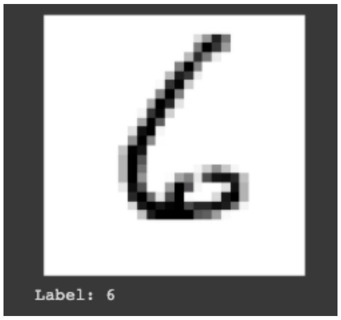
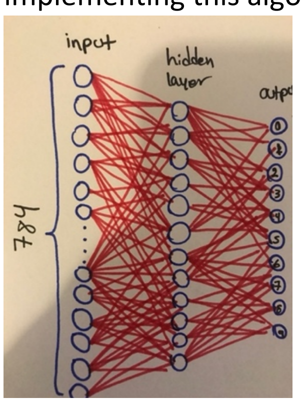
Dataset and Preprocessing

I am using the MNIST dataset to evaluate and compare model architecture and parameters. Both networks presented take an image (28x28 pixels) of a handwritten digit (between 0 and 9) and it will try to predict its class. Below is an example of a handwritten letter and the correct label.



After loading the MNIST dataset, all images were normalized from 0–255 pixel data to 0-1 to allow for faster learning of input node parameters (normalized to avoid exploding gradients). Before using the dataset, the labels (0-9) had to be converted from ordinal values to sets (this mean that value 1 is not smaller than 2, they are just two different sets of classification), this was done by one hot encoding the labels, resulting in 10 output nodes which was easier to code than how thecscikit-learn library handles it.

ANN Design

- Both models have the same input nodes. Input layer is defined as a vector with 784 entries (from the 28x28 image).
- This model has 1 hidden layer with 64 nodes and uses the sigmoid activation function
- The output layer is 10 nodes corresponding to numbers, it layer uses the softmax activation function to normalize all values between 0 and 1 (sum of 1), this way output values exist as probabilities and the largest probability is chosen as the prediction.
- Backpropagation or gradient descent was used to find best results (minima of curve). By implementing this algorithm, possibly underfitted graphs could be fitted more optimally.
- 

Model 1 – No Library Multilayer Neural Net

- **Initial weights** = Initial weights and biases were randomized. I chose to randomly initialize all weights since the data has been normalized and the spread is in a very small distribution. The initial weights can be a factor in accuracy since a model can converge to a local minima, but I feel that this is mitigated by normalizing the dataset and training until the model converges.
- **Hidden layer** = Since the data is not linearly separable, a hidden layer is required. I avoided adding more hidden layers as one is usually sufficient for majority of simple problems.
- **Nodes** = I initially started with only 12 neurons in the hidden layer, keeping all other parameters the same, it seemed the model was underfitting because the training accuracies were quite low. I then increased the number of nodes to 32 and then 64. In my experience 64 nodes was enough to observe a significant improvement in accuracy and sufficient enough as loss started to level off.
- **Learning rate** = This is a parameter of gradient descent, learning rate informs the model on the level of change depending on the estimated error after weight update. I first chose a very small learning rate of 0.001, however it resulted in a very long training time, therefore I chose the value of 1 as it is also not too big where it would result in sub-optimal set of weights.
- **Momentum** = I decided to add momentum which replaces gradient with a momentum which is an aggregation of all gradients, after adding momentum and trying various values I noticed a decent improvement in training time.
- **Epochs** = When a dataset, such as this one, is large, epochs and batches can be used to divide the data into smaller sizes and given to the model in steps and updates the weights at the end of every step to fit the given data, therefore only having one epoch could lead to underfitting. However, having too many epochs could also lead to overfitting so I used various values to observe changes in model performance. It seemed that after 200-250 epochs the model could be overfitting, so I set the epoch number to 200.
- **Batches** = Batch represents the subset of training examples in a single batch before the model's internal parameters are updated. I decided to include batch in my network because it requires less memory and improves gradient descent. A large batch number will make coarse updates to the weights whereas a smaller batch number implies the model updates often. I noticed that a very large batch size (1000) decreased training time significantly, however it also does much worse with test data compared to smaller batch numbers. Another factor I noticed was that using very high batch numbers took much longer for loss to converge. Using batch, allows the model to converge faster and avoid excessive number of epochs if correctly set.
- **Weight updates** = instead of loops, I used matrix multiplication during back propagation to reduce training time. Doing so reduced my model training time from approximately 90+ minutes to 1-5 minutes

