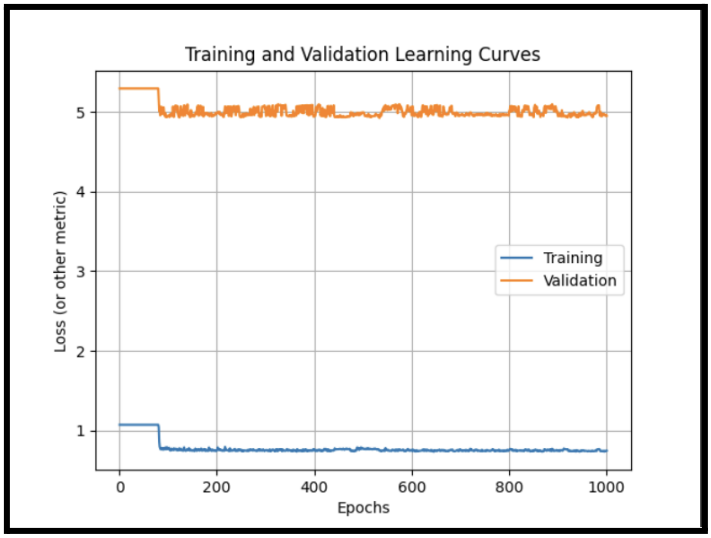
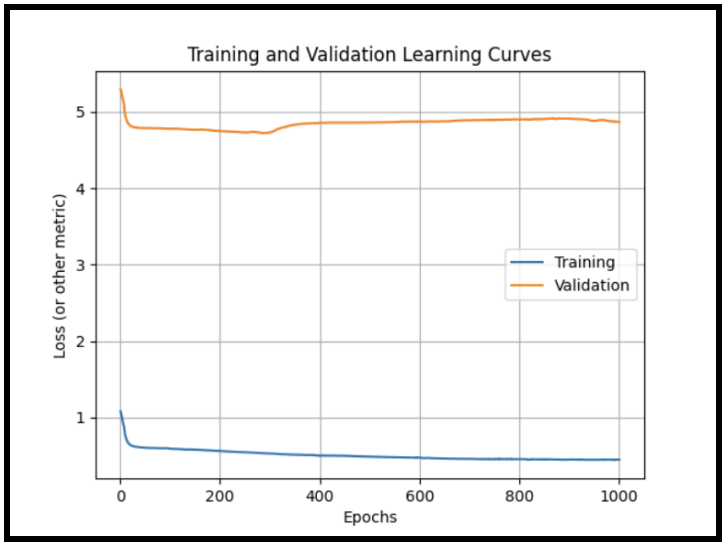
MSAI-349, Fall 2023
Homework #3: Neural Networks

Rohan Gupta

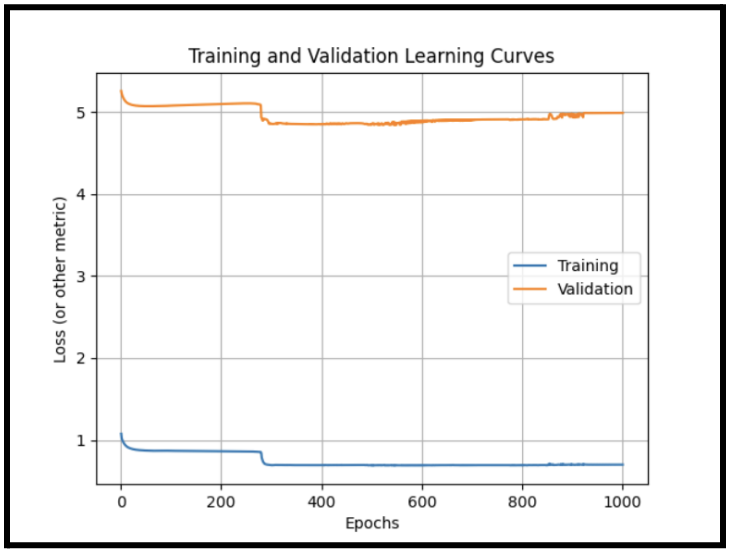
1. (4.0 points) Implement a simple feed-forward neural network in PyTorch to make classifications on the “insurability” dataset. The architecture should be the same as the one covered in class (see Slide 12-8). You can implement the neural network as a class or by using in-line code. You should use the SGD optimizer(), and you must implement the softmax() function yourself. You may apply any transformations to the data that you deem important. Train your network one observation at a time. Experiment with three different hyper-parameter settings of your choice (e.g. bias/no bias terms, learning rate, learning rate decay schedule, stopping criteria, temperature, etc.). In addition to your code, please provide (i) learning curves for the training and validation datasets, (ii) final test results as a confusion matrix and F1 score, (iii) a description of why using a neural network on this dataset is a bad idea, and (iv) short discussion of the hyper-parameters that you selected and their impact.
    a. **No** transformations were deemed necessary as all the data was relatively **comparable** in magnitude and there were no null values found.
    b. Results
        i. Learning Curves for 3 different parameter settings
            1. Bias Term - Learning Rate 0.01.
               Path to model - **HW3/src/model/trained/q1/q1-model-bias-lr-01**



