Bryant Har, Eric Sun

Part 1

The gradient of the loss is derived as follows:

$$R(h_W) = \frac{1}{\mathcal{D}} \sum L(h_W(x), y) + \frac{\gamma}{2}||W||^2$$

$$\nabla_W R(h_W) = \frac{1}{\mathcal{D}} \sum \nabla_W L(h_W(x), y) + \gamma W$$

$$\nabla_W R(h_W) = \frac{1}{\mathcal{D}} \sum_{\mathcal{D}} (\text{softmax}(Wx_i) - y_i)x_i^T + \gamma W$$

The gradient step is then:

$$w_{n+1} = w_n - \alpha \nabla R(h_W)$$

$$w_{n+1} = w_n - \alpha \left[\frac{1}{\mathcal{D}} \sum_{\mathcal{D}} (\text{softmax}(Wx_i) - y_i)x_i^T + \gamma W \right]$$

Parts 2 + 3 Timing Results

Non-Vectorized Gradient Descent

Below, for the non-vectorized GD, running 10 iterations took 21.781 seconds. Multiplying by 100, this would take 2,178 seconds or around half an hour for 1000 iterations.

```
[[-0.76723335 -0.02453466 -0.17042811 ...  0.69658406  0.88337969
  -0.34701082]
 [-0.97785297  0.45572859 -0.21204986 ...  0.18844613 -0.68401932
  -0.55696959]
 [ 0.69313035 -0.88233778 -0.0928733  ...  0.23537242 -0.53279844
  -0.28471874]
 ...
 [-0.39514742 -0.0547279  -0.95475117 ... -0.5297635  -0.92737043
   0.31955092]
 [-0.92989989  0.67117376  0.77031557 ... -0.47776568 -0.25448596
  -0.28537155]
 [-0.9226367   0.51405598  0.56886991 ... -0.97756347  0.95484518
  -0.32698318]]
~~~~~~~~~~
[[-0.76646646 -0.02451014 -0.17025776 ...  0.69588779  0.88249671
  -0.34666396]
 [-0.97687555  0.45527307 -0.21183791 ...  0.18825777 -0.68333561
  -0.55641287]
 [ 0.69243753 -0.88145584 -0.09278047 ...  0.23513715 -0.53226588
  -0.28443415]
 ...
 [-0.39475245 -0.0546732  -0.95379685 ... -0.52923397 -0.92644348
   0.31923151]
 [-0.92897041  0.67050288  0.7695456  ... -0.47728813 -0.25423158
  -0.2850863 ]
 [-0.92171448  0.51354216  0.5683013  ... -0.97658635  0.95389076
  -0.32665635]]
Took 21.781009912490845 Seconds
```

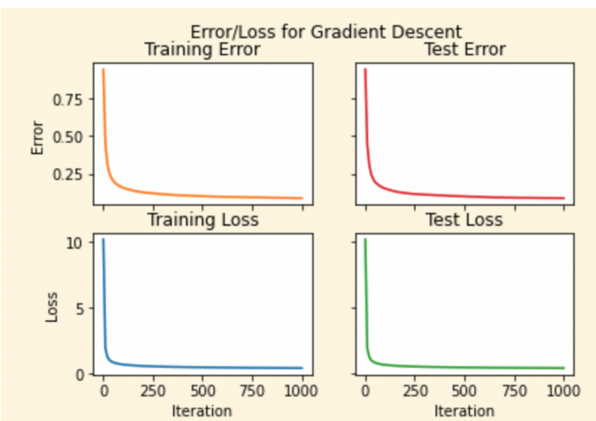
Vectorized Gradient Descent

The results using numpy vectorization is shown below. Using numpy optimizations, we observe a 30x speedup. This is significantly faster (21.781s compared to 0.779s). This is due to using C structures and vectorization abstracted away by numpy libraries.

