CSE 154 Deep Learning: PA 2 Part II

Classification on MNIST Dataset

Xinyi He
A13561164
Rinka Yoshida
A13491595

October 27, 2019

Abstract

Programming assignment 2 implements multi-layer neural networks via mini-batch stochastic gradient descent with softmax output activation function. We apply the classification on MNIST dataset to identify the handwritten digits(0-9). We alter configurations such as the learning rate, the momentum, the activation function for the hidden units, the regularization parameter, and the network topology to achieve best performance. Section 3 includes our discovery that for identifying hand written digits in MNIST dataset, using 0.9 as momentum_gamma results in around 90% test accuracy. Section 4 includes our discovery that using L2 regularization increases the test accuracy to around 93%. In section 5, we experimented with different activation function and found out that the test accuracy is the highest with ReLU activation function. In section 6, we found out the neural network topology affects its performance in different ways.

1 Load MNIST data

Please refer to function load_data for implementation.

2 Implement backpropagation

Please refer Neuralnetwork backward_pass for implementation.
Take $\epsilon = 10^{-2}$.

	gradient	numerical approximation
output bias	-0.0711812453645377	-0.0711808917236656
hidden bias	0.0007861338703251932	0.0007861342113280045
hidden to output 1	0.3005404328615423	0.30054034620183856
hidden to output 2	-0.006744991367732083	-0.006744985833240236
input to hidden 1	0.0012747609874263369	0.001274761493830212
input to hidden 2	-0.0007886136649042693	-0.0007886140057866697

	difference
output bias	3.5364087209999173e-07
hidden bias	3.410028113831273e-10
hidden to output 1	8.665970374632792e-08
hidden to output 2	5.534491847453138e-09
input to hidden 1	5.064038752135985e-10
input to hidden 2	3.4088240043195217e-10

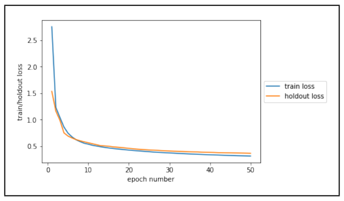
Therefore, it agrees that weight after backpropagation is within ($O(\epsilon^2)$) of the gradient obtained by numerical approximation.

3 Learn a classifier for MNIST

For each epoch, we randomize the data, separate the data into mini batches of equal size, and pass each batch through (forward_pass) the neural network. Then we backpropagate (backward_pass) through the neural network and update weights at each layer using learning rate and momentum. Once all the mini-batches have gone through the network, we do validation test and use the calculated validation loss to prevent over-fitting (early stop the model training). We early stop the training when there is threshold number of consecutive increase in validation loss. We repeat this procedure for each epoch and train the neural network.

We experimented with different learning rate using different network topology and tanh activation function, and found out that $lr = 0.001$ consistently results in the best test accuracy.

The below report is done on the following configuration: $[784, 50, 10]$ as the network topology (one hidden layer of 50 units), tanh as the activation_function, 0.001 as the learning_rate, 0.9 as the momentum_gamma, 50 as the number of epochs, 1000 as the batch size, and 5 as the early stopping threshold.



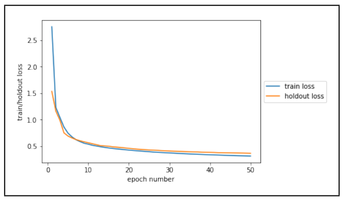


Figure 1: Train and validation loss vs. number of epochs on the left. Train and validation accuracy(%) vs. number of epochs on the right.

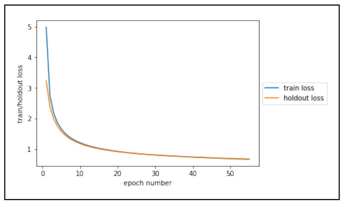
$$test\_accuracy \quad = \quad 89.49\%$$

We can see that the neural network learns the MNIST classification relatively fast, reaching around 90% for test accuracy using the best weights found.

4 Experiment with regularization

The below experiments are done on the following configuration: [784,50,10] as the network topology (one hidden layer of 50 units), 0.0001 as the learning_rate, 0.9 as the momentum_gamma, 55 as the number of epochs (10% more than 50 epochs used in section 3), 1000 as the batch size, and 5 as the early stopping threshold.

