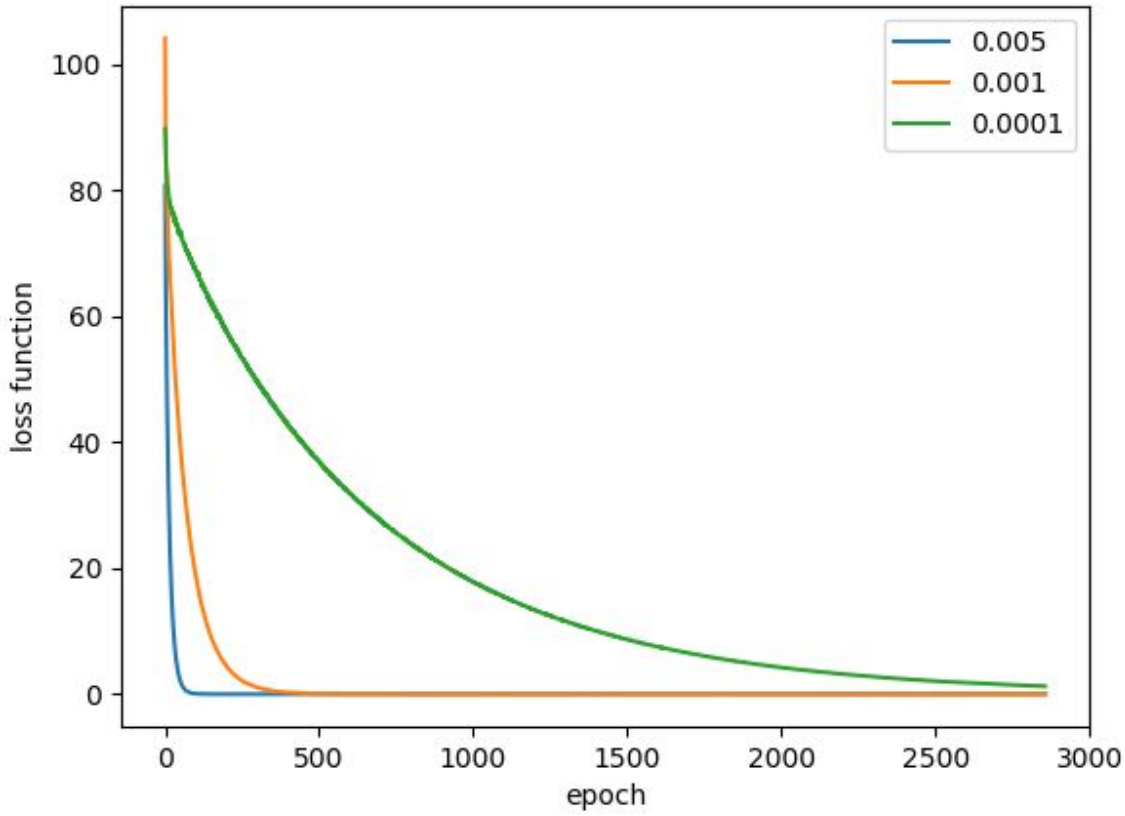
ECE521 Assignment 2

Contribution%	Name	Student Number
5	Ziwei Hu	1001197457
80	Tianqi Liu	1000456534
100	Elton Wong	1000625433

*Please note that the python code we submitted may not work with the tensorflow installed on UG machines.

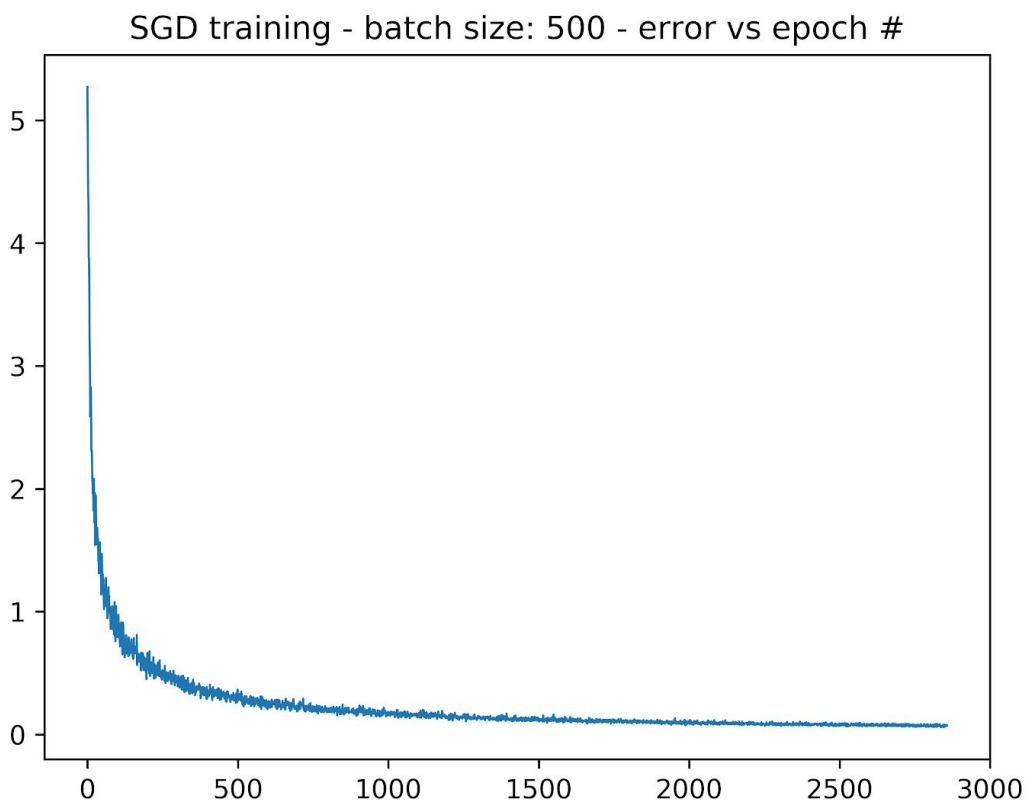
1.1)



