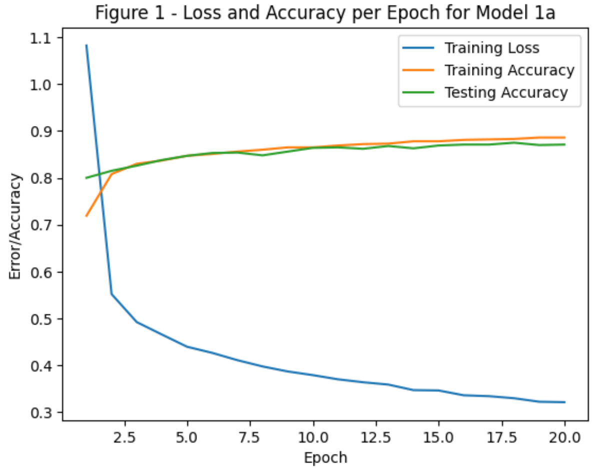
https://github.com/Christian-Martens-UNCC/ECGR-4106/tree/main/Homework_0-Multi_Layer_Perceptron

Problem 1:

A) It does not appear that my network needs more epochs for full training as the training and validation accuracy have converged and are pretty high. There does appear to be some very slight overfitting at the very end of the training since the training accuracy is climbing slightly while the testing accuracy is pretty much stationary.

```
Epoch 1:
        Duration = 1.662 seconds
        Training Loss: 1.08243
        Training Accuracy: 0.719
        Validation Accuracy: 0.8
Epoch 5:
        Duration = 1.523 seconds
        Training Loss: 0.43935
        Training Accuracy: 0.847
        Validation Accuracy: 0.847
Epoch 10:
        Duration = 1.544 seconds
        Training Loss: 0.37876
        Training Accuracy: 0.865
        Validation Accuracy: 0.864
Epoch 15:
        Duration = 1.58 seconds
        Training Loss: 0.34601
        Training Accuracy: 0.878
        Validation Accuracy: 0.869
Epoch 20:
        Duration = 1.563 seconds
        Training Loss: 0.321
        Training Accuracy: 0.886
        Validation Accuracy: 0.871

Total Training Time = 34.137 seconds
Average Training Time per Epoch = 1.707 seconds
```

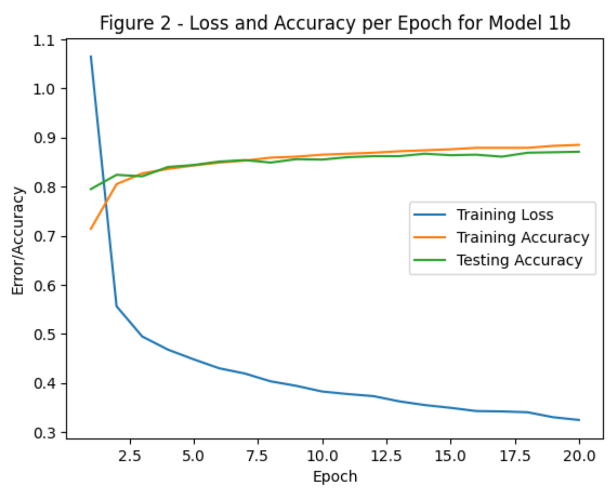
Figure 1 - Loss and Accuracy per Epoch for Model 1a



B) The training results are very similar to the baseline. The validation and training accuracies are more inconsistent but there also seems to be less overfitting compared to the baseline model. During other training sessions, this method also resulted in a ~2% increase in validation accuracy.

```
In [234]: model_2 = nn.Sequential(
                      nn.Identity(),
                      nn.Linear(784, 1024),    # First Hidden Layer
                      nn.ReLU(),
                      nn.Identity(),
                      nn.Linear(1024, 512),    # Second Hidden Layer
                      nn.ReLU(),
                      nn.Identity(),
                      nn.Linear(512, 256),   # Third Hidden Layer
                      nn.ReLU(),
                      nn.Linear(256, 10)).to(device=try_gpu())   # Output Layer

          optimizer_2 = optim.SGD(model_2.parameters(), lr = 1e-3, weight_decay=1e-4)

          model_2.eval()

Out[234]: Sequential(
            (0): Identity()
            (1): Linear(in_features=784, out_features=1024, bias=True)
            (2): ReLU()
            (3): Identity()
            (4): Linear(in_features=1024, out_features=512, bias=True)
            (5): ReLU()
            (6): Identity()
            (7): Linear(in_features=512, out_features=256, bias=True)
            (8): ReLU()
            (9): Linear(in_features=256, out_features=10, bias=True)
          )
```

```
Epoch 1:
        Duration = 1.694 seconds
        Training Loss: 1.06484
        Training Accuracy: 0.714
        Validation Accuracy: 0.795
Epoch 5:
        Duration = 1.852 seconds
        Training Loss: 0.44831
        Training Accuracy: 0.843
        Validation Accuracy: 0.844
Epoch 10:
        Duration = 1.772 seconds
        Training Loss: 0.3828
        Training Accuracy: 0.865
        Validation Accuracy: 0.855
Epoch 15:
        Duration = 1.693 seconds
        Training Loss: 0.34954
        Training Accuracy: 0.876
        Validation Accuracy: 0.864
Epoch 20:
        Duration = 1.829 seconds
        Training Loss: 0.32481
        Training Accuracy: 0.885
        Validation Accuracy: 0.871

Total Training Time = 37.19 seconds
Average Training Time per Epoch = 1.86 seconds
```

