TensorFlow人工智能引擎入门教程之七 DNN深度神经网络 的原理 以及 使用

DNN 深度神经网络，就是 把原有的多层神经网络 扩展到深度学习里面，加上了BP 反馈，是的整理上 loss 收敛 直至不变，同时也有dropout 前面 有很多这个词 出现，dropout 是指 随机用一定概率 把一些 节点失效，进行参与训练 放置数据整理上陷入overfitting 局部最优解。

OK 我们现在打开前面的AlexNet的网络

DNN ,就是去掉C 之后 使用全连接层+dropout下降+relu激活 一层一层的WX+B的 网络模式

```
import input_data mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)  import tensorflo
w as tf  # Parameters learning_rate = 0.001 training_iters = 100000 batch_size = 128 display_step = 10
 # Network Parameters n_input = 784 # MNIST data input (img shape: 28*28) n_classes = 10 # MNIS
T total classes (0-9 digits) dropout = 0.75 # Dropout, probability to keep units  # tf Graph input x = t
f.placeholder(tf.float32, [None, n_input]) y = tf.placeholder(tf.float32, [None, n_classes]) keep_prob = t
f.placeholder(tf.float32) #dropout (keep probability)  # Create model def conv2d(img, w, b):   return t
f.nn.relu(tf.nn.bias_add(tf.nn.conv2d(img, w, strides=[1, 1, 1, 1], padding='SAME'),b))  def max_pool(im
g, k):   return tf.nn.max_pool(img, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME')  def conv_ne
t(_X, _weights, _biases, _dropout):   # Reshape input picture   _X = tf.reshape(_X, shape=[-1, 28, 28, 1])
   # Convolution Layer   conv1 = conv2d(_X, _weights['wc1'], _biases['bc1'])   # Max Pooling (down-
sampling)   conv1 = max_pool(conv1, k=2)   # Apply Dropout   conv1 = tf.nn.dropout(conv1, _drop
out)   # Convolution Layer   conv2 = conv2d(conv1, _weights['wc2'], _biases['bc2'])   # Max Poolin
g (down-sampling)   conv2 = max_pool(conv2, k=2)   # Apply Dropout   conv2 = tf.nn.dropout(con
v2, _dropout)   # Fully connected layer   dense1 = tf.reshape(conv2, [-1, _weights['wd1'].get_shap
e().as_list()[0]]) # Reshape conv2 output to fit dense layer input   dense1 = tf.nn.relu(tf.add(tf.matmul(d
ense1, _weights['wd1']), _biases['bd1'])) # Relu activation   dense1 = tf.nn.dropout(dense1, _dropou
t) # Apply Dropout   # Output, class prediction   out = tf.add(tf.matmul(dense1, _weights['out']), _bia
ses['out'])   return out  # Store layers weight & bias weights = {   'wc1': tf.Variable(tf.random_norma
l([5, 5, 1, 32])), # 5x5 conv, 1 input, 32 outputs   'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])), # 5
x5 conv, 32 inputs, 64 outputs   'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])), # fully connecte
d, 7*7*64 inputs, 1024 outputs   'out': tf.Variable(tf.random_normal([1024, n_classes])) # 1024 input
s, 10 outputs (class prediction) }  biases = {   'bc1': tf.Variable(tf.random_normal([32])),   'bc2': tf.Varia
ble(tf.random_normal([64])),   'bd1': tf.Variable(tf.random_normal([1024])),   'out': tf.Variable(tf.rando
m_normal([n_classes])) }  # Construct model pred = conv_net(x, weights, biases, keep_prob)  # Define l
oss and optimizer cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y)) optimizer = t
f.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)  # Evaluate model correct_pred = tf.e
qual(tf.argmax(pred,1), tf.argmax(y,1)) accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))  # Ini
tializing the variables init = tf.initialize_all_variables()  # Launch the graph with tf.Session() as sess:   ses
s.run(init)   step = 1   # Keep training until reach max iterations   while step * batch_size < training_it
ers:   batch_xs, batch_ys = mnist.train.next_batch(batch_size)   # Fit training using batch data
   sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys, keep_prob: dropout})   if step % displa
y_step == 0:   # Calculate batch accuracy   acc = sess.run(accuracy, feed_dict={x: batch_x
s, y: batch_ys, keep_prob: 1.})   # Calculate batch loss   loss = sess.run(cost, feed_dict={x: b
atch_xs, y: batch_ys, keep_prob: 1.})   print "Iter " + str(step*batch_size) + ", Minibatch Los
s= " + "{:.6f}".format(loss) + ", Training Accuracy= " + "{:.5f}".format(acc)   step += 1   print "Opti
mization Finished!"   # Calculate accuracy for 256 mnist test images   print "Testing Accuracy:", sess.r
un(accuracy, feed_dict={x: mnist.test.images[:256], y: mnist.test.labels[:256], keep_prob: 1.})
```

