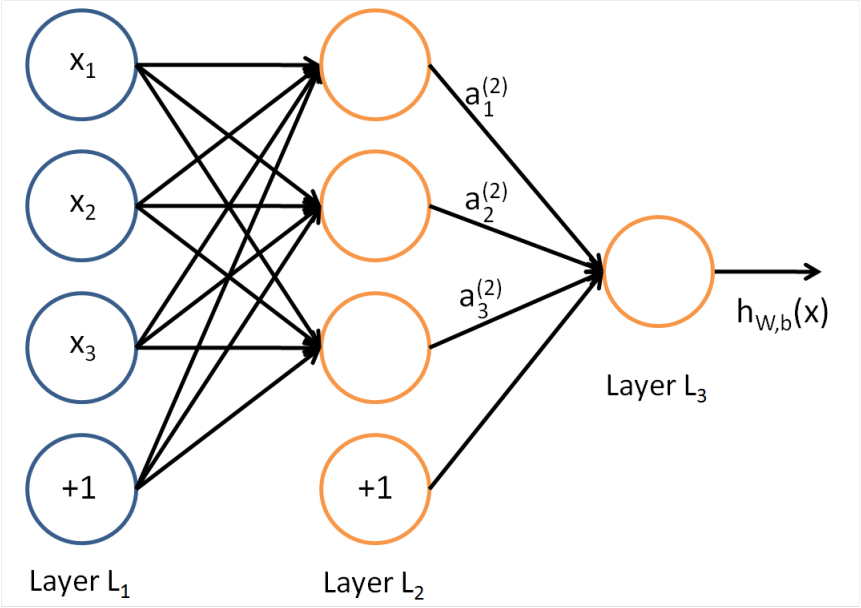
TensorFlow人工智能引擎入门教程之二 CNN卷积神经网络的基本定义理解。

2016-05-15 蓝莓对冲... 阅 20

首先 上面讲了神经元 Y=WX+B ，通过输入的参数X ==========》Y 深度学习每一个batch来说 其实就是 多项公式



我们知道 在数学里面 求多项公式 其实 就是 矩阵 W 矩阵 乘与 X 加上 B 矩阵 = Y矩阵 ，矩阵 二元数组在tensorflow 也是一个tensor ndarray，通常 我们知道 因为relu 收敛效果要比sigmod 与tanh 要好，所以在cnn中常用relu，所以 其实 对于输出o=relu(w x+b)，

1.TensorFlow卷积

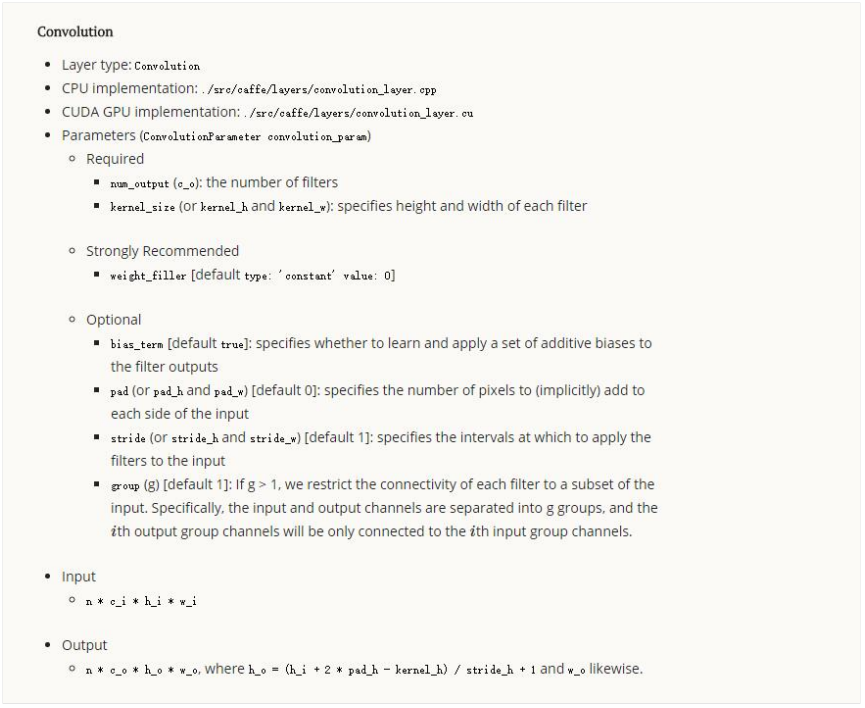
比如 下面 是tensorflow卷积定义 relu(W*X+B) W 矩阵 * X矩阵 + B矩阵 = W 权重variable变量 * X (placeholder占位符外部输入)variable变量 + B偏重变量，因为深度学习 会自动 不断地计算loss损失 BP 来调整 w b 所以 w b初始化 可以随便 全部都是0 都行，所以 其实 就是 X 以及 Y 对于 X来说其实我们知道 就是 我们图像数据 Y 是图像的标签，但是Y需要转为数学可以计算的值，所以采用one-hot 数组记录 标签的索引就行，比如 xx1 xx2 xx3 相应的y1=[1,0,0] y2=[0 1 0] y3=[0 0 1]