```
[[-0.61718895 -0.22341677 -0.12206275 ...  0.09720566  0.64661168
   0.9037707 ]
 [-0.44781121  0.36554002  0.09229313 ...  0.3197573  -0.68323162
  -0.91016185]
 [-0.2676272   0.94364364  0.5493082  ...  0.08307416 -0.85001726
   0.71071592]
 ...
 [-0.51502511  0.20157841  0.89043481 ...  0.47052526  0.2190349
   0.57640983]
 [ 0.65520247 -0.44301076 -0.56409091 ... -0.31602092  0.17820254
  -0.32338706]
 [-0.40144334  0.69703143 -0.3257527  ...  0.19335304  0.98856365
  -0.32434167]]
Min: -0.9994966460420656, Max: 0.9994066374891519
~~~~~~~~~~~~~~~~~~~~~~~~
[[-0.61718895 -0.22341677 -0.12206275 ...  0.09720566  0.64661168
   0.9037707 ]
 [-0.44781121  0.36554002  0.09229313 ...  0.3197573  -0.68323162
  -0.91016185]
 [-0.2676272   0.94364364  0.5493082  ...  0.08307416 -0.85001726
   0.71071592]
 ...
 [-0.51502511  0.20157841  0.89043481 ...  0.47052526  0.2190349
   0.57640983]
 [ 0.65520247 -0.44301076 -0.56409091 ... -0.31602092  0.17820254
  -0.32338706]
 [-0.40144334  0.69703143 -0.3257527  ...  0.19335304  0.98856365
  -0.32434167]]
Min: -0.9985852842171392, Max: 2.0671477986599207
~~~~~~~~~~~~~~~~~~~~~~~~
Took 0.7794671058654785 Seconds
```

Part 4

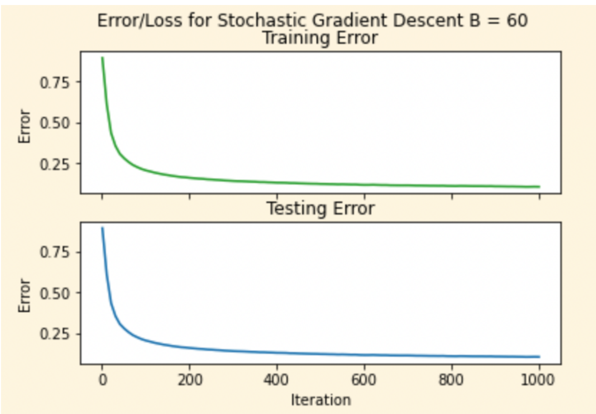
Plots of the training and testing error and loss over 1000 iterations are shown below. Loss is given by our loss function, while error is related to the percentage of incorrect predictions. Clearly, across all graphs, we see the typical rapid decrease in loss/error over iterations.

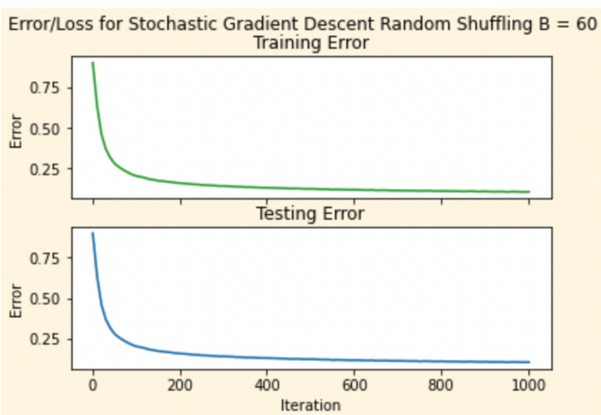


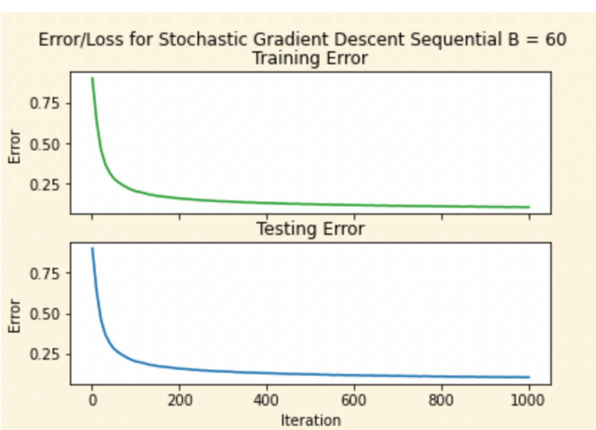
Part 5