Without regularization:



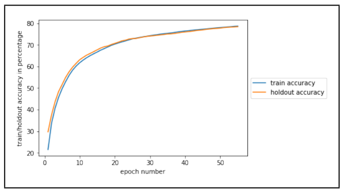


Figure 2: Train and validation loss vs. number of epochs on the left. Train and validation accuracy(%) vs. number of epochs on the right.

$$test\_accuracy \quad = \quad 78.58\%$$

Take regularization $\lambda = 0.001$.

$$test\_accuracy \quad = \quad 93.01\%$$

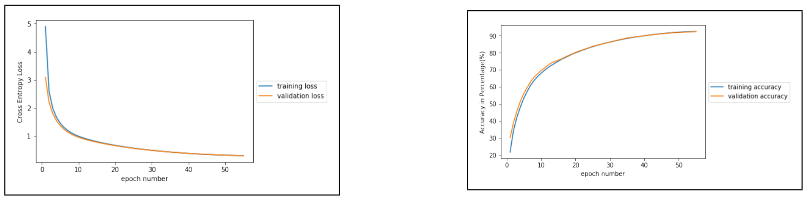


Figure 3: Train and validation loss vs. number of epochs on the left. Train and validation accuracy(%) vs. number of epochs on the right.

Take regularization $\lambda = 0.0001$.

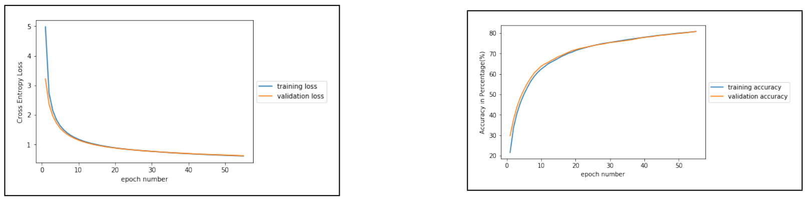


Figure 4: Train and validation loss vs. number of epochs on the left. Train and validation accuracy(%) vs. number of epochs on the right.

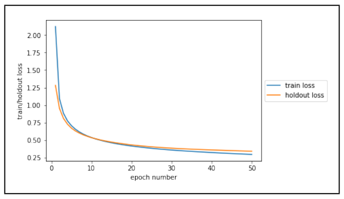
$$test\_accuracy \quad = \quad 80.46\%$$

Experiment with regularization gives us a better performance with a over 90% test accuracy when regularization parameter is 0.001. However, when regularization parameter is 0.0001, the test accuracy declines to around 80%. The reason is that regularization discourages the complexity of the model and helps avoid overfitting which results in higher test accuracy. But when the regularization parameter is too small it makes little difference.

5 Experiment with activation functions

The below experiments are done on the following configuration: $[784, 50, 10]$ as the network topology (one hidden layer of 50 units), 0.001 as the learning_rate, 0.9 as the momentum_gamma, 50 as the number of epochs, 1000 as the batch size, and 5 as the early stopping threshold.

5.1 Using sigmoid activation function



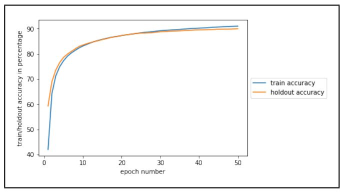


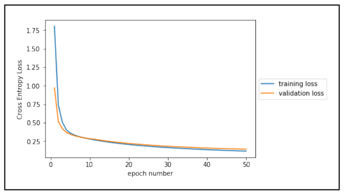
Figure 5: Train and validation loss vs. number of epochs on the left. Train and validation accuracy(%) vs. number of epochs on the right.

$$test\_accuracy \quad = \quad 90.36\%$$

When we use sigmoid function as the activation function at each layer, the test accuracy increases and the validation is smoothly continuously increases. The performance is similar to tanh activation function as tanh is a scaled sigmoid function.

5.2 Using ReLU activation function

We scale the initial weights by a factor of $1/(fan_{in} + fan_{out})$ for a given node and scale the learning rate by a factor of 10 (hyper-parameter) in order to resolve the gradient explosion problem.



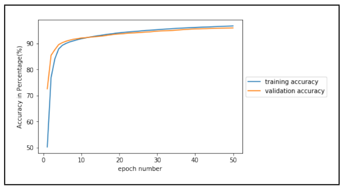


Figure 6: Train and validation loss vs. number of epochs on the left. Train and validation accuracy(%) vs. number of epochs on the right.

$$test\_accuracy \quad = \quad 96.22\%$$

When we use ReLU activation function, it gives a better performance with a faster convergence and a higher test accuracy over 90%. The reason is that ReLU activation function reduces the likelihood of gradient vanish and results in faster learning.