那么 其实 就是 X 图像的像素 通过外部输入 placeholder占位符 Y值 外部输入 通过 placeholder占位符

我们知道 W*X 矩阵 相乘 必须 符合 MXN NXM =MXM 也就是说W的 列 必须 与 X的行数目相同 这是要注意的，所以上一张shape来规范维度计算 ，下面是一个卷积层 定义 relu(wx+b) 下面是tensorflow来表示relu(wx+b)的公式

其中要注意参数 strides 是卷积滑动的步长 你可以配置更多的系数，

```
def conv2d(x, w, b):    return tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(x, w, strides=[1, 1, 1, 1], padding='SAME'),b))
```



Convolution

- Layer type: `Convolution`
- CPU implementation: `./src/caffe/layers/convolution_layer.cpp`
- CUDA GPU implementation: `./src/caffe/layers/convolution_layer.cu`
- Parameters (`ConvolutionParameter convolution_param`)
  - Required
    - `num_output` (`c_o`): the number of filters
    - `kernel_size` (or `kernel_h` and `kernel_w`): specifies height and width of each filter
  - Strongly Recommended
    - `weight_filler` [default `type: 'constant' value: 0`]
  - Optional
    - `bias_term` [default `true`]: specifies whether to learn and apply a set of additive biases to the filter outputs
    - `pad` (or `pad_h` and `pad_w`) [default 0]: specifies the number of pixels to (implicitly) add to each side of the input
    - `stride` (or `stride_h` and `stride_w`) [default 1]: specifies the intervals at which to apply the filters to the input
    - `group` (g) [default 1]: If g > 1, we restrict the connectivity of each filter to a subset of the input. Specifically, the input and output channels are separated into g groups, and the ith output group channels will be only connected to the ith input group channels.
- Input
  - `n * c_i * h_i * w_i`
- Output
  - `n * c_o * h_o * w_o`, where `h_o = (h_i + 2 * pad_h - kernel_h) / stride_h + 1` and `w_o` likewise.

上面我们理解了， 其实就是 W矩阵 * X矩阵 +B 矩阵 =Y矩阵

Y矩阵 好算 ，W矩阵 +B矩阵 已知 可以随意后面要bp调整。那么X矩阵 是图像 也就是说输入的X

```
x = tf.placeholder(tf.float32, [None, w*h]) #w*h 因为批次训练 数据可以任意所以第一个是None ，第二个图像是个w * h的图像 ，可以展开得到 w * h 的数组
```