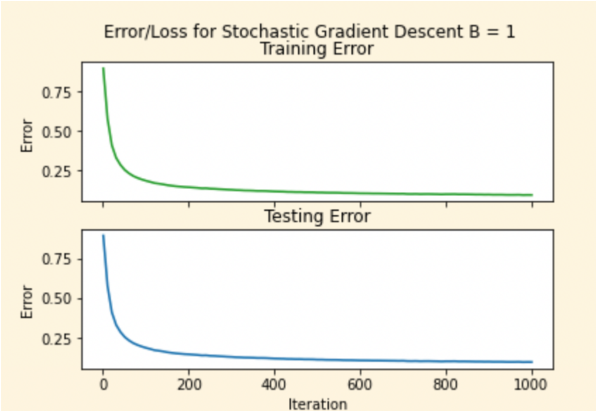
Overall, the six methods seem to converge to similar noise balls. By the final iterations, accuracy seems to stabilize at around 89% to 91% correct on the datasets with low variance across the models. Notably, however, the ***runtime*** was significantly different across the variations. Across both batch sizes, sequential stochastic gradient descent consistently ran the fastest, followed by random shuffling without replacement stochastic gradient, and unmodified gradient descent. Random shuffling without replacement did seem to converge slightly faster than normal sgd, which can be seen in the slightly steeper loss graphs. However, the effect was not very strong in this case. There was a dramatic difference in runtimes between batch sizes. For a batch size of 1, runtime over the datasets 24-27 seconds, while for a batch size of 60, that runtime is 9-10 seconds. This is almost a 3x speedup, considering the same number of samples were processed across all models. Further, compared to gradient descent, which took ~75s, this corresponds to a 3x and 10x speedup respectively.

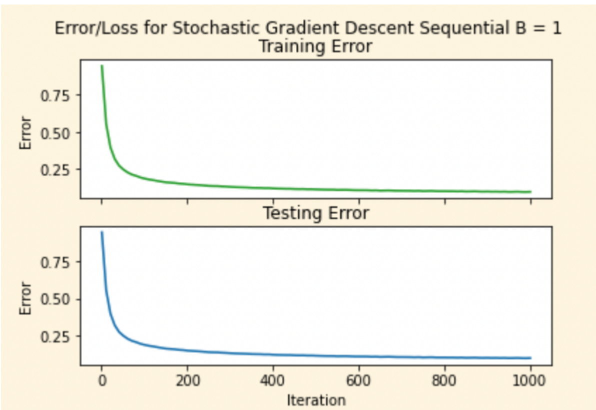
Note: *The graphs were plotted against iterations, which offers a finer picture than by epochs. If epochs are desired, you can equivalently consider the plot at every 100 iterations, since there were 10 epochs. Or I can regenerate the plots if requested. All future plots are plotted in epochs as desired.*

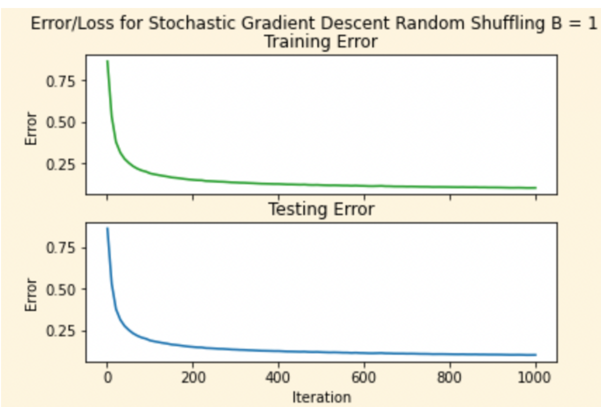










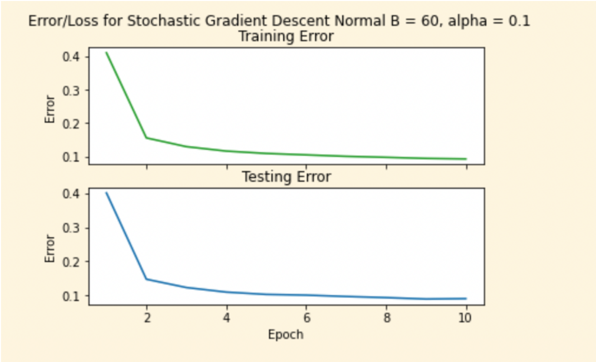


Part 6

1. Using step size other than alpha = 0.05 in Part 5