先去掉卷积部分 以及 maxpool部分

下面遵循 WX+B 即可 输入时候[Batchsize,768]

那么 W 应该 需要是[768 ,n] 接下来应该是[n,m]在接下来应该是[m,p]

也就是满足矩阵乘法

下面来看看我们定义的

```
def dnn(_X, _weights, _biases, _dropout):
    # Reshape input picture


    _X = tf.nn.dropout(_X, _dropout)#//这里可以让dropout都不同 我就一样了
    d1 = tf.nn.relu(tf.nn.bias_add(tf.matmul(_X,_weights['wd1']),_biases['bd1']), name="d1")

    d2x = tf.nn.dropout(d1, _dropout)
    d2 =  tf.nn.relu(tf.nn.bias_add(tf.matmul(d2x,_weights['wd2']),_biases['bd2']), name="d2")

    #dense1 = tf.nn.relu(tf.matmul(dense1, _weights['wd1']) + _biases['bd1'], name='fc1') # Relu activation

    #dense2 = tf.nn.relu(tf.matmul(dense1, _weights['wd2']) + _biases['bd2'], name='fc2') # Relu activation
    dout =tf.nn.dropout(d2,_dropout)
    # Output, class prediction
    out = tf.matmul(dout, _weights['out']) + _biases['out']
    return out

# Store layers weight & bias
```

修改相应的变量

```
weights = {    'wd1': tf.Variable(tf.random_normal([784,600], stddev=0.01)),     'wd2': tf.Variable(tf.random_normal([600,480], stddev=0.01)),     'out': tf.Variable(tf.random_normal([480, 10])) } biases = {    'bd1': tf.Variable(tf.random_normal([600])),     'bd2': tf.Variable(tf.random_normal([480])),     'out': tf.Variable(tf.random_normal([10])), }
```

其实我们看出来了 就是三个全连接层 只不过通过dropout保证 loss一致收敛，不会陷入最优解问题，其实可能实际上的还会有norm 其他一些层 等优化 ，也许是tanh 或者 sigmoid 的激活函数 这是网络设计的问题，上面的就是一个简单的DNN网络 利用深度学习 比传统的多层网络有了更好的效果以及准确率

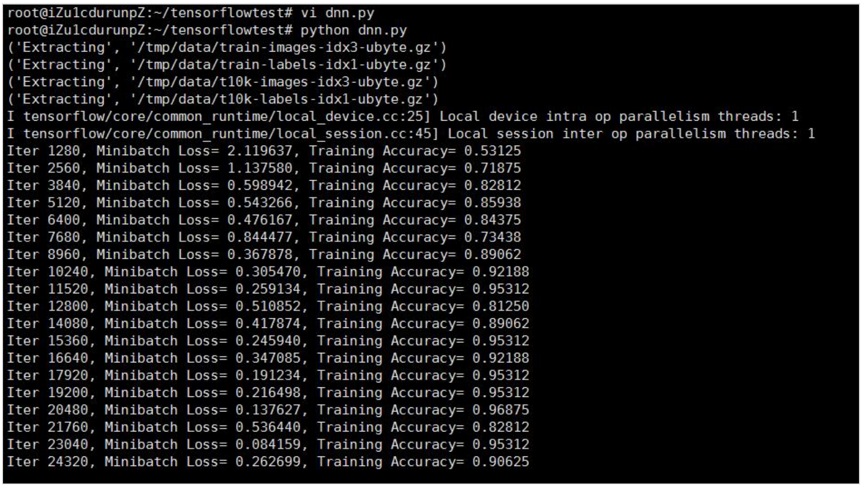
看上面Nonex768 768X600 600X480 480X10 = None x 10

