ECE521 Assignment 2 Report

Luyuan Chen 999516082
Balu Viswanathan 1001328850
Yu Luo 998121651

Mar 2018

Work breakdown

Everyone implemented a version of the code. Balu completed the report for linear regression while Luyuan did the part of logistic regression.

The description of source code is in the Appendix

1 Linear Regression

1.1 Tuning Learning Rate:

The training run in this section consisted of 20000 iterations over the two-class notMNIST data set, with a batch-size of 500, as specified in the instructions. Our script simply ran this training loop three times for each learning rate $\eta = \{0.005, 0.001, 0.0001\}$, while using tf.train.GradientDescentOptimizer to minimize the MSE loss function. If training loss (MSE) is used as the metric, our findings were $\eta = 0.005$ is the best learning rate.

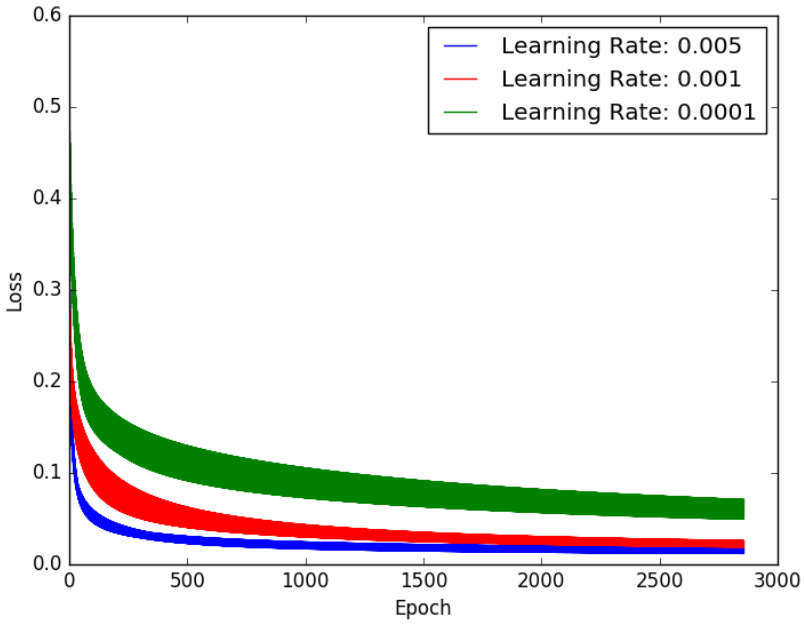

Figure 1: Training Loss at Different Learning Rates vs. Number of Epochs

From Figure 1, based on this data set and the learning parameters, the smaller the learning rate, the longer it takes for the training to converge.

1.2 Effect of Mini-Batch Size:

In order to check the effect of the mini-batch size, we used the same script as the previous section, but iterated three times for B= $\{0.005, 0.001, 0.0001\}$ at the tuned learning rate $\eta = 0.005$ If training loss is used as the metric, our findings were B= 1500 is the best batch size.

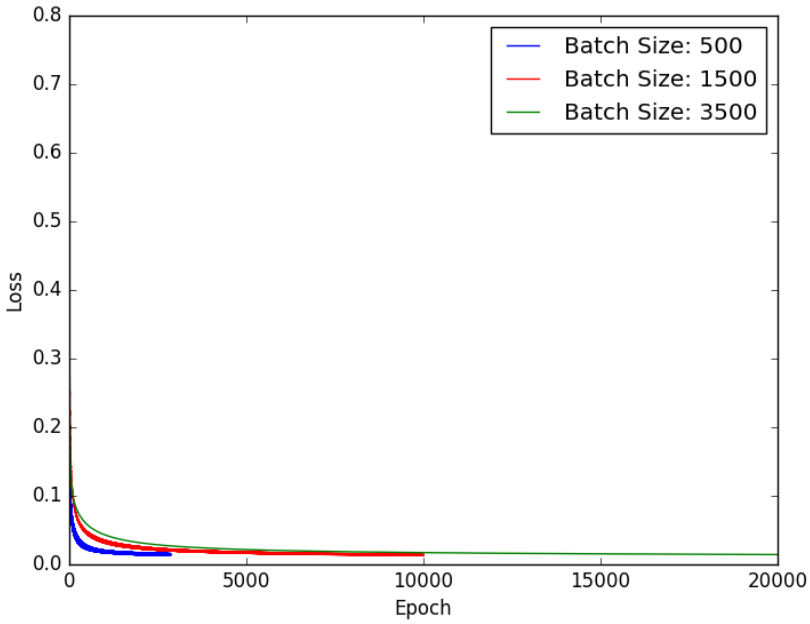

Figure 2: Training Loss at Different Batch Sizes vs. Number of Epochs

From Figure 2, the varying batch sizes do not introduce that much variation in training convergence, but batch size of 1500 has marginally smaller final training loss, as you can see in the table 1 below.

