## Introduction

This project entailed implementing and evaluating stochastic gradient descent (SGD) on the MNIST dataset. Four different versions of SGD were implemented:

1. SGD
2. Sequential Sampling SGD
3. SGD with Minibatching
4. SGD with Minibatching and Sequential Sampling

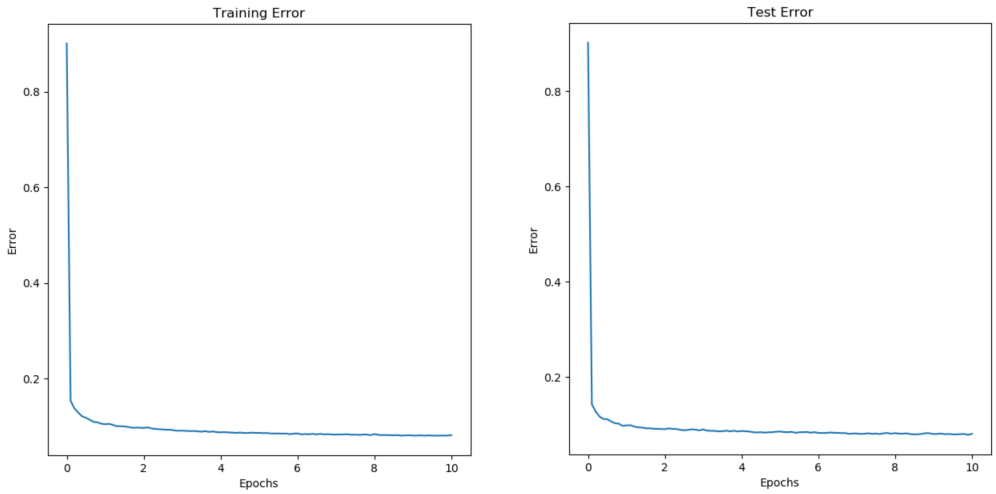
After implementation, the effects of learning rates and batch sizes were evaluated on the performance of our models. The run time of each algorithm was also evaluated along with the training and testing error.

## Part 1: Implementation

The implementation of the four variations of stochastic gradient descent can be observed in `main.py`. Listed below are plots of the training and testing error per epoch through the entire training/test set of each algorithm. The final error is listed and evaluated for each algorithm as well.

## Algorithm 1: SGD

In regular stochastic gradient descent, we saw a final training error of 0.081 and a testing error of 0.0797.

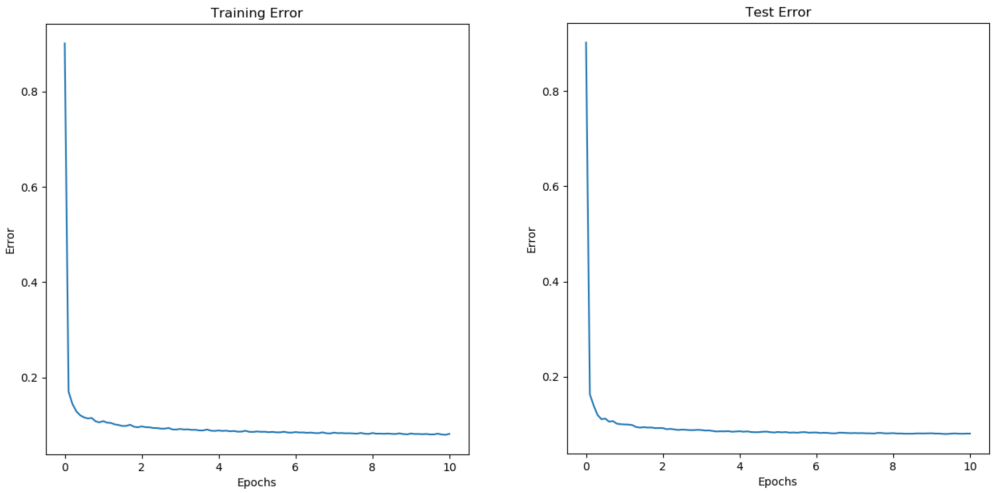



(a) Training Error (b) Testing Error

Figure 1: Algorithm 1 Errors

## Algorithm 2: Sequential SGD

In SGD with sequential sampling, we saw a final training error of 0.0816 and a testing error of 0.0804.