            2. Bias Term - Learning Rate 0.001.
               Path to model - **HW3/src/model/trained/q1/q1-model-bias-lr-001**

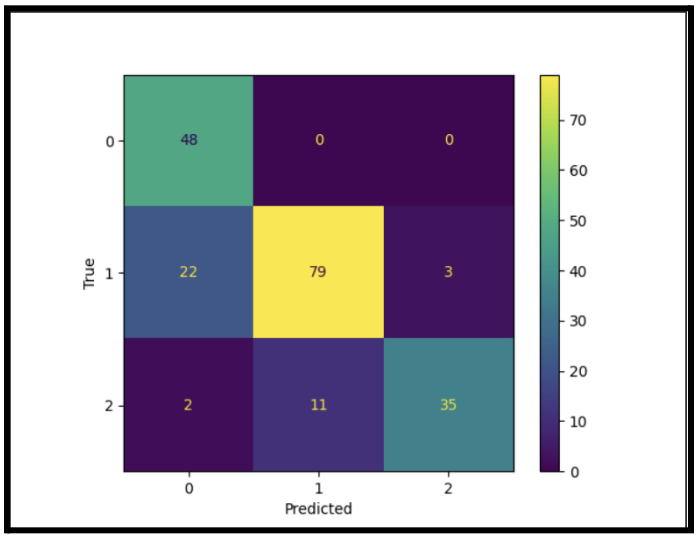


3. No Bias Term - Learning Rate 0.001.
   Path to model - **HW3/src/model/trained/q1/q1-model-no-bias-lr-001**



ii. **Final test dataset results** for parameter configuration “Bias Term - Learning Rate 0.001“

1. Confusion Matrix



2. Accuracy and F1 score on Test Dataset

```
Test accuracy / Q1:  0.81
F1 score / Q1:  0.81
```

iii. Using a neural net on the insurability dataset is a bad idea because
   1. Low count of data points within the dataset are too low to train a neural net properly
   2. Low count of meaningful features. Neural nets require a high dimension feature space to learn a lot of necessary parameters.

iv. We note that a **bias** term is helpful to **improve** accuracy as it **provides flexibility** to fit training data that may not pass through origin. We also note that a learning rate of **0.001** is better as it **prevents overshooting**.

2. (4.0 points) Implement neural network in PyTorch with an architecture of your choosing (a deep feed-forward neural network is fine) to perform 10-class classification on the MNIST dataset. Apply the same data transformations that you used for Homework #2. You are encouraged to use a different optimizer and the cross-entropy loss function that comes with PyTorch.Describe your design choices, provide a short rationale for each and explain how this network differs from the one you implemented for Part 1. Compare your results to Homework #2 and analyze why your results may be different. In addition, please provide the learning curves and confusion matrix as described in Part 1.
   a. Architectural choices
      i. In order for a NN to be considered “deep” it has to have **at least 2** hidden layers. The previous network only has 1 hidden layer which is why it's termed as a “simple” NN.
      ii. The count of neurons in the hidden layer is kept to be the average of input and output neurons. **So average of 784 and 10 which is 397.**

iii. I experimented with a quicker learning rate of 0.01 but the training accuracy was too low so I went with **0.001**.

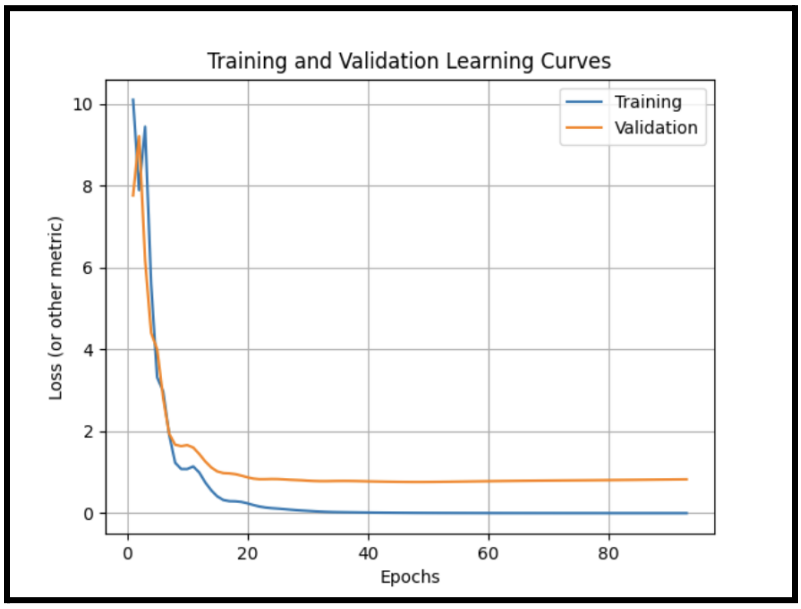
iv. Additionally, I also implemented an early training stop condition based on validation accuracy to avoid overfitting.

