Experiment on Classification of KMNIST by Multi-layer Neural Networks with Softmax Outputs

Wenbo Hu, Wenxiao Li, Huaning Liu
University of California, San Diego
`w1hu@ucsd.edu, wel032@ucsd.edu, h9liu@ucsd.edu`

Abstract

In this project, based on the softmax regression achieved from the previous project, we further implement and discuss the performance of softmax regression with multi-layer neural networks on KMNIST dataset, assigning one-fold cross-validation this time. Then, a numerical approach to the back propagation is achieved and used to compare with the real gradient. Also, several experiments regarding regularization, activation and network topology are executed to detect some key principles and possibilities of better model performance. For regularization experiments, the model with L1 regularization reports a test accuracy of 0.8644; while that with L2 performs a test accuracy of 0.8769. For the activation experiment, the model with the Sigmoid function reports a test accuracy of about 0.8501; while that of ReLU gives a test performance of 0.8639, and the test accuracy of the model with the Tanh function is presented to be 0.8403. For network topology, halving and doubling the number of hidden units brings a test accuracy of about 0.8496, 0.8769; while changing the number of hidden layers brings a test accuracy of about 0.864. Further analysis and reference behind the experiment will also be included.

1 Introduction

1.1 Background Overview

As the last project discovered, the KMNIST dataset is a Hiragana character set that is part of the Japanese writing system. Accordingly, a train set with matching labels and a test set with matching labels is given as data. Our task at this time is an extension of the previous project, which is executing and training a softmax regression model to classify the KMNIST dataset, i.e. matching images to corresponding labels. With more explorations this time, we look in-depth to do experiments and discuss the possible applications of regularization, activation function, and network topology, accordingly detecting their impact on the model performance.

1.2 Dataset Introduction

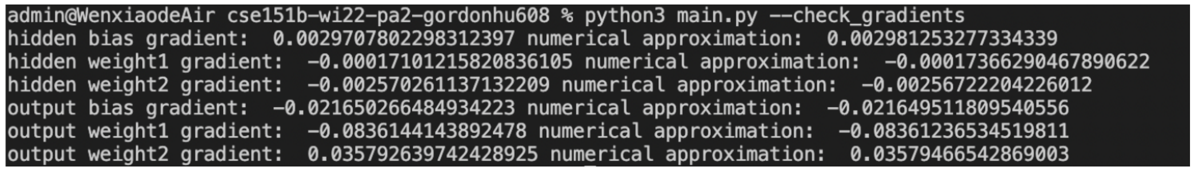
As the last project mentioned, there are 70,000 images (28×28 grayscale) in total, with each of the classes averagely taking 7,000 images. The train set has 60,000 images and the test set takes 10,000. Please refer to the CSE151B PA1 project report from Wenxiao-Wenbo-Huaning's group for further details.

2 Backpropagation

ϵ We choose is $1e-2$.

Category	Gradient	Numerical Approximation
Output bias weight	-0.02165026648	-0.02164951180
Hidden to output weight1	-0.08361441438	-0.08361236534
Hidden to output weight2	0.03579263974	0.03579466542
Hidden bias weight	0.00297078022	0.00298125327
Input to hidden weight 1	-0.00017101215	-0.00017366290
Input to hidden weight 2	-0.00257026113	-0.00256722204

Figure 1: Back Propagation Comparison Summary



```
admin@WenxiaodeAir cse151b-wi22-pa2-gordonhu608 % python3 main.py --check_gradients
hidden bias gradient:  0.0029707802298312397 numerical approximation:  0.002981253277334339
hidden weight1 gradient:  -0.00017101215820836105 numerical approximation:  -0.00017366290467890622
hidden weight2 gradient:  -0.002570261137132209 numerical approximation:  -0.00256722204226012
output bias gradient:  -0.021650266484934223 numerical approximation:  -0.021649511809540556
output weight1 gradient:  -0.0836144143892478 numerical approximation:  -0.08361236534519811
output weight2 gradient:  0.035792639742428925 numerical approximation:  0.03579466542869003
```

