CSE253 Programming Assignment2

Chao Yu
UCSD ECE
PID: A99049546
chy018@eng.ucsd.edu

Yau Mo Chan
UCSD CSE
AID: A53067459
ymchan@eng.ucsd.edu

Abstract

In assignment 2 we applied one hidden layer neural network to classify handwritten digits from Yann LeCun's website for the MNIST Database. We adjusted the initialization by making the input data from -1 and 1.In addition, we applied gradient check to prove our algorithm is right.We also add the "Tricks of the Trade." We shuffled the data in mini-batch, initialized weights using a distribution with 0 mean and standard deviation 1/sqrt(fan-in) and use momentum. We also halving and doubling the number of hidden units and construct a 2 hidden layer neural network. In the end, we achieved 99.112 percent accuracy on Training set, 97.590 in Testing set and 97.520 on validation set. using two hidden layer neural network.

1 Introduction

Neural Network is a very powerful Machine Learning method in classification and regression problem. In the assignment we use neural network to do the classification problem. In the training process, we apply mini-batch to accelerate the training time and use regularization, early stop to prevent over fitting. We apply different initializations and momentum in the training process and compare the results of different nodes on hidden layers. At last, we apply a two hidden layer model and test the performance of the model using ReLU activation function.

2 Classification

2.1 Data Pre-Processing

We split the training set of 60000 images to two subsets: one training set with 50000 training images and one validation set with 10000 images. The pixel values in the image are all in the range 0 to 255. We divide the pixel by 127.5 and subtract 1 to make all the values in the range -1 to 1. We added bias into the pixel image matrix by prepending 1 into the beginning.

2.2 Building One Hidden Layer Neural Network

For the hidden layer, we choose the hidden layer size to be 64 at the beginning. We use sigmoid function as activation function in the hidden layer to make the output of hidden layer in the range 0 to 1.

2.3 Gradient Check

The reason to do gradient check is because it is very easy to make mistakes in back-propagation. Doing gradient check can make sure we do not have bugs in back-propagation calculation.
We random pick a small set of data and apply the following equation below with $\epsilon = 0.01$. If the

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

differences between the slope calculated by the following equation and using back propagation is less than 10e-4, then the result is reasonable.

$$\frac{\partial E^n}{\partial w_{ij}} \approx \frac{E^n\left(w_{ij}+\epsilon\right)-E^n\left(w_{ij}-\epsilon\right)}{2\epsilon} \tag{1}$$

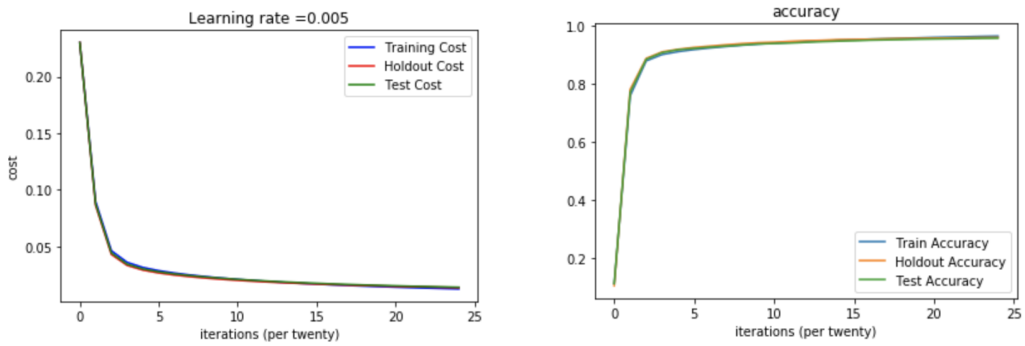
We input the weights plus ϵ and ϵ minus ϵ and calculate the slope of the weights. Then we compare the slope to the correspond gradient we calculate using back-propagation.
At first we tried changing input bias and random pick one Training sample for 5 times, the difference were 4.113e-07,1.4446e-5,2.3114e-6,3.1211e-5,-6.1222e-5. which were less than or in range 10e-4. Then we tried change output bias, the difference were 1.024e-4, -3.0306e-5,-2.0146e-6,9.4146e-6. We changed the weights of input to hidden, the difference were 3.1451e-5, 7.8699e-5, 1.4565e-4, -7.508e-5,-8.60961e-5. Then we did the same calculation on output weight, the difference were -1.9716e-5,-7.8320e-6,1.9716e-5,-7.8320e-5, -4.12336e-7. We concluded that our implementation of backward propagation was correct.