v. Our results are **slightly worse** in terms of accuracy for this HW. This is because a NN may be **overkill** for a relatively smaller dataset. And a simpler model like k means might be suitable.

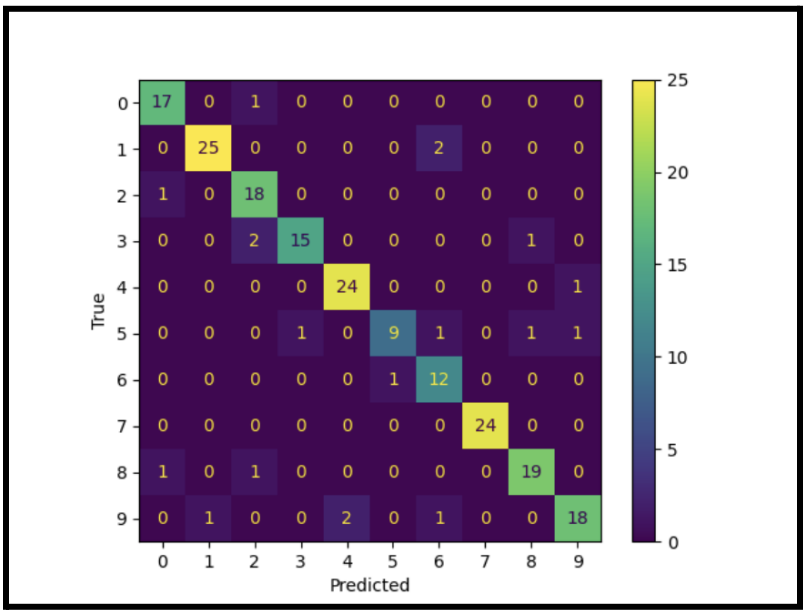
b. Learning Curve

i. Bias - Learning Rate 0.01

Path to mode - **HW3/src/model/trained/q2/model**



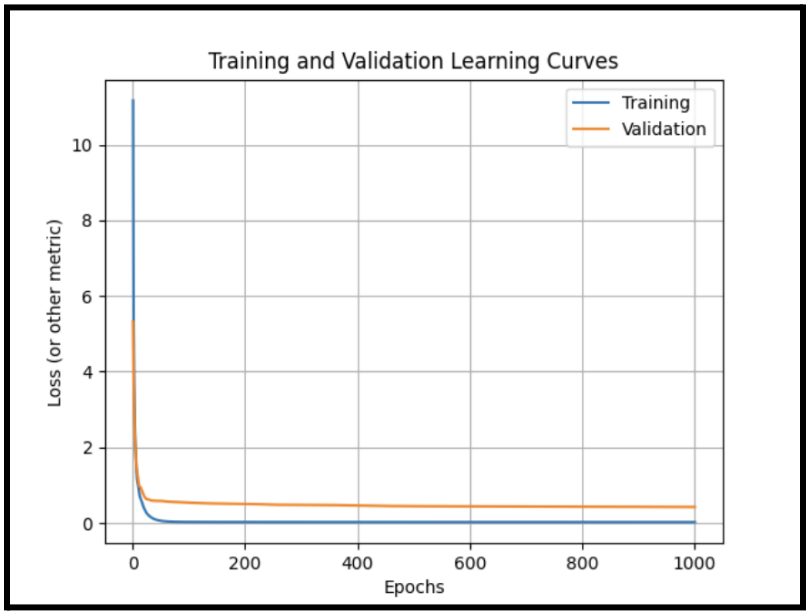
ii. Confusion matrix



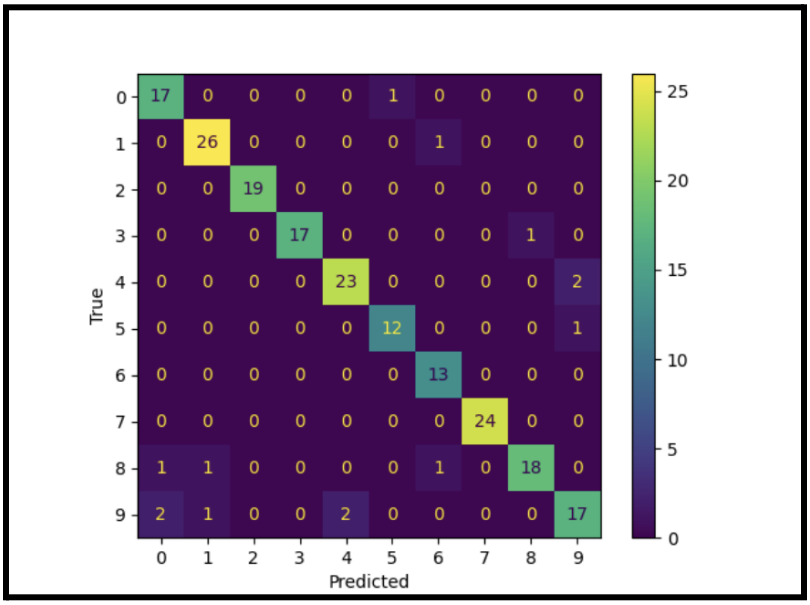
iii. Accuracy and F1 score on Test Dataset

```
Test accuracy / Q2:  0.905
F1 score / Q2:  0.905
```

3. (2.0 points) Add a regularizer of you choice to the 10-class classifier that you implemented for Part 2. Describe your regularization, the motivation for your choice and analyze the impact of the performance of the classifier. This analysis should include a comparison of learning curves and performance.
    a. I chose **L2 regularization** because L1 is more suitable where we want our model to prioritize learning from certain features over others. The MNIST dataset doesn't really have distinct features as its just capturing a pixel. L2 regularization is more general and can be used to further prevent overfitting.