Figure 2: Gradient Comparison Output for Backpropagation

Check the backpropagation gradient by using

$$\frac{d}{dw}E^n(w) \approx \frac{E^n(w+\epsilon) - E^n(w-\epsilon)}{2\epsilon}$$

If the difference between the gradient in the network and the numerical approximation is within the boundary of ϵ^2, then it passed the check. If the difference between the gradient in the network and the numerical approximation is within the boundary of ϵ^2, then it passed the check. According to the table, we can see that the difference between the actual gradient and numerical approximation are within 1e-4. So, back propagation is implemented correctly.

3 Gradient Descent

In this part, following the update rule from the written part of CSE151B PA2, a multi-label classifier is trained to map each of the input examples into the labels from 0 to 9. With 1-fold validation here, the model is trained on the train set and related performance is evaluated on the validation set. Stochastic gradient descent is applied here, such that several mini-batches of data are grabbed and trained within each epoch. For each minibatch, related network forward pass and backpropagation are achieved following the forward and backward formula, and weights are accordingly updated following the update formula displayed on previous parts of this assignment, i.e. $w_{ij} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}}$, with a fixed momentum $\gamma = 0.9$ to be involved here. Meanwhile, related statistics, including train loss and training set accuracy, are recorded for further plotting and presenting. After the training on all mini-batches under a single batch is finished, validation loss and validation accuracy are also calculated and recorded to better evaluate the model.

Furthermore, another two strategies are applied here. First, during the training process, we always save the best model based on the performance of the model by comparing the validation loss. That is to say, by checking if the current validation loss is even smaller than the minimum one recorded, we decided if the update (model) in the current epoch needs to be discarded, which to some degree ensure the quality of the model under each epoch as well as the final model we achieve. The other strategy will be early stopping to potentially avoid the case of overfitting. That is, when the validation loss remains larger than the recorded minimum for 5 consecutive epochs (i.e. the weight is not updated for 5 consecutive epochs), the training process will be terminated immediately and the current stored model will be outputted as the best model.

We also choose to stick to the default learning rate 0.005, since it yields the best validation accuracy of 0.9298. While setting the learning rate to be 0.001 will give validation accuracy of 0.9250, and setting the learning rate to be 0.01 will give 0.9276. After all, epochs are done (or early stopped), the training process comes to an end, during which, as figure 3 shows, the training loss changes from 0.8 to 0.05, training accuracy grows from 0.45 to 0.98, validation loss changes from 0.55 to 0.30,

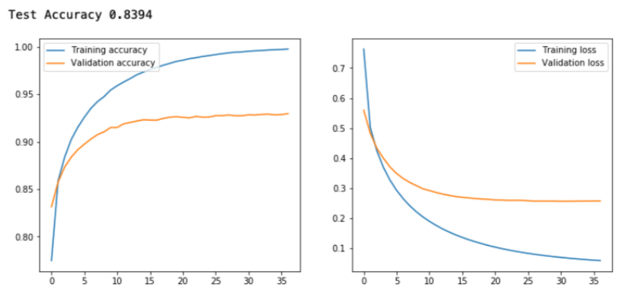


Figure 3: Loss plot and accuracy plot for Gradient Descent Part

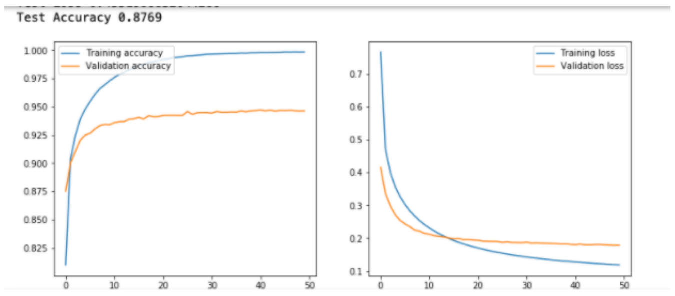


Figure 4: L2 Penalty Term: $1e-3$

and the validation accuracy grows from 0.84 to 0.92. Then the saved model is tested based on the test set – the test accuracy is reported to be 0.8394, with a test loss of 0.52.

