Graded Homework 1 Machine Learning

Qiaowei Li qxl161930

The parameters can be changed include: the standard deviation of initial weights. The number of nodes in the hidden layer and the number of layers. Learning rate and weight update steps.

First of all, adjusting parameter individually and independently to find which parameter has the most powerful influence on the accuracy. We have the following tables:

Wt Ud Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
750	1024	0.1	0.5	0.857
1500	1024	0.1	0.5	0.879
3000	1024	0.1	0.5	0.893
6000	1024	0.1	0.5	0.906
12000	1024	0.1	0.5	0.915
24000	1024	0.1	0.5	0.924

If just adjust the weight update steps, we will see the trend is always going up. But the process should be within 10 minutes. Thus the weight update steps should be from 1500 to 12000. (This is an initial range, all of the parameter range can be modified if there exist a better one.)

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
1500	1024	0.1	0.5	0.879
1500	512	0.1	0.5	0.867
1500	2048	0.1	0.5	0.886
1500	4096	0.1	0.5	0.733
1500	8192	0.1	0.5	0.848
1500	256	0.1	0.5	0.848

If just adjust number of nodes, we will see it is not the case that the more the better. The range should between 256 to 2048.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
1500	1024	0.1	0.5	0.879
1500	1024	0.05	0.5	0.099
1500	1024	0.2	0.5	0.887
1500	1024	0.4	0.5	0.545
1500	1024	0.3	0.5	0.652
1500	1024	0.25	0.5	0.842

If just adjust the sd of initial weights, we can see the range should be from 0.1 to 0.25. And during the training the process, the bigger sd may cause the results varying heavily. The 1500 weight update steps seem to limit the training because the accuracy is all the way up and still has a lot potential when the training is done.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
1500	1024	0.1	0.5	0.879
1500	1024	0.1	0.25	0.862

1500	1024	0.1	1	0.892
1500	1024	0.1	2	0.442
1500	1024	0.1	1.5	0.69
1500	1024	0.1	1.25	0.894

The learning is from 0.25 to 1.25. It should be as large as possible and will not cause much oscillation in accuracy.

All the accuracy we got is an average number of three times with the same parameters except some particular trainings which is much longer than 10 minutes.

As the tables suggest, the number of epoch is the most powerful parameter. Adjust the number of epoch to find a suitable one. Choose weight update step as 3000 first and adjust other parameters. When weights updated steps may become a limitation to other parameters, that is, accuracy is all the way up and still has potential but steps are over, adjust the steps bigger.

Because the learning rate seems to be the second powerful parameter, it will become the second parameter I will adjust. Then we have the following table:

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
3000	1024	0.1	1.25	0.904
3000	1024	0.1	1	0.904
3000	1024	0.1	0.75	0.903

It is clear that the learning rate in this case is not so important. But I still prefer the bigger one in order to reduce the training time. Learning rate = 1.25. Then we choose number of nodes as the parameter to be adjusted.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
3000	512	0.1	1.25	0.903
3000	1024	0.1	1.25	0.906
3000	1536	0.1	1.25	0.478
3000	1200	0.1	1.25	0.871
3000	256	0.1	1.25	0.903
3000	128	0.1	1.25	0.902
3000	64	0.1	1.25	0.9

When adjusting the number of nodes, we find that the range should be updated to 64 to 1024. The less nodes we use, the more we can improve this network. So we choose 128 as the number of nodes. Then we need to modify the sd of weights.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
3000	128	0.1	1.25	0.902
3000	128	0.15	1.25	0.903
3000	128	0.2	1.25	0.905
3000	128	0.25	1.25	0.903
3000	128	0.3	1.25	0.902

When adjusting the sd of weights, it is clear that sd = 0.2 is the highest one with accuracy. So we choose sd = 0.2.

Then we need to expand the weight update steps to 6000. Because when we are adjusting sd and num of nodes, the weight update step has become a limitation. The training can still go further and get a better result.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
3000	128	0.2	1.25	0.905
6000	128	0.2	1.25	0.92
12000	128	0.2	1.25	0.933
24000	128	0.2	1.25	0.949
48000	128	0.2	1.25	0.965
96000	128	0.2	1.25	0.972

When expanding the steps, we can find the accuracy varies rapidly. So we keep expanding the steps and finally when it comes to 96000 steps, the time assumption is 8 minutes and the accuracy is 0.972. But we can find that in several epochs the accuracy varies a lot. So we try to modify the sd of weights and learning rate to see if we can improve the training.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
48000	128	0.2	1.25	0.965
48000	128	0.1	1.25	0.961
48000	128	0.1	0.5	0.943
48000	128	0.2	0.5	0.944
48000	128	0.2	1.5	0.967
48000	128	0.1	1.5	0.966

From this table, we can see sd is not very important. Is just need to be in a reasonable value. But learning is very important. It means the network learns things very fast so we can get a higher accuracy in the same period. The next thing we need to test is number of nodes.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
48000	64	0.2	1.25	0.959
48000	128	0.2	1.25	0.965

We can see from this table that 128 is better. So number of nodes 128 is what we want. Since the limitation of time is 10 minutes, expand the weight update step again to see what accuracy we will get.

Wt Upd Stps	Num Of Nodes	Sd Of Weights	Learning Rate	Accuracy
96000	128	0.2	1.5	0.974

This is the final result we get. In conclusion, the right number of nodes is the most important key to this network. The learning rate can help the network program finish in 10 minutes. And the sd of weights should be within a particular range to prevent very different accuracy values. The weight update steps help train the network and get a higher accuracy value. In the end, the accuracy is 0.974. The program terminates within 10 minutes. The source code is as follows:

```
# Learning with old style neural network


# MNIST_data is a collection of 2D gray level images.
# Each image is a picture of    a digit from 0..9
# Each image is of size 28 x 28 pixels


from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

import tensorflow as tf
sess = tf.InteractiveSession()

# xi is an image of size n. yi is the N labels of the image
# X is mxn. Row xi of X is an image
# Y is mxN. Row yi of Y is the labels of xi
X = tf.placeholder(tf.float32, shape=[None, 784])
Y = tf.placeholder(tf.float32, shape=[None, 10])

# a method for initializing weights. Initialize to small random values

def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.2)
    return tf.Variable(initial)

# a method for initializing bias. Initialize to 0.1

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# Densely Connected Hidden Layer of 1024 nodes

W_fc1 = weight_variable([784, 128])
b_fc1 = bias_variable([128])
v_fc1 = tf.nn.sigmoid(tf.matmul(X, W_fc1) + b_fc1) # v_fc1 ?x1024

# Readout Layer

W_fc2 = weight_variable([128, 10])
b_fc2 = bias_variable([10])
```

```
v_fc2 = tf.nn.sigmoid(tf.matmul(v_fc1, W_fc2) + b_fc2) # v_fc2 ?x10

predicted_Y = v_fc2;

sess.run(tf.global_variables_initializer())

mse = tf.losses.mean_squared_error(Y, predicted_Y)

train_step = tf.train.GradientDescentOptimizer(1.25).minimize(mse)

for i in range(96000):
    batch = mnist.train.next_batch(100)
    if i % 100 == 0:
        correct_prediction = tf.equal(tf.argmax(predicted_Y,1), tf.argmax(Y,1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        print(i, accuracy.eval(feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
    train_step.run(feed_dict={X: batch[0], Y: batch[1]})

correct_prediction = tf.equal(tf.argmax(predicted_Y,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

print(accuracy.eval(feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```