Batch size	Training MSE	Training time(s)
500	0.017	30.744
1500	0.012	78.974
3500	0.015	192.787

Table 1: Training MSE and time for different batch size

However, it should be noted here that while $B = 1500$ performed the best in terms of final MSE on several training runs, the margin of difference in final MSE is so small between the different batch sizes that it cannot be concluded that any batch size is superior for minimizing training loss. As such, computation time should be the only criteria used for

picking batch size, and clearly we must choose $B = 500$, which has the shortest training time at approximately 30.744 s. It appears that smaller mini-batch sizes are faster for SGD optimization. In general though, mini-batches should not be too small, otherwise the optimization will not converge to an accurate solution that is representative of the entire training data.

1.3 Generalization - Tuning Weight Decay Coefficient:

Again, a script that is highly similar to the previous sections was used to tune the weight decay coefficient λ. We ran our training loop four times for $\lambda = \{0., 0.001, 0.1, 1\}$. The difference in the script for this section was the calculation for accuracy, which was done for both the validation and test data. From the script provided in the lab handout, the positive class 'C' was denoted with $Target = 1$, and the negative class 'J' was denoted with $Target = 0$. A classification threshold of 0.5 was chosen; i.e. if a value from the linear regression prediction is greater than 0.5, the class prediction is 'C', the positive class. The class predictions are then compared to the true target labels, and fraction of accurately classified input data is calculated as accuracy. Fig 3 shows the validation accuracy results below.

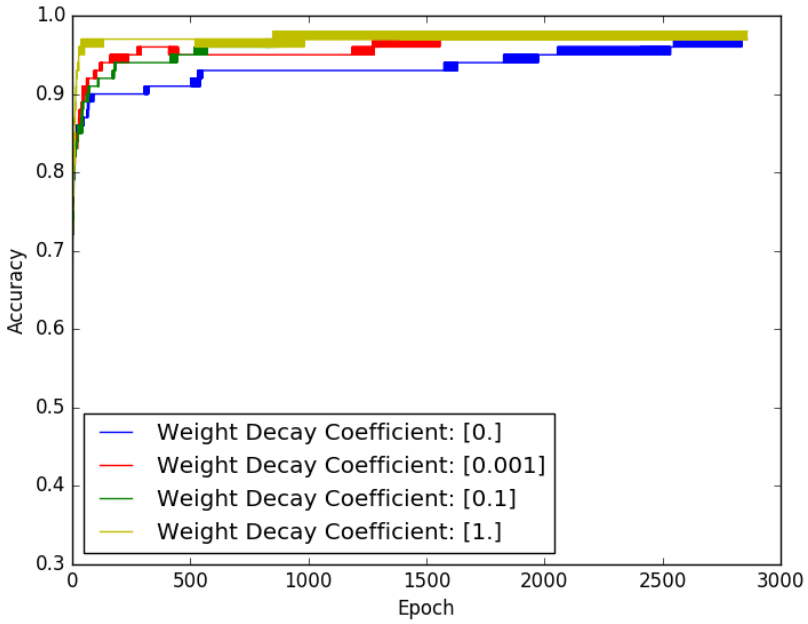

Figure 3: Validation Accuracy at Different Weight Decay Coefficients vs. Number of Epochs

The graph shows $\lambda = 0.001$ giving the best validation accuracy. The table 2 below summarizes the final results for validation and test accuracy for the different values of λ.

λ	Validation accuracy	Test accuracy
0	0.97	0.965
0.001	0.99	0.959
0.1	0.98	0.972
1	0.98	0.966

Table 2: Accuracy of validation and test set

For $\lambda = 0.001$, which has the best final validation accuracy, the final test accuracy is 0.959. It is obvious that the value of the weight decay has to be neither too big or too small. Regularization is used to keep the weights small so that the model trained does not "remember" the training set. A big λ will suppress the weights (forcing the second momentum of weights to be small) too much. On one hand it did prevent overfitting, on the other it limited the model's ability to fit the training set, in other words, underfit. Same with the small λ value, it's too small to be effective, leading to overfitting.