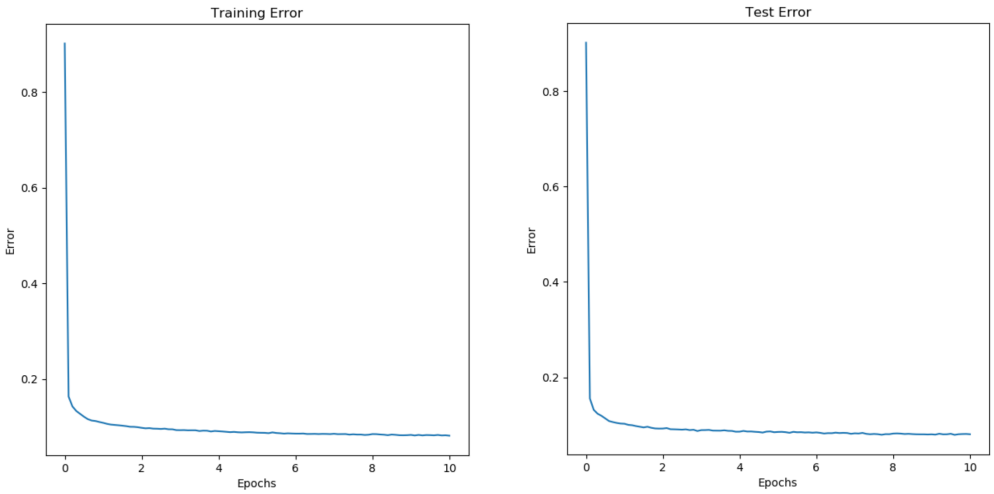



(a) Training Error (b) Testing Error

Figure 2: Algorithm 2 Errors

## Algorithm 3: SGD with Minibatching

In SGD with minibatching, we saw a final training error of 0.0815 and a testing error of 0.0807.

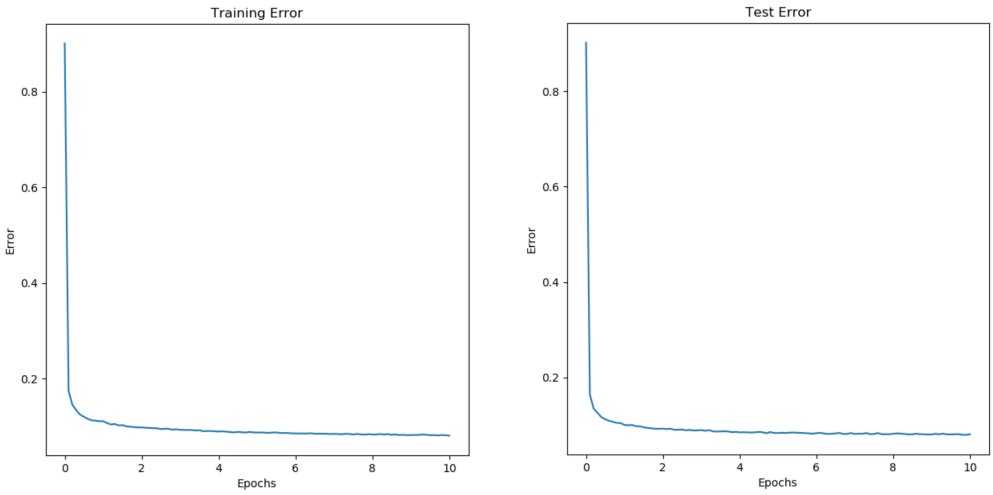



(a) Training Error (b) Testing Error

Figure 3: Algorithm 3 Errors

## Algorithm 4: SGD with Minibatching and Sequential Sampling

In SGD with minibatching and sequential sampling, we saw a final training error of 0.0813 and a testing error of 0.0797.




(a) Training Error (b) Testing Error

Figure 4: Algorithm 4 Errors

## Comparison of Error

As seen above, each algorithm converges in roughly the same time - with small change from epoch 3 onwards. The training and testing error of each algorithm at the end of run time is also roughly the same which shows that speeding up algorithms doesn't always sacrifice algorithm accuracy.

## Part 2: Exploration

With the implementation of the algorithms complete, we now turn to hyperparameter tuning to increase the performance of each algorithm.