6 Experiment with network topology

The below experiments are done on the following configuration: tanh as the activation_function, 0.001 as the learning_rate, 0.9 as the momentum_gamma, 50 as the number of epochs, 1000 as the batch size, and 5 as the early stopping threshold.

6.1 Changing the number of hidden units

Halving the number of hidden units $[784 - 25 - 10]$:

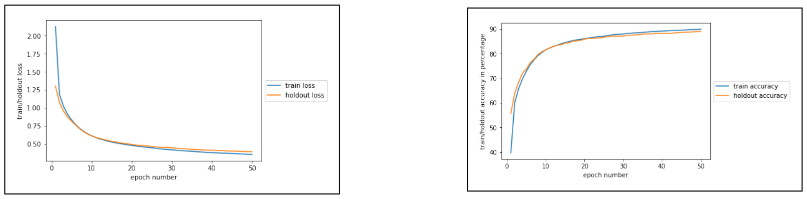


Figure 7: One hidden layer with 25 hidden units.

$$test\_accuracy \quad = \quad 89.45\%$$

Doubling the number of hidden units $[784 - 100 - 10]$:

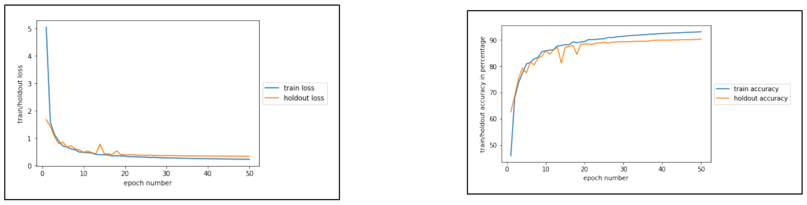
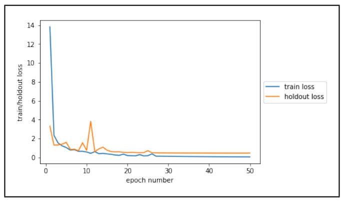


Figure 8: One hidden layer with 100 hidden units.

$$test\_accuracy \quad = \quad 90.65\%$$

Even more number of hidden units $[784 - 350 - 10]$:



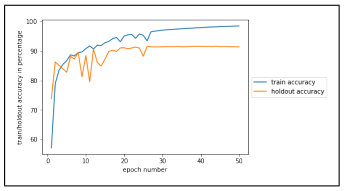


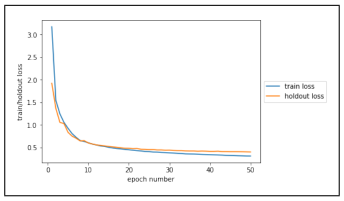
Figure 9: One hidden layer with 350 hidden units.

$$test\_accuracy \quad = \quad 91.39\%$$

When there is one hidden layer, the test accuracy increases slightly as the number of hidden units increases. We believe the cause of this increase in performance is that more features/representations of MNIST dataset are learnt by the neural network. When the number of hidden units is 350, which is around the mean of input and output units, it resulted in the best test accuracy and train accuracy. We suspect the test accuracy does not increase dramatically because MNIST is a relatively simple dataset. For single hidden layer neural network, the zigzag pattern manifests when there is more neurons in that layer. We suspect it is because they learn the features/representations of the dataset, but it also learns the noise at the same time.

6.2 Changing the number of hidden layers

Two hidden layers $[784 - 45 - 45 - 10]$:



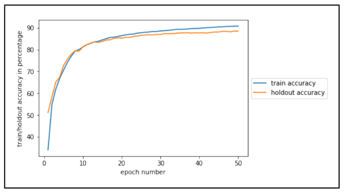


Figure 10: Two hidden layers each with 45 hidden units in each.

$$test\_accuracy \quad = \quad 89.19\%$$

The performance does not improve because MNIST dataset is a easy enough problem where one hidden layer should suffice. All four data—train loss, validation loss, train accuracy, and validation accuracy—reaches around the same value at the end of training, aka. epoch 50.

7 Individual contributions to the project

We collectively implemented backpropagation and the neural network model training function.

Xinyi He wrote the report for abstract, section 3 and 4.

Rinka Yoshida wrote the report for section 2, 5, and 6.