        i. As a result of defining a **weight decay** of **0.1** the model consistently shows a better accuracy over the previous model in Q2.
        ii. Learning Curve
            1. Bias - Learning Rate - 0.001 - Weight Decay - 0.1



2. Confusion Matrix

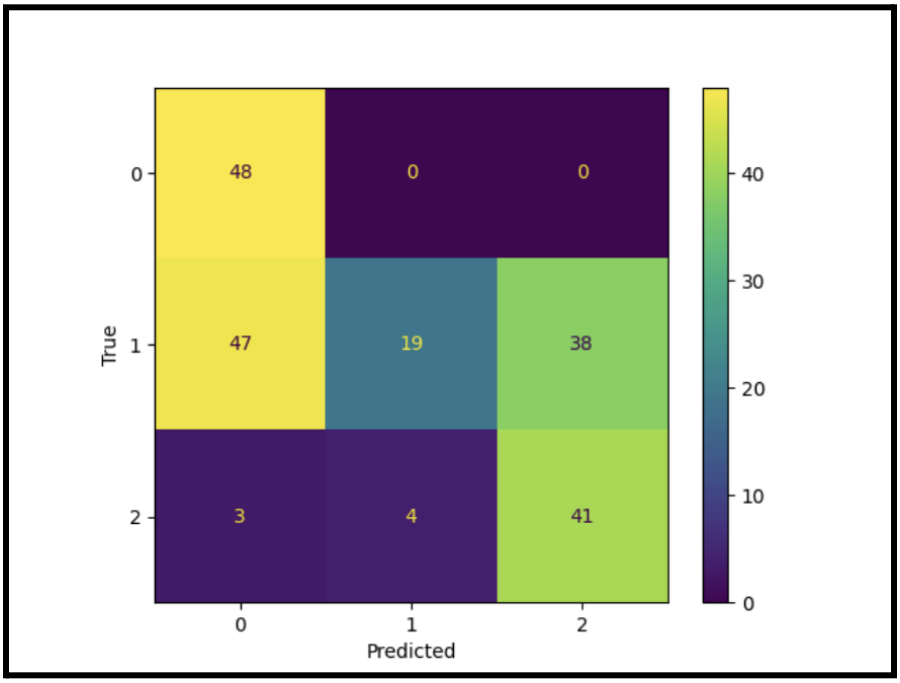


3. Accuracy and F1 score

```
Test accuracy / Q3:  0.93
F1 score / Q3:  0.93
```

4. (1.0 bonus points) Re-implement the 3-class classification feed-forward neural network from Part 1 by calculating and applying the gradients without a PyTorch optimizer. Do not use a bias term for this part of the assignment. Also, it may be easier to perform Part 4 with in-line code (e.g., weight vectors, matrix algebra operations and your own sigmoid() function). Note: You can check your gradient calculations by comparing before/after update weight matrices against a parallel implementation using a PyTorch optimizer.
    a. Path to model - **HW3/src/model/trained/q4**
    b. Path to code - **HW3/src/model/trained/simple_ffnn.py:89**

c. Confusion Matrix



d. Accuracy and F1 Score

```
Test accuracy / Q4:  0.54
F1 score / Q4:  0.54
```