Classification on MNIST dataset with Neural Networks

UCSD CSE151/251B/ WI24 PA1
Ziqi Shang zishang@ucsd.edu CSE251B
Lian Gan lgan@ucsd.edu CSE151B

January 26, 2024

Abstract

In this report, we are going to use Neural Networks to do classification on MNIST dataset. We will try different experiments on networks to see if we get different accuracy on the test set, such as number of layers, learning rate, batch size, numerical approximation of gradients, momentum method, regularization and activation functions. The best accuracy we get on the test set is 0.9404, with mini-batch stochastic gradient descent (SGD) approach, 128 batch size, 3 layers and no regularization.

1 Data Loading

In our study, we utilized the MNIST database, a large collection of handwritten digits comprising a training set of 60,000 examples and a test set of 10,000 examples. This database is derived from a larger NIST Special Database, featuring digits penned by U.S. Census Bureau employees and high school students. The digits have been size-normalized and centered in a fixed-size image. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. For each example, there are 784 features; and there are ten classes with respect to digit 0-9. We randomly shuffle the train set and then use the first 48,000 examples as training set and last 12,000 examples as validation set. For each example image, we normalize the features by z-scoring it, which is subtract each pixel value by average pixel value and then divided by standard deviation over the single image. For any one train image, the mean is 0 and std value is 1. In the initial stages of our preprocessing, we also implemented one-hot encoding for the labels using a basic iterative approach. This involved creating a zero matrix of dimensions N (number of examples) by 10 (number of classes) and setting the appropriate index to 1 for each label.

2 Softmax Regression

In this implementation, we configure the neural network with an input layer consisting of 784 units, reflecting the dimensionality of our input data, and an output layer comprising 10 units, corresponding to the number of classes in our multi-class classification task. To optimize our model, we employ a mini-batch stochastic gradient descent (SGD) approach, with 128 batch size. The softmax activation function is utilized to compute the probabilities of each class at the output layer, while the cross-entropy loss function is adopted to quantify the model's prediction error.

For model training, we set the process to iterate over 100 epochs, incorporating an early stopping mechanism to prevent overfitting, which halts training if no improvement is observed for a minimum duration of 3 epochs. We select a learning rate of 0.001 to balance the speed and stability of the convergence. Through this setup, our model achieves a peak accuracy of 92.04 on the training set and 93.04 on the validation set. When evaluated on the test set, the model demonstrates a commendable accuracy of 92.45, indicating its robustness and generalization capability.

Figure 1: train/ validation loss and train/validation accuracy

3 Numerical Approximation of Gradients

We choose $\epsilon = 10^{(} - 2)$ for this experiment.

Table 1: Comparison of Gradients

Type of Weight	Gradient (Numerical Approx.)	Gradient (Backpropagation)	Absolute Difference
Output Bias	1.0018240106	1.0018119968	0.0000120139
Hidden Bias	-0.0173982719	-0.0175006099	0.0001023380
Hidden to Output 1	-0.0280635019	-0.0280634954	0.0000000065
Hidden to Output 2	0.0267335519	0.0267335499	0.0000000020
Input to Hidden 1	0.0030368552	0.0030714313	0.0000345760
Input to Hidden 2	0.0059757034	0.0059714871	0.0000042163

4 Momentum Experiments

In this implementation, we are still doing mini batch SGD with 128 batch size and cross entropy loss function. However, here we add tanh non-linear activation function and two hidden layers, one with 258 units and one with 128 units. With learning rate 0.02, we get 0.9361 accuracy on the train set and 0.9521 accuracy. The accuracy on the test set is 0.9404. After adding hidden layers, the network becomes more complex and the accuracy increases. However, our model slightly overfits, since accuracy on the test is 0.012 lower than the validation set.

Figure 2: train/ validation loss and train/validation accuracy for momentum experiment

5 Regularization Experiments

In this implementation, we are still doing mini batch SGD with 128 batch size , cross entropy loss function, tanh activation function, one hidden layer with 128 units. However, we are not using the momentum method and we enabled L1 regularization. We choose $\lambda = 0.0001$ for both L1 and L2 regularization. Both regularizations apply a penalty to the complexity of the model, so the accuracy on the test set decreases compared to the previous network. Two regularizations have the similar accuracy on test test, but L1 is slightly better than L2.

Figure 3: L1 train/ validation loss and train/validation accuracy

Figure 4: L2 train/ validation loss and train/validation accuracy

6 Activation Experiments

In this implementation, we are still doing mini batch SGD with 128 batch size , cross entropy loss function, one hidden layer with 128 units. Here, we disabled regularization and momentum. And we will try Sigmoid and ReLU activation functions in this step.

Sigmoid: Accuracy is 0.9118 on the train set, 0.9190 on the validation set and 0.918 on the test set.

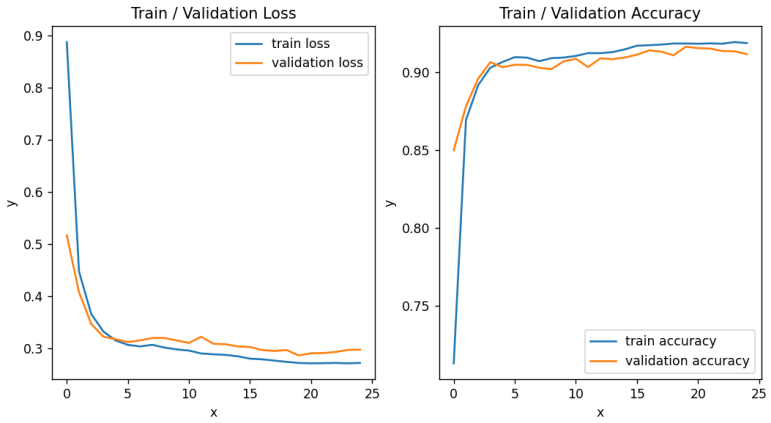

Figure 5: Enter Caption

ReLU: Accuracy is 0.7828 on the train set, 0.7609 on the validation set and 0.7769 on the test set.

Figure 6: Enter Caption

7 Team contributions

Ziqi Shang mainly works on neuralnet.py, main.py, train.py, hypertuning and write half reports. Lian Gan mainly works on gradient.py and util.py, debug and write half reports.