4 Experiment with Regularization

When L2 penalty term is 1e-3, it penalized the network the most with weight complexity. It pushes model to generalize better in the validation and test dataset which gives a test accuracy of 0.8769. When L2 penalty term is 1e-6, it penalized weight complexity only slightly. So desipe a very low training loss, it performs badly in validation set, resulting in overfitting, with an according test accuracy of 0.8648. When trying L1 regularization. We pick the same 1e-3 for L1 penalty term. However, it penalized the network really much that the training accuracy is relatively low and 0.8055 for test accuracy. It's underfitting, meaning not learning very well. L1 regularization punishes the network more than L2 since the gradient of L1 regularization is bigger from a mathematical perspective. That's why the validation accuracy in the graph vibrates back and forth without converging smoothly. So, we want to lower is L1 penalty term. When it's 1e-6, the network generalizes better with a test accuracy of 0.8644. Figure 4 5 6 and 7 are presented for visualization towards the four cases under comparison.

5 Experiment with Activations

In this scenario, we only change the type of activation function, with other hyperparameters staying fixed. For other unchanged parameters, we set hidden unit = 128, learning rate = 0.005, batch size = 110, epochs = 100, early stop epoch = 5, gamma = 0.9. For Tanh as the activation function, the training process early stops at epoch 29, with validation accuracy ending up with 0.9232. The testing accuracy here is reported to be 0.8403. When the activation function is Sigmoid, training stops at exactly epoch 20, with a validation accuracy of 0.9386. In this case, the testing accuracy is 0.8501. Then, the training with ReLU stops at epoch 19, with a validation accuracy of 0.9436. And, the testing accuracy is 0.8639. A quick summary for related output is presented on figure 8, 9, 10 and 11. Overall, ReLU has the best performance among the three activation functions. ReLU not only

