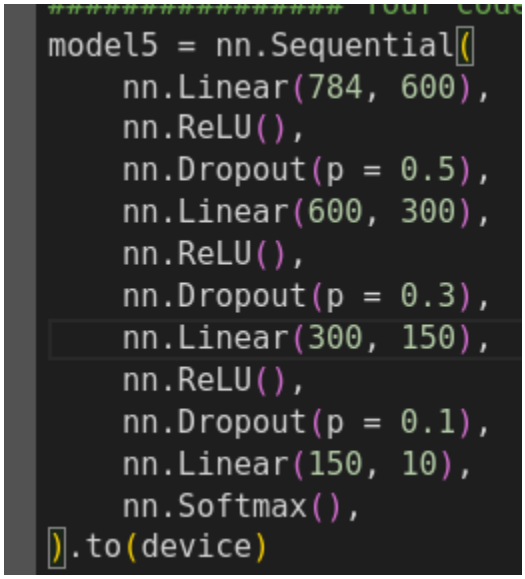
The model still is overfitted but it is much better than the previous

c)

For the L2 regularization as we can see the model's performance increases(I add wight decay of 1e-4):

d)

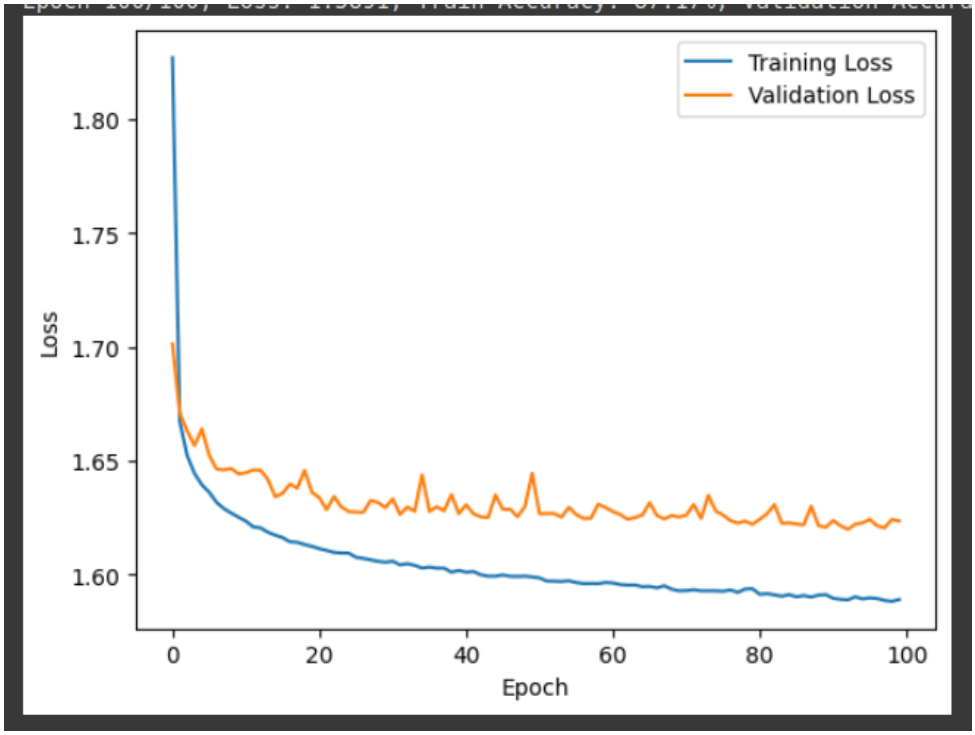
For this part I added dropout layers to the model like this:

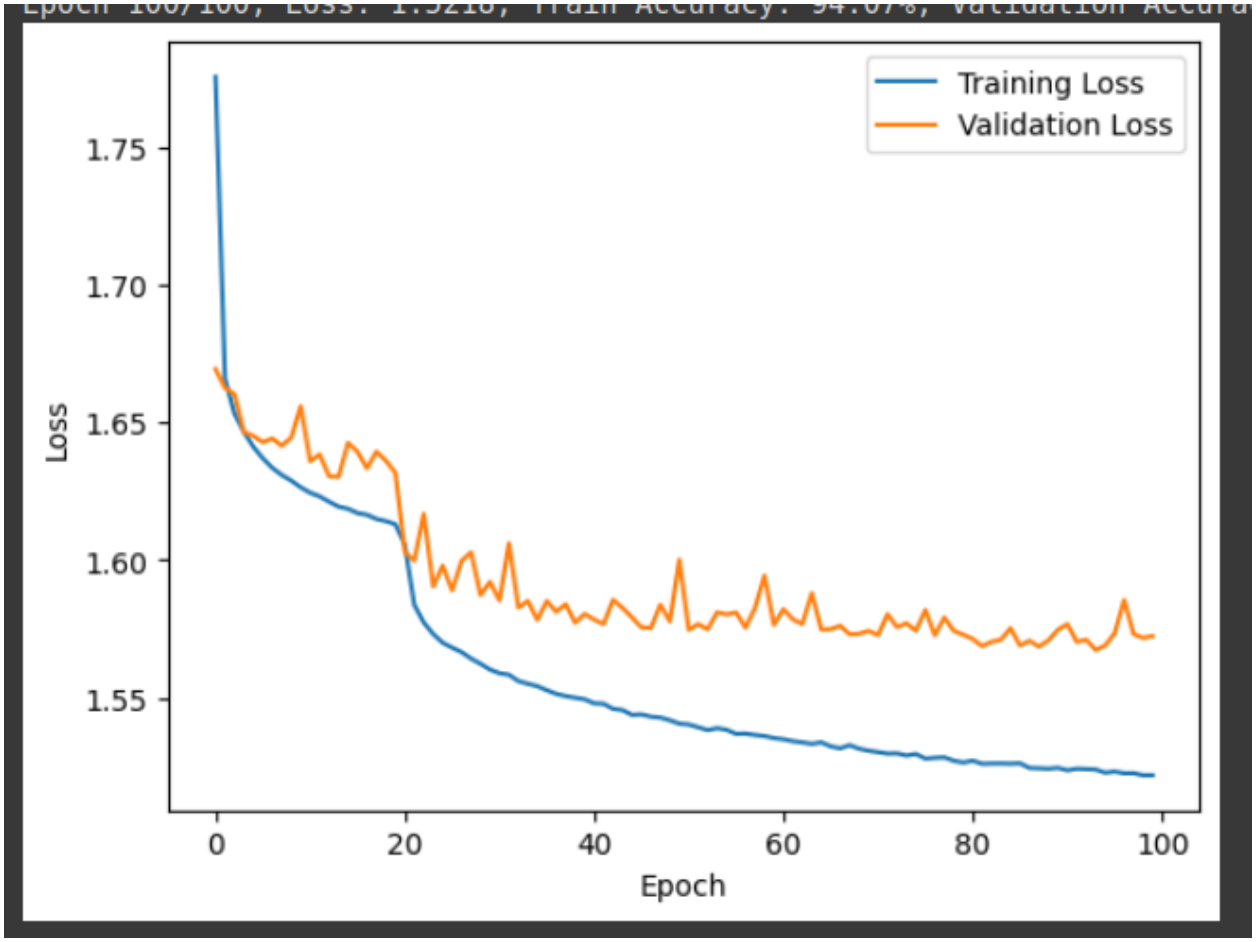


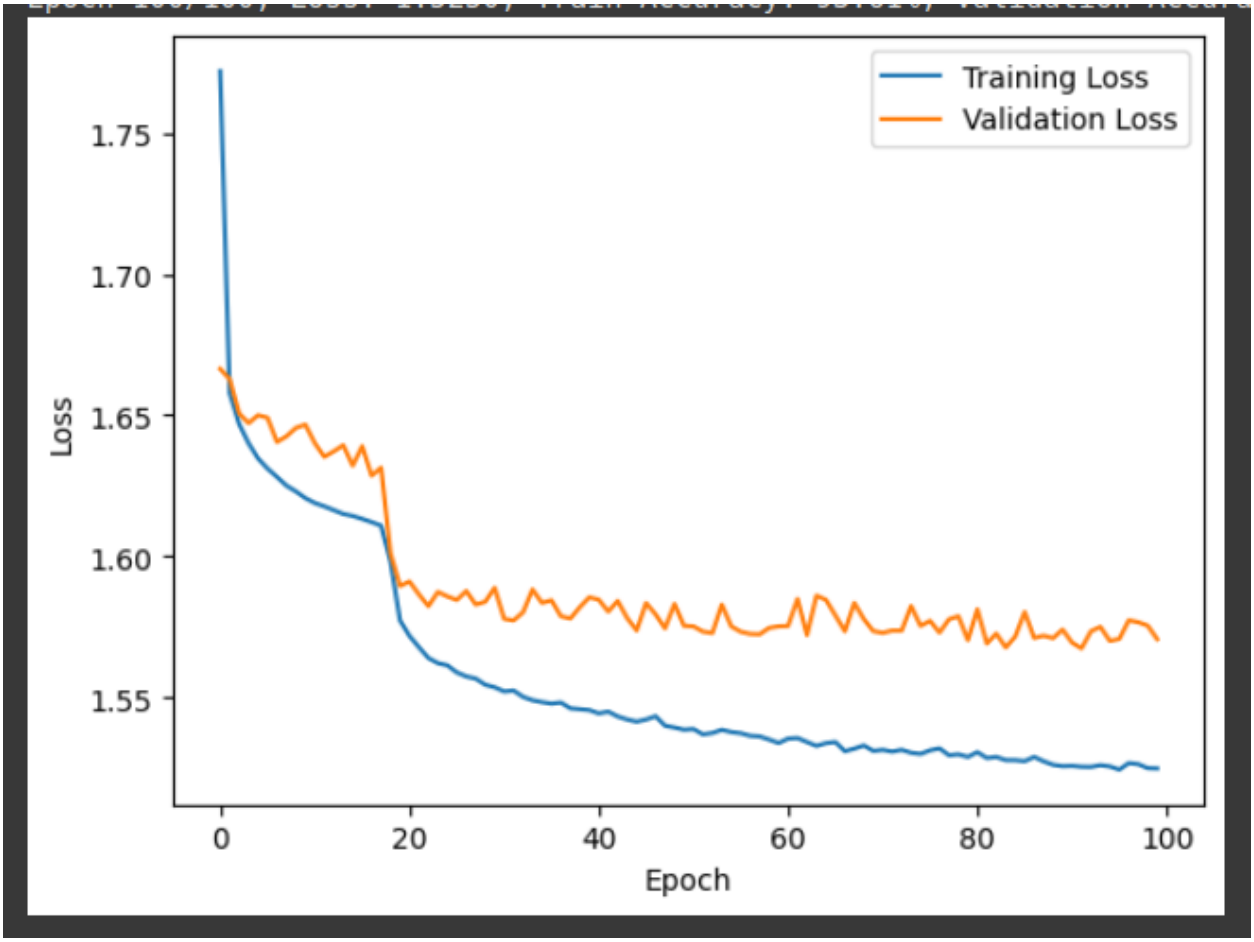
```
model5 = nn.Sequential(
    nn.Linear(784, 600),
    nn.ReLU(),
    nn.Dropout(p = 0.5),
    nn.Linear(600, 300),
    nn.ReLU(),
    nn.Dropout(p = 0.3),
    nn.Linear(300, 150),
    nn.ReLU(),
    nn.Dropout(p = 0.1),
    nn.Linear(150, 10),
    nn.Softmax(),
).to(device)
```