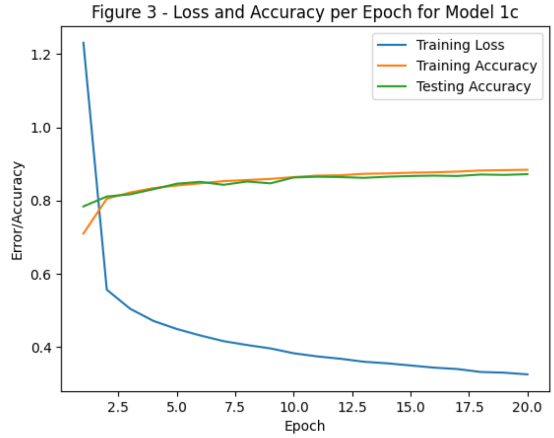


C) The baseline model and the weight decay model both performed similarly to the dropout model. This method was consistently faster than the other two however, likely due to the reduced number of computations that needed completing. During other training sessions, this method generally performed with the highest validation accuracy, sometimes reaching as high as 90%.

```
In [237]:  model_3 = nn.Sequential(
                       nn.Dropout(p=0.3),
                       nn.Linear(784, 1024),    # First Hidden Layer
                       nn.ReLU(),
                       nn.Dropout(p=0.3),
                       nn.Linear(1024, 512),    # Second Hidden Layer
                       nn.ReLU(),
                       nn.Dropout(p=0.3),
                       nn.Linear(512, 256),    # Third Hidden Layer
                       nn.ReLU(),
                       nn.Linear(256, 10)).to(device=try_gpu())  # Output Layer

           optimizer_3 = optim.SGD(model_3.parameters(), lr = 1e-3)

           model_3.eval()

Out[237]:  Sequential(
             (0): Dropout(p=0.3, inplace=False)
             (1): Linear(in_features=784, out_features=1024, bias=True)
             (2): ReLU()
             (3): Dropout(p=0.3, inplace=False)
             (4): Linear(in_features=1024, out_features=512, bias=True)
             (5): ReLU()
             (6): Dropout(p=0.3, inplace=False)
             (7): Linear(in_features=512, out_features=256, bias=True)
             (8): ReLU()
             (9): Linear(in_features=256, out_features=10, bias=True)
           )
```

```
Epoch 1:
        Duration = 2.05 seconds
        Training Loss: 1.23049
        Training Accuracy: 0.71
        Validation Accuracy: 0.784
Epoch 5:
        Duration = 1.537 seconds
        Training Loss: 0.44936
        Training Accuracy: 0.841
        Validation Accuracy: 0.846
Epoch 10:
        Duration = 1.564 seconds
        Training Loss: 0.38318
        Training Accuracy: 0.864
        Validation Accuracy: 0.863
Epoch 15:
        Duration = 1.516 seconds
        Training Loss: 0.34995
        Training Accuracy: 0.876
        Validation Accuracy: 0.867
Epoch 20:
        Duration = 1.525 seconds
        Training Loss: 0.32561
        Training Accuracy: 0.884
        Validation Accuracy: 0.872

Total Training Time = 34.5 seconds
Average Training Time per Epoch = 1.725 seconds
```

Figure 3 - Loss and Accuracy per Epoch for Model 1c



D) The training times for these both models were not much better than the untrained models. As we can see, there was not much change between the weights of the baseline model and the weights of the weight decay and dropout models. This method did allow us to further train our model, which was used to confirm that training had completed and that the models was slowly drifting towards overfitting if they continued for additional epochs.

Figure 4 - Loss and Accuracy per Epoch for Model 1b with Preloaded Weights

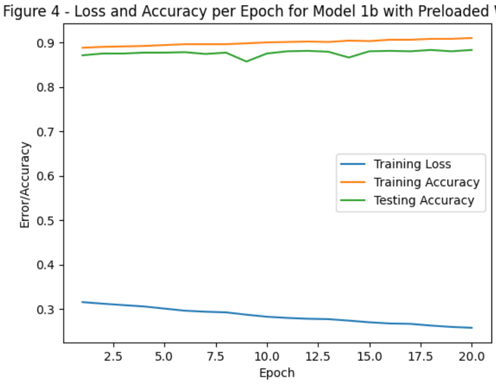
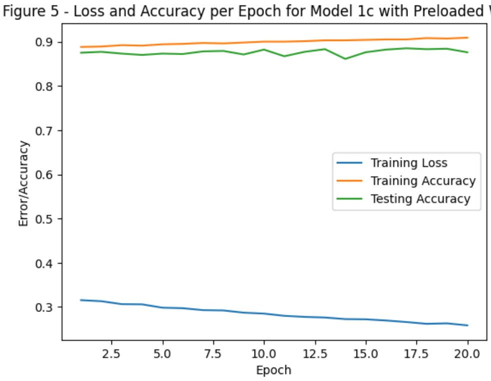