我们知道 卷积运算 其实是卷积的个数 M 展开的卷积核长宽维度 K * 卷积能够提取的数目 K * 提取图像的维度N =MXN

也就说可以通过 卷积核 让图像 W*H 转变成 一个 卷积的个数 的行 ，N 为提取图像卷积特征的每一行数据的的矩阵

还是遵循WX+B

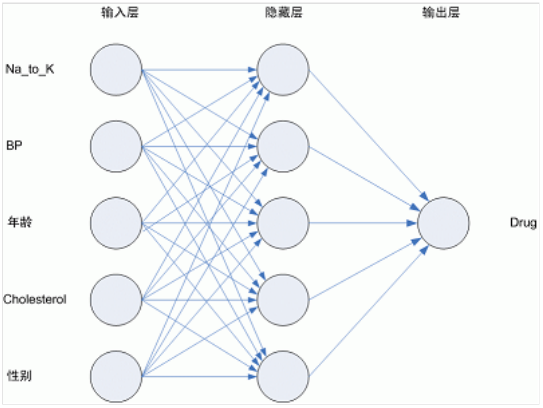
我们定义好W

我们可以把整个网络看成wx+b 忽略掉那些relu 等等 其实 就是一个y=wx+b 我们知道输入 是上面的

```
x = tf.placeholder(tf.float32, [None, w*h])  y = tf.placeholder(tf.float32, [None, ysize])#y的数目个数 比如3个类 就是3
```

那么 一个x 需 要 * W = y 计入batch为50 y为10 那么[50,224*224] * W= [50 ,10]

那么W 需要是一个[224*224，10] 的矩阵 ，也就是说 至少224*224*10*50 个连接



下面继续 讲X [None ,w*h] 对于每一个 w*h 是一个矩阵 每一层的w 也是一个矩阵 每一层的 b也是一个矩阵 ，每一层的输出y1 也是一个矩阵 y=[w*h]*w+b 为了减少系数，我们使用卷积，把它转换成MXN的值 ，这里就是跟全连接层的不同，使用了卷积转换成了一个MXN的卷积特征 而全连接层就是 y=wx+b (这里省略了那些relu(wx+b) tanh(wx+b))

所以我们现在来看看每一层的w 定义

因为卷积层的w是需要 与 w*h 提取的MXK来做矩阵相乘 所以 他是跟卷积核相关 以及输入 输出 相关，对于每一张图像

输入为1 输出分解为多个输出用于 第二层运算

```
wc=tf.Variable(tf.random_normal([3, 3, 1, 64]) #3 3 分别为3x3大小的卷积核 1位输入数目 因为是第一层所以是1 输出我们配置的64  所以我们知道了 如果下一次卷积wc2=[5,5,64,256] //5x5 是我们配置的卷积核大小，第三位表示输入数目 我们通过上面知道 上面的输出 也就是下一层的输入 所以 就是64 了输出我们定义成256 这个随你喜好，关键要迭代看效果，一般都是某一个v*v的值
```

这样我们知道了 wc*(卷积得到的x) + b =y1下一层输入，我们也能知道 一下层输入 mxn xo 分别为输出O个卷积核特征MXN 我们也能知道b 大小 那个肯定跟O一样，比如上面是64输出，所以

```
b1=tf.Variable(tf.random_normal([64])) 同样可以知道b2=[256]
```

下面我们讲一讲池化层pool 池化层 不会减少输出，只会把MXNXO ，把卷积提取的特征做进一步卷积 取MAX AVG MIN等 用局部代替整理进一步缩小特征值大小

下面是一个用kxk 的核做maxpooling的定义

```
def max_pool_kxk(x):   return tf.nn.max_pool(x, ksize=[1, k, k, 1],                              strides=[1, k, k, 1], padding='SAME')
```

这样我们可以定义一个简单的CNN了

```
c1 = tf.nn.relu(tf.nn.conv2d(X, w, [1, 1, 1, 1], 'SAME'))     m1 = tf.nn.max_pool(l1a, ksize=[1, k, k, 1],                         strides=[1, k, k, 1], padding='SAME')     d1 = tf.nn.dropout(l1, p_keep_conv)     c2 = tf.nn.relu(tf.nn.conv2d(l1, w2, [1, 1, 1, 1], 'SAME'))     m2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1],                         strides=[1, 2, 2, 1], padding='SAME')     d2= tf.nn.dropout(l2, p_keep_conv)     c3 = tf.nn.relu(tf.nn.conv2d(l2, w3, [1, 1, 1, 1], 'SAME'))     m3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1],                         strides=[1, 2, 2, 1], padding='SAME')          d3 = tf.nn.dropout(tf.reshape(l3, [-1, w4.get_shape().as_list()[0]]), p_keep_conv)     #d3表示倒数第二层的输出 也就是倒数第一层一层的输入x 我们代
```