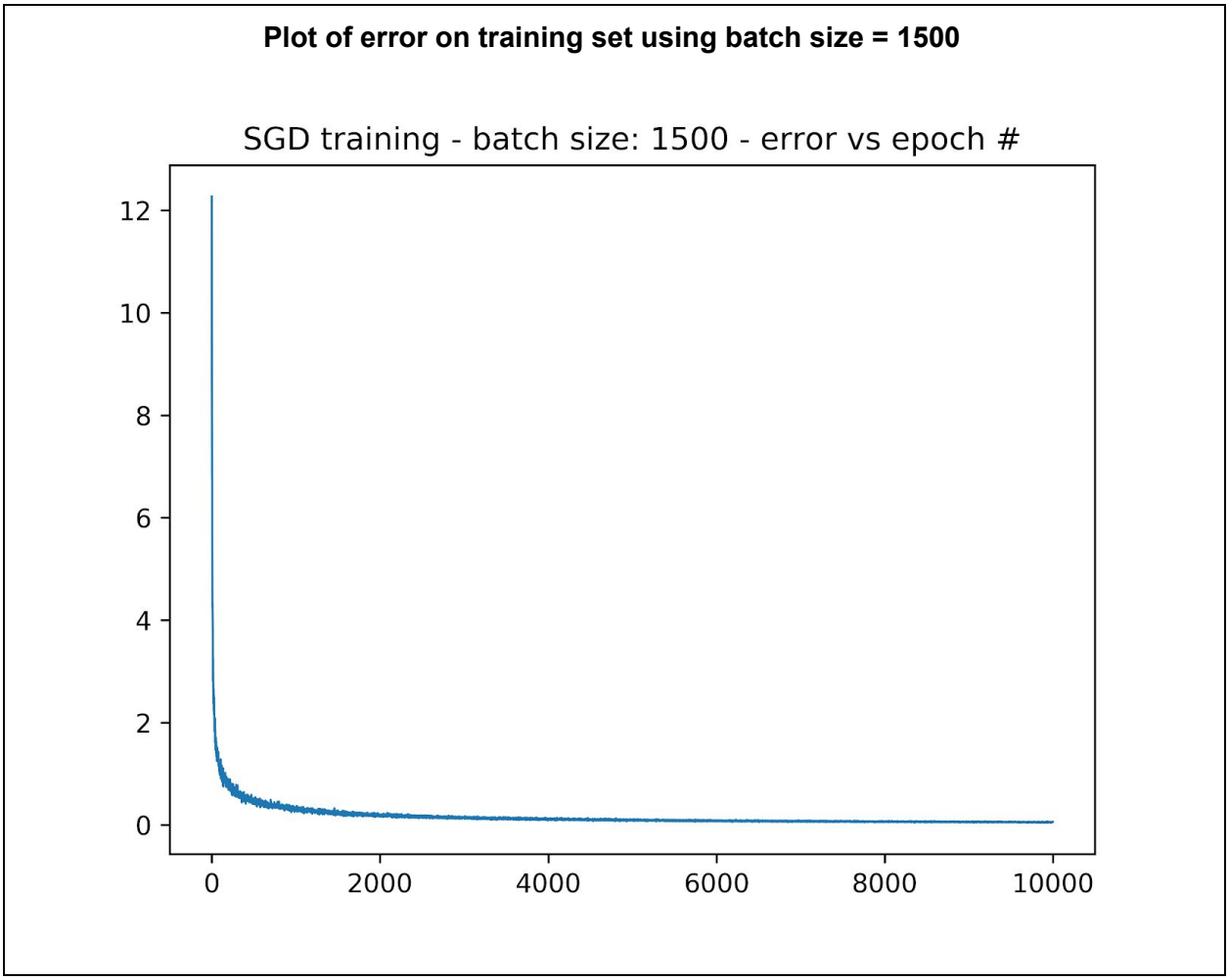
The best learning rate was 0.005, it had the fastest convergence because it takes the largest reasonable step size when updating the weights. As learning rate increases, the step size increases, which results in faster convergence. However, if the learning rate chosen is sufficiently large, the gradient of the next step may diverge. Conversely, if the learning rate is too small, convergence will be very slow.

1.2)

