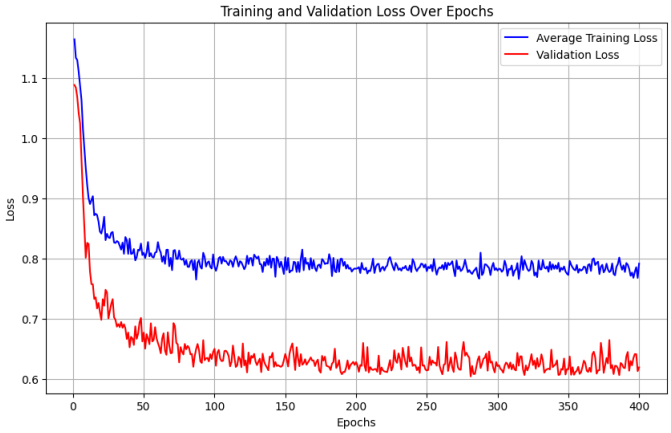
CS 349 Homework 3 Writeup

Coel Morcott and Vatsal Bhargava
CS 349 Fall ‘23

Question 1:

In this section we will discuss what hyper-parameters we used for our classify_insurability() function as well as go over why using a neural network on this data can be bad. We will start with the latter first. As this data deals with classifying how insurable a person is based on three features (age, exercise, and cigarettes), there can be some ethical concerns with this neural net. The predictions we make have massive impacts on people, as it could potentially prevent them from getting insurance and life supporting care. While neural nets have great use cases, it is important to recognize the context in which you use them and the impacts they can have on people. You are reducing impactful life services to purely statistics, so we need to make sure as engineers we are considering ethical factors, transparency, and consistency with our data. Moving onto the hyper-parameters, we chose to experiment with learning rate, epochs, and loss function. For the loss function we played around with both mean squared error and cross entropy, but ultimately found CRE to be better. We believe this result to hold because our model is outputting a probabilistic classification, which is what we want, where a MSE loss function would be more applicable to regression problems. For the epochs, we kept adjusting this number to empirically determine the optimal number. Too low and your data is not trained enough, but too high can cause overfitting. Lastly, we played around with the learning rate for our model as well. As the learning rate is the step size during the optimization process, if we set it too large we may overshoot any minima, but if it is too small then convergence can be too slow and may get stuck in saddle points. After adjusting this number, we went with a rate of 1e-3.

Learning Curves for Training and Validation Sets:



Confusion Matrix and F1 Score:

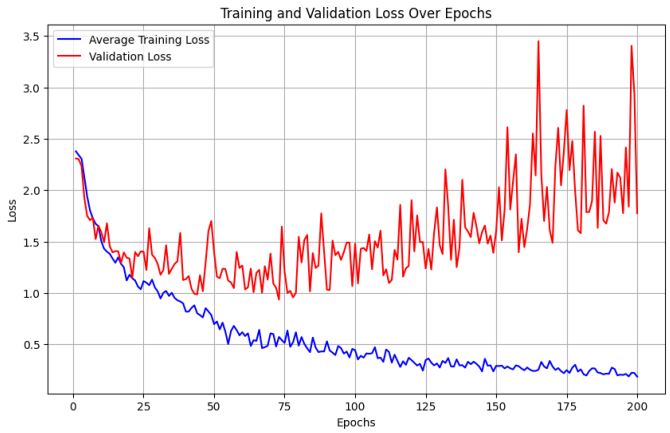
```
F1 Score: 0.9105683453681798
Confusion Matrix:
```

```
[[48  0  0]
[11 93  0]
[ 0  7 41]]
```

Question 2:

To classify MNIST, we implemented a deep feed forward neural network. It has 3 more hidden layers than in part 1 and 10 output neurons (the numbers 0-9). We added a bottle neck in the hidden layer, used LeakReLU, cross entropy loss function, and an adam optimizer. We used the bottleneck because it can act as a regularizer in a way by limiting the capacity of the network and also adds feature compression and abstraction. We made the hidden layer deeper because we were instructed to, but it also allows for the model to learn more complex relationships. We chose LeakyReLU because it allows a small constant gradient leak, maintaining a flow of gradients through the network and assurance of continued learning. The adam optimizer was chosen because it can converge quickly and is effective in large data sets, which we have here. Lastly, we used Cross Entropy Loss because it is a good choice for classification models. While there are many similarities between this network and part 1, the increased depth it has allows for more complex relationships and patterns to be learned. If we compare it to our work in Homework 2 (KNN and KMeans), we can see that our accuracy for those models actually outperformed this neural network by a few percentage points. This could be because of a couple factors. First off, while the dataset is large in terms of 784 input features for each row, there are not really complex relationships, they are just pixel values. Deep neural networks are based on complex feature abstraction, so their power may not be realized here. Also, while we were happy with our accuracy, it could be possible that different tunings to our network would have made the model better. Lastly, with a DNN, you always run the risk of overfitting, leading to poor predictions.