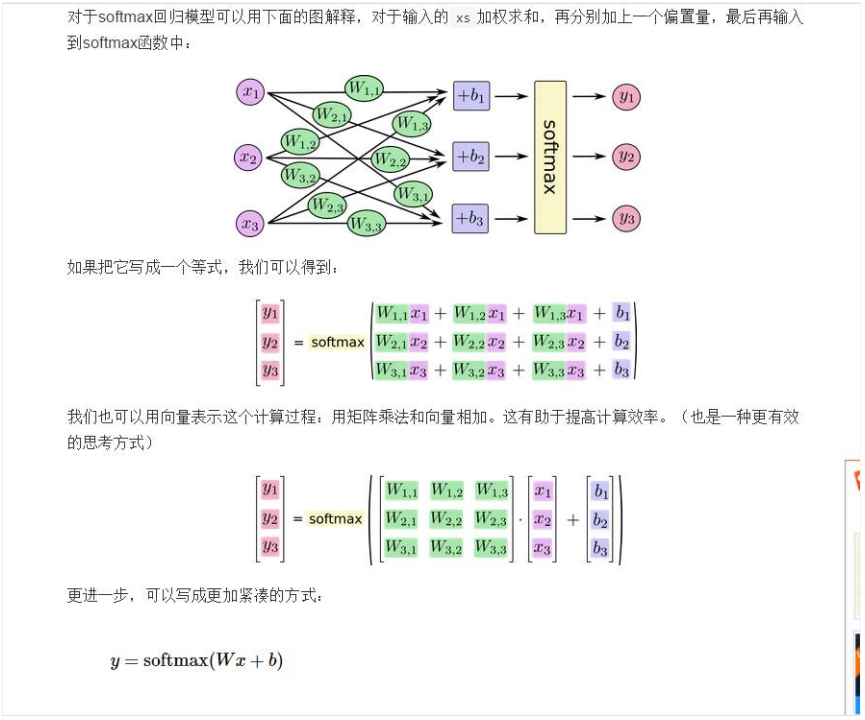
入 y=wx+b 也就是w*x=w4*d3+b y = tf.nn.bias_add(tf.matmul(d3, w4),b4))#matmul 顾名思义mat矩阵mul相乘 matmul w*x

上面的用图表示就是 X,Y ========>conv1======>relu======>drop梯度下降=======》conv2===>relu2===drop2====>conv3====>relu======>drop===full全连接 （ y=wx+b ）

官方给了 一个example 关于alexnet的 我贴一下代码 ，大家只要理解了上面讲的内容 就可以知道怎么稍微修改 就可以自己定义一个 CNN网络了

http://www.tensorfly.cn/tfdoc/tutorials/deep_cnn.html

http://www.tensorfly.cn/tfdoc/tutorials/mnist_pros.html



大家可以去执行一下，这个我执行多次了，下一章DNN ，DNN 就是CNN去掉卷积层，的深度学习网络 通过全连接层 梯度下降层构成的的DNN网络

大家有疑问就是后面训练完了怎么使用，其实训练完了 保存每一层的w b参数 起来就是一个网络模型model 然后使用model进行预测 分类了，这个后面要讲怎么保存并使用model