Model 1 (100 epochs)

```
Final loss: 0.7298848203148632

Train data stats:
[[5131    1  179   89   40  267  126   80  116   54]
 [   1 6293  135   48   21   45   46  148  207   17]
 [ 116  133 4376  223   99  111  287  170  230   65]
 [  83   14  247 4592   28  538   18   72  480  124]
 [  16   12  165   34 4480  155  140  221  169  491]
 [ 242   28   65  522   63 3445  173   85  399  141]
 [ 190   12  265   30  220  188 4969   36  134   62]
 [  53   29  179   75  114   47   26 4976   77  490]
 [  78  204  307  415  188  451   82  100 3715  167]
 [  13   16   40  103  589  174   51  377  324 4338]]
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      6083
           1       0.93      0.90      0.92      6961
           2       0.73      0.75      0.74      5810
           3       0.75      0.74      0.75      6196
           4       0.77      0.76      0.76      5883
           5       0.64      0.67      0.65      5163
           6       0.84      0.81      0.83      6106
           7       0.79      0.82      0.81      6066
           8       0.63      0.65      0.64      5707
           9       0.73      0.72      0.72      6025

    accuracy                           0.77     60000
   macro avg       0.77      0.77      0.77     60000
weighted avg       0.77      0.77      0.77     60000
```

```
Final loss: 0.6927345992527022
Test data stats:
[[ 867    0   31   13    5   43   32   10   21    9]
 [   0 1052   31    2    3    4    3   27   11    5]
 [  21   19  763   30   13   24   52   31   25   11]
 [   7    4   53  806    2   95    2   19   93   14]
 [   5    1   26    4  762   23   33   25   29   82]
 [  40    3   10   78    7  591   27   10   57   27]
 [  25    2   43    2   35   27  786    3   28    6]
 [   5    5   28   12   17    7    6  815   16   58]
 [   9   48   41   52   27   59   12   28  627   13]
 [   1    1    6   11  111   19    5   60   67  784]]
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1031
           1       0.93      0.92      0.93      1138
           2       0.74      0.77      0.76       989
           3       0.80      0.74      0.77      1095
           4       0.78      0.77      0.77       990
           5       0.66      0.70      0.68       850
           6       0.82      0.82      0.82       957
           7       0.79      0.84      0.82       969
           8       0.64      0.68      0.66       916
           9       0.78      0.74      0.76      1065

    accuracy                           0.79     10000
   macro avg       0.78      0.78      0.78     10000
weighted avg       0.79      0.79      0.79     10000
```

(left is train, right is test)

Model 1 (200 epochs)

```
Final loss: 0.5518701951222055

Train data stats:
[[5343    1  124   69   28  197  100   56   69   42]
 [   1 6400   92   41   15   43   37  115  166   15]
 [  71   70 4777  201   64   82  175  147  164   62]
 [  56   18  182 4919   18  416   13   49  355  112]
 [  12   11  153   19 4855  108  107  142  109  399]
 [ 191   30   48  385   44 3933  133   44  318   99]
 [ 136   11  193   31  165  153 5257   16   90   32]
 [  41   26  143   69   67   39   17 5318   59  309]
 [  57  156  212  320  121  319   60   65 4284  144]
 [  15   19   34   77  465  131   19  313  237 4735]]
              precision    recall  f1-score   support

           0       0.90      0.89      0.89      6029
           1       0.95      0.92      0.94      6925
           2       0.80      0.82      0.81      5813
           3       0.80      0.80      0.80      6138
           4       0.83      0.82      0.83      5915
           5       0.73      0.75      0.74      5225
           6       0.89      0.86      0.88      6084
           7       0.85      0.87      0.86      6088
           8       0.73      0.75      0.74      5738
           9       0.80      0.78      0.79      6045

    accuracy                           0.83     60000
   macro avg       0.83      0.83      0.83     60000
weighted avg       0.83      0.83      0.83     60000
```

```
Final loss: 0.5264352336787862
Test data stats:
[[ 901    0   26    8    4   38   27    7   17    6]
 [   0 1073   16    3    2    1    3   27    7    4]
 [  14    8  841   29    9   14   24   34   21    7]
 [   3    4   33  851    1   69    1   12   71    9]
 [   2    2   22    2  826   21   26   18   19   70]
 [  32    2    4   61    7  662   25    4   54   22]
 [  18    3   31    4   21   25  841    2   23    5]
 [   3    3   23    9   10    5    2  853   18   42]
 [   7   39   30   35   17   46    6   21  700   16]
 [   0    1    6    8   85   11    3   50   44  828]]
              precision    recall  f1-score   support

           0       0.92      0.87      0.89      1034
           1       0.95      0.94      0.94      1136
           2       0.81      0.84      0.83      1001
           3       0.84      0.81      0.82      1054
           4       0.84      0.82      0.83      1008
           5       0.74      0.76      0.75       873
           6       0.88      0.86      0.87       973
           7       0.83      0.88      0.85       968
           8       0.72      0.76      0.74       917
           9       0.82      0.80      0.81      1036

    accuracy                           0.84     10000
   macro avg       0.84      0.83      0.83     10000
weighted avg       0.84      0.84      0.84     10000
```