Figure 5 - Loss and Accuracy per Epoch for Model 1c with Preloaded Weights

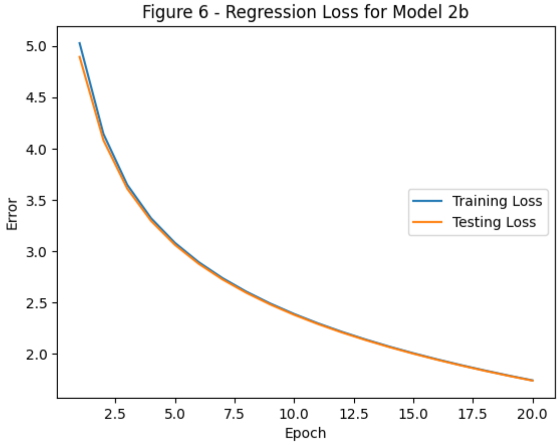


Problem 2:

A) When we standardize the continuous numerical features of a dataset, the features are condensed such that all the features are relatively the same size compared to one another. This makes sure that the model doesn't get dominated by features with large input values and allows features with smaller input values to contribute to training the model. This overall helps with generalization.

B) The model complexity was increased by adding in more hidden layers with more neurons. It is hard to compare training times since the model in class was ran on a different computer, but it appears that my model has a lower validation loss than the model in the lecture. I would expect the model to take longer to train given it has more layers and thus more computations.

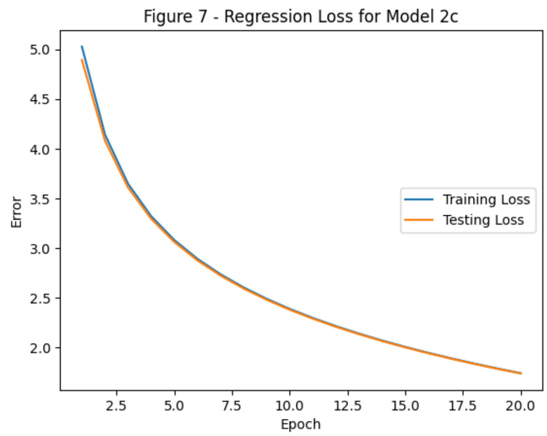
Figure 6 - Regression Loss for Model 2b




```
In [548]: model_house_1 = nn.Sequential(
                nn.Identity(),
                nn.LazyLinear(512),   # First Hidden Layer
                nn.ReLU(),
                nn.Identity(),
                nn.Linear(512, 256),   # Second Hidden Layer
                nn.ReLU(),
                nn.Identity(),
                nn.Linear(256, 128),   # Third Hidden Layer
                nn.ReLU(),
                nn.Linear(128, 1)).to(device=try_gpu())  # Output Layer

          optimizer_house_1 = optim.SGD(model_house_1.parameters(), lr = 1e-3)

          model_house_1.eval()

Out[548]: Sequential(
            (0): Identity()
            (1): LazyLinear(in_features=0, out_features=512, bias=True)
            (2): ReLU()
            (3): Identity()
            (4): Linear(in_features=512, out_features=256, bias=True)
            (5): ReLU()
            (6): Identity()
            (7): Linear(in_features=256, out_features=128, bias=True)
            (8): ReLU()
            (9): Linear(in_features=128, out_features=1, bias=True)
          )
```

C) This model trained significantly faster than the previous model (roughly a 10% decrease in training time). Additionally, we can see that these methods did not increase the validation loss.

```
In [555]: model_house_2 = nn.Sequential(
                nn.Dropout(p=0.3),
                nn.LazyLinear(512),   # First Hidden Layer
                nn.ReLU(),
                nn.Dropout(p=0.3),
                nn.Linear(512, 256),   # Second Hidden Layer
                nn.ReLU(),
                nn.Dropout(p=0.3),
                nn.Linear(256, 128),   # Third Hidden Layer
                nn.ReLU(),
                nn.Linear(128, 1)).to(device=try_gpu())  # Output Layer

          optimizer_house_2 = optim.SGD(model_house_2.parameters(), lr = 1e-3, weight_decay=1e-4)

          model_house_2.eval()

Out[555]: Sequential(
            (0): Dropout(p=0.3, inplace=False)
            (1): LazyLinear(in_features=0, out_features=512, bias=True)
            (2): ReLU()
            (3): Dropout(p=0.3, inplace=False)
            (4): Linear(in_features=512, out_features=256, bias=True)
            (5): ReLU()
            (6): Dropout(p=0.3, inplace=False)
            (7): Linear(in_features=256, out_features=128, bias=True)
            (8): ReLU()
            (9): Linear(in_features=128, out_features=1, bias=True)
          )
```



D) My predictions to Kaggle were not as good as I would've liked. Had I more time, I would've gone back and tried other ways of implementing Pytorch so that the regression is more accurate. My predictions were all very high compared to what I was expecting.