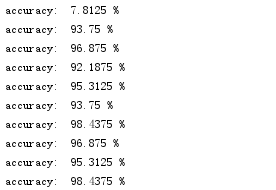
# 使用Layer API搭建卷积神经网络

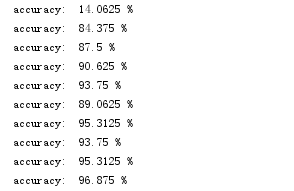
以下用一般方法搭建了一个神经网络：

|  |  |
| --- | --- |
| *#coding=utf-8* **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data data=input\_data.read\_data\_sets(**'F:/tmp/data'**, one\_hot=**True**)  IMG\_SIZE=28 IMG\_SIZE\_FLAT=IMG\_SIZE\*IMG\_SIZE IMG\_SHAPE=(IMG\_SIZE,IMG\_SIZE) NUM\_CHANNEL=1 NUM\_CLASSES=10 BATCH\_SIZE = 64 NUM\_ITERATIONS=1000  x=tf.placeholder(tf.float32,shape=[**None**, IMG\_SIZE\_FLAT], name=**'x'**) x\_image=tf.reshape(x, [-1, IMG\_SIZE, IMG\_SIZE, NUM\_CHANNEL]) y\_true=tf.placeholder(tf.float32,shape=[**None**, NUM\_CLASSES], name=**'y\_true'**) y\_true\_cls=tf.argmax(y\_true, dimension=1)  net=x\_image   |  | | --- | | conv1\_weight=tf.get\_variable(**"conv1\_weight"**, [5,5,1,16],initializer=tf.truncated\_normal\_initializer(stddev=0.1)) conv1\_biases=tf.get\_variable(**'conv1\_biases'**,[16],initializer=tf.constant\_initializer(0.0)) net=tf.nn.conv2d(input=net, filter=conv1\_weight, strides=[1,1,1,1], name=**'layer\_conv1'**, padding=**'SAME'**) net=tf.nn.relu(tf.nn.bias\_add(net,conv1\_biases)) net=tf.nn.max\_pool(net, ksize=[1,2,2,1], strides=[1,2,2,1], padding=**'SAME'**)  conv2\_weight=tf.get\_variable(**"conv2\_weight"**, [5,5,16,128],initializer=tf.truncated\_normal\_initializer(stddev=0.1)) conv2\_biases=tf.get\_variable(**'conv2\_biases'**,[128],initializer=tf.constant\_initializer(0.0)) net=tf.nn.conv2d(input=net, filter=conv2\_weight, strides=[1,1,1,1], name=**'layer\_conv2'**, padding=**'SAME'**) net=tf.nn.relu(tf.nn.bias\_add(net,conv2\_biases)) net=tf.nn.max\_pool(net, ksize=[1,2,2,1], strides=[1,2,2,1], padding=**'SAME'**)  pool\_shape=net.get\_shape().as\_list() nodes=pool\_shape[1] \* pool\_shape[2] \* pool\_shape[3]  net=tf.reshape(net, [BATCH\_SIZE, nodes]) fc1\_weights=tf.get\_variable(**"fc1\_weight"**, [nodes, 128]) fc1\_biases=tf.get\_variable(**"fc1\_biases"**, [128]) net=tf.nn.relu(tf.matmul(net, fc1\_weights) + fc1\_biases)  fc2\_weights=tf.get\_variable(**"fc2\_weight"**, [128, 10]) fc2\_biases=tf.get\_variable(**"fc2\_biases"**, [10]) net=tf.matmul(net, fc2\_weights) + fc2\_biases |   logits=net y\_pred=tf.nn.softmax(logits=logits) y\_pred\_cls=tf.argmax(y\_pred,dimension=1) cross\_entropy=tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_true, logits=logits) loss=tf.reduce\_mean(cross\_entropy)  optimizer=tf.train.AdamOptimizer(learning\_rate=0.0001).minimize(loss) correct\_prediction=tf.equal(y\_pred\_cls,y\_true\_cls) accuracy=tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  **with** tf.Session() **as** sess:  sess.run(tf.global\_variables\_initializer())  **for** i **in** range(NUM\_ITERATIONS):  x\_batch, y\_true\_batch = data.train.next\_batch(BATCH\_SIZE)  feed\_dict\_train={x:x\_batch,y\_true:y\_true\_batch}  sess.run(optimizer, feed\_dict=feed\_dict\_train)  **if** i%100 == 0:  acc=sess.run(accuracy, feed\_dict=feed\_dict\_train)  print(**'accuracy: '**, acc \* 100, **'%'**) |