In practice, we usually look at accuracy of the validation set. In fact, the model's ability to adapt to the training set is not as important as it's capability to perform well on new data. This is why base evaluation on "unseen" data.

1.4 Comparing SGD with Normal Equation:

Our script ran an SGD training run with 20000 iterations, $B = 500$, $\lambda = 0$, and $\eta = 0.005$, and compared the final training MSE, accuracy, and computation with the same metrics for a closed-form normal equation. The formula presented in lecture (least-squares) was used to calculate the optimal W. The results are presented in the table 3 below.

Method	Final MSE	Training accuracy	Training time(s)
SGD	0.016	0.982	42.664
Normal Equation	0.010	0.992	0.594

Table 3: MSE and accuracy with normal equation and SGD

From final results, the normal equation is better in every respect. It has slightly lower MSE, slightly better accuracy and an order of magnitude better computation time. This makes sense for our specific data set because it is relatively small and the closed form normal equation is guaranteed to find the optimal W. However, in real-world application SGD may be more optimal as the data sets are much larger. Huge data sets will make calculating the matrix inverse very computationally intensive, to the point where SGD will have much better training time.

2 Logistic Regression

2.1 Learning

With the weight decay coefficient $\lambda = 0.01$ and 5000 iterations, the SGD performed by the gradient descent optimizer achieved validation accuracy of 98%. The learning rate used is 0.1. The loss values are plotted in figure 4 and the accuracy values are plotted in figure 5

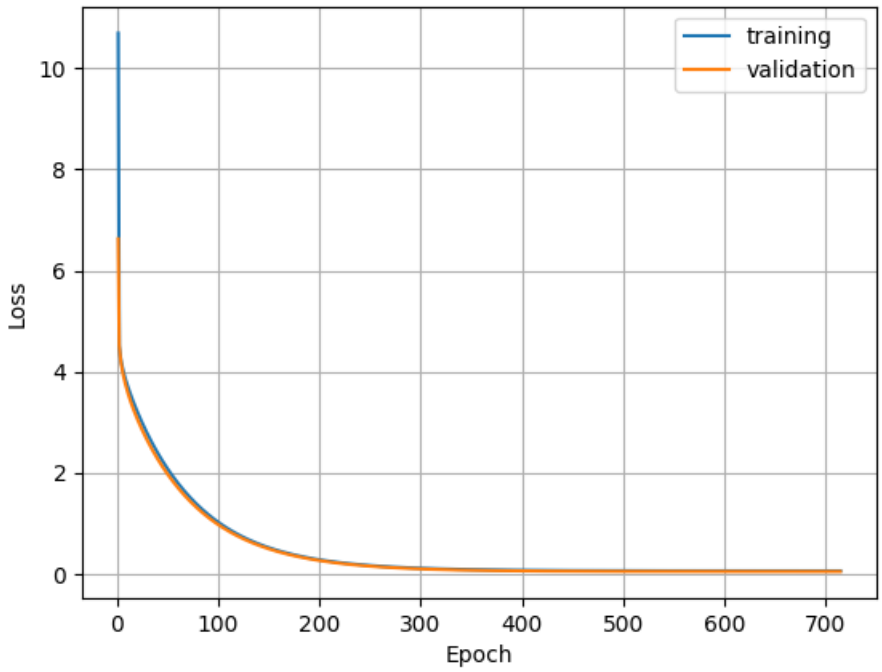

Figure 4: Loss, training vs. validation

2.2 Beyond SGD

With $\lambda = 0.01$ and $\eta = 0.001$ and the Adam optimizer performed a much better job than the regular SGD in that the rate of descent is much faster. The experiment result is shown in the figure 6 below.

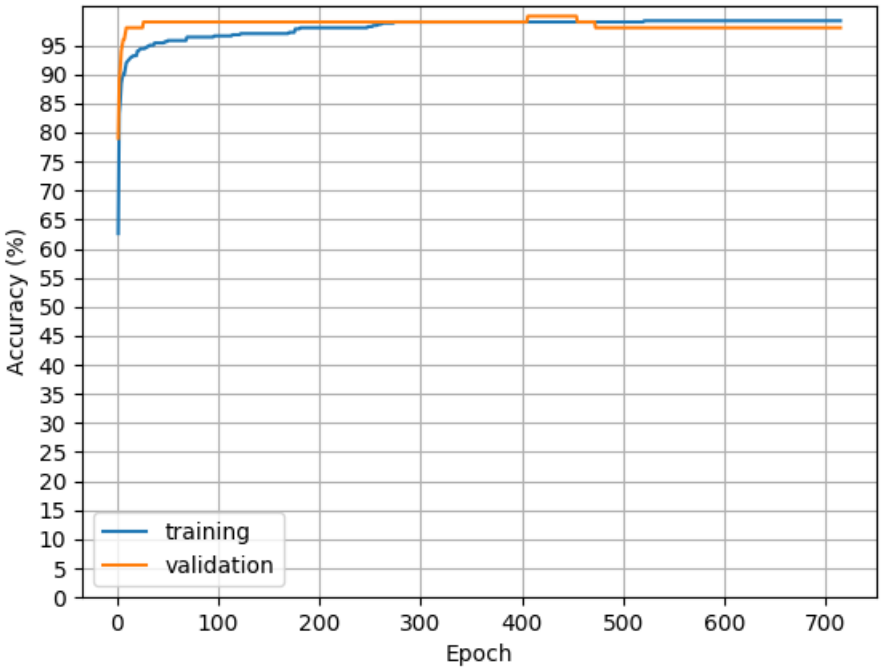

Figure 5: Accuracy, training vs. validation

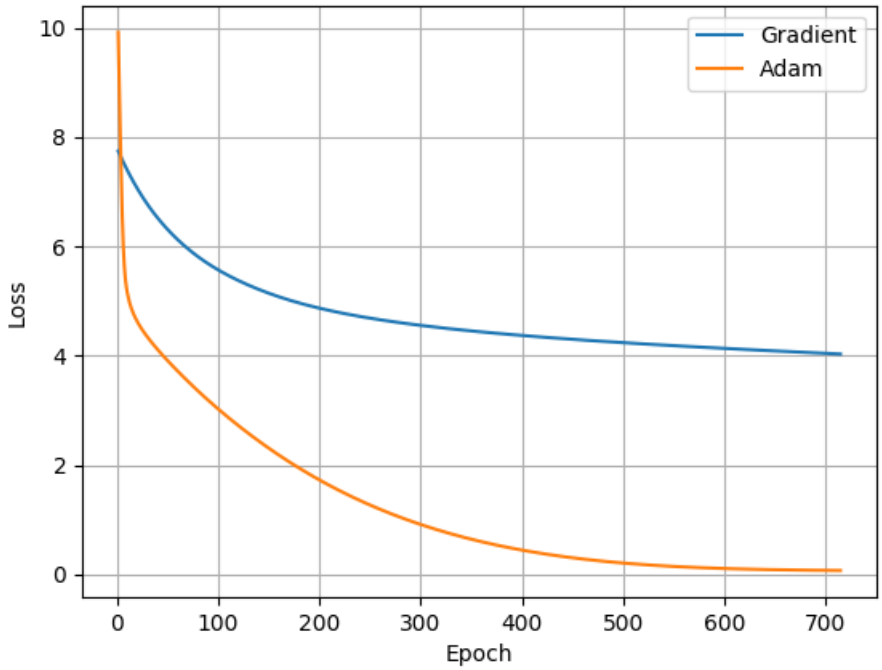

Figure 6: Adam vs Gradient descent optimizer

After some research, it is found that the Adam optimizer uses the estimated first and second moments of the gradient to tune the hyper-parameters. It also include a momentum in every iterations of the gradient descent. This is to say, a part of the "momentum" in the previous iteration will be carried over to the current one. This way, the loss will drop much faster than ordinary gradient descent.

2.3 Comparison with linear regression

The results of the classification accuracy is summarized in table 4. Logistic regression performed slightly better on validation and testing accuracy.

The logistic regression model is trained with $\lambda = 0$ and $\eta = 0.1$ (learning rate re-tuned as instructed). The loss vs epoch plot is shown in figure 7. Although it's not very clear in the plot, the cross-entropy error value dropped very fast. As for accuracy, linear regression took about 15000 iterations to reach over 90% while logistic regression's accuracy climbed much steeper, as shown in figure 8.

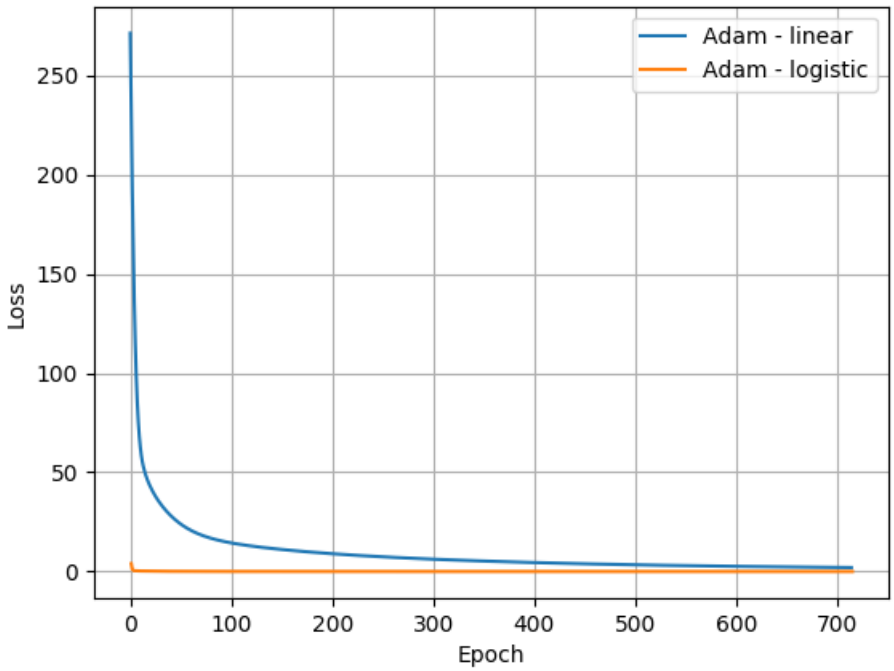

Figure 7: Square and cross entropy loss with Adam optimizer

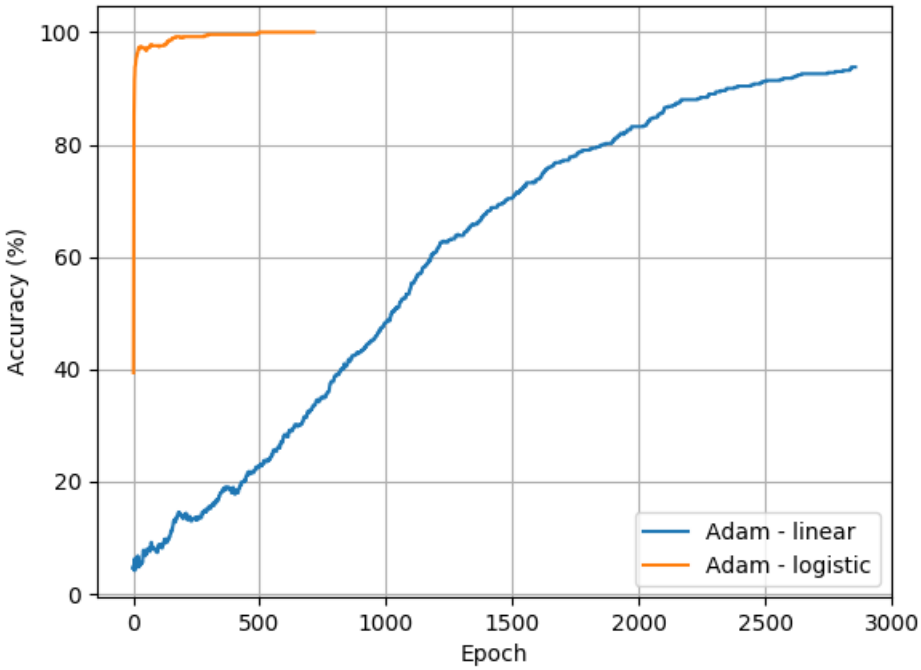

Figure 8: Accuracy of linear and logistic regression with Adam optimizer

	Training	Validation	Testing
Logistic regression	98.2	99.0	97.2
Normal equation	98.4	96.0	91.7

Table 4: Accuracy with normal equation and logistic regression

Explanation To understand this behaviour, let's take a look at the error (loss) function, assuming that $\hat{y} \in [0, 1]$ and $y = 0$. The error functions are:

$$e(y, \hat{y}) = \begin{cases} \hat{y}^2, & \text{square error} \\ -\log(1 - \hat{y}), & \text{cross entropy error} \end{cases}$$

One can see that, as $\lim_{x \to 0} \log(x) = -\infty$

$$\lim_{\hat{y} \to 1} e(y, \hat{y}) = \begin{cases} 1, & \text{square error} \\ \infty, & \text{cross entropy error} \end{cases}$$

This gives the cross entropy error a huge advantage when the classification is incorrect. Furthermore, if we take the difference of the two error functions:

$$
\begin{aligned}
f(y) &= y^2 + \log(1-y) \\
f'(y) &= 2y + \frac{1}{y-1} < 0, \forall y \in \mathbb{R} \text{ and } f(0) = 0 \\
\implies f(y) &< 0, \forall y \in [0, 1)
\end{aligned}
$$

This is to say, the error output of cross entropy error will always be bigger at the same condition.

2.4 Multiclass logistic regression

MNIST dataset Training rates of $0.1, 0.01, 0.001, 0.0005$ are tested. Figure 9 shows the loss from a set of learning rates that helped us to located the optimal $\eta = 0.01$. Note in the figure α means the learning rate.

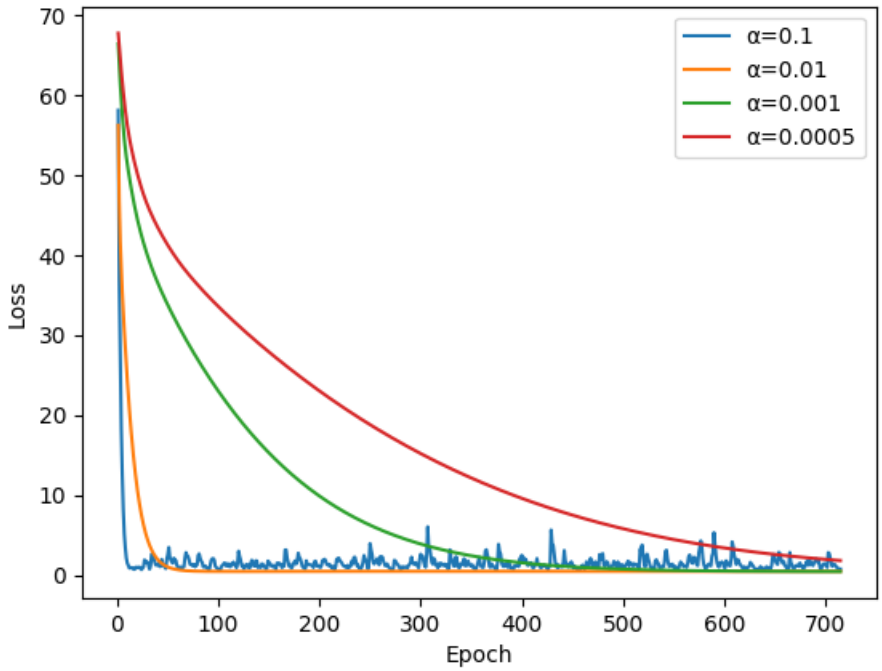

Figure 9: Loss over multiple training rates

Under the setting of $\lambda = 0.01$ and $\eta = 0.01$, figures 10 and 11(page 11) plots accuracy and loss respectively, against the number of epoch.

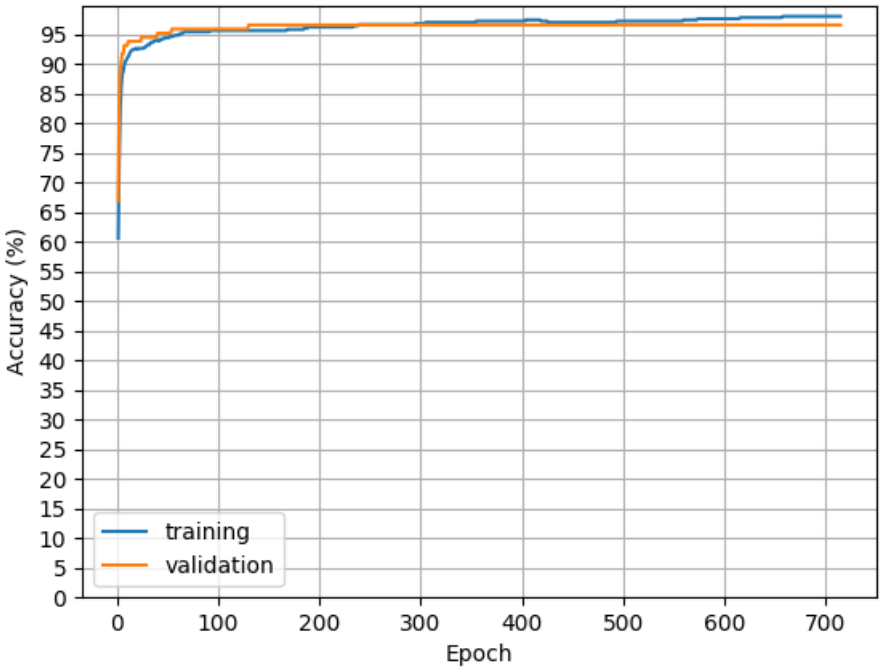

Figure 10: Training vs validation accuracy

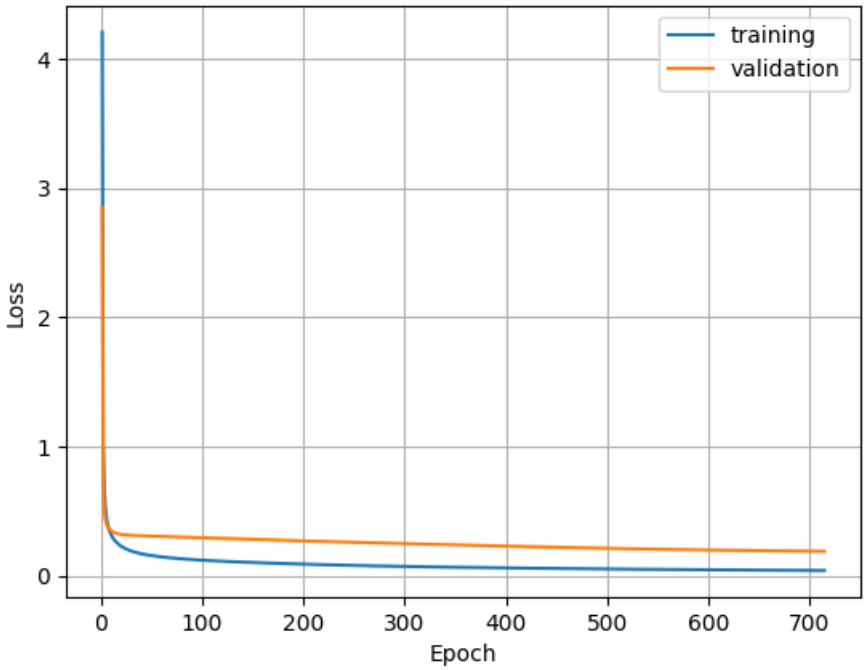

Figure 11: Training vs validation cross entropy error

The best validation accuracy got from multi-class logistic regression is 97.9%. While the performance is less optimal compared to the binary class case, it's understandable. The multi-class problem takes on a probablitic approach over 10 classes. The probability density is estimated with a *softmax* function, where in the binary case, the *sigmoid* function is more like a *hardmax*. In our *softmax* case, the prediction is from the largest probability density, despite there might be close draws. This explains the slight performance drop.

Facerub dataset We only performed name recognition (6-class) in this part (the gender part is binary classification). The model parameters are set to $\lambda = 0.01$ and $\eta = 0.001$ after doing experiments with the values. figures 12(page 12) and 13(page 13) shows accuracy and loss vs the number of epoch (here 10000 iterations are performed).

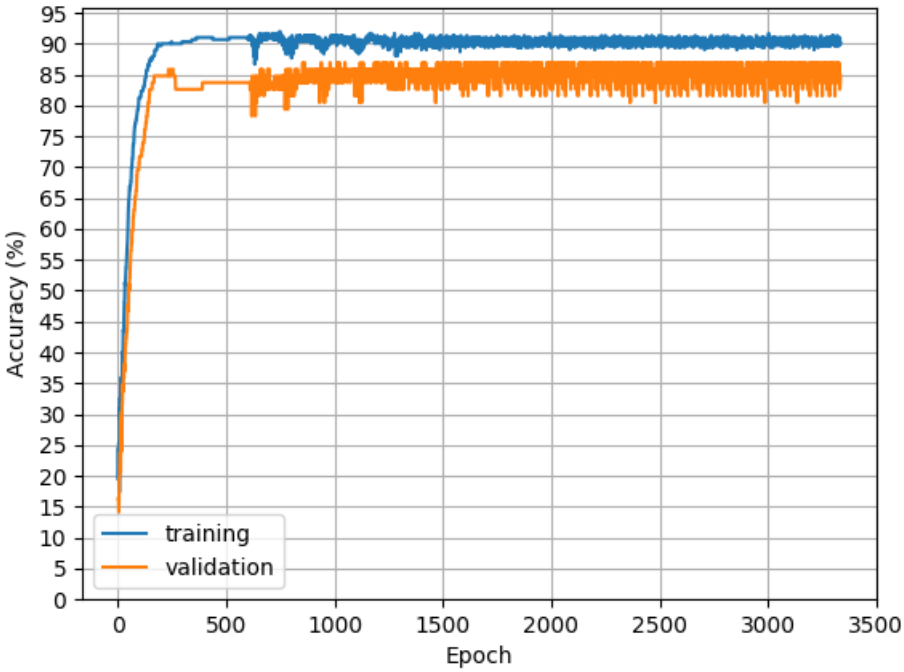

Figure 12: Training vs validation accuracy

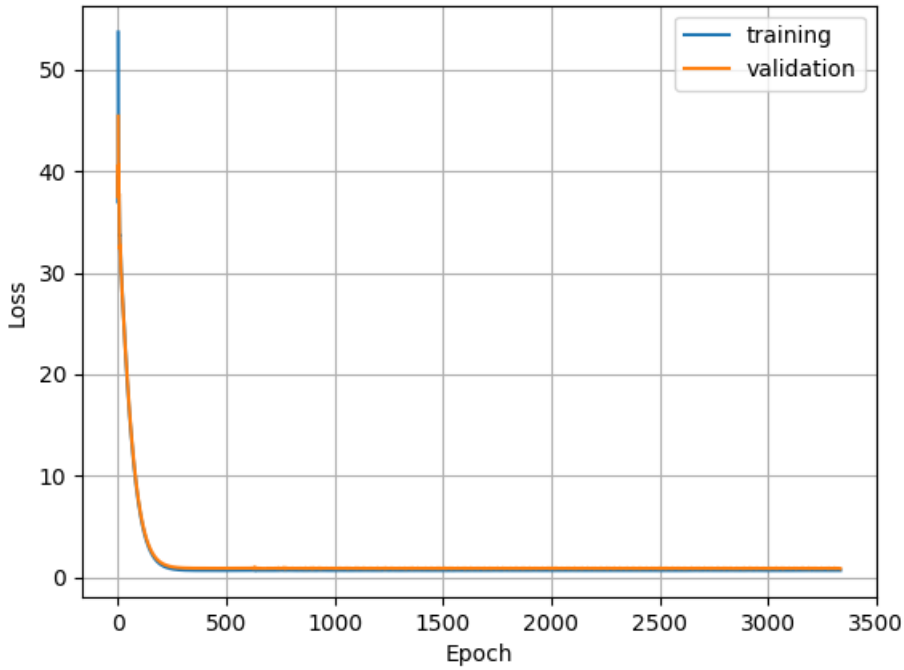

Figure 13: Training vs validation cross entropy error

The best validation accuracy achieved on this dataset is 87.0%, which is much better than the k-NN accuracy (71.0%). As mentioned in our previous report, k-NN simply gets the distance from the luminosity of the pixels. Logistic regression, although not taking into account the local features, managed to capture useful data by using a weight for every pixel.

Appendix

There are two versions of source code, both works for all the parts. These versions, along with a pdf of this report, are included in a compressed file `ECE521Asst2.zip` sent to ece521ta2018@gmail.com

Version 1 `version1` contains the code version 1, specifically,

- `Linear_L.py` contains the code for the first part, including the normal equation part
- `Logistic_L.py` has the code for binary logistic regression
- `Logistic_multi_L.py` has the code for multiple class logistic regression for MNIST data set
- `Logistic_multi_L2.py` has the code for multiple class logistic regression for face data set
- `Util.py` generates batched input

Version 2 `version2` contains the code version 2, and the files are labeled with the respective question numbers.