2.4 Training

We initialize the weights to be mean at 0 and stand deviation be 0.001 for both w1 and w2 and implement mini-batch gradient descent in the assignment with mini batch size equals to 128. w1 represents the weights of input to hidden layer and w2 represents the weights of hidden layer to output. We use 0.005 as learning rate. We apply easy stop to decide when to stop the training process. If the validation set error continues to increase for 3 epochs, then we break the training process. We also apply L2 regularization with λ to be 0.0001 to prevent over fitting. Since we will not be able to see the test set in real world, so the weights we choose will be the weights to achieve highest validation set accuracy.

2.5 Result

Figure 1(a) shows the cost and Figure 1(b) shows the accuracy of the one layer neural network. The training accuracy is 95.436 percent, the validation accuracy is 95.18 percent and the test accuracy is 94.921 percent. The accuracy of the test and validation set are very close to Training set. One problem is the distribution of the training set might be different from the validation set, then Our next step is to shuffle the data.
The converge speed is very low, the cost is not very smooth and it takes 480 epochs to converge. In Figure 1. We think it is because the bad initialization. We will use better initialization in section 3.



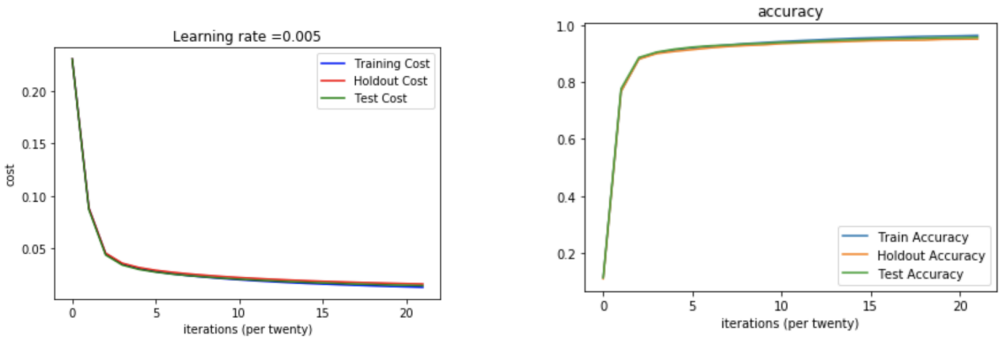
(a) *costerror* (b) *accuracy*

Figure 1: No Shuffle, No Fan-in initialization, No momentum

3 Adding the "Tricks of the Trade"

3.1 shuffle training data

We shuffle the total training data first and then pick top 50000 as training set, the left 10000 as validation set, to make sure training set is the same distribution as the validation set. Then we run our mini-batch gradient descent on our model with batch size 128. For each epoch, we reshuffle the training set after one epoch. In Figure 2, the accuracy for training set is 96.782 percent, validation set is 95.84 percent and test set is 96.12 percent. The use of shuffle data is to make the distribution of training set is the same as validation set. It takes 420 epochs to converge with the learning rate 0.005. The shuffle of data does not boost the accuracy of the model very much compared to the non-shuffle model. The training speed does not increase much as well.



(a) *costerror* (b) *accuracy*

Figure 2: Shuffle, No Fan-in initialization, No momentum

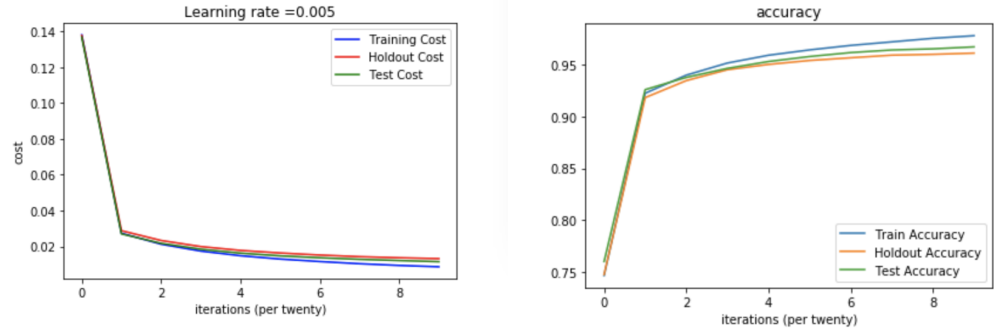
3.2 tanh

We used tanh function as activation function instead of sigmoid function.