下面使用tensorflow的Layer API搭建一个类似的网络：

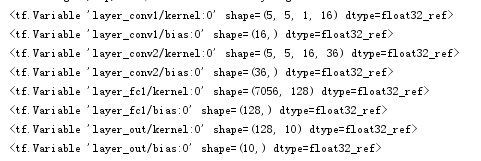
|  |  |
| --- | --- |
| *#coding=utf-8* **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data data=input\_data.read\_data\_sets(**'F:/tmp/data'**, one\_hot=**True**)  IMG\_SIZE=28 IMG\_SIZE\_FLAT=IMG\_SIZE\*IMG\_SIZE IMG\_SHAPE=(IMG\_SIZE,IMG\_SIZE) NUM\_CHANNEL=1 NUM\_CLASSES=10 BATCH\_SIZE = 64 NUM\_ITERATIONS=1000  **def** get\_weights\_variable(layer\_name):  **with** tf.variable\_scope(layer\_name, reuse=**True**):  variable=tf.get\_variable(**'kernel'**)  **return** variable  x=tf.placeholder(tf.float32,shape=[**None**, IMG\_SIZE\_FLAT], name=**'x'**) x\_image=tf.reshape(x, [-1, IMG\_SIZE, IMG\_SIZE, NUM\_CHANNEL]) y\_true=tf.placeholder(tf.float32,shape=[**None**, NUM\_CLASSES], name=**'y\_true'**) y\_true\_cls=tf.argmax(y\_true, dimension=1) net=x\_image   |  | | --- | | net=tf.layers.conv2d(inputs=net, name=**'layer\_conv1'**, padding=**'same'**, filters=16, kernel\_size=5,activation=tf.nn.relu) layer1\_conv1=net net=tf.layers.conv2d(inputs=net, name=**'layer\_conv2'**, padding=**'same'**, filters=36, kernel\_size=5,activation=tf.nn.relu) layer2\_conv2=net net=tf.layers.max\_pooling2d(inputs=net, pool\_size=2,strides=2) net=tf.contrib.layers.flatten(net) net=tf.layers.dense(inputs=net,name=**'layer\_fc1'**, units=128, activation=tf.nn.relu) net=tf.layers.dense(inputs=net,name=**'layer\_out'**, units=NUM\_CLASSES, activation=**None**) |   logits=net y\_pred=tf.nn.softmax(logits=logits) y\_pred\_cls=tf.argmax(y\_pred,dimension=1) cross\_entropy=tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_true, logits=logits) loss=tf.reduce\_mean(cross\_entropy)  optimizer=tf.train.AdamOptimizer(learning\_rate=0.0001).minimize(loss) correct\_prediction=tf.equal(y\_pred\_cls,y\_true\_cls) accuracy=tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  *#打印所有的变量，可以发现每一个层里都有两个变量，一个kernel，一个bias # for var in tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES): # print(var)* weights\_conv1 = get\_weights\_variable(layer\_name=**'layer\_conv1'**) weights\_conv2 = get\_weights\_variable(layer\_name=**'layer\_conv2'**)  **with** tf.Session() **as** sess:  sess.run(tf.global\_variables\_initializer())  **for** i **in** range(NUM\_ITERATIONS):  x\_batch, y\_true\_batch = data.train.next\_batch(BATCH\_SIZE)  feed\_dict\_train={x:x\_batch,y\_true:y\_true\_batch}  sess.run(optimizer, feed\_dict=feed\_dict\_train)  **if** i%100 == 0:  acc=sess.run(accuracy, feed\_dict=feed\_dict\_train)  print(**'accuracy: '**, acc \* 100, **'%'**) |



对比两种方法，在网络搭建时，使用Layer API要简单得多，使用Layer API建立的网络，都在内部包含有两个重要的变量，一个为kernel一个为bias，即为权重和偏置值，使用如下代码可以将所有的变量打印出来：

|  |
| --- |
| **for** var **in** tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES):  print(var) |

结果如下：



另外，注意，将特征图展开时使用的方法是tf.contrib.layers.flatten()，该方法在tf.contrib.layers下，而不是在tf.layers下。全连接层使用的方法是tf.layers.dense()。