ok 下面贴出全部的代码

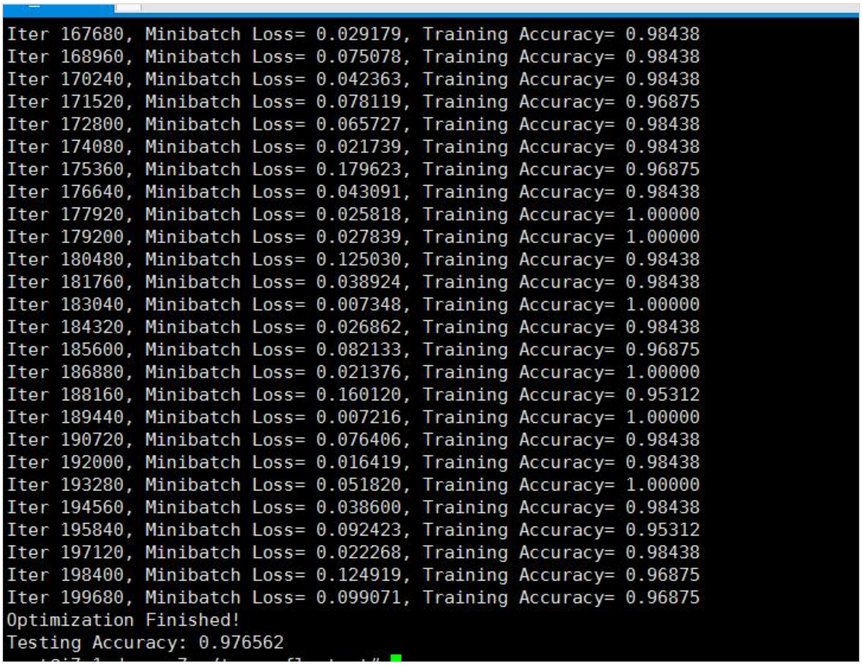
```
import input_data mnist = input_data.read_data_sets("/tmp/data/", one_hot=True) import tensorflow as tf # Parameters learning_rate = 0.001 training_iters = 200000 batch_size = 64 display_step = 20 # Network Parameters n_input = 784 # MNIST data input (img shape: 28*28) n_classes = 10 # MNIST total classes (0-9 digits) dropout = 0.8 # Dropout, probability to keep units # tf Graph input x = tf.placeholder(tf.float32, [None, n_input]) y = tf.placeholder(tf.float32, [None, n_classes]) keep_prob = tf.placeholder(tf.float32) # dropout (keep probability) def init_weights(shape):     return tf.Variable(tf.random_normal(shape, stddev=0.01)) # Create custom model def conv2d(name, l_input, w, b):     return tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(l_input, w, strides=[1, 1, 1, 1], padding='SAME'),b), name=name) def max_pool(name, l_input, k):     return tf.nn.max_pool(l_input, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME', name=name) def norm(name, l_input, lsize=4):     return tf.nn.lrn(l_input, lsize, bias=1.0, alpha=0.001 / 9.0, beta=0.75, name=name) def dnn(_X, _weights, _biases, _dropout):     # Reshape input picture      _X = tf.nn.dropout(_X, _dropout)#//这里可以让dropout都不同 我就一样了     d1 = tf.nn.relu(tf.nn.bias_add(tf.matmul(_X,_weights['wd1']),_biases['bd1']), name="d1")     d2x = tf.nn.dropout(d1, _dropout)     d2 =  tf.nn.relu(tf.nn.bias_add(tf.matmul(d2x,_weights['wd2']),_biases['bd2']), name="d2")     #dense1 = tf.nn.relu(tf.matmul(dense1, _weights['wd1']) + _biases['bd1'], name='fc1') # Relu activation     #dense2 = tf.nn.relu(tf.matmul(dense1, _weights['wd2']) + _biases['bd2'], name='fc2') # Relu activation     dout =tf.nn.dropout(d2,_dropout)     # Output, class prediction     out = tf.matmul(dout, _weights['out']) + _biases['out']     return out # Store layers weight & bias weights = {    'wd1': tf.Variable(tf.random_normal([784,600], stddev=0.01)),     'wd2': tf.Variable(tf.random_normal([600,480], stddev=0.01)),     'out': tf.Variable(tf.random_normal([480, 10])) } biases = {    'bd1': tf.Variable(tf.random_normal([600])),     'bd2': tf.Variable(tf.random_normal([480])),     'out': tf.Variable(tf.random_normal([10])), } # Construct model pred = dnn(x, weights, biases, keep_prob) # Define loss and optimizer cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y)) optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # Evaluate model correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1)) accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32)) # Initializing the variables init = tf.initialize_all_variables() # Launch the graph with tf.Session() as sess:     sess.run(init)     step = 1     # Keep training until reach max iterations     while step * batch_size < training_iters:         batch_xs, batch_ys = mnist.train.next_batch(batch_size)         # Fit training using batch data         sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys, keep_prob: dropout})         if step % display_step == 0:             # Calculate batch accuracy             acc = sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys, keep_prob: 1.})             # Calculate batch loss             loss = sess.run(cost, feed_dict={x: batch_xs, y: batch_ys, keep_prob: 1.})             print "Iter " + str(step*batch_size) + ", Minibatch Loss= " + "{:.6f}".format(loss) + ", Training Accuracy= " + "{:.5f}".format(acc)         step += 1     print "Optimization Finished!"     # Calculate accuracy for 256 mnist test images     print "Testing Accuracy:", sess.run(accuracy, feed_dict={x: mnist.test.images[:256], y: mnist.test.labels[:256], keep_prob: 1.})
```