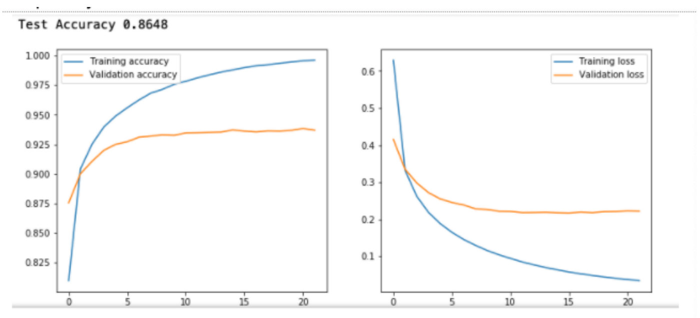

Test Accuracy 0.8648

Figure 5: L2 Penalty Term: $1e-6$

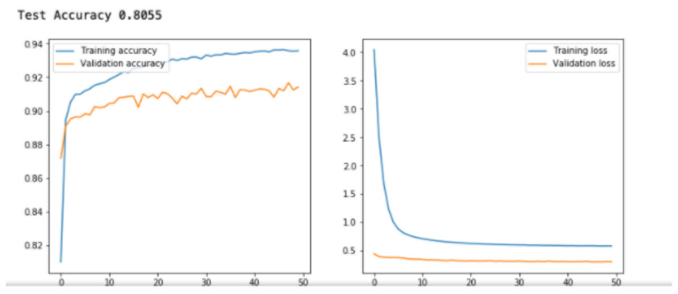

Test Accuracy 0.8055

Figure 6: L1 Penalty Term: $1e-3$

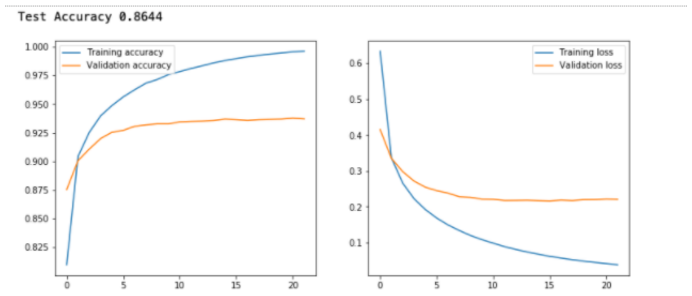

Test Accuracy 0.8644

Figure 7: L1 Penalty Term: $1e-6$

Function Name	Numerical Representation	Stop Epoch	Validation Accuracy	Test Accuracy
Tanh	$f(z) = \tanh(z)$	29	0.9232	0.8403
Sigmoid	$f(z) = \frac{1}{1+e^{-z}}$	20	0.9386	0.8501
ReLU	$f(z) = max(0, z)$	19	0.9436	0.8639

Figure 8: Activation Experiment Comparison

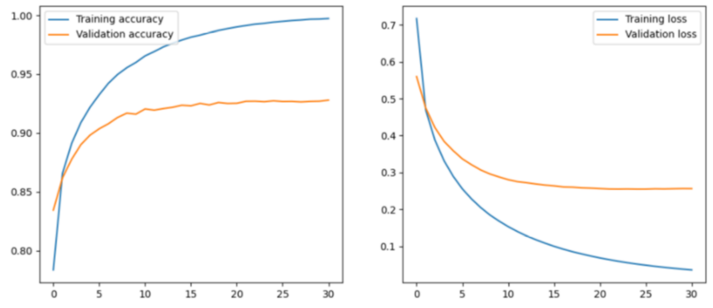


Figure 9: Activation Experiment with Tanh Function

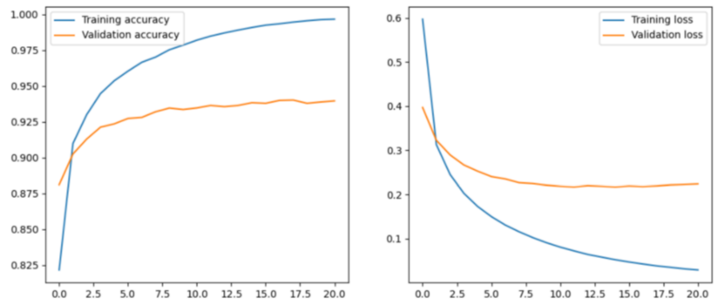


Figure 10: Activation Experiment with Sigmoid Function

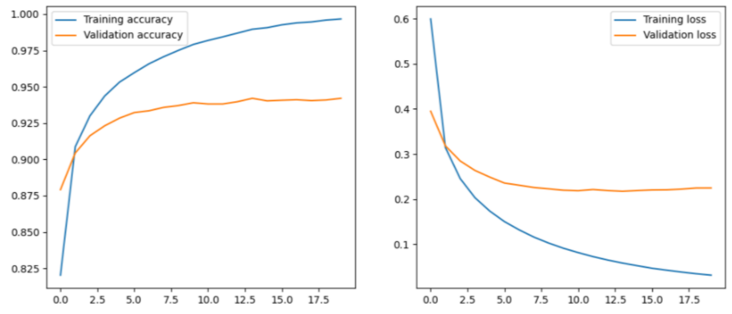


Figure 11: Activation Experiment with ReLU Function

has the best validation accuracy, and testing accuracy, but also reaches this accuracy in relatively smaller epochs, which indicates faster performance.

6 Experiment with Network Topology

Single Layer with different hidden units: In this scenario, we only change the number of hidden units. For other unchanged parameters, we have the activation function fixed to be ReLU, learning rate by 0.005, batch size by 110, epochs by 100, early stop epoch to be 5, and γ to be 0.9. Under the case that there are 128 hidden Units, the training process stops at epoch 19, with validation accuracy 0.9436. Accordingly, the testing accuracy in this case is 0.8639. When the hidden units are set to

Hidden Units	Stop Epoch	Validation Accuracy	Test Accuracy
128	19	0.9436	0.8639
64	18	0.9323	0.8496
256	20	0.9482	0.8769