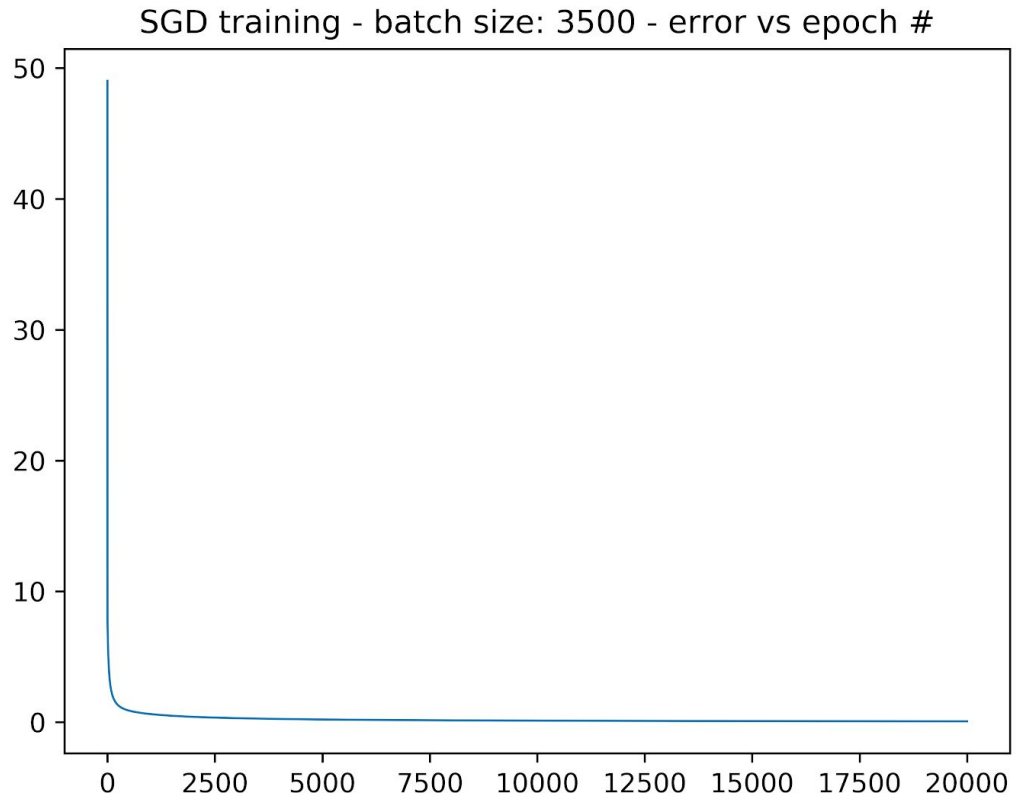
Plot of error on training set using batch size = 500



Plot of error on training set using batch size = 1500



Plot of error on training set using batch size = 3500



	Batch size = 500	Batch size = 1500	Batch size = 3500
Final training MSE	0.0719496	0.0626899	0.0683133
Time taken (CPU)	63.29763889312744 sec	205.0373787879943 8 sec	392.5694708824157 7 sec

The training time increases as the batch size increases. This is because increasing batch size will increase the number of rows of data matrix, thus it should naturally takes more time to compute the gradient of loss function - the computation complexity is O(N*d) where N is number of samples and d is length of data vector. Although CPU/GPU can do parallel computation, but they cannot handle it very well when the matrix is two large like 3500.
The performance of each batch size is similar because the expected value of error gradient are the same for different batch size. Their MSEs also take similar amount of iteration to converge - somewhere around 2000 iterations.

1.3)

	λ = 0	**λ** = 0.001	**λ** = 0.1	**λ** = 1
Final MSE	0.08611933	0.094435096	0.021802453	0.030090094
Validation Accuracy	92%	94%	93%	95%

Best **λ**: 1

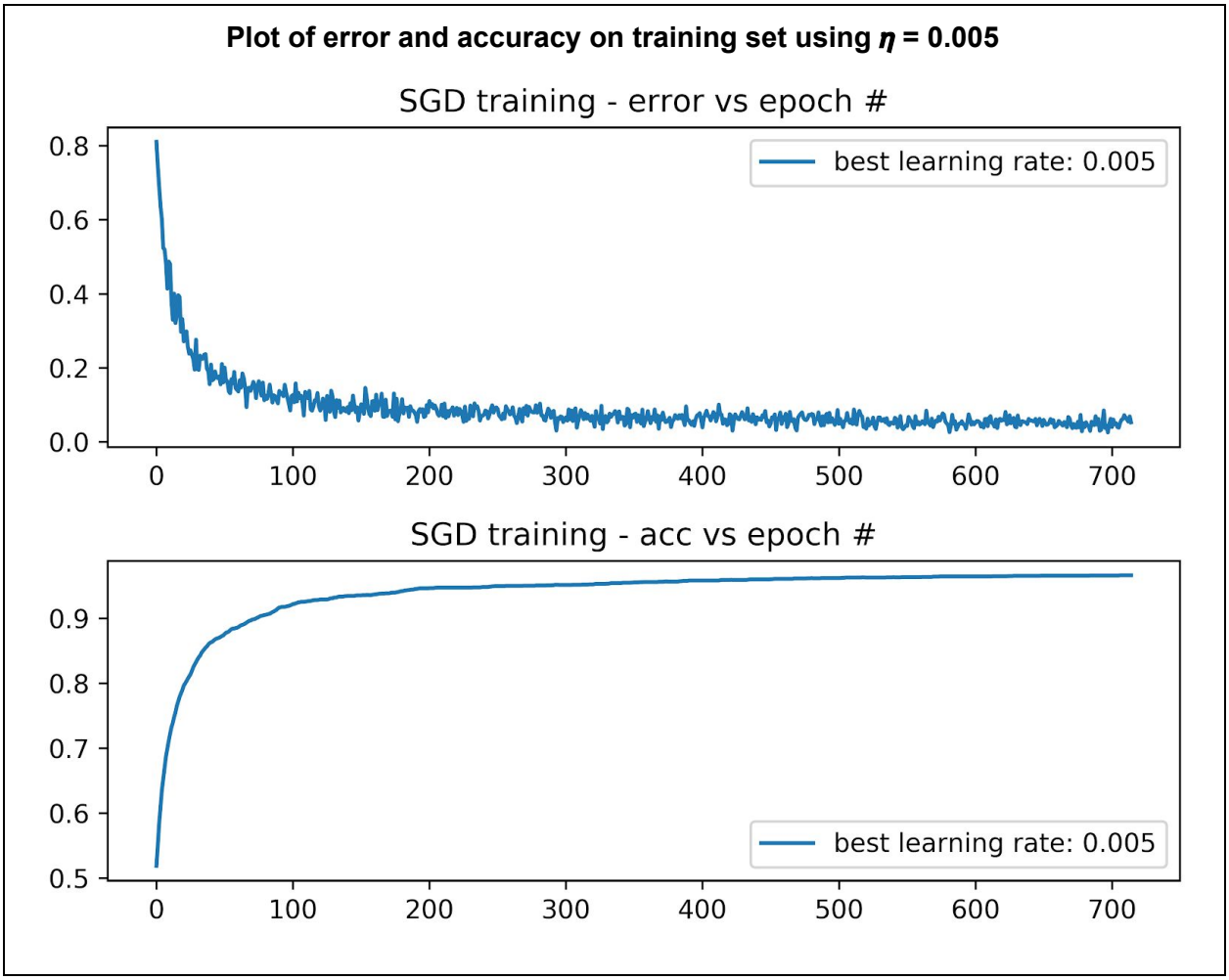
	λ = 1
Test Accuracy	0.9517241379310345 = 95.17241379310345%