下面我们来运行测试看看

```
root@iZu1cdurunpZ:~/tensorflowtest#
root@iZu1cdurunpZ:~/tensorflowtest# python dnn.py
('Extracting', '/tmp/data/train-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
```

```
root@iZu1cdurunpZ:~/tensorflowtest# vi dnn.py
root@iZu1cdurunpZ:~/tensorflowtest# python dnn.py
('Extracting', '/tmp/data/train-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
Iter 1280, Minibatch Loss= 2.119637, Training Accuracy= 0.53125
Iter 2560, Minibatch Loss= 1.137580, Training Accuracy= 0.71875
Iter 3840, Minibatch Loss= 0.598942, Training Accuracy= 0.82812
Iter 5120, Minibatch Loss= 0.543266, Training Accuracy= 0.85938
Iter 6400, Minibatch Loss= 0.476167, Training Accuracy= 0.84375
Iter 7680, Minibatch Loss= 0.844477, Training Accuracy= 0.73438
Iter 8960, Minibatch Loss= 0.367878, Training Accuracy= 0.89062
Iter 10240, Minibatch Loss= 0.305470, Training Accuracy= 0.92188
Iter 11520, Minibatch Loss= 0.259134, Training Accuracy= 0.95312
Iter 12800, Minibatch Loss= 0.510852, Training Accuracy= 0.81250
Iter 14080, Minibatch Loss= 0.417874, Training Accuracy= 0.89062
Iter 15360, Minibatch Loss= 0.245940, Training Accuracy= 0.95312
Iter 16640, Minibatch Loss= 0.347085, Training Accuracy= 0.92188
Iter 17920, Minibatch Loss= 0.191234, Training Accuracy= 0.95312
Iter 19200, Minibatch Loss= 0.216498, Training Accuracy= 0.95312
Iter 20480, Minibatch Loss= 0.137627, Training Accuracy= 0.96875
Iter 21760, Minibatch Loss= 0.536440, Training Accuracy= 0.82812
Iter 23040, Minibatch Loss= 0.084159, Training Accuracy= 0.95312
Iter 24320, Minibatch Loss= 0.262699, Training Accuracy= 0.90625
```



```
Iter 167680, Minibatch Loss= 0.029179, Training Accuracy= 0.98438
Iter 168960, Minibatch Loss= 0.075078, Training Accuracy= 0.98438
Iter 170240, Minibatch Loss= 0.042363, Training Accuracy= 0.98438
Iter 171520, Minibatch Loss= 0.078119, Training Accuracy= 0.96875
Iter 172800, Minibatch Loss= 0.065727, Training Accuracy= 0.98438
Iter 174080, Minibatch Loss= 0.021739, Training Accuracy= 0.98438
Iter 175360, Minibatch Loss= 0.179623, Training Accuracy= 0.96875
Iter 176640, Minibatch Loss= 0.043091, Training Accuracy= 0.98438
Iter 177920, Minibatch Loss= 0.025818, Training Accuracy= 1.00000
Iter 179200, Minibatch Loss= 0.027839, Training Accuracy= 1.00000
Iter 180480, Minibatch Loss= 0.125030, Training Accuracy= 0.98438
Iter 181760, Minibatch Loss= 0.038924, Training Accuracy= 0.98438
Iter 183040, Minibatch Loss= 0.007348, Training Accuracy= 1.00000
Iter 184320, Minibatch Loss= 0.026862, Training Accuracy= 0.98438
Iter 185600, Minibatch Loss= 0.082133, Training Accuracy= 0.96875
Iter 186880, Minibatch Loss= 0.021376, Training Accuracy= 1.00000
Iter 188160, Minibatch Loss= 0.160120, Training Accuracy= 0.95312
Iter 189440, Minibatch Loss= 0.007216, Training Accuracy= 1.00000
Iter 190720, Minibatch Loss= 0.076406, Training Accuracy= 0.98438
Iter 192000, Minibatch Loss= 0.016419, Training Accuracy= 0.98438
Iter 193280, Minibatch Loss= 0.051820, Training Accuracy= 1.00000
Iter 194560, Minibatch Loss= 0.038600, Training Accuracy= 0.98438
Iter 195840, Minibatch Loss= 0.092423, Training Accuracy= 0.95312
Iter 197120, Minibatch Loss= 0.022268, Training Accuracy= 0.98438
Iter 198400, Minibatch Loss= 0.124919, Training Accuracy= 0.96875
Iter 199680, Minibatch Loss= 0.099071, Training Accuracy= 0.96875
Optimization Finished!
Testing Accuracy: 0.976562
```