Figure 12: Network Topology Experiment Comparison

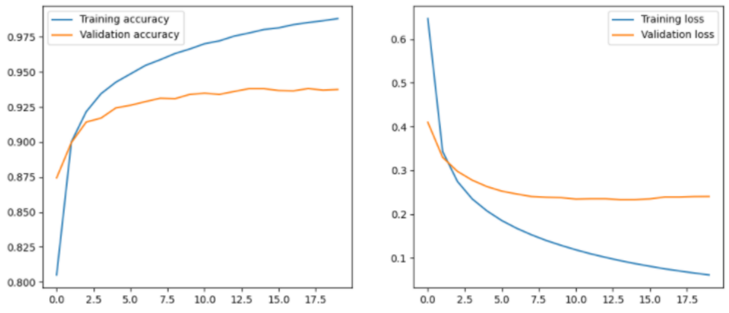


Figure 13: Network Topology Experiment with 64 hidden units

be 64, the model training stops at epoch 18, with a validation accuracy ending up with 0.9323. The testing accuracy here turns out to be 0.8496. Then, setting the hidden units to be 256, the training process stops at epoch 20, with an ending validation accuracy of 0.9482. For this setting, the testing accuracy is 0.8769. Related statistics is presented on 12, and the three related figures are also shown for 13, 14 and 15.

A quick summary for this single experiment is present on corresponding figures. When the number of hidden units is 256, the model achieves the best performance. The reason behind more hidden units leading to better accuracy is likely because Japanese Characters share similarities in nature. Thus, we need more principal components to distinguish between different characters. More hidden units might be able to grasp more principal components, thereby giving better accuracy eventually.

Double hidden layer, with related visualization presented on 16 with 128 Hidden Units for each, which means the total number of hidden units remain the same. The number of parameters will also roughly be the same. The model stops at epoch 14, with a validation accuracy of 0.9414. And, the testing accuracy is 0.864.
So, even though this is slightly better that a single hidden layer with 128 units, it is worse than a single hidden layer with 256 units. The possible reason for this phenomenon is the model being too complicated. And, it converges faster (with only 14 epochs), so it may come to overfit the training data. Thus, it cannot perform the best accuracy for the testing data.