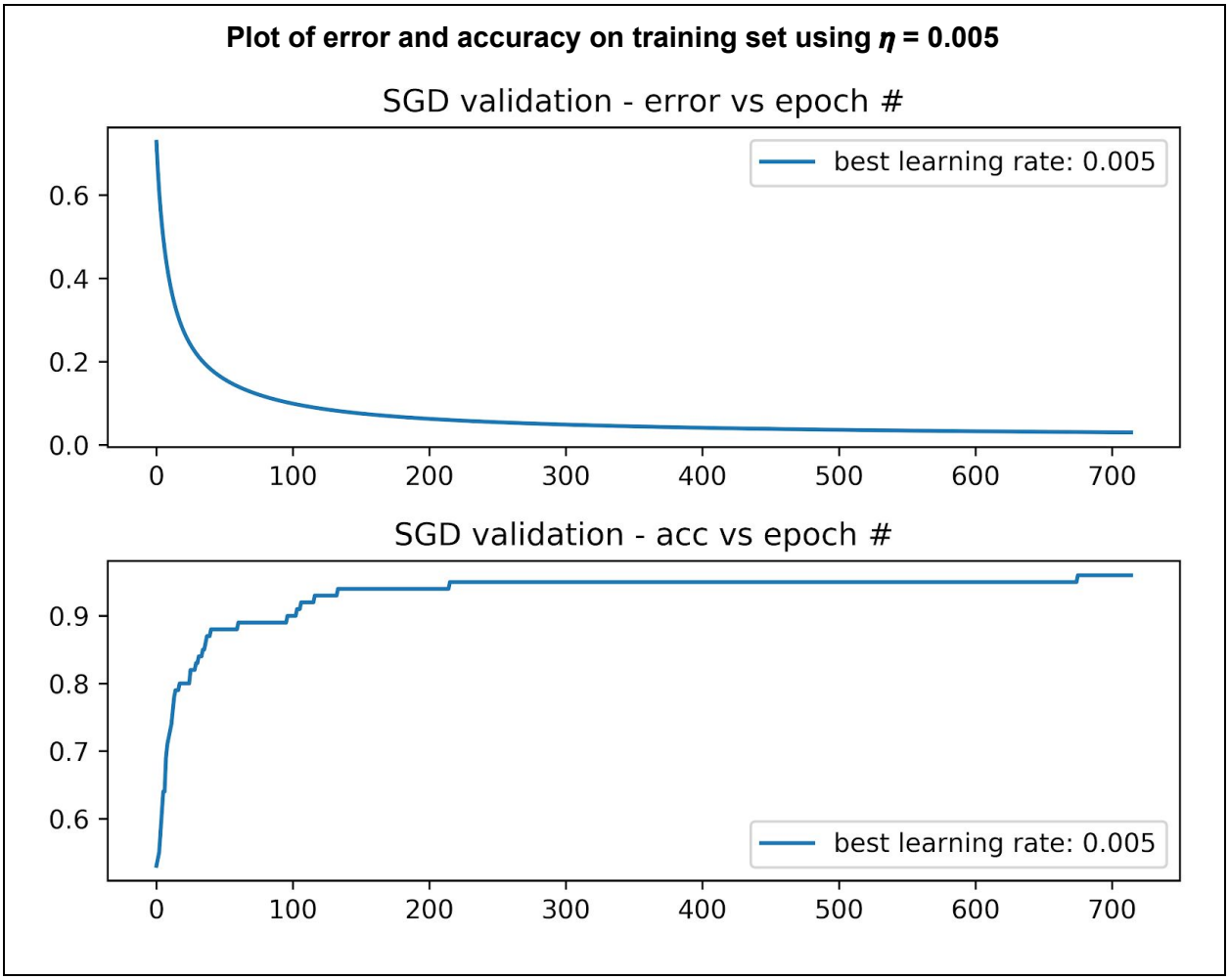
Comment on λ for performance:
As the **λ** becomes appropriately large, the performance of the model becomes better as we see from the validation accuracies for different **λ**.
When **λ** becomes large, the regularization term of the loss function becomes large, so the loss function become large. Since we are optimizing the loss, giving a rise in **λ** will penalize weight matrix which means the elements in our weight matrix will not grow arbitrarily and thus become smaller. This can be useful for reducing overfitting.