Model 1 (200 batch, 20 epoch)

```
Final loss: 0.1666956036373846

Train data stats:
[[5797    0   27    9    7   32   30    8   16   21]
 [   0 6598   28   12    7   10    7   21   41    9]
 [  12   45 5648   82   26   23   28   48   38   12]
 [   8   15   51 5754    6  106    2   17   86   53]
 [   7    9   34    5 5570   16   32   27   16  134]
 [  21    7   15  125    7 5062   53    7   81   25]
 [  33    6   30    5   42   54 5736    2   34    4]
 [   7   20   47   38   20   10    0 6015   12   98]
 [  33   27   65   66   25   73   28   16 5478   40]
 [   5   15   13   35  132   35    2  104   49 5553]]
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      5947
           1       0.98      0.98      0.98      6733
           2       0.95      0.95      0.95      5962
           3       0.94      0.94      0.94      6098
           4       0.95      0.95      0.95      5850
           5       0.93      0.94      0.94      5403
           6       0.97      0.96      0.97      5946
           7       0.96      0.96      0.96      6267
           8       0.94      0.94      0.94      5851
           9       0.93      0.93      0.93      5943

    accuracy                           0.95     60000
   macro avg       0.95      0.95      0.95     60000
weighted avg       0.95      0.95      0.95     60000
```

```
Final loss: 0.20033133006317197
Test data stats:
[[ 963    0    9    1    1    8   10    1   10    6]
 [   0 1115    2    4    2    1    3   11    3    6]
 [   1    3  965   13    8    2    4   19    7    2]
 [   2    2   16  949    1   34    1   14   18    8]
 [   0    0    7    1  914    4    7    5    9   26]
 [   4    1    1   21    0  792    8    1   17    8]
 [   8    5   11    1   16   18  923    0   14    1]
 [   2    2    6    7    3    4    0  947    4    9]
 [   0    7   13   10    4   21    2    3  880    5]
 [   0    0    2    3   33    8    0   27   12  938]]
              precision    recall  f1-score   support

           0       0.98      0.95      0.97      1009
           1       0.98      0.97      0.98      1147
           2       0.94      0.94      0.94      1024
           3       0.94      0.91      0.92      1045
           4       0.93      0.94      0.94       973
           5       0.89      0.93      0.91       853
           6       0.96      0.93      0.94       997
           7       0.92      0.96      0.94       984
           8       0.90      0.93      0.92       945
           9       0.93      0.92      0.92      1023

    accuracy                           0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000
```

(left is train, right is test)