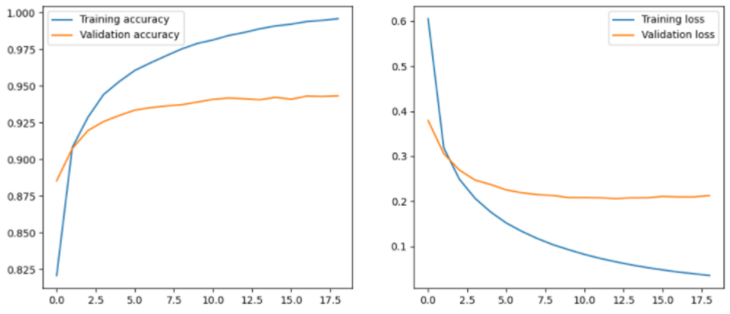


Figure 14: Network Topology Experiment with 128 hidden units

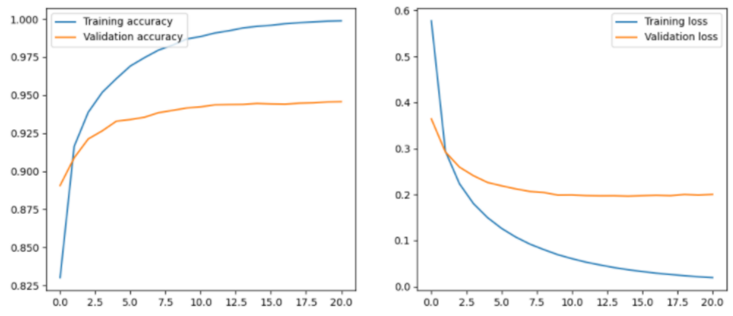


Figure 15: Network Topology Experiment with 256 hidden units

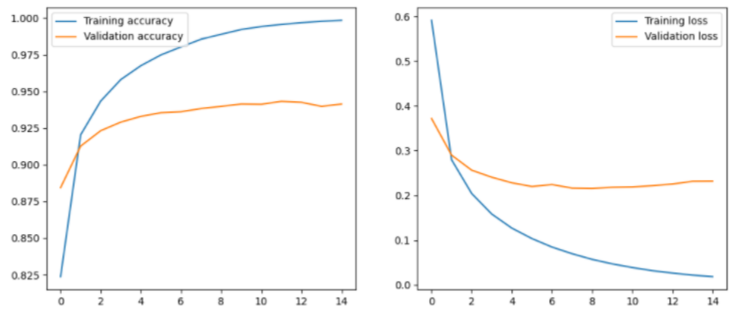
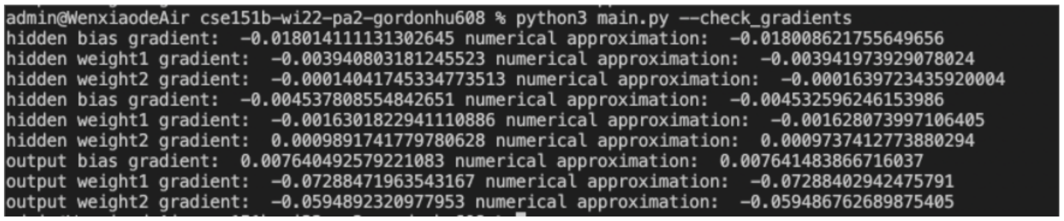


Figure 16: Network Topology Experiment with 2 layers with 128 hidden units

In case the backpropagation is not right, we checked them by 2b and they all passed the check, whose table is accordingly presented in 17.

7 Conclusion

In this project, utilizing stochastic gradient descent as well as several techniques including early stopping and k-fold validation (1-fold here), we implement a neural network for the KMNIST dataset with softmax output. Besides, a numerical approximation for back propagation is achieved and its comparison with the real gradient is accordingly presented and analyzed, which infers a relatively small difference. Then, three experiments regarding regularization methodology, activation function and network topology are executed; related plots and output, including test performance and validation performance, are reported for further comparison and exploration. In all cases, the test performance (accuracy) flows over 80%, and the reasons between the differences and varies between these accuracies will be meaningful for in-depth reasonings.



```
admin@WenxiaodeAir cse151b-wi22-pa2-gordonhu608 % python3 main.py --check_gradients
hidden bias gradient:  -0.018014111131302645 numerical approximation:  -0.018008621755649656
hidden weight1 gradient:  -0.003940803181245523 numerical approximation:  -0.003941973929078024
hidden weight2 gradient:  -0.00014041745334773513 numerical approximation:  -0.0001639723435920004
hidden bias gradient:  -0.004537808554842651 numerical approximation:  -0.004532596246153986
hidden weight1 gradient:  -0.0016301822941110886 numerical approximation:  -0.001628073997106405
hidden weight2 gradient:  0.0009891741779780628 numerical approximation:  0.0009737412773880294
output bias gradient:  0.007640492579221083 numerical approximation:  0.007641483866716037
output weight1 gradient:  -0.07288471963543167 numerical approximation:  -0.07288402942475791
output weight2 gradient:  -0.0594892320977953 numerical approximation:  -0.059486762689875405
```

Figure 17: Backprpagation Double Check

8 Team Contributions

Wenbo Hu: Implementation of multilayer perceptron, training network. Experimented checking network gradients by numerical approximation and experiment with L2 and L1 regularization.
Wenxiao Li: Implemented helper functions to calculate the loss functions and gradients of them. Generate minibatches, and used SGD on these minibatches to update weights. Connected the network so data was correctly handled in different layers. Experimented on the best learning rate, activation functions, and network topology, and wrote reports for these experiments sections.
Huaning Liu: Experimented and plotted on activation experiment and network topology experiment; completed text part for title, abstract, introduction, dataset, activation experiment, network topology experiment and conclusion; organize, format and wrap up latex report.