$$tanh = 2\sigma(2x) - 1 \tag{2}$$

Because the slope of sigmoid function is $\sigma(x)^{'} = \sigma(x) * (1 - \sigma(x))$, we can apply chain rule to the function to get slope of tanh function. Then the slope will be:

$$\begin{aligned} tanh'(x) &= 2\sigma(2x) * (1 - \sigma(2x)) * 2 \\ &= 4\sigma(2x) * (1 - \sigma(2x)) \end{aligned} \tag{3}$$



(a) *costerror* (b) *accuracy*

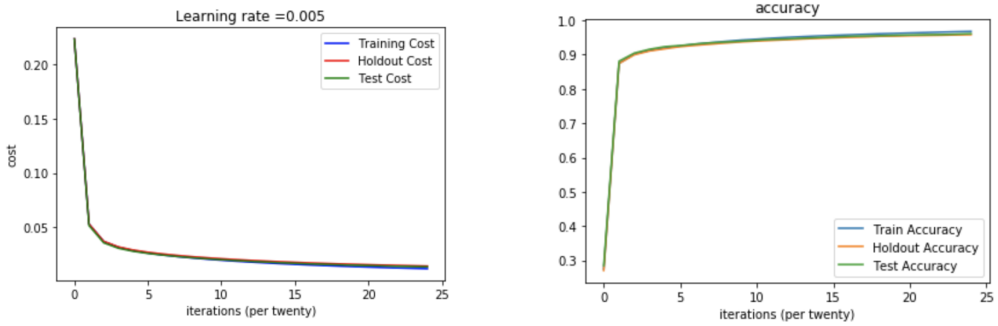
Figure 3: Shuffle, tanh

The results of using tanh as activation function work well. We got 97.786 percent in training set, 96.11 in validation set and 96.71 in test set. The accuracy increase about one percent compared to

using sigmoid function. In addition, using tanh boosting the training speed a lot. It only takes 180 epochs to finish training with learning rate = 0.005. We think it is because the range of tanh function is from -1 to 1 so that it is centered at 0, which is good for speeding up the training process.

3.3 Initialize the input weights

We Initialize the input weights by using a distribution of 0 mean and 1/sqrt(fan-in) standard deviation where fan-in is the training size. The accuracy of training set is 96.04 percent, validation set is 95.82 percent, test set is 95.31 percent in Figure 4(b). The result does not getting much better than Figure 2. However, when we compare the image in Figure 2(a) and Figure 3(a), we find the cost of function drops faster when we initialize the input weights using 0 mean and 1/sqrt(fan-in) standard deviation. The good initialization will increase the learning speed of the model, it takes 10 fewer epoch for the cost to converge than our original initialization.



(a) $costerror$ (b) $accuracy$

Figure 4: Shuffle, Fan-in initialization, No momentum

3.4 Momentum

Momentum update is another approach that almost always enjoys better converge rates on deep networks. we use moment to update the parameters using the equation:

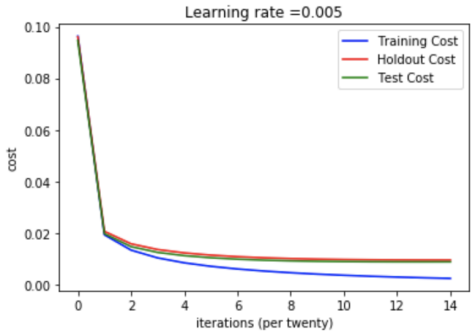
$$\begin{aligned} v &= \alpha * v - learningrate * dw \\ w &= w + v \end{aligned} \tag{4}$$

We use α = 0.9. From Figure 5, the training accuracy is 99.69 percent, validation accuracy is 97.17 percent and test accuracy is 97.36 percent. The performance of the model increase a lot compared to use simple gradient to update parameters. The learning speed is also faster than gradient descent. The cost drops very quickly at the beginning 20 epochs and become very smooth afterwards. We use the learning rate 0.005 and it takes 280 epochs to converge.