```
alpha = 0.1
batch = 60
gamma = 0.0001
```

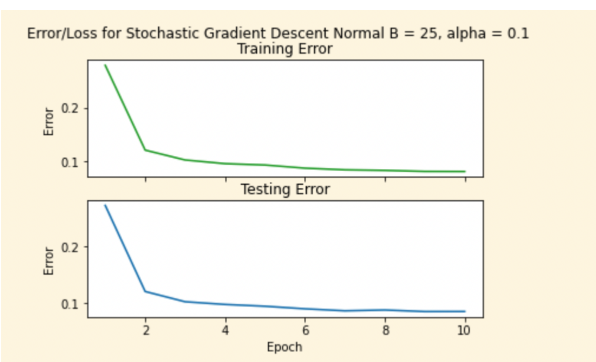
Changed alpha from 0.05 to 0.1. Final accuracy was very similar at ~0.91 in 9 seconds. Using 0.05, accuracy was close at around 0.895.



2. Improved Accuracy.

```
alpha = 0.1
batch = 25
gamma = 0.0001
```

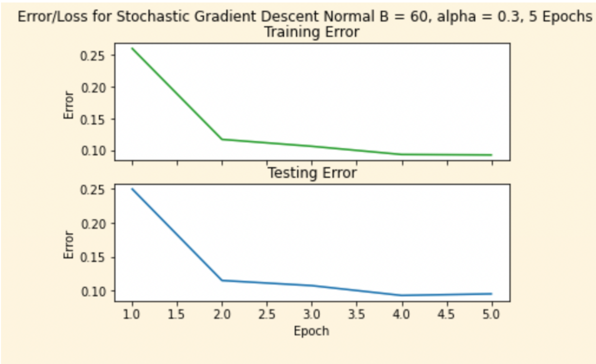
Of the hyperparameter sets I experimented with, this yielded the best test accuracy (0.92) given how fast it ran (~2-4 seconds). With respect to the training error (error is $1 - \text{accuracy}$), this meant it performed a raw 1-2% better than the given hyperparameters. On the test error, it performed similarly well, yielding an accuracy of around 0.92 and error of 0.08. However, from the graph, you can see some instability around the 8th epoch, when test error slightly jumps before recovering. Test error also decreased.



3. Only 5 epochs.

```
alpha = 0.3
batch = 60
gamma = 0.0001
```

We aim for around 89-91% accuracy using 5 epochs. Using the same parameters as before but over 5 epochs, we achieve 0.91465 test accuracy over 2 seconds. This is around the same performance as the best runs of SGD, so it is an improvement and it is faster. This corresponds to a test error of about 0.085.



Plot

The 3 required plots are given in the three parts above. Typically the test error follows the training error closely, which is a good sign. In general, I only retained the best hyperparameter sets, so all three plots perform either the same or better than the baseline in Part 5. Plots 2 and 3 have close to 0.92 accuracy, which is better than the baseline.

Part 7

Typical single results are shown below in the plots in the section below. Summarized and averaged over 5 runs, the runtimes are as follows:
Note that measurements were taken 10x an epoch, slowing overall runtime.

Average Over 5 Runs

- 5.2 - Batch Size = 1
  - SGD Normal: **27 seconds**
  - SGD Sequential: **24 seconds**
  - SGD Random Shuffling: **26 seconds**
- 5.3 - Batch Size = 60
  - SGD Normal: **11.5 seconds**
  - SGD Sequential: **9 seconds**
  - SGD Random Shuffling: **10 seconds**

Runtime Analysis

In general, batch size of 60 ran better than batch size of 1. This is because a batch size of 60 leverages numpy vectorization much better than processing each example individually. Processing multiple samples at once significantly speeds up the average processing time per sample. This leads to an 3x speedup.

Between the three types of models, we clearly observe that sequential beats random shuffling without replacement which beats the normal algorithm. This is because the sequential algorithm exploits memory locality to ensure that accesses to each sample are faster across batch iterations, making it the fastest of all. Random shuffling without replacement similarly exploits locality to speed up runtime, but shuffling incurs an overhead penalty that places it behind sequential. It also tends to converge very slightly faster than normal SGD. Finally, normal SGD takes the longest, since it does not use any of these speedup methods and repeatedly generating a new set of random indices at every batch is expensive.

Various Characteristic Accuracy/Runtime Results (Labeled)

