深度学习方法与实践第八次作业

梯度截断

姓名：杨玉雷
学号：18023040

1.连体网络 MINIST 优化

要求：在 lenet MNIST 分类中，应用梯度截断，使得梯度更新时，让每个变量的梯度分量保持在 min=-0.001, max=0.001 的范围内。
比较使用如上要求的梯度截断，和不使用梯度截断时，训练过程中，loss 的变化情况。网络采用 lenet,batch size=8，iter=1000，每隔 10 步打印一次 mnist.validation.next_batch(100) 的 loss 和 accuracy。

提交：代码和文档。文档中有训练中的 loss，accuracy 更新截图。

2.实验过程

本次实验和前面的改动不大，主要在于体会加了梯度截断和不加的训练效果对比。

实验代码：

（1）lenet.py

```
class Lenet:

    def __init__(self,learning_rate,sigma):
        self.learning_rate=learning_rate
        self.sigma=sigma
        #在创建的时候运行画图
        self._build_graph()

    #涉及网络的所有画图 build graph 过程,常用一个 build graph 封起来
    def _build_graph(self, network_name='Lenet'):
        self._setup_placeholders_graph()
        self._build_network_graph(network_name)
        self._compute_loss_graph()
        self._create_train_op_graph()
        self._compute_acc_graph()

    def _cnn_layer(self,scope_name, W_name, b_name,x,filter_shape,conv_strides, b_shape,padding_tag='VALID'):
        with tf.variable_scope(scope_name) as scope:
```

```
            conv_weights = tf.get_variable(name=W_name,shape=filter_shape,i
nitializer=tf.truncated_normal_initializer(stddev=self.sigma))
            conv_biases = tf.get_variable(name=b_name,shape=b_shape,initial
izer=tf.constant_initializer(0.1))
            conv = tf.nn.conv2d(x, conv_weights, strides=conv_strides, padd
ing=padding_tag)
            act=tf.nn.relu(tf.nn.bias_add(conv, conv_biases))
            tf.summary.histogram(W_name, conv_weights)
            tf.summary.histogram(b_name, conv_biases)
            return act

    def _pooling_layer(self, scope_name, relu, pool_ksize,pool_strides, pad
ding_tag='VALID'):
        with tf.variable_scope(scope_name) as scope:
            return tf.nn.max_pool(relu, ksize=pool_ksize, strides=pool_stri
des, padding=padding_tag)

    def _flatten(self,pool2):
        #将 x 拉直
        pool_shape=pool2.get_shape().as_list()
        length= pool_shape[1] * pool_shape[2] * pool_shape[3]
        return tf.reshape(pool2,[pool_shape[0],length])

    def _fully_connected_layer(self,scope_name,W_name, b_name,x,W_shape,b_s
hape):
        with tf.variable_scope(scope_name) as scope:
            fc_weights = tf.get_variable(W_name,W_shape,initializer=tf.trun
cated_normal_initializer(stddev=self.sigma))
            fc_biases = tf.get_variable(b_name,b_shape,initializer=tf.const
ant_initializer(0.1))

            act = tf.nn.relu(tf.matmul(x, fc_weights) + fc_biases)
            tf.summary.histogram(W_name, fc_weights)
            tf.summary.histogram(b_name, fc_biases)
            if scope_name=="layer_6_fc2":
             with tf.name_scope('fc') as v_s:
                # scale weights to [0 1], type is still float
                x_min = tf.reduce_min(fc_weights)
                x_max = tf.reduce_max(fc_weights)
                fc_W_img = (fc_weights - x_min) / (x_max - x_min)
                fc_W_img_reshape = tf.reshape(fc_W_img, [-1, W_shape[0], W_
shape[1], 1])
                tf.summary.image(W_name, fc_W_img_reshape)
```

```
            return act

    #构建图
    def _setup_placeholders_graph(self):
            self.x  = tf.placeholder(tf.float32, (None, 32, 32, 1),name='in
put_x')
            print(self.x.shape)
            self.y= tf.placeholder(tf.int32, (None))  # 在模型中的占位

    def _build_network_graph(self, scope_name):
        with tf.variable_scope(scope_name):
            #第一个卷积层
            # Input = 32x32x1. Output = 28x28x6.
            #卷积核:[filter_height, filter_width, in_channels, out_channel
s]
            self.conv1_relu= self._cnn_layer('layer_1_conv','conv1_w', 'con
v1_b', self.x, (5,5,1,6),[1,1,1,1],[6])

            #第一个池化层
            #Input = 28x28x6. Output = 14x14x6.
            #图像序列 x 高 x 宽 x 通道序列;步长只设定在“高”和“宽”的维度
为 2。
            self.pool1 = self._pooling_layer('layer_2_pooling', self.conv1_
relu, [1, 2, 2, 1], [1, 2, 2, 1])

            #第二个卷积层
            #Output = 10x10x16.

            self.conv2_relu= self._cnn_layer('layer_3_conv', 'conv2_w', 'co
nv2_b', self.pool1, (5, 5,6,16), [1, 1, 1, 1],[16])

            #第二个池化层
            #Input = 10x10x16. Output = 5x5x16.
            self.pool2 = self._pooling_layer('layer_4_pooling', self.conv2_
relu, [1, 2, 2, 1], [1, 2, 2, 1])

            #Tensor to vector:输入维度由 Nx5x5x16 压平后变为 Nx400
            print("self.pool2.shape:",self.pool2.shape)
            self.pool2=flatten(self.pool2)

            #第一个全连接层
            #Input = 400. Output = 120.
            self.fc1_relu=self._fully_connected_layer('layer_5_fc1','fc1_w',
'fc1_b',self.pool2,(400,120),[120])
```

```
            #第二个全连接层
            #Input = 120. Output = 84.
            self.fc2_relu=self._fully_connected_layer('layer_6_fc2','fc2_w',
'fc2_b',self.fc1_relu,(120,84),[84])

            #第三个全连接层
            #Input = 84. Output = 10.
            self.fc3_relu=self._fully_connected_layer('layer_7_fc3','fc3_w',
'fc3_b',self.fc2_relu,(84,10),[10])

            self.digits=self.fc3_relu

            return self.digits

    def _compute_loss_graph(self):
            cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=
self.y, logits=self.digits)
            self.loss = tf.reduce_mean(cross_entropy)
            tf.summary.scalar("loss", self.loss)

    def _create_train_op_graph(self):
        # self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimi
ze(self.loss)
        optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate,
 beta1=0.5)
        # 获取 loss 对 var_list 中变量的梯度
        var_list = tf.trainable_variables()
        gradients = optimizer.compute_gradients(self.loss, var_list)
        # 对梯度进行截断
        capped_gradients = [(tf.clip_by_value(grad, -0.001, 0.001), var) fo
r grad, var in gradients if grad is not None]
        # 应用截断梯度来更新参数
        self.train_op = optimizer.apply_gradients(capped_gradients)

    def _compute_acc_graph(self):
            # calculate correct
            self.prediction=tf.equal(tf.argmax(self.digits,1), tf.argmax(se
lf.y, 1))
            self.accuracy = tf.reduce_mean(tf.cast(self.prediction, tf.floa
t32))
            tf.summary.scalar("accuracy", self.accuracy)
```