Comment on tuning hyper-parameter λ on validation set:
The reason we tune **λ** on validation set instead of training set is that tuning hyper-parameters like **λ** will cause a lot of overfitting on training set, which means our model will also learn a lot from the noise in training set, and will not generalize well. Since data in validation set are also random (ideally i.i.d), and are never seen by our model before, the validation set is therefore good for tuning hyper-parameters and evaluating the performance of our model.

1.4)
Normal equation took 5.142256259918213 seconds, with accuracy of 98.2% and final MSE of 0.02308755. SGD is more practical than using the normal equation when calculating multiplication and the inverses of Nxd matrices becomes too expensive. Calculating the multiplication and the inverse is on the order of O(N^3), which for large N becomes very expensive even for modern GPU.

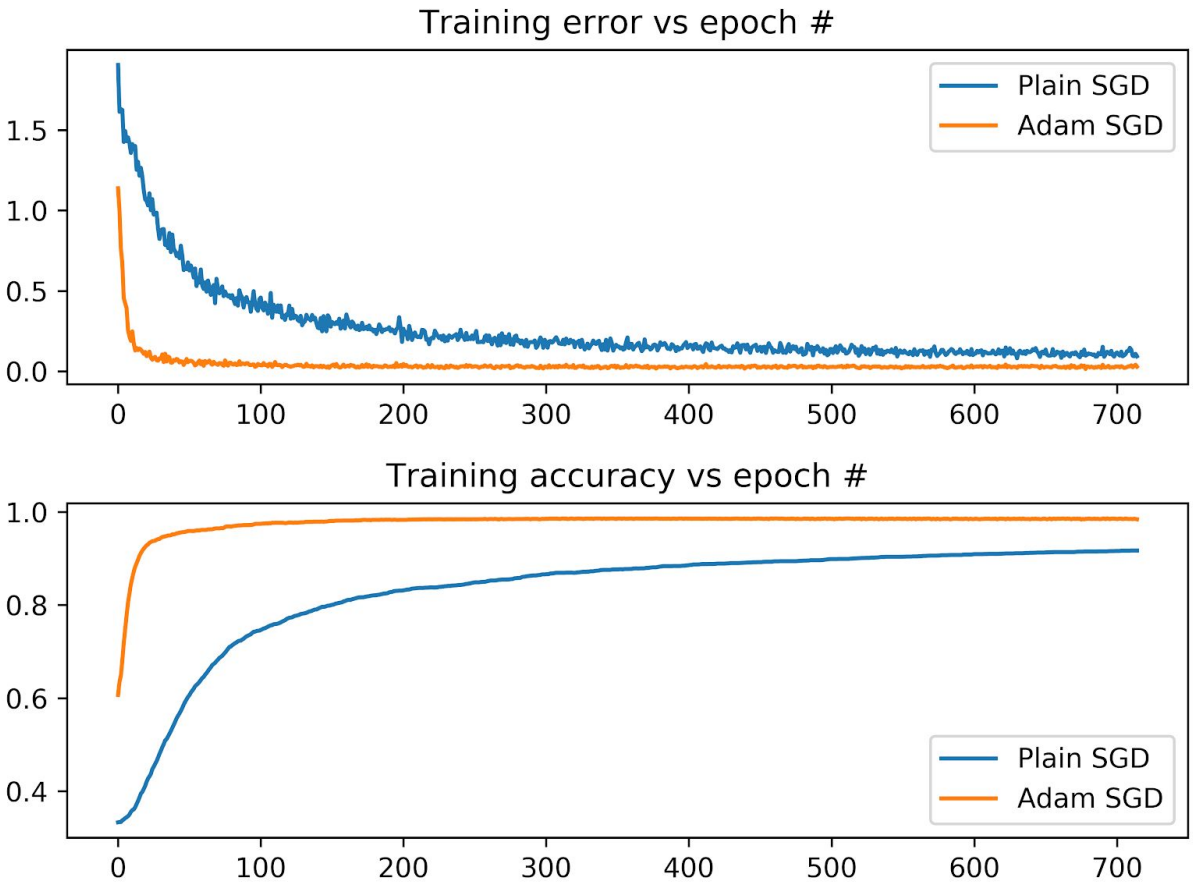
2.1.1) The **best learning rate η** for our model is **0.005**





Best test accuracy = 0.958620689655 = 95.8620689655%

2.1.2)