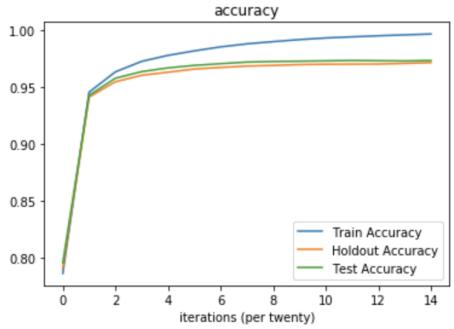
4 Experiment

4.1 halve and double hidden units

We changed the number of hidden units to 32 in Figure 6 and 128 in Figure 7. For 32 units layer model, the accuracy is 98.508 percent for training set, 96.14 percent for validation set and 96.41 percent for test set. For 128 units layer model, the accuracy is 99.968 percent for training set, 97.543 percent for validation set and 97.652 percent for test set. We compare the validation set accuracy between two models and find out the model with 128 units hidden layer has higher accuracy but it takes more time to train. With the learning rate equals to 0.005, it takes 200 epochs to train a 32 units layer model, 340 epochs to train a 128 units layer model. The result is the same as we expect. Because usually the more hidden layers the network have, the better performance the model will have when the model is not over fitting. Since the model has more units, it takes more time to train each unit.