（2）train.py

```
mninst = read_data_sets("MNIST_data/", reshape=False, one_hot=True)

def load_data(signal,batch):
   '''
   导入数据
   :return:训练集、测试集
   '''
   X_train, y_train = mninst.train.next_batch(batch)
   X_valid,y_valid=mninst.validation.next_batch(batch)

   assert (len(X_train) == len(y_train))
   assert (len(X_test) == len(y_test))


   # 将训练集进行填充
   # 因为 mnist 数据集的图片是 28*28*1 的格式，而 lenet 只接受 32*32 的格式
   # 所以只能在这个基础上填充
   X_train = np.pad(X_train, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')
   X_valid = np.pad(X_valid, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')

   if signal=="train":
     return X_train, y_train
   if signal=="validation":
     return X_valid,y_valid


batch_size = 8
LOGDIR="tensorboard"
iter = 1000 # 迭代次数


def main():
    lenet5 = Lenet(0.001, 0.1)
    merged_summary = tf.summary.merge_all()

    init = tf.global_variables_initializer()
    sess=tf.Session()

    writer = tf.summary.FileWriter(LOGDIR + "/" )

    writer.add_graph(sess.graph)
    sess.run(init)
```

```
    #开始训练
    print("Start Training...")
    for j in range(iter):
        X_train, y_train=load_data("train",batch_size)
        _, accuracy,loss=sess.run([lenet5.train_op,lenet5.accuracy,lenet5.loss], feed_dict={lenet5.x: X_train, lenet5.y: y_train})
        if j % 10 == 0:
            X_valid,y_valid=load_data("validation",100)
            s,accuracy,loss = sess.run([merged_summary,lenet5.accuracy,lenet5.loss], feed_dict={lenet5.x: X_valid, lenet5.y: y_valid})
            writer.add_summary(s, j)
            print("Step:",j," validation accuracy:",accuracy," loss:",loss)

if __name__ == '__main__':
    main()
```