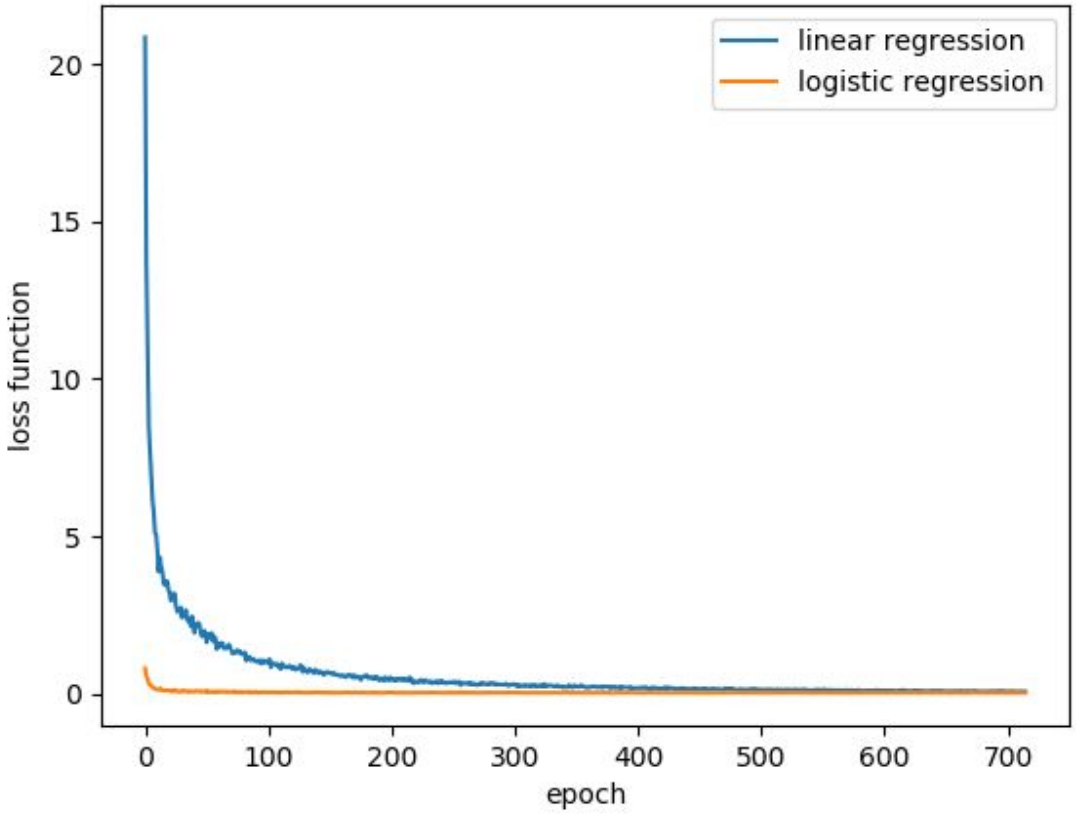


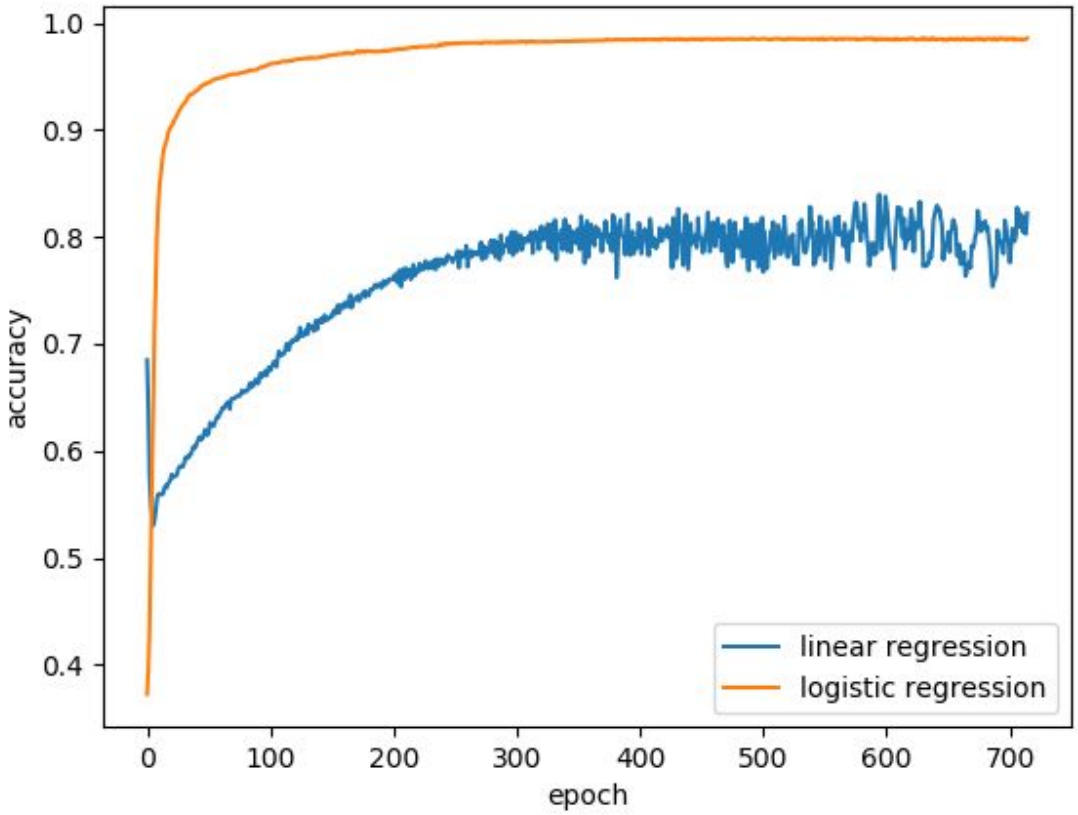
	Plain SGD	Adam
Final Training MSE	0.0932426	0.0286024
Final Training Accuracy	0.916857142857 = 91.6857142857%	0.984 = 98.4%

As shown above, Adam optimizer converges faster than plain gradient descent, and achieves lower training loss and accuracy. Adam may help use learning because it is more effective and stable when updating the weight matrix. For example, it uses a more smooth version of gradient instead of using just the raw gradient which may cause oscillation. With Adam, we can achieve better accuracy and lower training loss with the same number of iterations used before.