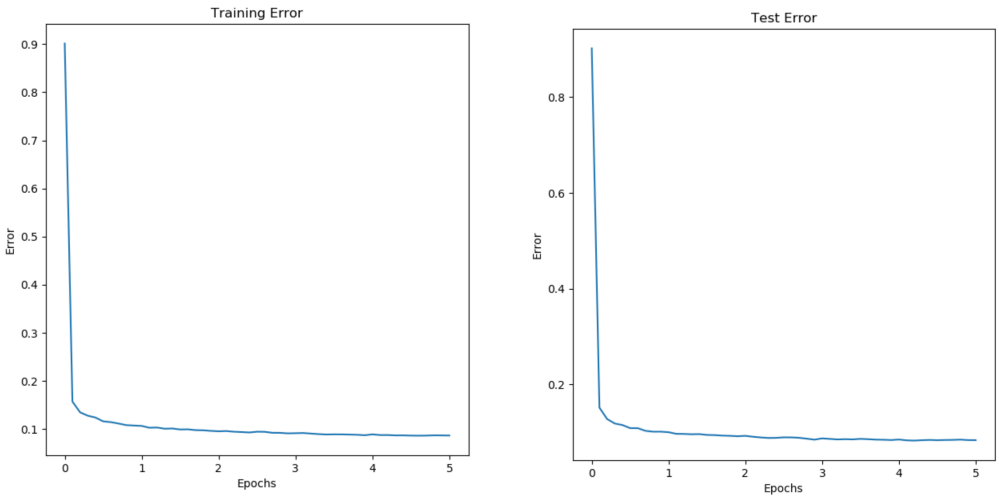
## Step Size Tuning - SGD

The step sizes that we evaluated in Part 2.1 for algorithm 1 were: 0.0005, 0.003, 0.006, 0.012, 0.05, 0.1 whose results can be seen the table below.

Table 1: Step Sizes for SGD

| $\alpha$ | Training Error |
|---|---|
| 0.0005 | 0.0859 |
| **0.001** | **0.0810** |
| 0.003 | 0.0777 |
| 0.006 | 0.0748 |
| 0.012 | 0.0845 |
| 0.05 | 0.0912 |

The step size with the lowest training error (0.006) also gave the lowest test error of 0.076. This is due to the model being able to converge towards its optima at a faster rate. We observe the training and testing error versus the number of epochs with this newfound best parameter in the figure below.




(a) Training Error (b) Testing Error

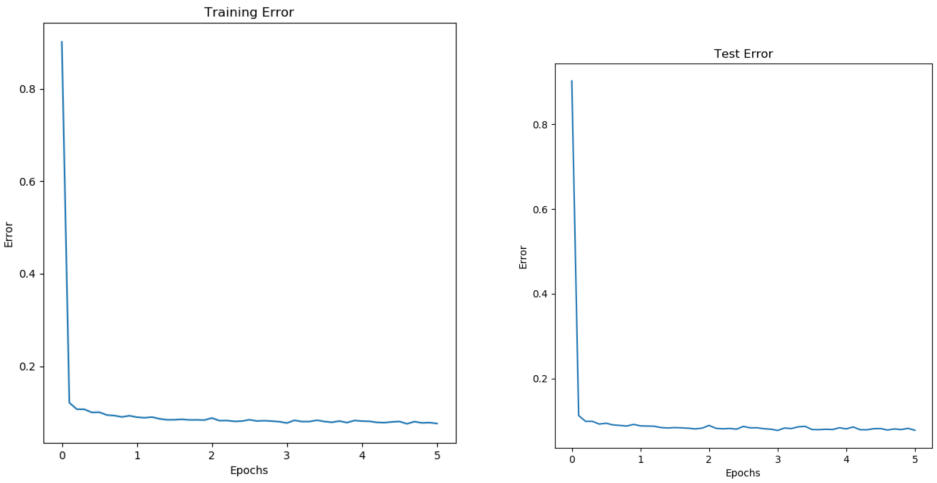
Figure 5: Error With $\alpha = 0.006$

### Reducing Epochs

Now the amount of epochs run was reduced from 10 to 5 and the step size was tuned to improve performance. The initial parameter (bolded) for 5 epochs is shown, as well as the error obtained with other step sizes. Note: the best performing value (italicized) is plotted.

Table 2: Step Sizes for SGD with 5 Epochs

| $\alpha$ | Training Error |
|---|---|
| **0.001** | **0.0810** |
| *0.006* | *0.0802* |
| 0.010 | 0.0867 |
| 0.025 | 0.0942 |
| 0.06 | 0.1262 |