3. 实验结果

（1）未使用梯度截断的训练过程输出截图如下图 1 所示。

```
Start Training...
Step: 0  validation accuracy: 0.09  loss: 2.31759
Step: 10  validation accuracy: 0.48  loss: 2.11247
Step: 20  validation accuracy: 0.54  loss: 1.79437
Step: 30  validation accuracy: 0.74  loss: 1.1539
Step: 40  validation accuracy: 0.68  loss: 1.07097
Step: 50  validation accuracy: 0.72  loss: 0.859021
Step: 60  validation accuracy: 0.73  loss: 0.833589
Step: 70  validation accuracy: 0.68  loss: 0.922204
Step: 80  validation accuracy: 0.65  loss: 0.971158
Step: 90  validation accuracy: 0.83  loss: 0.45558
Step: 100  validation accuracy: 0.81  loss: 0.524406
Step: 110  validation accuracy: 0.73  loss: 0.739084
Step: 120  validation accuracy: 0.66  loss: 0.882695
Step: 130  validation accuracy: 0.73  loss: 0.683623
Step: 140  validation accuracy: 0.73  loss: 0.6882
Step: 150  validation accuracy: 0.76  loss: 0.79761
Step: 160  validation accuracy: 0.76  loss: 0.653602

Step: 810  validation accuracy: 0.84  loss: 0.399176
Step: 820  validation accuracy: 0.77  loss: 0.65178
Step: 830  validation accuracy: 0.82  loss: 0.4496
Step: 840  validation accuracy: 0.78  loss: 0.560476
Step: 850  validation accuracy: 0.72  loss: 0.677289
Step: 860  validation accuracy: 0.78  loss: 0.520769
Step: 870  validation accuracy: 0.75  loss: 0.584319
Step: 880  validation accuracy: 0.68  loss: 0.744305
Step: 890  validation accuracy: 0.79  loss: 0.563866
Step: 900  validation accuracy: 0.81  loss: 0.455994
Step: 910  validation accuracy: 0.81  loss: 0.465697
Step: 920  validation accuracy: 0.79  loss: 0.521232
Step: 930  validation accuracy: 0.77  loss: 0.592529
Step: 940  validation accuracy: 0.76  loss: 0.568541
Step: 950  validation accuracy: 0.77  loss: 0.530707
Step: 960  validation accuracy: 0.67  loss: 0.790253
Step: 970  validation accuracy: 0.75  loss: 0.669188
Step: 980  validation accuracy: 0.72  loss: 0.625706
Step: 990  validation accuracy: 0.78  loss: 0.518784

Process finished with exit code 0
```