2.1.3)

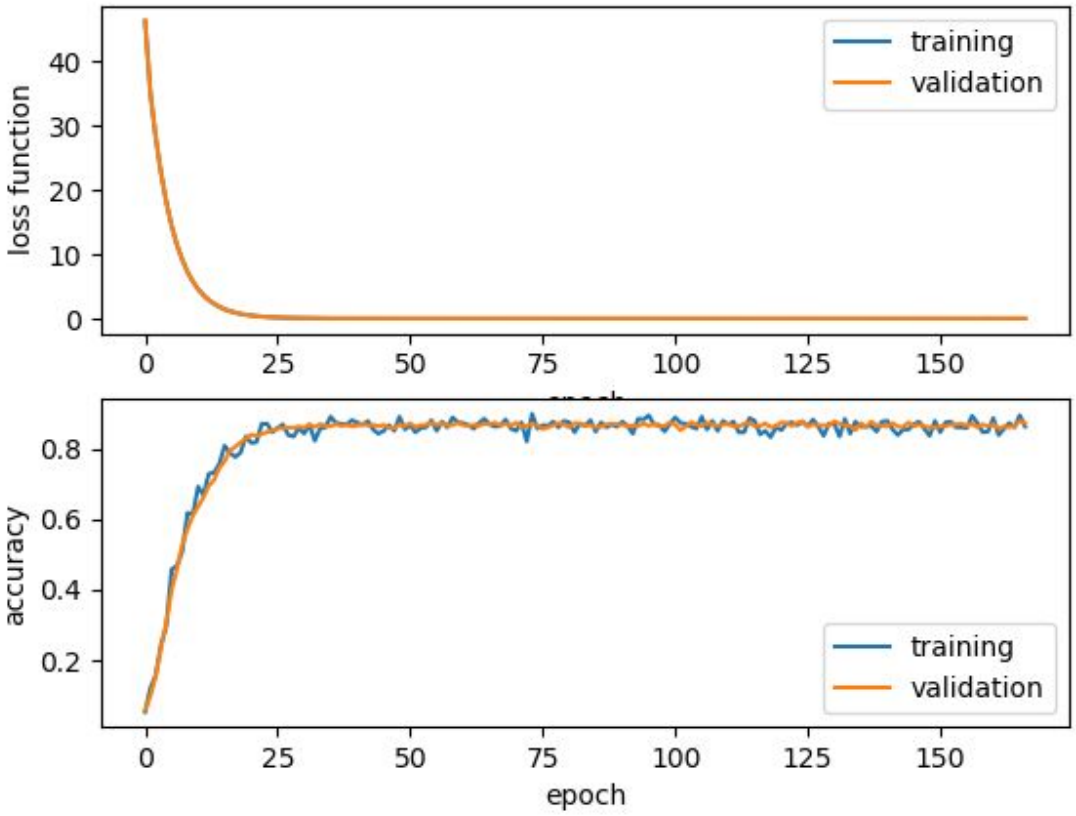
	Training accuracy	Validation accuracy	Test accuracy
Normal Equation	98%	96%	93.1%
Logistic regression	96.6%	96%	95.8%





Cross entropy loss converges faster than MSE. When the predicted value and true value have a large difference, the convergence speed is fast; otherwise, it is small, and this is our expectation.

2.2.1)



The test classification accuracy is 85.2%

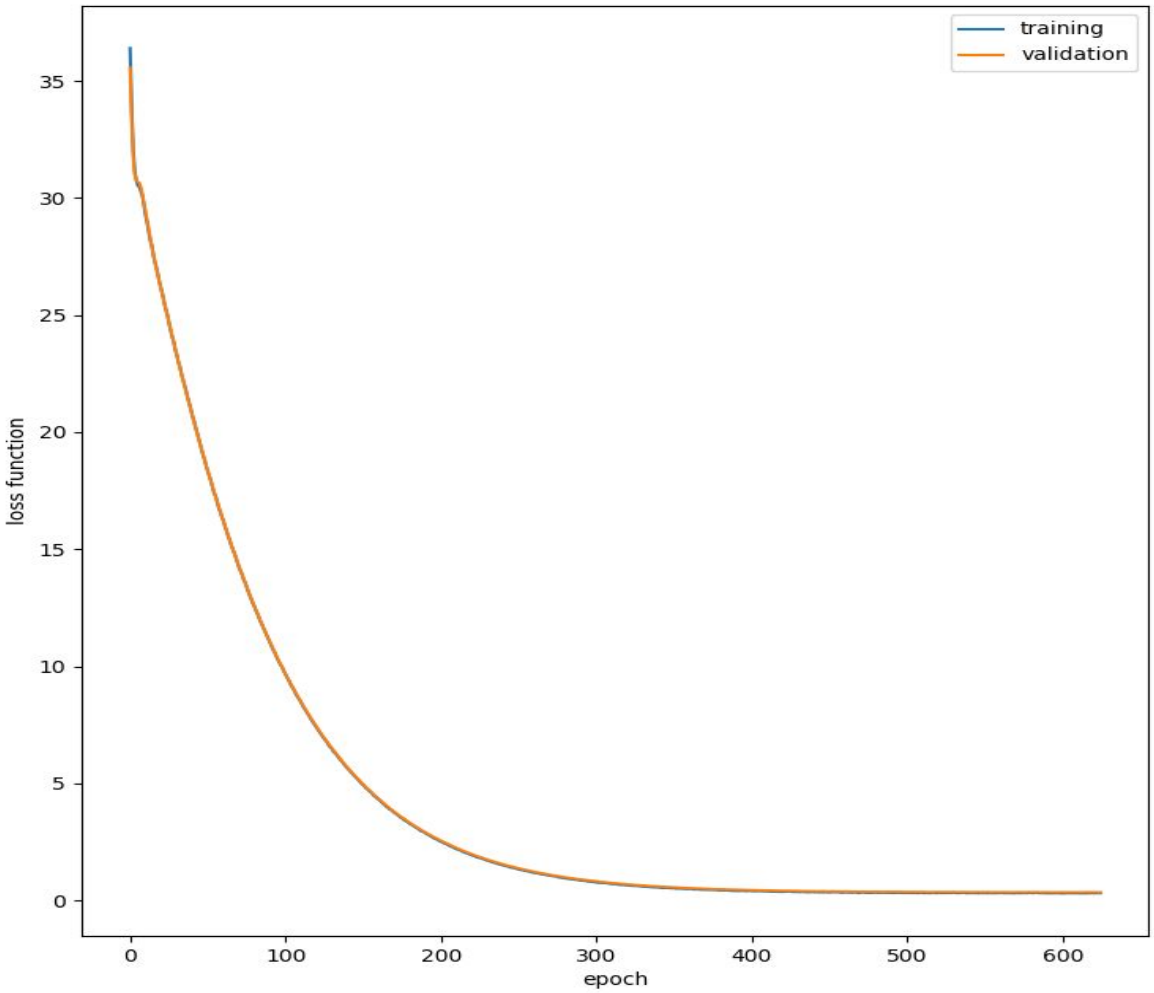
The multiclass classification performance is worse than the binary classification. The reason is that the the degree of complexity of the problem increases - i.e. from distinguishing 2 letters to distinguishing 10 letters - while the capacity of the model remains the same - i.e. still the same model.