(a) Training Error

(b) Testing Error

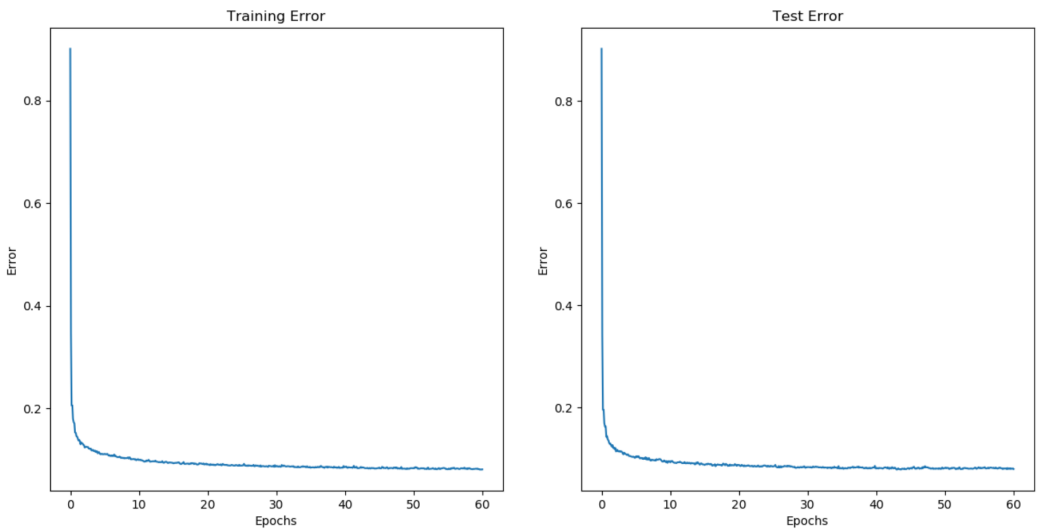
Figure 6: Error With $\alpha = 0.006$ and 5 Epochs

## Minibatched Sequential SGD Tuning

With the amount of epochs now reduced to 5, we explored different step size and batch size combinations. It was interesting to observe that even with only 5 epochs, we can tune our hyperparameters to get a training error of 0.0815, which is just barely larger than the best training error we got with running 10 epochs. The initial parameter (bolded) for 5 epochs is shown, as well as the error obtained with other step sizes. Note: the best performing value (italicized) is plotted.

Table 3: Tuning Algorithm 4

| $\alpha$ | B | Training Error |
|---|---|---|
| 0.001 | 5 | 0.1036 |
| 0.001 | 10 | 0.1168 |
| **0.001** | **60** | **0.1592** |
| 0.005 | 5 | 0.0860 |
| 0.005 | 10 | 0.0931 |
| 0.005 | 60 | 0.1199 |
| *0.01* | *5* | *0.0815* |
| 0.01 | 10 | 0.0862 |
| 0.01 | 60 | 0.1064 |




(a) Training Error (b) Testing Error

Figure 7: Error With $\alpha = 0.01$, $B = 5$ and 5 Epochs

## Part 3: Systems Evaluation

Turning away from hyper parameter tuning we now move towards systems evaluation. The average measured run-times for the 4 algorithms are respectively as follows: 76.441s, 52.442s, 5.618s, 5.843s. The specific run-times for each of the 5 iterations of each of the 4 algorithms can be seen as described in the table below:

Table 4: Completion Time of Algorithms

| Algorithm #1 | Algorithm #2 | Algorithm #3 | Algorithm #4 |
|---|---|---|---|
| 77.357s | 54.142s | 6.147s | 7.195s |
| 76.707s | 52.732s | 5.108s | 6.006s |
| 77.111s | 51.901s | 6.078s | 5.239s |
| 75.932s | 52.676s | 5.120s | 5.193s |
| 75.098s | 50.758s | 5.636s | 5.584s |

We see that algorithms 3 and 4, which utilize minibatching, are significantly faster than algorithms 1 and 2 which don't utilize minibatching. The huge time savings of batch gradient descent are present because instead of constantly updating the gradient over a single instance in the dataset, we are instead averaging the gradient over a subset $B$ of the entire dataset.

It is also evident that sequential sampling is faster than random sampling, which is especially seen in the big time difference between algorithm 1 and 2. The reason that random sampling is much is slower is because it doesn't take advantage of spatial locality. Sequential sampling, on the other hand, will be able to cache the training examples that are located next to each other and thus has much quicker lookup times.

## Conclusion

Over the course of this project, we learned the various ways that we can tweak the Stochastic Gradient Descent algorithm and the implications of both time and accuracy as a result. Algorithms 3 and 4, which including minibatching, had significantly faster runtimes than algorithms 1 and 2, which did not include minibatching, as seen in Systems Evaluation. It was especially interesting for us to see how minibatching significantly improved the runtime efficiency of Stochastic Gradient Descent without actually sacrificing nearly any accuracy with respect to both the training and the test error.