图 1 未使用梯度截断的训练过程截图

（2）未使用梯度截断的 loss 和 accuracy 变化情况图 2 所示。

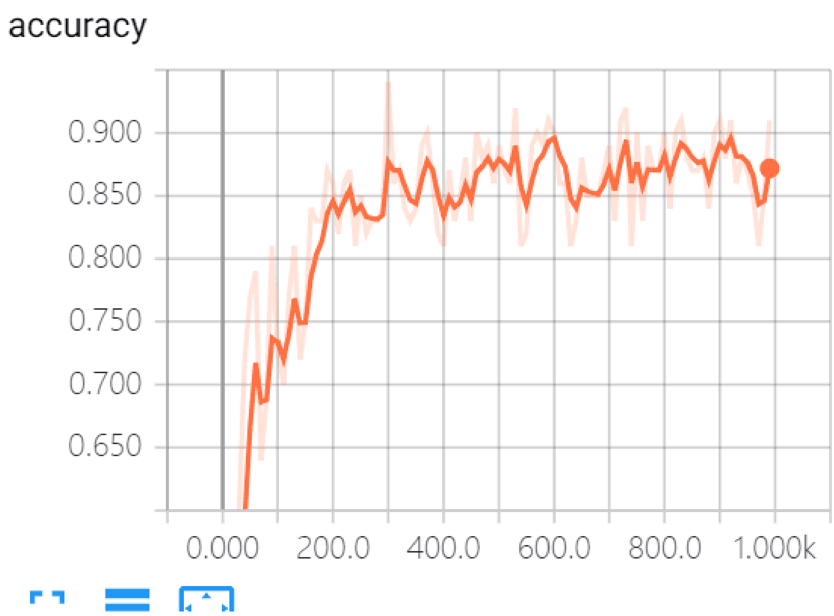
图 2 未使用梯度截断的 accuracy 和 loss 变化图

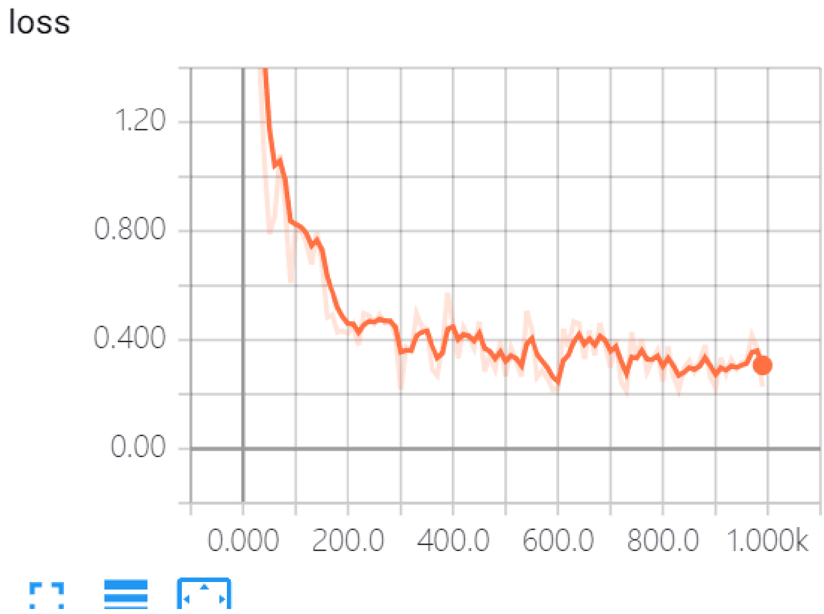
（3）使用了梯度截断的训练过程如图 3 所示。

```
Start Training...
Step: 0  validation accuracy: 0.12  loss: 2.27151
Step: 10  validation accuracy: 0.43  loss: 2.11076
Step: 20  validation accuracy: 0.49  loss: 1.77926
Step: 30  validation accuracy: 0.6  loss: 1.47876
Step: 40  validation accuracy: 0.72  loss: 1.07109
Step: 50  validation accuracy: 0.77  loss: 0.789671
Step: 60  validation accuracy: 0.79  loss: 0.85255
Step: 70  validation accuracy: 0.64  loss: 1.08135
Step: 80  validation accuracy: 0.69  loss: 0.883808
Step: 90  validation accuracy: 0.81  loss: 0.610164
Step: 100  validation accuracy: 0.73  loss: 0.808328
Step: 110  validation accuracy: 0.7  loss: 0.800499
Step: 120  validation accuracy: 0.77  loss: 0.758099
Step: 130  validation accuracy: 0.81  loss: 0.678861
Step: 140  validation accuracy: 0.72  loss: 0.797747
Step: 150  validation accuracy: 0.75  loss: 0.67581
Step: 830  validation accuracy: 0.91  loss: 0.214861
Step: 840  validation accuracy: 0.88  loss: 0.297752
Step: 850  validation accuracy: 0.87  loss: 0.323589
Step: 860  validation accuracy: 0.87  loss: 0.283547
Step: 870  validation accuracy: 0.88  loss: 0.322375
Step: 880  validation accuracy: 0.84  loss: 0.380799
Step: 890  validation accuracy: 0.9  loss: 0.264416
Step: 900  validation accuracy: 0.91  loss: 0.223202
Step: 910  validation accuracy: 0.88  loss: 0.334621
Step: 920  validation accuracy: 0.91  loss: 0.276488
Step: 930  validation accuracy: 0.86  loss: 0.32769
Step: 940  validation accuracy: 0.88  loss: 0.293057
Step: 950  validation accuracy: 0.87  loss: 0.316966
Step: 960  validation accuracy: 0.85  loss: 0.325269
Step: 970  validation accuracy: 0.81  loss: 0.416823
Step: 980  validation accuracy: 0.85  loss: 0.367848
Step: 990  validation accuracy: 0.91  loss: 0.228207
```

图 3 使用梯度截断的训练过程截图

（4）使用梯度截断的 loss 和 accuracy 变化情况图 4 所示。




loss
1.20
0.800
0.400
0.00
0.000
200.0
400.0
600.0
800.0
1.000k