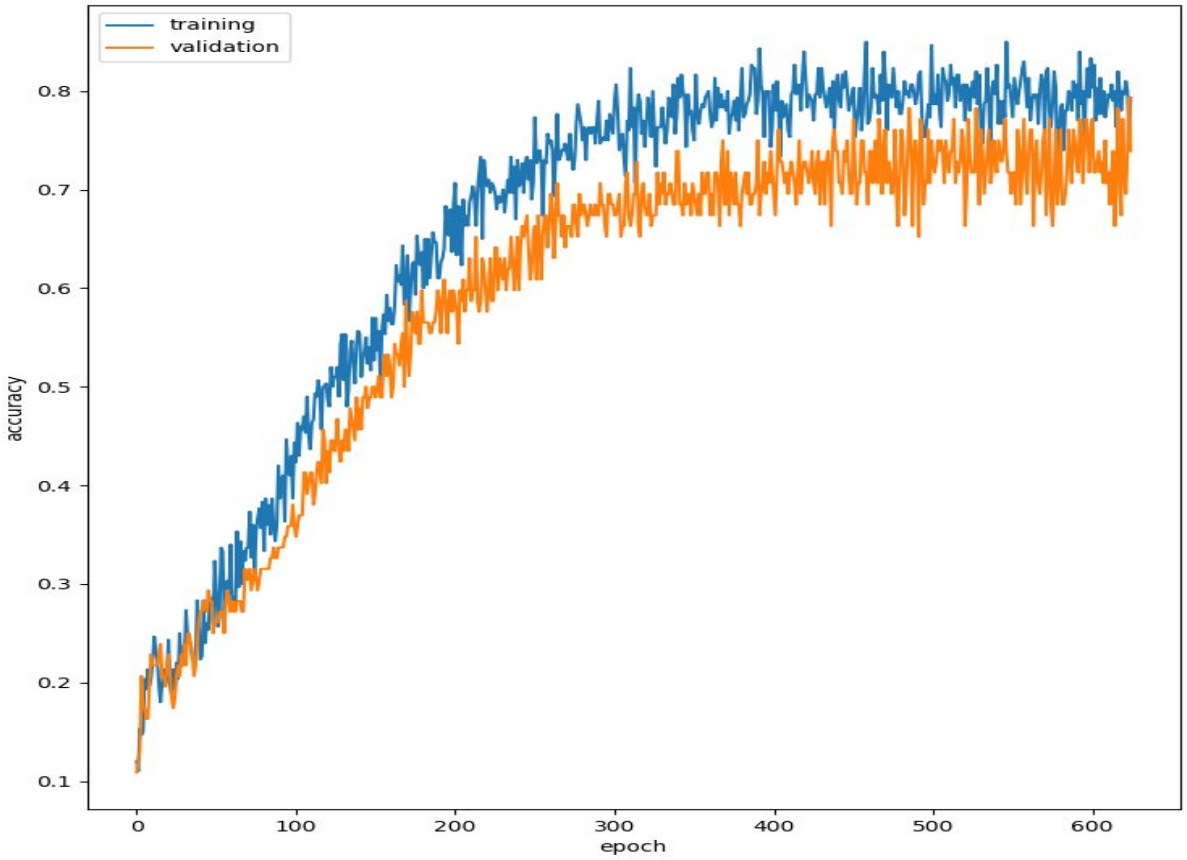
For example, in the binary problem, we classify between letter **"C"** and **"J"**. These two letters are visually very different from each other in most cases of handwriting, so it is reasonable we have high performance on the binary classification.

However, for the multi-classification problem, we are given letter **"A"** to **"J"**, and notice that letters **"E"** and **"F"** are similar and can be hard to distinguish depends on the actual handwriting. The same is true for letters **"I"** and **"J"**.

Therefore, it is reasonable that multi-classification performs worse than the binary classification.

2.2.2)


training
validation
loss function
epoch
0
5
10
15
20
25
30
35
0
100
200
300
400
500
600



The test classification accuracy is 79.26% for 1250 iterations, and does not really improve when we tried 6000 iterations and 10000 iterations.

The accuracy of logistic regression is better than what we had for k-NN which was 70.9%. One of the reasons why logistic regression performed somewhat better is that k-NN is vulnerable to overfitting on small data sets and it does not learn from the data sets it has seen, whereas logistic regression learns from input data sets and adjust the weight matrix accordingly, so it generalizes well when the testing data is not far off from what it has learnt.