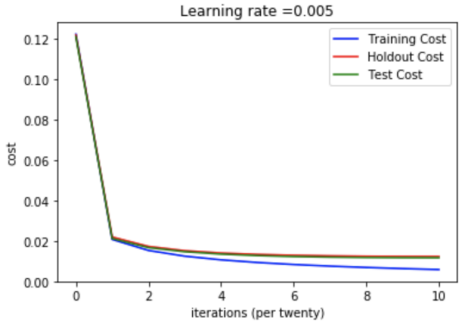


(a) *costerror*

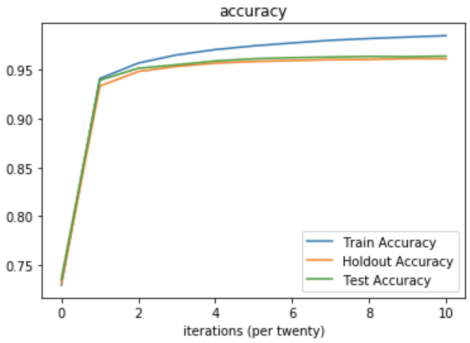


(b) *accuracy*

Figure 5: Shuffle, Fan-in initialization, momentum

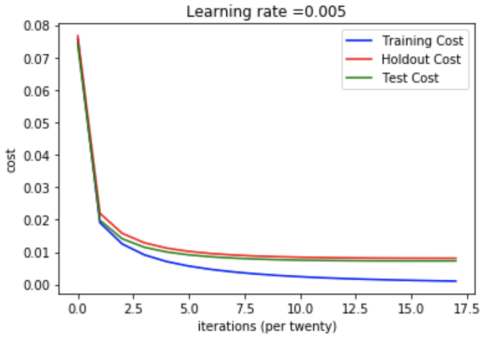


(a) *costerror*

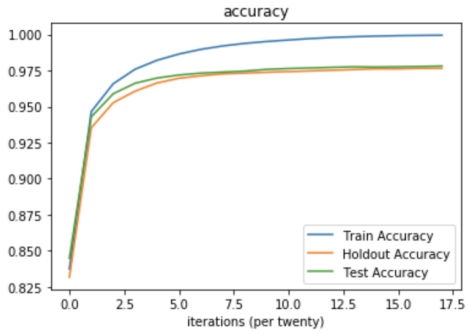


(b) *accuracy*

Figure 6: 32 units of hidden layer



(a) *costerror*

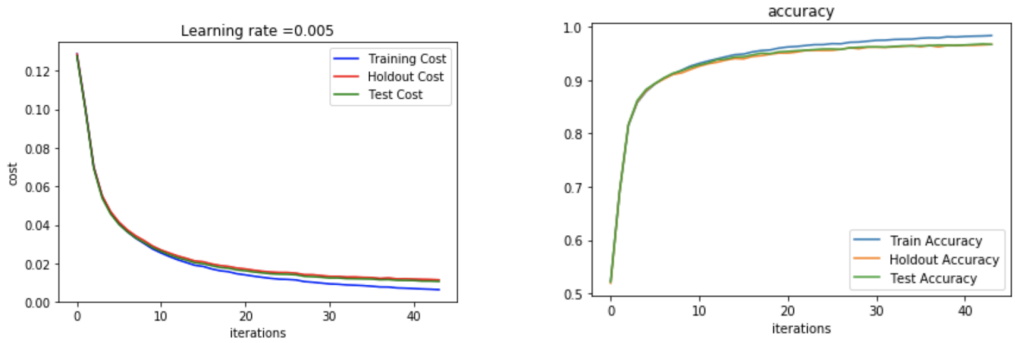


(b) *accuracy*

Figure 7: 128 units of hidden layer

4.2 2 hidden layer neural network

We change the structure of one hidden layer to two hidden layer and set units for both hidden layer to be 64 units. For each layer, we add bias. In Figure 8, the accuracy is 99.112 percent for training set, 97.390 for validation set and 97.520 for test set. Because there are two hidden layers in the model, so the model is very easy to overfit. We change the early stop factor to 2, when the cost of training model continue to increase for two epochs, we stop the training process. The training process is very quick, it only takes 38 epochs with learning rate 0.005 to finish training process. The accuracy is slightly higher than single layer neural network. Overall the performance of two hidden layer neural network has high accuracy and fast training speed.



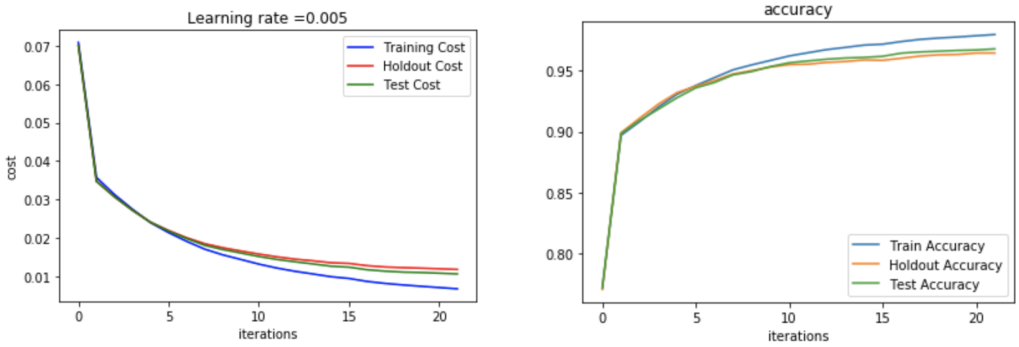
(a) $costerror$ (b) $accuracy$

Figure 8: 2 hidden layer

4.3 ReLU

We apply ReLU as the activation function instead of using sigmoid function. In Figure 9, the accuracy of training data is 98.598 percent, validation set is 97.561 percent and test set 97.554 percent. We compare the result of validation set between using ReLU as activation function and using sigmoid function, we find out the model using ReLU has higher accuracy. The training process is also very fast with 0.005 learning rate, it takes 22 epochs to finish the training. Therefore using ReLU will also decrease the training time compared to using sigmoid function.

$$f(x) = max(0, x) \tag{5}$$



(a) $costerror$ (b) $accuracy$

Figure 9: 2 hidden layer using Relu

4.4 Discussion

The sigmoid function will make the output in the range 0 to 1. When the neuron's activation saturates at either tail of 0 or 1, the gradient will become very close to zero. When the local gradient is very small, almost no signal will flow through the neuron to its weights and recursively to its data. Then it has a huge negative influence in the training process.For ReLU activation function, which is zero when $x < 0$ and then linear with slope 1 when $x > 0$. The slope is 1 which is much bigger than the slope in sigmoid function, so it will accelerate the training process.

5 Conclusion

We started with a single hidden layer of neural network with sigmoid function at the hidden layer activation and cross entropy function with regularization at the output layer. We applied several methods in "Tricks of the Trade" and observed the performance difference. We found that shuffling and initializating better initial weights, with 0 mean and 1/sqrt(fan-in) standard deviation, did not improve much on the accuracy. However, the better initial weights resulted in faster learning rate as expected. We found that momentum updates gave much improvement in both accuracy and convergence speed. It increased accuracy by around 4% and fasten the learning rate almost by half.

We then compared the performance difference by modifying the neural network structure, including using more and fewer hidden units and adding 1 more hidden layer. More hidden units gave better accuracy but longer training time. More hidden layers resulted in both higher accuracy and convergence speed, but it could easily overfit so early stop factor has to be applied carefully.

Lastly, we compared the performance with using ReLU as the activation function instead of sigmoid. We found that the accuracy was significantly higher and the convergence was very fast.

In conclusion, the studies of different methods and structures of the neural network were significant in understanding how a neural network could be designed with performance enhancement in practice, with both accuracies and learning time in consideration.

6 Contribution

In Assignment 2, Chao calculated the backward propagation and constructed 1 hidden layer and 2 hidden layer neural network. Yau did gradient check and graph analysis.