不知道怎么上传附件。麻烦 懒得放外联了 。其中input_data模块

地址为 https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/input_data.py

```
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 = 100000 batch_size = 128 display_step = 10 # Network Parameters n_input = 784 # MNIST data input (img shape: 28*28) n_classes = 10 # MNIST total classes (0-9 digits) dropout = 0.75 # 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) # Create model def conv2d(img, w, b):     return tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(img, w, strides=[1, 1, 1, 1], padding='SAME'),b)) def max_pool(img, k):     return tf.nn.max_pool(img, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME') def conv_net(_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, _dropout)     # Convolution Layer     conv2 = conv2d(conv1, _weights['wc2'], _biases['bc2'])     # Max Pooling (down-sampling)     conv2 = max_pool(conv2, k=2)     # Apply Dropout     conv2 = tf.nn.dropout(conv2, _dropout)
```

```
    # Fully connected layer     dense1 = tf.reshape(conv2, [-1, _weights['wd1'].get_shape().as_list()[0]]) # R
eshape conv2 output to fit dense layer input     dense1 = tf.nn.relu(tf.add(tf.matmul(dense1, _weights['w
d1']), _biases['bd1'])) # Relu activation     dense1 = tf.nn.dropout(dense1, _dropout) # Apply Dropout
    # Output, class prediction     out = tf.add(tf.matmul(dense1, _weights['out']), _biases['out'])     return o
ut # Store layers weight & bias weights = {     'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])), # 5x5 co
nv, 1 input, 32 outputs     'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])), # 5x5 conv, 32 inputs, 64 o
utputs     'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])), # fully connected, 7*7*64 inputs, 1024 o
utputs     'out': tf.Variable(tf.random_normal([1024, n_classes])) # 1024 inputs, 10 outputs (class predicti
on) } biases = {     'bc1': tf.Variable(tf.random_normal([32])),     'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),     'out': tf.Variable(tf.random_normal([n_classes])) } # Con
struct model pred = conv_net(x, weights, biases, keep_prob) # Define loss and optimizer cost = tf.reduc
e_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y)) optimizer = tf.train.AdamOptimizer(learning_ra
te=learning_rate).minimize(cost) # Evaluate model correct_pred = tf.equal(tf.argmax(pred,1), tf.argma
x(y,1)) accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32)) # Initializing the variables init = tf.initi
alize_all_variables() # Launch the graph with tf.Session() as sess:     sess.run(init)     step = 1     # Keep tra
ining until reach max iterations     while step * batch_size < training_iters:         batch_xs, batch_ys = mni
st.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 bat
ch 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_pro
b: 1.})             print "Iter " + str(step*batch_size) + ", Minibatch Loss= " + "{:.6f}".format(loss) + ", Traini
ng Accuracy= " + "{:.5f}".format(acc)         step += 1     print "Optimization Finished!"     # Calculate ac
curacy for 256 mnist test images     print "Testing Accuracy:", sess.run(accuracy, feed_dict={x: mnist.tes
t.images[:256], y: mnist.test.labels[:256], keep_prob: 1.})
```

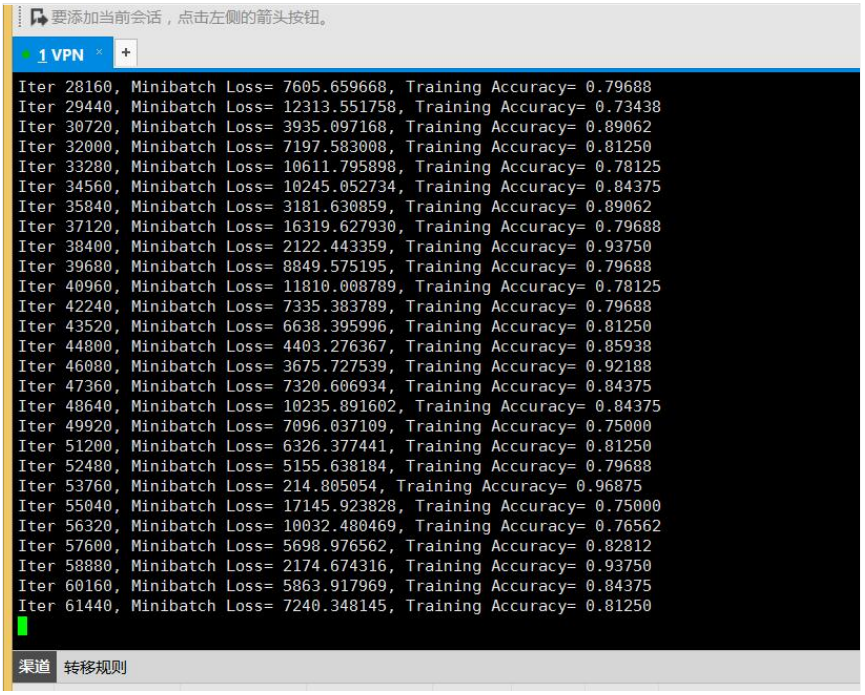
运行截图========== accuracy为精度 loss为损失函数，有多种计算方式 这里是 交叉熵" 。下面看到 accuracy第三次为0.40625 通过多次迭代后 不断的提高，运行多次 最后保存精度最高的那个为模型就好了，这里因为alexnet本来就是一个很简单的模型，所以精度也不怎么高。 我服务器是1核1G的CPU服务器所以很慢。懒得等他运行完了

