KMNIST Dataset Classification

Yifei Wang
Department of Data Science
University of California San Diego
yiw014@ucsd.edu

Rui Zhang
Department of Data Science
University of California San Diego
r2zhang@ucsd.edu

Abstract

To handle two-class and multi-class classification tasks on labeled dataset, logistic regression and SoftMax regression are classical supervised machine learning methods. We propose to classify two-class KMNIST dataset using a one-layer neural network with logistic regression as the activation function and classify multi-class KMNIST dataset using SoftMax Regression. Stochastic gradient descent is used in the weight update for this linear neural network. The objective function in this network is built using binary cross entropy and multiple class cross entropy. Finally, we provide rigorous testing to demonstrate the proposed neural network's great performance. Aside from demonstrating the power of logistic regression and softmax regression in the context of handwritten digits classification, we also conduct hyperparameter selections based on the performance of the models to which we applied different sets of hyperparameters. For the model performance, we attain a test accuracy of 98.34% for class 0 vs class 6 classification. For class 2 vs class 6, we have a test accuracy of 87.95%. With early stopping and proper hyperparameter settings, we get a 70.34 % test accuracy on class 0-9 categorization.

1 Introduction

In this paper, we propose utilizing a one-layer neural network with logistic regression as the activation function to classify two-class KMNIST datasets and SoftMax Regression to classify multi-class KMNIST datasets. We will experiment with discerning between two classes of data (class 2 vs class 6, class 0 vs class 6), and with multi-class classification. We first implement logistic regression for the two-class classification. Softmax regression is the generalization of logistic regression for multiple class classification, so we also use SoftMax to classify class 0-9. We apply K-fold cross validation and early stopping technique to prevent overfitting. We also utilize mini batches for stochastic gradient descent in our approach, which makes the algorithm far more efficient than the one that is implemented with original gradient descent.

We experiment with different combinations of hyperparameters, specifically we pay attention to the learning rate, normalization method and early stopping method. We find that with appropriate learning rate, we could boost the class 0 vs class 2 classification test accuracy from % to %. As a result, for class 0 vs class 6 classification, we are able to achieve a test accuracy of 98.34%. We obtain a 87.95% test accuracy for class 2 vs class 6. We also achieve a 70.34% accuracy on class 0-9 classification with early stopping and appropriate hyper parameters.

2 Dataset Description

Kuzushiji-MNIST(KMNIST) is a replacement for the MNIST dataset (28x28 grayscale, 70,000 images), where one character is chosen to represent each of the 10 rows of Hiragana [1]. To further investigate the KMNIST dataset, we have showed one randomly sampled example from each class

(0-9) in Figure 1:

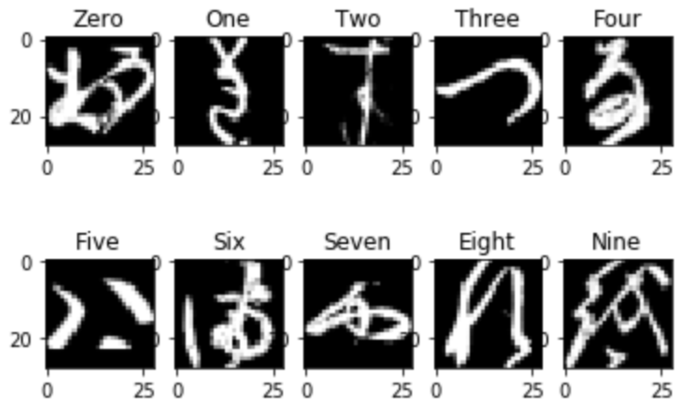

Figure 1: Sample KMNIST Digit

For the prepossessing of the data, we apply min-max normalization to normalize the data and we also append bias to the patterns data so that we could have an extra error dimension for the pattern recognition. When we enter the multiple classification stage, we also one-hot encode the 10 class KMNIST Digit labels for the SoftMax Regression.

To better illustrate how the KMNIST dataset is split, we include a table showcasing the statistics for the size of train and test data for the different classes in the KMNIST dataset:

Table 1: Train-Test size for each class

Class	**Train Set Size**	**Test Set Size**
Zero	6000	1000
One	6000	1000
Two	6000	1000
Three	6000	1000
Four	6000	1000
Five	6000	1000
Six	6000	1000
Seven	6000	1000
Eight	6000	1000
Nine	6000	1000

3 Related Work

KMNIST dataset has been widely used in the image classification area. In “Domain2vec: Domain embedding for unsupervised domain adaptation supplementary material”, Peng et al. applied domain2vec method to KMNIST dataset to uncover the domain embedding technique for unsupervised neural network learning and they summarized the category information in the open set domain adaptation experiments [2].

Linda Studer and her colleagues adopted inter-disciplinary approaches to KMNIST dataset to enhance the comprehension in automatic analysis of historical documents, they obtained mixed results by applying semantic-segmentation at pixel level, and they discovered that ImageNet pre-training has a positive influence on classification and content-based retrieval across a variety of network architectures [3].

4 Logistic Regression

4.1 Description of Method

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a sigmoid function [4]. And we use stochastic gradient descent to update the weights in our logistic regression linear neural network. To classify the two class labels in the KMNIST dataset, we used logistic regression as the activation method along with binary cross entropy as the loss function to build the linear neural network.

4.2 Class 0 vs Class 6

(a) Training/Validation Loss

To accomplish the binary classification task of differentiating between class 0 and class 6, we build a Logistic Regression model by using 0.001 as our learning rate, setting the size of each minibatch to be 300, and using the z_score normalization method. We use 10-fold cross validation along with 100 epoches for each fold to avoid overfitting. We then plot a training/validation loss graph for each fold, and hoping that by comparing the Training Loss and Validation Loss for each epoch in Figure 2, we can determine the best number of epochs one should use when we train the logistic regression model.

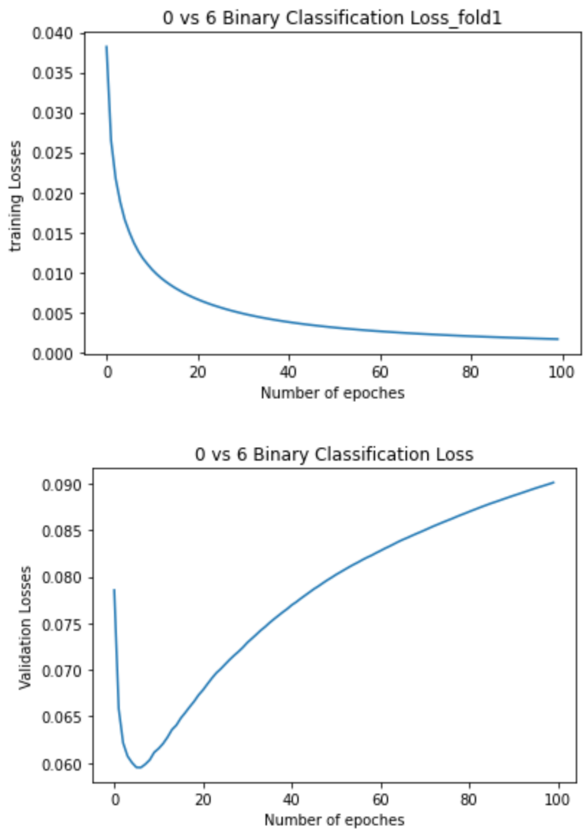

Figure 2: Logistic Training/Validation Loss for a single fold

We then change the size of each minibatch to be 64, and use the z_score normalization method. We use 10-fold cross validation along with 100 epoches for each fold to avoid overfitting. We provide a plot of a training/validation loss graph by averaging the average of loss during each epoch for each fold in Figure 3.

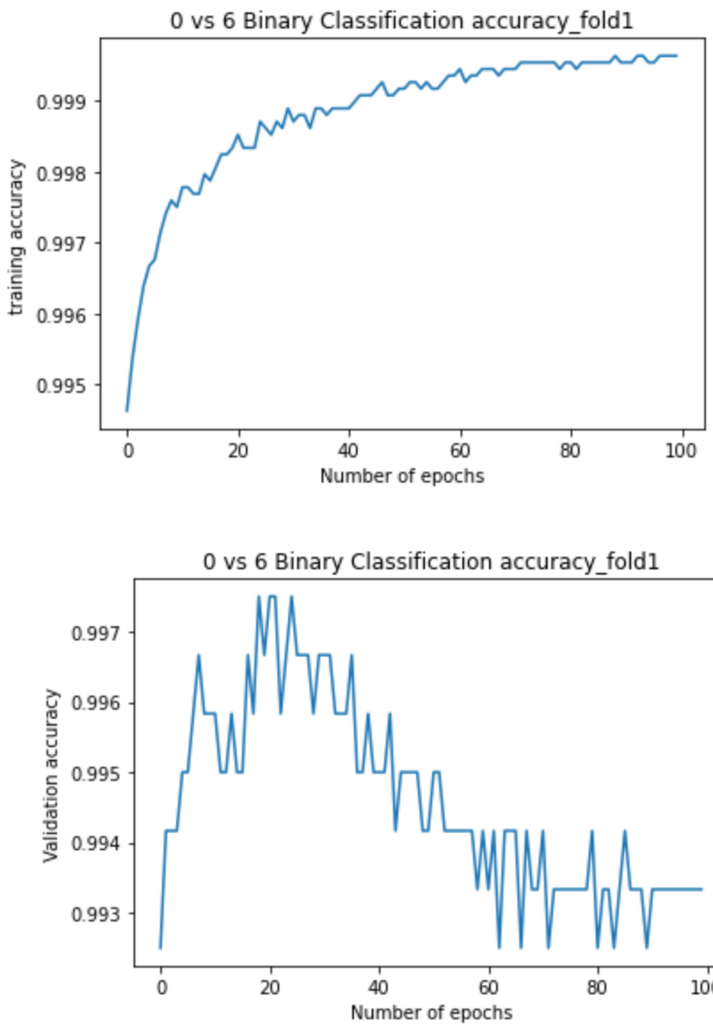

Figure 3: Logistic Training/Validation accuracy for a single fold

(b) Test Performance

By changing the batch size from 300 to 64, the test accuracy is 98.34%. The learning rate is set to be 0.001. The type of the normalization we are using is z-score normalization.

(c) Discussion of the Results and Future Improvement

Such an amazing result of our model should not be surprising to us. Despite the fact that the model we are using here is simple, it works surprisingly well on the classes of images whose pixels are distributed in a discriminative way. For example, as the readers might notice, for class 0, the white pixels are evenly spread over the image space. On the contrary, for class label 6, the white pixels are more concentrated over the right side of the image.

Also, due to the effect of the normalization, we could also see an improvement on the test accuracy. This illustrates the importance of data preprocessing steps. Ideally, if we improve the feature space of our input data and reduce some unnecessary noise, our algorithm's performance could improve dramatically.

We therefore decide to implement an early stop feature when our algorithm is trained based on this particular set of data. There are few challenges that one may face when trying to implement the early stop feature. 1) Due to the fact that we are using stochastic gradient descent algorithm to train our model, the validation lost may fluctuate, which may cause difficulty in determining whether the given increase in the loss is caused by overfitting or just randomness. 2) Since we are doing k-fold validation and train our model based on different sets of data, the best stop time is not so easy to determine since it is really depending on the data we are using.

Based on the above mentioned difficulty, we use a simple way to implement early stopping; We would try a range of early stopping thresholds, and pick the one with highest accuracy. In this case, given that our task is not computationally heavy, we set M(maximum number of epochs for training the model) to be between 15-20. This range is picked based on observation of the graphs we plot.

Then we record the test accuracies for different early stopping stages to see if early stopping does help improve the test accuracy. Below is the table documenting the accuracy when we implement early stop.

Table 2: Test Accuracy for different early stopping number of epochs

M	Test Accuracy
15	98.75%
16	98.7%
17	98.75%
10	98.75%
100	98.35%

One thing that readers should notice is that since our training data is sampled randomly from a larger dataset, the result of our experiment might be varying. Nevertheless, implementing early stopping during the training process can effectively prevent overfitting, and that is exactly what we are going to do for the next task: classifying data class 2 from data class 6.

4.3 Class 2 vs Class 6

(a) Training/Validation Loss

To accomplish the binary classification task of differentiating between class 2 and class 6, we build a Logistic Regression model by using 0.001 as our learning rate, setting the size of each minibatch to be 300, and using the z_score normalization method. We use 10-fold cross validation along with 100 epoches for each fold to avoid overfitting.

We then plot a training/validation loss graph for each fold in Figure 4-5, and hope that with the graph, we can determine the best number of epoche one should use when we train the logistic regression model. Since the graphs we are plotting exhibit the same pattern over and over again, we will select and present one set of them.

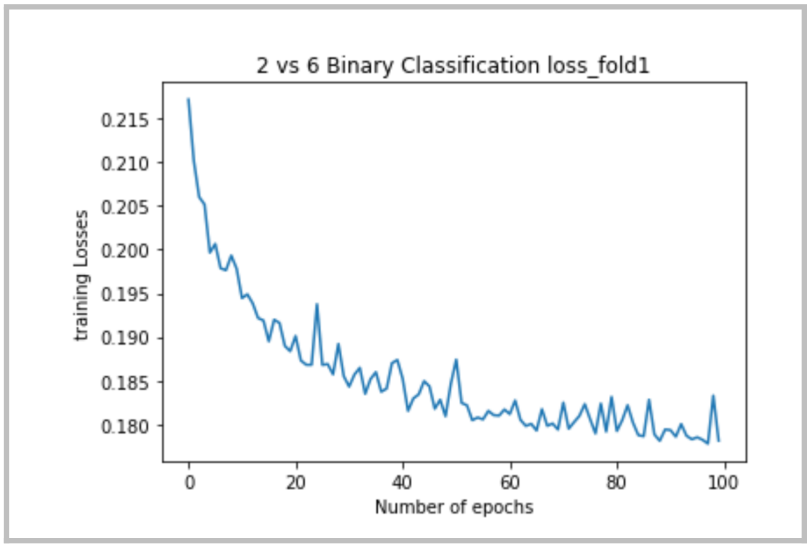

Figure 4: Logistic Training Loss

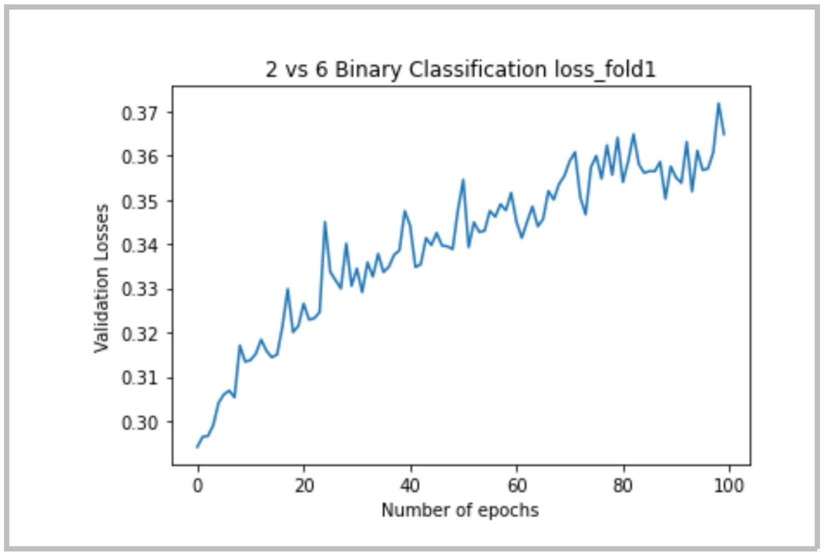

Figure 5: Logistic Validation Loss

We also include a plot of a single fold validation accuracy in Figure 6 to represent the fluctuations in the binary classification validation accuracy when we do not use Early Stopping in Figure 6.

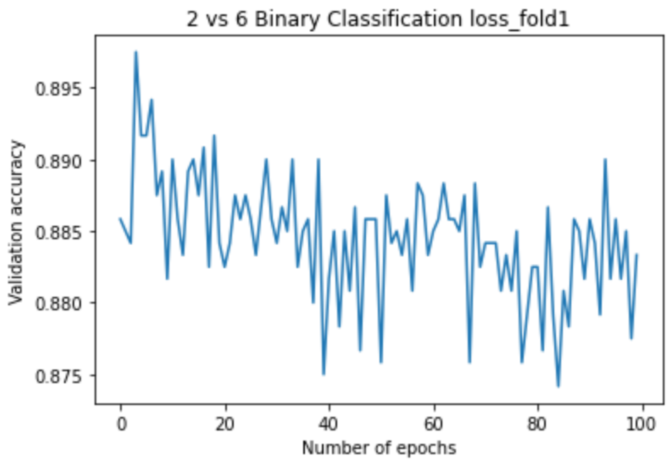

Figure 6: Logistic Validation Accuracy

(b) Test Performance

Given the best hyperparameters we set, without the implementation of early stopping, we reach the test accuracy of 87.95%.

After that, we plot the training/validation losses and try to eyeball the best early stopping threshold. However, as can be seen from the validation graph, the loss is not decreasing anywhere. Instead, the loss is increasing over time.

Then, a smart way would be looking at the validation set's accuracy instead of the losses, the graph is less obvious than what we have for 0/2 classification but still perceivable. The accuracy starts dropping after M=15. So that is what we set for M in our experiment.

Indeed, after setting M=15, we get a decent test accuracy boost: our model test accuracy becomes 0.889. Again, it might be higher in reality given the stochastic nature of our algorithm. But the important takeaway is that we should always do early stopping with the help of train/validation loss/accuracy visualizations.

(c) Discussion of the Results

One interesting question that might be asked is, why does the same piece of algorithm behave so differently for a different set of data? We could attribute the reason to the data itself. As can be seen from Figure 1, class 2 and class 6 have slightly more similar white pixel distribution than class 0 and class 6 have. This means that it would be harder for the algorithm to distinguish between class 2 and class 6. This kind of dilemma, or weakness, manifests itself in the logistic/softmax regression tasks when the data from each class we are trying to model are hard to differentiate.

In this case, we could use a deeper network or do clever feature engineering to get higher model performance. For example, PCA (Principal Component Analysis) might be a good choice when we try to discover the latent structure/representation of the data, and hence reduce its dimensions for clearer representation and pattern recognition.

5 SoftMax Regression

5.1 Description of Method

In *Dive into Deep Learning*, SoftMax regression was described as a method that interprets the outputs of a model as probabilities: parameters were optimized to produce probabilities that maximize the likelihood of the observed data. Then, to generate predictions, thresholds were set to choose the label with the maximum predicted probabilities [4]. To classify the ten classes in the KMNIST dataset, we used SoftMax regression as the activation method along with multiple class cross entropy as the loss function to build the linear neural network.

5.2 Multiple Class Classification

(a) Training/Validation Loss

To correctly classify class 0-9 different Hiragana characters, we build a SoftMax Regression model by using 0.001 as our learning rate, setting the size of each minibatch to be 300, and use the z_score normalization method. We use 10-fold cross validation along with 100 epoches for each fold to avoid overfitting. We then plot a training/validation loss graph by averaging the average of loss during each epoch for each fold:

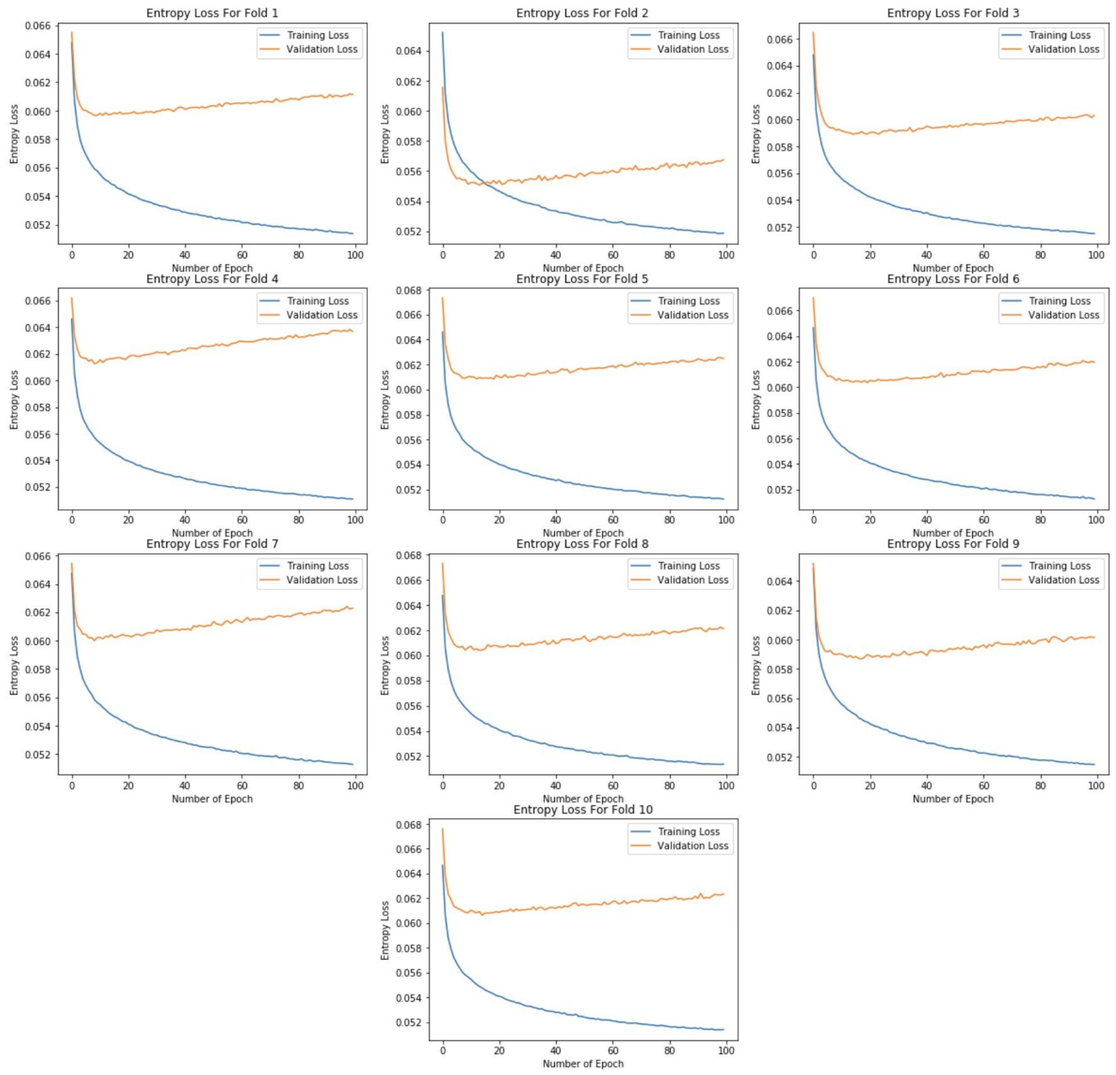

Figure 7: SoftMax Training/Validation Loss Each Fold without Early Stopping

From Figure 7, we could clearly see that without early stopping, although training loss continually decreases over the epochs, the validation cross entropy loss goes down in the first few epochs, then the validation loss fluctuates and gradually increases. This indicates that our model is overfitting, and that we should stop updating the weights for the model once the loss begins to increase.

We then implement early stopping to our chosen hyperparameters, and the early stopping runs 20 epochs for each fold in our 10-fold cross validation procedure for training the model. We plot a training/validation loss graph by averaging the average of loss during each epoch for each fold:

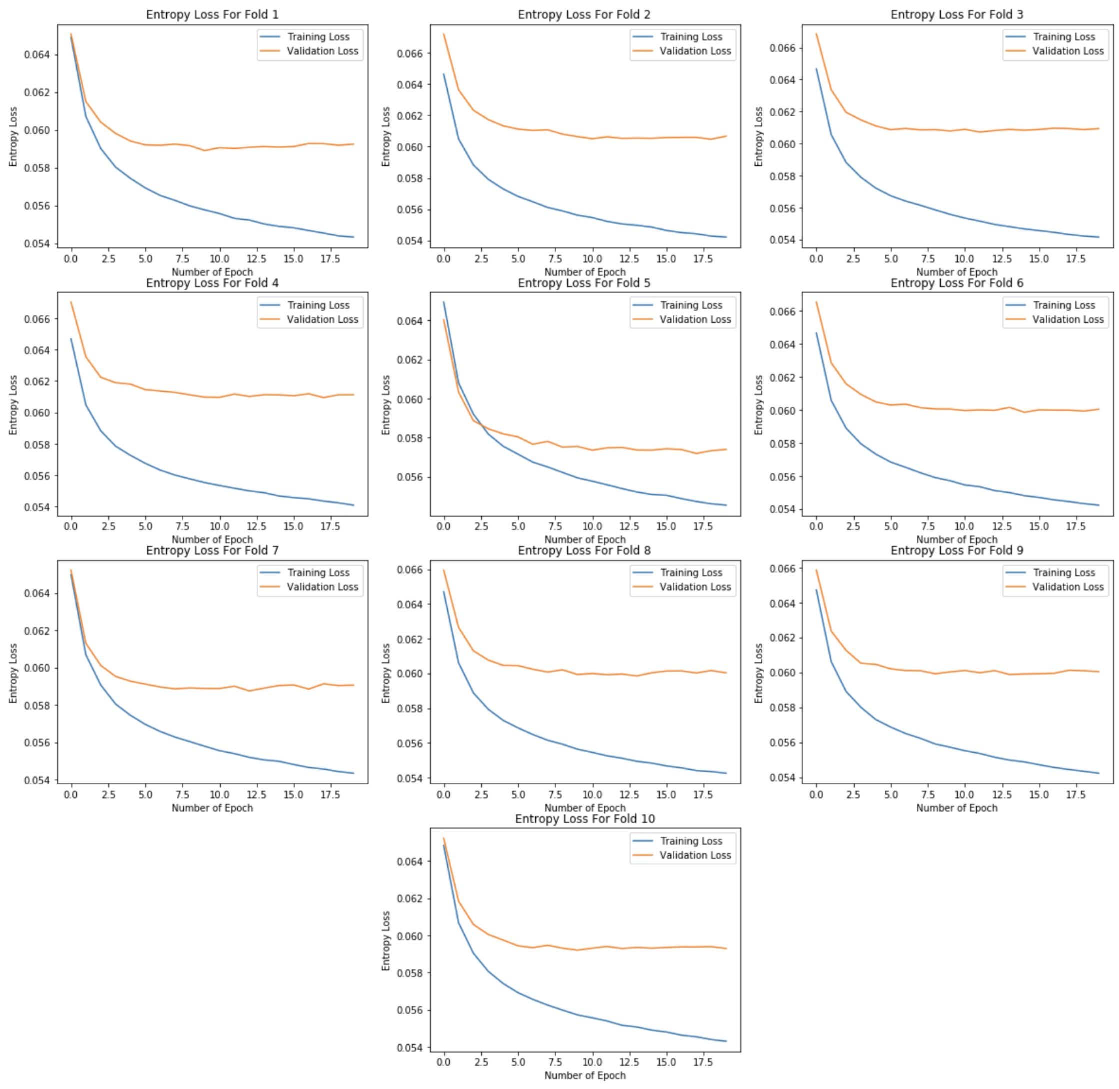

Figure 8: SoftMax Training/Validation Loss Each Fold with Early Stopping

From Figure 8, we could see that with early stopping, training loss gradually decreases over the epochs, the validation cross entropy loss goes down in the first few epochs, then the validation loss fluctuates and the validation loss would not increase significantly.

(b) Test Performance

Without early stopping, our test accuracy is only 67.92% for the chosen hyperparameters set. By implementing early stopping in our SoftMax Regression model, we are able to have a test accuracy of 70.34%, which clearly shows that early stopping could effectively prevent overfitting.

(c) Discussion of the Result

By comparing the multiple class entropy loss and the test accuracy of the model without early stopping and the model with early stopping, we find early stopping crucial to prevent overfitting in this multiple class categorization setting.

From Figure 7, we could clearly see that without early stopping, although training loss continually decreases over the epochs, the validation cross entropy loss goes down in the first few epochs, then the validation loss fluctuates and gradually increases. This indicates that our model is overfitting, and that we should stop updating the weights for the model once the loss begins to increase.

From Figure 8, we could see that with early stopping, training loss gradually decreases over the epochs, the validation cross entropy loss goes down in the first few epochs, then the validation loss fluctuates and the validation loss would not increase significantly.

(d) Visualize the Weights for Each Output Class

To further understand how a single layer softmax regression neural network would work , we provide a visualization of weights for each output class by dropping the bias term and reshape the weight vector for each class into 28x28 pixels images in Figure 8.

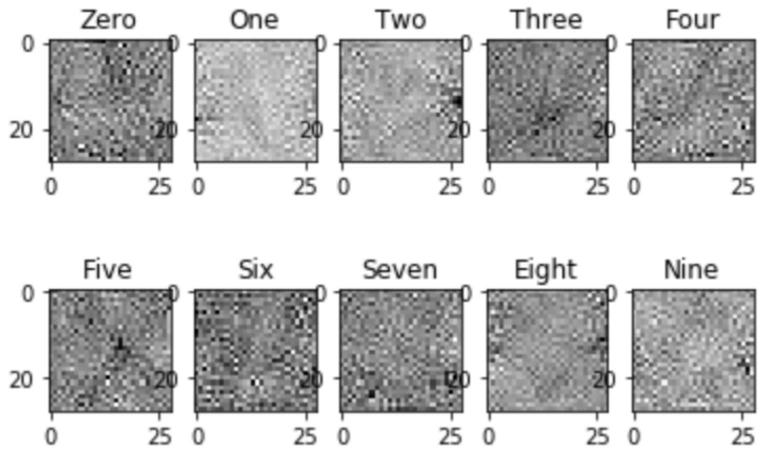

Figure 8: SoftMax Weights Visualization

(e) Findings from the Weight Visualization

As can be seen from Figure 8, each piece of visualization we plot shows latent characteristics of the underlying class of data we are modelling. If the visualization of input data is close to one of the weight visualizations, then our model is likely to classify the input example to the class where the weight visualization plot resembles that of the data.

The visualization of weight vectors for each output class is an abstraction of the data within that class, we could identify the resemblance of Figure 8 and Figure 1.

6 Team contributions

Rui Zhang implemented the majority of code for logistic regression and SoftMax regression. Yifei Wang compiled most of the report and graphs. Yifei Wang and Rui Zhang together worked on figuring out the dimensions of the dataset in the training method for the main Network and tested several combinations of hyperparameters.

References

[1] Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., & Ha, D. (2018). Deep learning for classical Japanese literature. arXiv preprint arXiv:1812.01718.

[2] Peng, X., Li, Y., & Saenko, K. (2020). Domain2vec: Domain embedding for unsupervised domain adaptation supplementary material. ECCV.

[3] Studer, L., Alberti, M., Pondenkandath, V., Goktepe, P., Kolonko, T., Fischer, A., ... & Ingold, R. (2019, September). A comprehensive study of imagenet pre-training for historical document image analysis. In 2019 International Conference on Document Analysis and Recognition (ICDAR) (pp. 720-725). IEEE.

[4] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. arXiv preprint arXiv:2106.11342.