```
Accuracy Measurement 90: 0.9012833333333333
Accuracy Measurement 91: 0.9012833333333333
Accuracy Measurement 92: 0.9012833333333333
Accuracy Measurement 93: 0.9012833333333333
Accuracy Measurement 94: 0.9012833333333333
Accuracy Measurement 95: 0.9012833333333333
Accuracy Measurement 96: 0.9012833333333333
Accuracy Measurement 97: 0.9012833333333333
Accuracy Measurement 98: 0.9012833333333333
Accuracy Measurement 99: 0.9012833333333333
Accuracy Measurement 100: 0.9012833333333333
Accuracy Measurement 101: 0.9012833333333333
Gradient Descent (1000 Iters) Took 76.30830788612366 Seconds
```

```
Accuracy Measurement 94: 0.9010333333333334
Accuracy Measurement 95: 0.9018666666666667
Accuracy Measurement 96: 0.90205
Accuracy Measurement 97: 0.9020166666666667
Accuracy Measurement 98: 0.9014833333333333
Accuracy Measurement 99: 0.9029833333333334
Accuracy Measurement 100: 0.9024333333333333
Accuracy Measurement 101: 0.9027166666666666
SGD Normal B=1, A=0.001, gamma=0.0001 Took 27.719687938690186 Seconds
```

```
Accuracy Measurement 94: 0.9017
Accuracy Measurement 95: 0.9010166666666667
Accuracy Measurement 96: 0.90245
Accuracy Measurement 97: 0.9020833333333333
Accuracy Measurement 98: 0.9011833333333333
Accuracy Measurement 99: 0.9025333333333333
Accuracy Measurement 100: 0.9029666666666667
Accuracy Measurement 101: 0.9019666666666667
SGD Sequential B=1, A=0.001, gamma=0.0001 Took 24.439671993255615 Seconds
```

```
Accuracy Measurement 94: 0.8995833333333333
Accuracy Measurement 95: 0.9001333333333333
Accuracy Measurement 96: 0.9001833333333333
Accuracy Measurement 97: 0.8998166666666667
Accuracy Measurement 98: 0.9003333333333333
Accuracy Measurement 99: 0.9012
Accuracy Measurement 100: 0.9010666666666667
Accuracy Measurement 101: 0.9008166666666667
SGD Random Shuffling B=1, A=0.001, gamma=0.0001 Took 25.456284284591675 Seconds
```

```
Accuracy Measurement 90: 0.9009166666666667
Accuracy Measurement 91: 0.9009333333333334
Accuracy Measurement 92: 0.9009333333333334
Accuracy Measurement 93: 0.90095
Accuracy Measurement 94: 0.9009666666666667
Accuracy Measurement 95: 0.901
Accuracy Measurement 96: 0.9010166666666667
Accuracy Measurement 97: 0.9010166666666667
Accuracy Measurement 98: 0.9010166666666667
Accuracy Measurement 99: 0.9010166666666667
Accuracy Measurement 100: 0.90105
Accuracy Measurement 101: 0.90105
SGD Normal B=60, A=0.05, gamma=0.0001 Took 10.556522846221924 Seconds
```

```
Accuracy Measurement 91: 0.9009166666666667
Accuracy Measurement 92: 0.9009333333333334
Accuracy Measurement 93: 0.9009333333333334
Accuracy Measurement 94: 0.9009666666666667
Accuracy Measurement 95: 0.901
Accuracy Measurement 96: 0.9010166666666667
Accuracy Measurement 97: 0.9010333333333334
Accuracy Measurement 98: 0.9010333333333334
Accuracy Measurement 99: 0.9010333333333334
Accuracy Measurement 100: 0.9010333333333334
Accuracy Measurement 101: 0.9010333333333334
SGD Sequential B=60, A=0.05, gamma=0.0001 Took 9.481458187103271 Seconds
```

```
Accuracy Measurement 90: 0.9009166666666667
Accuracy Measurement 91: 0.9009166666666667
Accuracy Measurement 92: 0.9009333333333334
Accuracy Measurement 93: 0.90095
Accuracy Measurement 94: 0.9010166666666667
Accuracy Measurement 95: 0.9010166666666667
Accuracy Measurement 96: 0.9010166666666667
Accuracy Measurement 97: 0.9010166666666667
Accuracy Measurement 98: 0.9010166666666667
Accuracy Measurement 99: 0.9010333333333334
Accuracy Measurement 100: 0.90105
Accuracy Measurement 101: 0.90105
SGD Random Shuffling B=60, A=0.05, gamma=0.0001 Took 10.594208002090454 Seconds
```