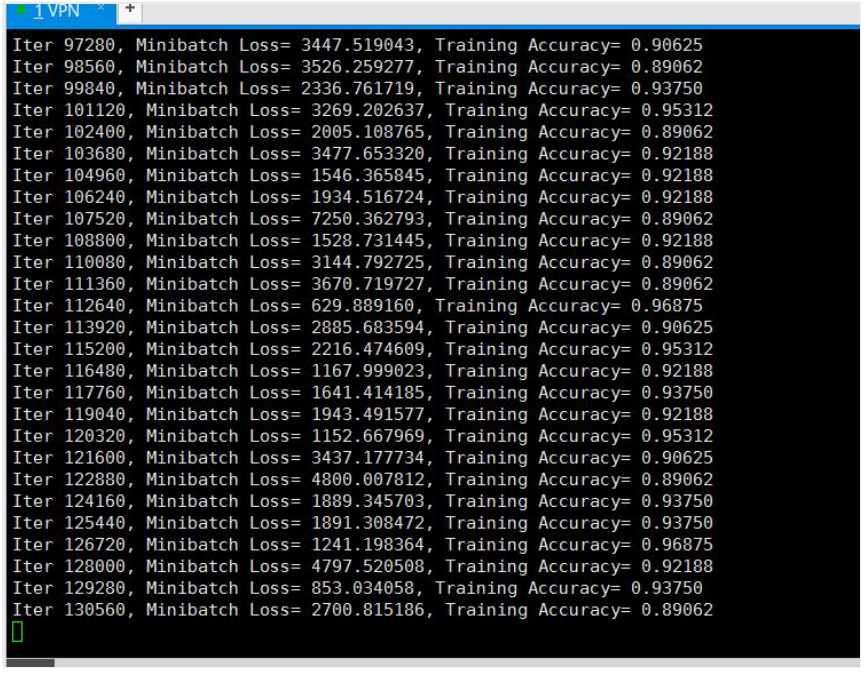
```
root@iZu1cdurunpZ:~/tensorflowtest# python alexnet.py
('Succesfully downloaded', 'train-images-idx3-ubyte.gz', 9912422, 'bytes.')
('Extracting', '/tmp/data/train-images-idx3-ubyte.gz')
('Succesfully downloaded', 'train-labels-idx1-ubyte.gz', 28881, 'bytes.')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Succesfully downloaded', 't10k-images-idx3-ubyte.gz', 1648877, 'bytes.')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Succesfully downloaded', 't10k-labels-idx1-ubyte.gz', 4542, 'bytes.')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism thre
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism th
Iter 1280, Minibatch Loss= 189811.281250, Training Accuracy= 0.14062
Iter 2560, Minibatch Loss= 79509.171875, Training Accuracy= 0.42188
Iter 3840, Minibatch Loss= 58623.582031, Training Accuracy= 0.40625
```