Learning Curves for Training and Validation Sets:



Confusion Matrix and F1 Score:

```
F1 Score: 0.7879484988853036
Confusion Matrix:
 [[12  0  2  0  0  2  2  0  0  0]
 [ 0 25  0  0  0  0  1  0  1  0]
 [ 0  0 17  2  0  0  0  0  0  0]
 [ 0  0  2 15  0  0  0  1  0  0]
 [ 0  0  0  2 22  0  0  0  1  0]
 [ 0  0  0  0  0  6  2  0  4  1]
 [ 0  0  0  0  0  0 12  0  1  0]
 [ 0  0  0  0  1  0  0 20  0  3]
 [ 2  0  0  2  1  1  0  0 15  0]
 [ 0  0  0  0  5  0  2  1  0 14]]
```

Question 3:

Describe what regularizer we used, the motivation behind it, and the impact it had on performance.

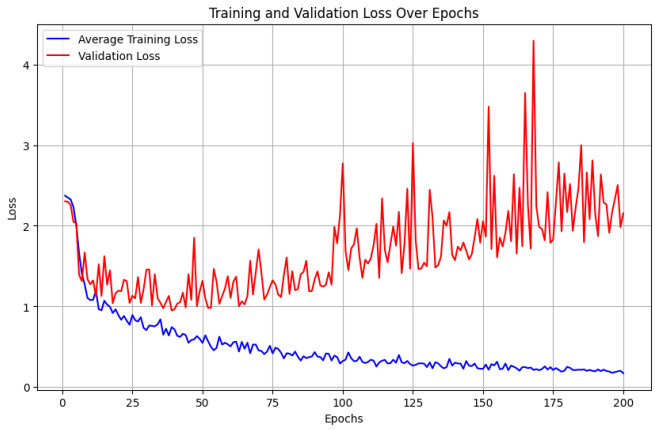
We used L2 regularization in our code as can be seen in this line:

```
   optimizer = torch.optim.Adam(ff.parameters(), lr=1e-3, weight_decay=1e-3)
```

This weight_decay parameter is part of torch and allows us to add a regularizer with ease. We chose this method because the L2 regularizer in theory, reduces overfitting of the model by penalizing larger weights and keeping things more spread out. Also, the simplicity of adding it played a factor into our choice, simply adding a single parameter is time efficient and simple.

Through empirical calculations of tinkering the learning rate, weight decay, and epochs we determined that a learning rate of 1e-3 with weight decay of 1e-5 and 75 epochs yielded our best results.

Learning Curve with Regularizer:



F1 and Confusion Matrix:

```
F1 Score: 0.7812393216679981
Confusion Matrix:
 [[14  0  0  0  0  1  3  0  0  0]
 [ 0 23  1  2  0  0  0  1  0  0]
 [ 0  0 14  4  0  0  0  0  1  0]
 [ 0  0  3 10  0  1  1  0  3  0]
 [ 0  0  0  0 22  1  0  0  1  1]
 [ 1  0  0  0  1  8  0  0  3  0]
 [ 0  0  0  0  0  0 13  0  0  0]
 [ 1  0  0  1  1  0  0 21  0  0]
 [ 0  0  0  1  0  2  1  1 15  1]
 [ 0  0  0  0  3  0  1  2  0 16]]
```

Comparison of Performance for the two:

As you can tell by our F1 scores, the model with the regularizer slightly outperformed our mnist classifier without the regularizer. This is expected as adding a regularizer should reduce overfitting and thus improve results on the test set. To me, there is no real pattern in the misclassification for either matrix so I will chalk up the misclassifications to randomness.

The learning curves for the 2 graphs although are similar in shape, have some keu nuanced differences. The first one being the gap between the validation loss and training loss after an ideal number of epochs. This is the main takeaway from the graphs as it shows how the model is much less overfit because the performance on validation is not spiraling down.