有人问我 图片文件夹 怎么读取。。。数据怎么读取。。。。。。可以看看input的代码，，，这是python的知识。。。不属于tensorflow 不过tensorflow也带了一些record 读取。后面比如数据推荐系统就可能会使用哪个电影文件的 ，那时候我在使用下。

上一篇：原 TensorFlow人工智能引擎入门教程之六 训练的模型Model 保存 文件 并使用

下一篇：原 TensorFlow人工智能引擎入门教程之八 接着补充一章MLP多层感知器网络原理以及 使用

类似文章 更多

- TensorFlow之CNN和GPU
- 【机器学习】Tensorflow学习笔记
- 深入MNIST code测试
- 用Tensorflow基于Deep Q Learning DQN 玩....
- 当TensorFlow遇见CNTK
- 基于Python的卷积神经网络和特征提取&...
- Deep learning：四十二(Denoise Autoenc...
- Theano3.5-练习之深度卷积网络

精选文章

- 百图请欣赏,张张是精品
- 教你制作年夜饭菜肴
- 你在过什么样的生活
- 古代有多少以狗自诩的名人
- 运用潜意识的力量考取北大
- 椎盘突出—外用药热敷治愈2年
- 拆招之办公室那些阴招狠招
- 【长寿秘方】三样中药让老人精神气爽！告别...

发表评论：

请 登录 或者 注册 后再进行评论 社交帐号登录：