渠道 转移规则

ID	类型	从	到	创建时间	接收到...	发送的...

```
('Succesfully downloaded', 'train-labels-idx1-ubyte.gz', 28881, 'bytes.')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Succesfully downloaded', 't10k-images-idx3-ubyte.gz', 1648877, 'bytes.')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Succesfully downloaded', 't10k-labels-idx1-ubyte.gz', 4542, 'bytes.')
('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= 189811.281250, Training Accuracy= 0.14062
Iter 2560, Minibatch Loss= 79509.171875, Training Accuracy= 0.42188
Iter 3840, Minibatch Loss= 58623.582031, Training Accuracy= 0.40625
Iter 5120, Minibatch Loss= 34966.421875, Training Accuracy= 0.46875
Iter 6400, Minibatch Loss= 27468.613281, Training Accuracy= 0.65625
Iter 7680, Minibatch Loss= 48998.773438, Training Accuracy= 0.42188
Iter 8960, Minibatch Loss= 30511.707031, Training Accuracy= 0.59375
Iter 10240, Minibatch Loss= 23466.617188, Training Accuracy= 0.68750
Iter 11520, Minibatch Loss= 20749.056641, Training Accuracy= 0.67188
Iter 12800, Minibatch Loss= 28307.523438, Training Accuracy= 0.65625
Iter 14080, Minibatch Loss= 27130.816406, Training Accuracy= 0.59375
Iter 15360, Minibatch Loss= 15809.673828, Training Accuracy= 0.70312
Iter 16640, Minibatch Loss= 15052.324219, Training Accuracy= 0.78125
Iter 17920, Minibatch Loss= 11442.550781, Training Accuracy= 0.78125
Iter 19200, Minibatch Loss= 8619.753906, Training Accuracy= 0.81250
Iter 20480, Minibatch Loss= 11589.886719, Training Accuracy= 0.78125
Iter 21760, Minibatch Loss= 11748.119141, Training Accuracy= 0.75000
Iter 23040, Minibatch Loss= 6764.416992, Training Accuracy= 0.79688
Iter 24320, Minibatch Loss= 12309.477539, Training Accuracy= 0.75000
```


要添加当前会话，点击左侧的箭头按钮。
1 VPN
Iter 28160, Minibatch Loss= 7605.659668, Training Accuracy= 0.79688
Iter 29440, Minibatch Loss= 12313.551758, Training Accuracy= 0.73438
Iter 30720, Minibatch Loss= 3935.097168, Training Accuracy= 0.89062
Iter 32000, Minibatch Loss= 7197.583008, Training Accuracy= 0.81250
Iter 33280, Minibatch Loss= 10611.795898, Training Accuracy= 0.78125
Iter 34560, Minibatch Loss= 10245.052734, Training Accuracy= 0.84375
Iter 35840, Minibatch Loss= 3181.630859, Training Accuracy= 0.89062
Iter 37120, Minibatch Loss= 16319.627930, Training Accuracy= 0.79688
Iter 38400, Minibatch Loss= 2122.443359, Training Accuracy= 0.93750
Iter 39680, Minibatch Loss= 8849.575195, Training Accuracy= 0.79688
Iter 40960, Minibatch Loss= 11810.008789, Training Accuracy= 0.78125
Iter 42240, Minibatch Loss= 7335.383789, Training Accuracy= 0.79688
Iter 43520, Minibatch Loss= 6638.395996, Training Accuracy= 0.81250
Iter 44800, Minibatch Loss= 4403.276367, Training Accuracy= 0.85938
Iter 46080, Minibatch Loss= 3675.727539, Training Accuracy= 0.92188
Iter 47360, Minibatch Loss= 7320.606934, Training Accuracy= 0.84375
Iter 48640, Minibatch Loss= 10235.891602, Training Accuracy= 0.84375
Iter 49920, Minibatch Loss= 7096.037109, Training Accuracy= 0.75000
Iter 51200, Minibatch Loss= 6326.377441, Training Accuracy= 0.81250
Iter 52480, Minibatch Loss= 5155.638184, Training Accuracy= 0.79688
Iter 53760, Minibatch Loss= 214.805054, Training Accuracy= 0.96875
Iter 55040, Minibatch Loss= 17145.923828, Training Accuracy= 0.75000
Iter 56320, Minibatch Loss= 10032.480469, Training Accuracy= 0.76562
Iter 57600, Minibatch Loss= 5698.976562, Training Accuracy= 0.82812
Iter 58880, Minibatch Loss= 2174.674316, Training Accuracy= 0.93750
Iter 60160, Minibatch Loss= 5863.917969, Training Accuracy= 0.84375
Iter 61440, Minibatch Loss= 7240.348145, Training Accuracy= 0.81250
渠道 转移规则


1 VPN
Iter 97280, Minibatch Loss= 3447.519043, Training Accuracy= 0.90625
Iter 98560, Minibatch Loss= 3526.259277, Training Accuracy= 0.89062
Iter 99840, