

---

# Image Classification in Practice: Implementing Logistic and Softmax Regression Using Numpy

---

**Linghang Kong**

Halicioğlu Data Science Institute  
University of California San Diego  
San Diego, CA 92093  
l3kong@ucsd.edu

**Weiyue Li**

Halicioğlu Data Science Institute  
University of California San Diego  
San Diego, CA 92093  
wel019@ucsd.edu

**Yi Li**

Halicioğlu Data Science Institute  
University of California San Diego  
San Diego, CA 92093  
yil115@ucsd.edu

## Abstract

In this project, we were using Numpy to implement both Logistic Regression and Softmax Regression by using Stochastic Gradient Descent. We have used these regression classifiers to classify Japanese Hiragana hand writing from dataset KM-NIST. The accuracy for Logistic Regression classifier on **class 0** (お) and **class 6** (ま) is above 98%, the accuracy for Logistic Regression classifier on **class 2** (す) and **class 6** (ま) is above 85%, and the accuracy for Softmax Regression classifier on test set including **all 10 classes** of characters is approximately 70%. To visualize the training progress, we have use Matplotlib to perform visualizations on the loss history.

## 1 Introduction

We trained a logistic regression and a softmax regression classifier to identify Japanese Hiragana hand writing. We utilized Python library Numpy for model implementation and Matplotlib for training visualization. By training our models on a training set of size 60,000, we are able to obtain a simple neural network that can classify Hiragana hand writing into the corresponding labels with high accuracy. After normalization and one-hot encoding, we feed the data into the network, where weight vectors are simulated via stochastic gradient descent (SGD), and calculate the accuracy and loss on the training sets, validation sets, and test sets. Based on the results from test sets, we obtained an accuracy of approximately 98% on the logistic regression between target class 0 and target class 6, 88% on the logistic regression between target class 2 and target class 6, and 70% on softmax regression.

## 2 Related work

[1] Clanuwat, Tarin, et al. "Deep learning for classical japanese literature." arXiv preprint arXiv:1812.01718 (2018).

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

### 3 Dataset

#### 3.1 Dataset introduction

The Hiragana handwriting dataset we use in this project is called KMNIST(Kuzushiji-MNIST). It is a drop-in replacement for the original MNIST dataset. Since the original MNIST dataset only have ten classes, KMNIST is restricted to 10 classes as well. In this case, the 10 classes represent 10 different Hiragana characters. The images are sizes of 28\*28, and are reshaped into 1 dimension array of length 784 for input purposes.

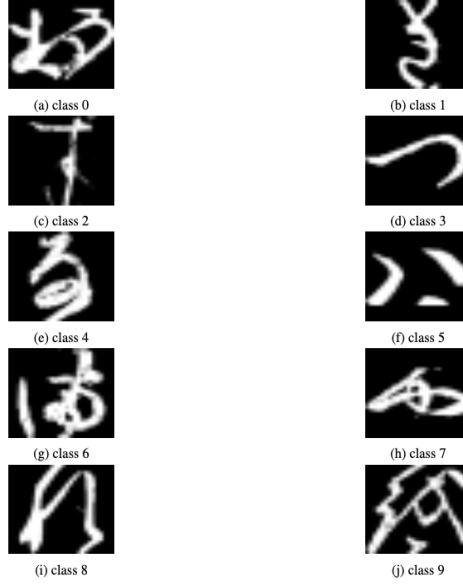


Figure 1: Sample figure caption. Source: KMNIST Dataset website

#### 3.2 Data distribution

Number of samples per class in training set and test set is shown in Table 1. According to Table 1, there are 6000 samples per class in the training set, 60000 samples in total, and 1000 samples per class in the test set, 10000 samples in total.

Table 1: Sample distribution among classes

Class	Number of samples (Train)	Number of samples (Test)
0	6000	1000
1	6000	1000
2	6000	1000
3	6000	1000
4	6000	1000
5	6000	1000
6	6000	1000
7	6000	1000
8	6000	1000
9	6000	1000

#### 3.3 Data preprocessing

The labels for each sample will be one-hot encoded as vectors of length 10, in which each index indicates whether this sample belongs to the corresponding class or not, with 1 represents true and 0 represents

false. The patterns are normalized so that tasks can be performed on a common scale without distorting differences in the range of values or lose information. Two normalization methods, z-score normalization and min-max normalization are implemented, and can be altered through change in hyperparameters. The z-score normalization is calculated with the formula  $f(x) = \frac{(x-\mu)}{\sigma}$ , where  $\mu$  is mean of  $x$  and  $\sigma$  is standard deviation of  $x$ . The min-max normalization is computed by the formula  $f(x) = \frac{x-\min(x)}{\max(x)-\min(x)}$ .  $x$  stands for the data to normalize. Afterwards, before starting the training process, to augment the data, we added bias  $w_0$  to the samples in the form of value 1 at the beginning of the array, thus changing the dimension of each individual sample input from 784 to 785.

During the training process, in order to estimate the model's performance on unseen data before testing as well as prevent over-fitting, we will use a k-fold cross validation method to split a portion of the training data into validation data at each fold. In our circumstance, we choose  $k$  as 10, so that data is splitted into 10 folds, and 9 out of the 10 folds will be used as training data, and the rest will be validation data. The procedure is shown in the training procedure below.<sup>1</sup>

---

```

1: procedure TRAINING PROCEDURE
2:   folds =  $k$  mutex split of training data;
3:   for fold = 1 to  $k$  do
4:     val_set  $\leftarrow$  folds[fold];
5:     train_set  $\leftarrow$  remaining folds;
6:     for epoch = 1 to  $M$  do
7:       train the model with train_set, and test the performance on val_set every epoch
8:       record train_set, val_set loss for plotting and accuracy on val_set for hyperparameter
        tuning
9:       save the best model based on val_set performance
10:    plot the training and validation loss curves
11:    use the best model to report accuracy on test

```

---

## 4 Logistic regression

Logistic Regression is used for binary classification, we have used a sigmoid function to serve as our activation function, where the range of the sigmoid function is  $(0, 1)$ .

$$g(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})} \quad (1)$$

For binary classification, we classify our images of Hiragana hand writing into two classes. For convenience, we call one class as class 0 and another class as class 1. It is useful to know that the probability of an image  $x$  being classified to class 1 is the prediction of the activation function:

$$P(C_1 | \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})} = y \quad (2)$$

Since it is a binary classification, the probability of being classified into class 0 and 1 should sum to one, which implies:

$$P(C_0 | \mathbf{x}) = 1 - y \quad (3)$$

For logistic regression, we will be using binary cross-entropy cost function to calculate our loss.:

$$E(w) = - \sum_{n=1}^N \{t^n \ln(y^n) + (1 - t^n) \ln(1 - y^n)\}. \quad (4)$$

where  $t^n \in \{0, 1\}$  is the label for image  $n$ , and we will minimize it via Stochastic Gradient Descent. The gradient for this loss function is:

$$\frac{\partial E(w)}{\partial w_j} = - \sum_{n=1}^N (t^n - y^n) x_j^n \quad (5)$$

---

<sup>1</sup>CSE 151B Winter 2022 PA1 Writeup

To implement stochastic gradient descent, we follow the pseudo code in the programming assignment write up in **Algorithm 1** shown below.<sup>2</sup>

Intuitively, as we initialize our weight vector to 0, and multiply it to the augmented sample values in our dataset, we obtain a weighted sample that reflects how the model learned from the image. Under the effect of sigmoid activation function, the result can be mapped into the range (0,1) as a representation of the sample's probability of being a class 1 object. We use 0.5 as a threshold, indicating that those sample with a parameterized value higher than 0.5 after activation are going to be classified into class 1, while those lower than 0.5 are going to be classified into class 0. Since there will absolutely be differences between the sample's parameterized value and its observed value (0 and 1), as results calculated via sigmoid function cannot be equal to 1 or 0, we will be able to obtain a loss in our training using the binary cross entropy function. Our goal is to update our weight vector using gradient descent, and minimize the cross-entropy loss, such that our model's performance is optimized.

---

**Algorithm 1** Stochastic Gradient Descent

---

```

1: procedure STOCHASTIC GRADIENT DESCENT
2:    $w \leftarrow 0$ 
3:   for  $t = 1$  to  $M$  do ▷ Here, t is one epoch.
4:     randomize the order of the indices into the training set
5:     for  $j = 1$  to  $N$ , in steps of  $B$  do ▷ Here, N is number of examples and B is the batch size
6:        $start = j$ 
7:        $end = j+B$ 
8:        $w_{t+1} = w_t - learning\_rate * \sum_{n=start}^{end} \nabla E^n(w)$ 
9:   return  $w$ 

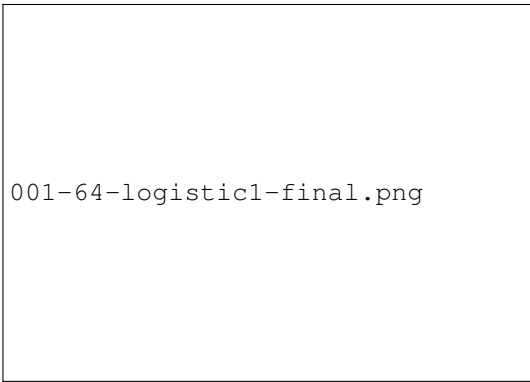
```

---

#### 4.1 Results for Class 0 vs Class 6

Table 2: Model Results on Class 0 vs Class 6

Learning Rate	Batch Size	Val Accuracy	Test Accuracy	Test Loss
0.1	64	0.9961	0.9875	0.0385
0.1	128	0.9966	0.9885	0.0370
0.1	256	0.9968	0.9875	0.0358
0.01	64	0.9952	0.9900	0.0371
0.01	128	0.9966	0.9880	0.0357
0.01	256	0.9970	0.9880	0.0361
<b>0.001</b>	<b>64</b>	0.9950	<b>0.9900</b>	<b>0.0366</b>
0.001	128	0.9943	0.9890	0.0394
0.001	256	0.9934	0.9890	0.0448



001-64-logistic1-final.png

Figure 2: Training and Validation Loss of the Final Result for Logistic Regression Between Class 0 and Class 6 (lr=0.01, batch-size=64)

---

<sup>2</sup>CSE 151B Winter 2022 PA1 Writeup

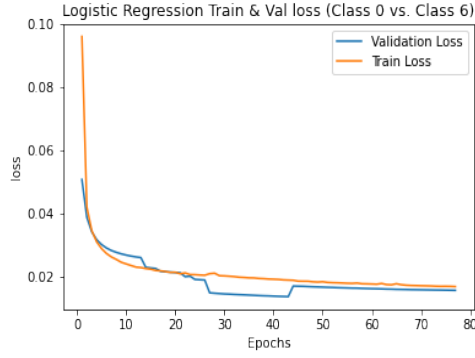


Figure 3: Training and Validation Loss of the Final Result for Logistic Regression Between Class 0 and Class 6 (lr=0.01, batch-size=128)

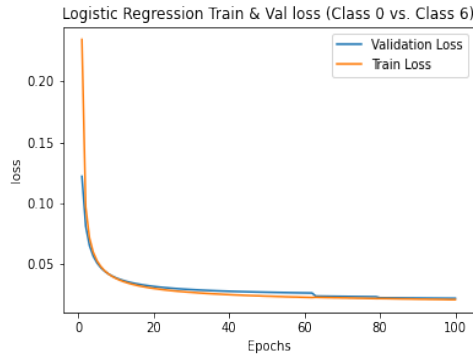


Figure 4: Training and Validation Loss of the Final Result for Logistic Regression Between Class 0 and Class 6 (lr=0.001, batch-size=64)

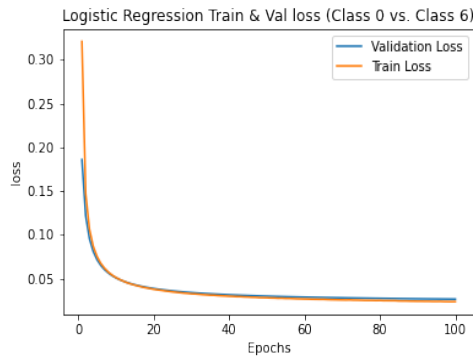


Figure 5: Training and Validation Loss of the Final Result for Logistic Regression Between Class 0 and Class 6 (lr=0.001, batch-size=128)

Figure 2 to 5 shows training and validation loss under 0.001 learning rate and batch size of 64 over 100 epochs. For fine-tuning, we applied three different sets of learning rate 0.1, 0.01, 0.001 combined with three sets of batch size 64, 128, 256. During the training and validation process, we noticed that the batch sizes we selected did not lead to significant changes in accuracy. On the other hand, the smaller the learning rate, the better the result. According to Table 2, our best accuracy is 0.99 on the test set with learning rate 0.001 and a loss of 0.0366.

## 4.2 Results for Class 2 vs Class 6

Table 3: Model Results on Class 2 vs Class 6

Learning Rate	Batch Size	Val Accuracy	Test Accuracy	Test Loss
0.1	64	0.9102	0.8815	0.2923
0.1	128	0.9158	0.8855	0.2838
0.1	256	0.9092	0.8805	0.2987
<b>0.01</b>	<b>64</b>	0.9177	<b>0.8860</b>	<b>0.2845</b>
0.01	128	0.9166	0.8840	0.2887
0.01	256	0.9166	0.8850	0.2873
0.001	64	0.9155	0.8835	0.2858
0.001	128	0.9152	0.8830	0.2867
0.001	256	0.9079	0.8820	0.2901

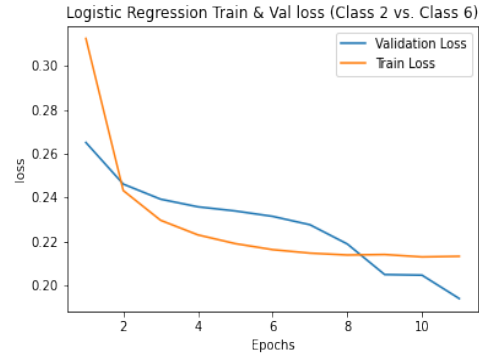


Figure 6: Training and Validation Loss of the Final Result for Logistic Regression Between Class 2 and Class 6 (lr=0.01, batch-size=64)

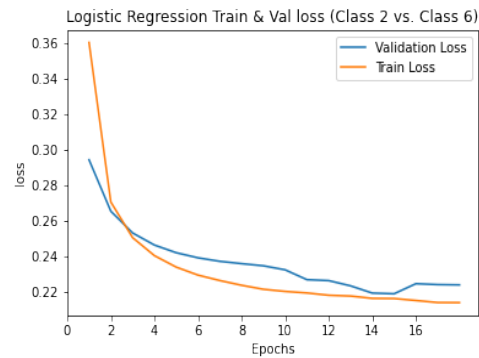


Figure 7: Training and Validation Loss of the Final Result for Logistic Regression Between Class 2 and Class 6 (lr=0.01, batch-size=128)

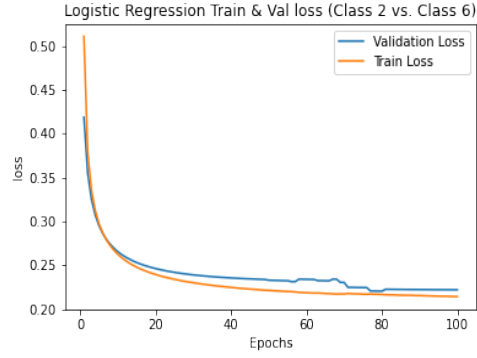


Figure 8: Training and Validation Loss of the Final Result for Logistic Regression Between Class 2 and Class 6 ( $lr=0.001$ ,  $batch-size=64$ )

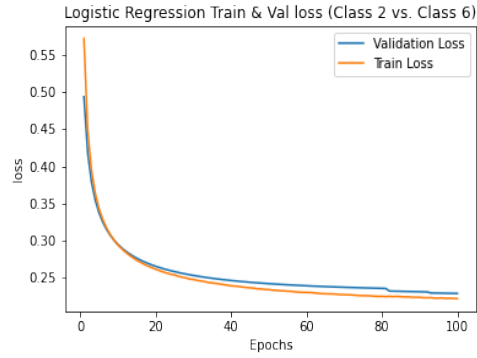


Figure 9: Training and Validation Loss of the Final Result for Logistic Regression Between Class 2 and Class 6 ( $lr=0.001$ ,  $batch-size=128$ )

Again, we applied the three sets of learning rate and three batch sizes to tune the model. Batch sizes did not have much effect on model performance. We achieved best accuracy of 0.8850 on test set with learning rate of 0.001 and batch size of 128.

Figure 6 to 9 represents training and validation loss under different learning rates and batch sizes over 100 epochs.

### 4.3 Discussion



Figure 10: More Samples of Class 0



Figure 11: More Samples of Class 2



Figure 12: More Samples of Class 6

Although we showed a sample from each class in section 3, one sample may not be representative. We extracted samples from class 0, 2, and 6 from KMNIST Github repository, and they are shown in Figure 10, 11, and 12. According to the images, from the perspective of a human being, class 0 characters are very different from class 2 and 6 in terms of shape and size. However, class 2 and class 6, even if different in original label, can be written in a similar way. Both of them have vertical strokes as the main character structure, and most of the horizontal strokes are distributed in a similar manner. We believe that due to this similarity in samples, the weight generated based on them tend to show homogeneous features. As a result, the model cannot be as accurate as it was on classification between class 0 and class 6 when decide on the features.

## 5 Softmax regression

Softmax Regression can be seen as logistic regression for multi-class variables. Instead of evaluating whether the input  $x$  belongs to a class or not based on the float output and threshold, softmax regression returns the probability, or the likelihood that input  $x$  belongs to every class  $k$ . The probability that input  $x$  belongs to a certain class  $k$  is calculated using the following equation:

$$y_k^n = \frac{\exp(a_k^n)}{\sum_{k'} \exp(a_{k'}^n)} \quad (6)$$

$$a_k^n = w_k^T x^n \quad (7)$$

Likewise, the cross-entropy cost function for softmax becomes a multi-class cross-entropy that sums loss across all classes. We define such multi-class cross-entropy as the equation below:

$$E = - \sum_n \sum_{k=1}^c t_k^n \ln y_k^n \quad (8)$$

The learning rule is basically the same as logistic regression:

$$-\frac{\partial E^n(w)}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n \quad (9)$$

### 5.1 Results for Softmax Regression

Table 4: Softmax Model Results

Learning Rate	Batch Size	Val Accuracy	Test Accuracy	Test Loss
0.1	64	0.8202	0.6818	1.0572
0.1	128	0.8247	0.7013	1.0436
0.1	256	0.8330	0.6985	1.0383
0.01	64	0.8341	0.7003	1.0232
0.01	128	0.8317	0.7029	1.0099
0.01	256	0.8328	0.7019	1.0095
<b>0.001</b>	<b>64</b>	0.8327	<b>0.7050</b>	<b>1.0057</b>
0.001	128	0.8326	0.7034	1.0083
0.001	256	0.8279	0.6982	1.0070





Figure 13: Training and Validation Loss of the Final Result for softmax Regression on Each Class (lr=0.01, batch-size=64)

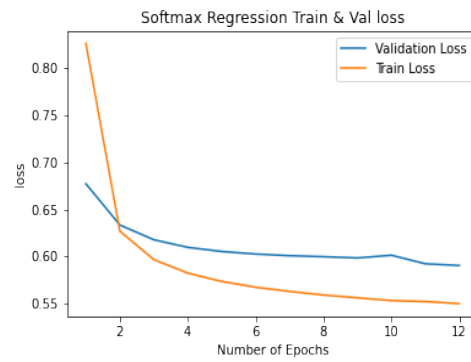


Figure 14: Training and Validation Loss of the Final Result for softmax Regression on Each Class (lr=0.01, batch-size=128)

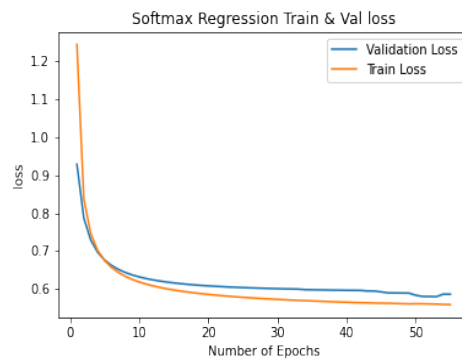


Figure 15: Training and Validation Loss of the Final Result for softmax Regression on Each Class (lr=0.001, batch-size=64)

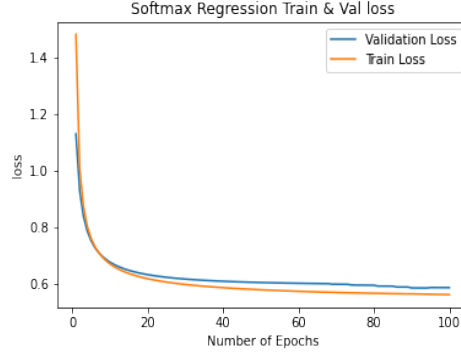


Figure 16: Training and Validation Loss of the Final Result for softmax Regression on Each Class (lr=0.001, batch-size=128)

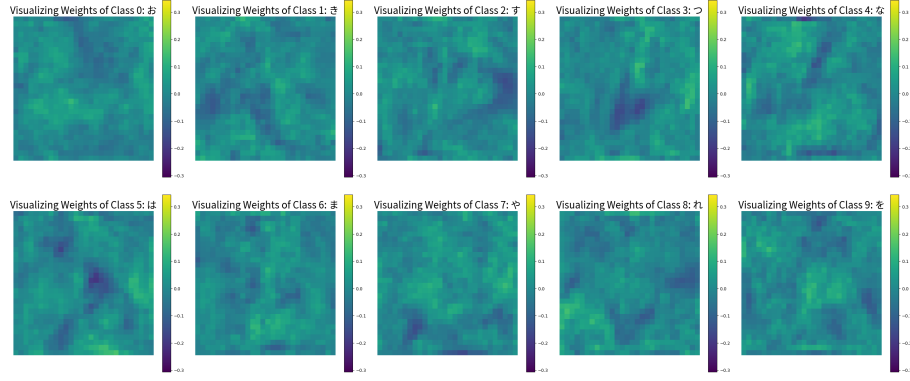


Figure 17: Visualizing Predicted Weights of Softmax Regression on Each Class

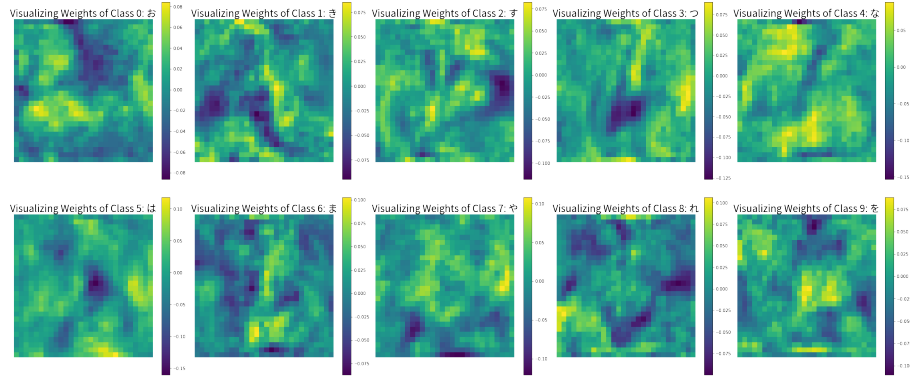


Figure 18: Visualizing Predicted Weights of Softmax Regression on Each Class before scaling

## 5.2 Discussion on loss

For softmax regression, we obtained the best accuracy of 0.7056 on test set using learning rate of 0.01 and batch size of 128.

Figure 16 represents training and validation loss under 0.01 learning rate and batch size of 128 over 100 epochs. We suspect that a greater loss due to implementation of softmax regression happens because we are classifying objects among 10 classes instead of 2. As the original samples from different classes are all cursive writings

with similar texture and patterns, there might be a lot of repetitive or common features among the classes learned by the model, thus decreasing overall accuracy.

### 5.3 Discussion on weight

According to the weight visualization (Figure 8 and Figure 9), we are able to see that the visualized weights are very similar to the original Hiragana Character. Taking class 4 as an example, the strokes in the original character can be viewed as two separate parts: one upper left part and one lower left part. The weight vector has these two parts highlighted accurately. Class 0 has a very different weight pattern from class 6, so that it is not surprising to have a very high accuracy when performing classification between them. Meanwhile, even though class 2 and class 6 have distinguishable label characters, their handwritings are very similar, as their weight vectors seem to have emphasized many similar parts. There are both vertical straight components in these two classes' weight vectors, possibly representing the vertical straight strokes in the label character. Hence, the classification between these two classes becomes more difficult. We concluded that as our model trains through the samples, the samples belong to the same class may not be identical, but they will eventually have many strokes in common positions, and these positions become "feature" for the class and are learned by the model, thus increasing the weights in these positions.

## 6 Conclusion

In this project, we implemented logistic regression and softmax regression methods using Python's numpy library to train a image classifier on KMNIST Hiragana handwriting data. After the data is augmented and one-hot encoded, we put the data through a cross validation training process, and split the data into 90% training set and 10% validation set on each fold. Weights are updated throughout the fixed 100 epochs using Stochastic gradient descent. During gradient descent, cross entropy loss between prediction value and observed value is calculated and used to update the weight vectors. After fine-tuning our model with different learning rates and batch sizes, we finally achieved accuracy over 99% in the logistic regression classification between class 0 and class 6, almost 90% in the logistic regression classification between class 2 and class 6, and over 70% in the softmax regression classification. Training and validation losses are plotted, and weight vector is visualized.

## 7 Team contributions

- Linghang Kong: In this project, I worked on data preprocessing, model pipeline first draft, and hyperparameter importing. I assisted Weiyue Li and Yi Li in optimization and debugging by reviewing code and providing ideas. I am also in charge of visualizing sample images, section 1, 3, and 6 of the report.
- Weiyue Li: In this project, I have worked on building the logistic regression, the `readme.md`, and `main.py`, simplifying our source code (converting for loops and list comprehensions to matrix multiplications in Numpy), and visualizing the predicted weights of the softmax regression. I have assisted Linghang and Yi through debugging process and I was in charge of the Abstract, part 2 and 4 of the report as well.
- Yi Li: In this project, I worked on implementing data preprocessing, logistic regression, softmax regression, and optimizing several functions (convert from for loop to vectorization or matrix multiplication). I am also in charge of visualizing line plot for validation loss and training loss, and assisted Linghang and Weiyue with the report.

## 8 Reference

- [1] Cottrell, Garrison W. 2022, *Logistic Regression Multinomial Regression*, lecture notes, Deep Learning CSE 151B, University of California San Diego, delivered 10 January 2022.
- [2] Cottrell, Garrison W. 2022, *Maximum Likelihood: The theoretical basis for our objective functions*, lecture notes, Deep Learning CSE 151B, University of California San Diego, delivered 14 January 2022.
- [3] Cottrell, Garrison W. 2022, *Perceptrons*, lecture notes, Deep Learning CSE 151B, University of California San Diego, delivered 7 January 2022.
- [4] "KMNIST Dataset" (created by CODH), adapted from "Kuzushiji Dataset" (created by NIJL and others), doi:10.20676/00000341