Model 2 – scikit-learn MLP Classifier

- First, I ran the MLP model with the same parameters (shown below) that I picked for the above model.
- ```
  # network parameters
  n_x = X_train.shape[0] # num input nodes
  n_h = 64               # num hidden layer nodes
  learning_rate = 1      # learning rate
  epochs = 20            # epochs
  momentum = 0.5         # momentum
  ```
- Epoch was set to 20 at first because MLP by default implements 200 batches just like model 1 (with batch). But I had to increase it to 50 because the solution wasn't optimal
- ```
  Train data stats:
  [[5923    0    0    0    0    0    0    0    0    0]
   [   0 6742    0    0    0    0    0    0    0    0]
   [   0    0 5958    0    0    0    0    0    0    0]
   [   0    0    0 6131    0    0    0    0    0    0]
   [   0    0    0    0 5842    0    0    0    0    0]
   [   0    0    0    0    0 5421    0    0    0    0]
   [   0    0    0    0    0    0 5918    0    0    0]
   [   0    0    0    0    0    0    0 6265    0    0]
   [   0    0    0    0    0    0    0    0 5851    0]
   [   0    0    0    0    0    0    0    0    0 5949]]
                precision    recall  f1-score   support

             0       1.00      1.00      1.00      5923
             1       1.00      1.00      1.00      6742
             2       1.00      1.00      1.00      5958
             3       1.00      1.00      1.00      6131
             4       1.00      1.00      1.00      5842
             5       1.00      1.00      1.00      5421
             6       1.00      1.00      1.00      5918
             7       1.00      1.00      1.00      6265
             8       1.00      1.00      1.00      5851
             9       1.00      1.00      1.00      5949

      accuracy                           1.00     60000
     macro avg       1.00      1.00      1.00     60000
  weighted avg       1.00      1.00      1.00     60000
  ```
  ```
  Test data stats:
  [[ 968    0    6    0    1    3    6    1    6    1]
   [   1 1122    3    0    0    0    2    6    0    2]
   [   1    4 1001    8    4    0    4   10    4    0]
   [   0    2    5  978    1   11    1    3    4    3]
   [   0    0    4    1  961    0    6    1    5    7]
   [   1    1    1    8    0  863    5    0    1    4]
   [   4    2    1    0    5    4  933    0    2    1]
   [   1    1    4    2    0    1    0  995    5    4]
   [   3    3    7    4    1    7    1    6  941    5]
   [   1    0    0    9    9    3    0    6    6  982]]
                precision    recall  f1-score   support

             0       0.99      0.98      0.98       992
             1       0.99      0.99      0.99      1136
             2       0.97      0.97      0.97      1036
             3       0.97      0.97      0.97      1008
             4       0.98      0.98      0.98       985
             5       0.97      0.98      0.97       884
             6       0.97      0.98      0.98       952
             7       0.97      0.98      0.98      1013
             8       0.97      0.96      0.96       978
             9       0.97      0.97      0.97      1016

      accuracy                           0.97     10000
     macro avg       0.97      0.97      0.97     10000
  weighted avg       0.97      0.97      0.97     10000
  ```

- Then, I decided to use Randomized Search to find optimal hyperparameters, and to also compare these with ones chosen for the above models. I will use these parameters in model 2 to observe performance.
-
```
param_grid = {
    'hidden_layer_sizes':[32,64,96,128],
    'activation':['identity','logistic','tanh','relu'],
    'learning_rate':['constant', 'invscaling', 'adaptive'],
    'solver':['lbfgs','sgd','adam'],
    'batch_size':[100,250,500]
}
```
- Following parameters were picked:
-
```
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed: 177.4min finished
best score 0.9776428571428573
best score {'solver': 'adam', 'learning_rate': 'constant', 'hidden_layer_sizes': 128, 'batch_size': 250, 'activation': 'logistic'}
```
-
```
Test data stats:
[[ 972    1    3    0    1    3    5    1    5    1]
 [    0 1122    0    0    0    1    2    2    0    2]
 [    1    4 1009    8    0    0    1   10    2    0]
 [    1    0    5  984    1    8    1    2    4    4]
 [    2    0    1    0  961    3    3    1    6   11]
 [    0    1    0    5    0  867    5    1    4    2]
 [    2    2    4    0    6    5  940    0    5    1]
 [    1    1    5    5    0    0    0 1005    5    3]
 [    1    4    5    3    2    3    1    2  941    5]
 [    0    0    0    5   11    2    0    4    2  980]]
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       992
           1       0.99      0.99      0.99      1129
           2       0.98      0.97      0.98      1035
           3       0.97      0.97      0.97      1010
           4       0.98      0.97      0.98       988
           5       0.97      0.98      0.98       885
           6       0.98      0.97      0.98       965
           7       0.98      0.98      0.98      1025
           8       0.97      0.97      0.97       967
           9       0.97      0.98      0.97      1004

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```
- As you can see, after parameter search, model 2's accuracy slightly improved but this was at the cost of additional processing as it took almost 3 hours to find these parameters. A bigger problem or list of parameters to search through will take exponentially longer.

Model Comparison and Critical analysis about possible causes for inaccuracies

- **Scores**: Model 2 (scikit-learn) overall had a better accuracy than model 1. This could be due to various reasons such as underfitting model 1. Model 1 achieved the best scores with batch implemented, however training scores did not exceed 95% and there are still misclassifications present. Perhaps training model 1 more could result in scores closer to those seen with model 2.
- **Time**: In terms of training time, model 1 variants generally took longer to run than MLP even with batch implemented. This could be because the library has more complex logic implemented such as solvers and adaptive learning rates which could more efficiently train model 2.