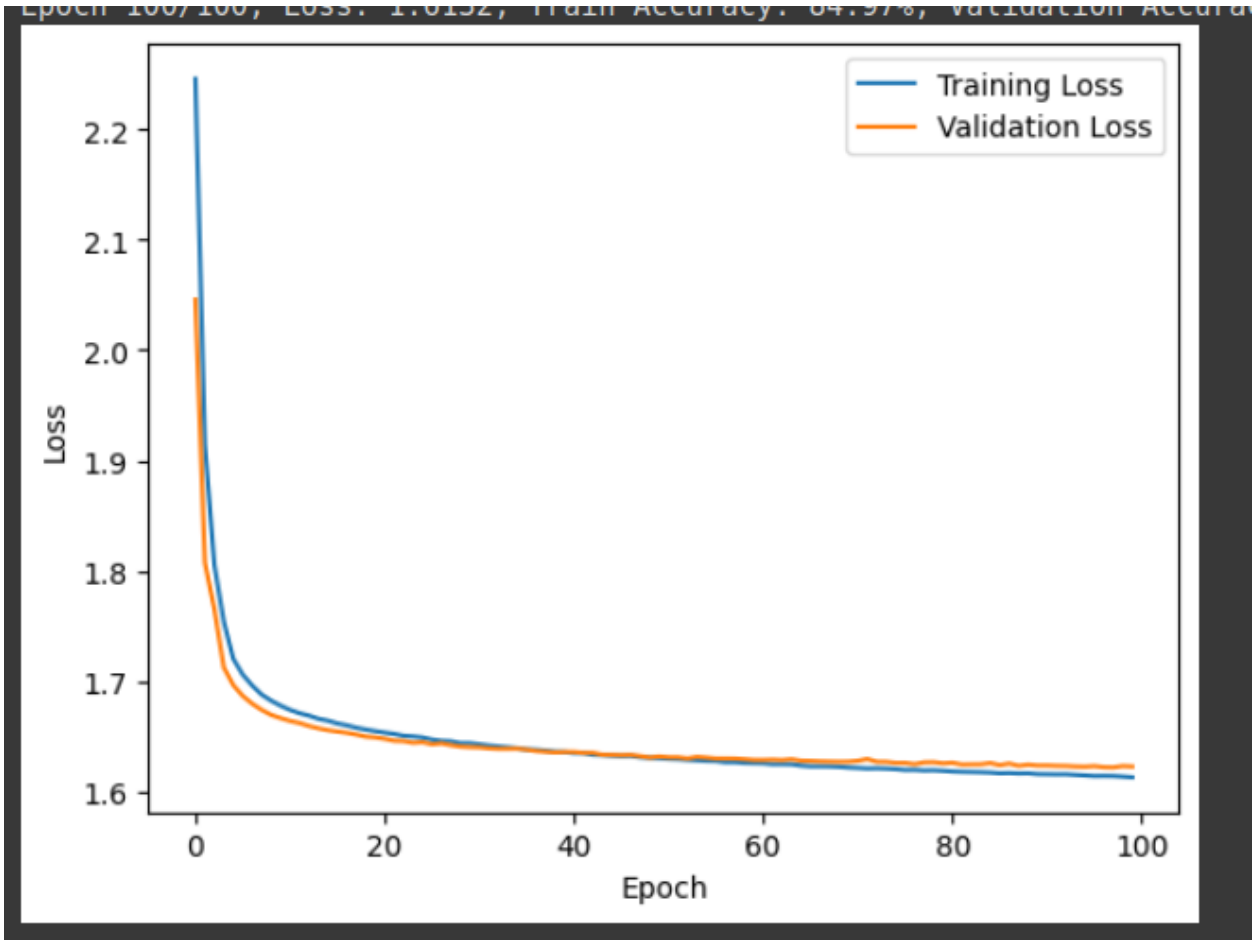
And Also I added weight decay of 0.4
And Also I change lr from 0.1 to 0.01.

The process of changing model is as follows:




Training Loss
Validation Loss
1.75
1.70
1.65
1.60
1.55
Loss
0
20
40
60
80
100
Epoch


Training Loss
Validation Loss
1.75
1.70
1.65
1.60
1.55
Loss
0
20
40
60
80
100
Epoch


Training Loss
Validation Loss
2.2
2.1
2.0
1.9
1.8
1.7
1.6
Loss
0
20
40
60
80
100
Epoch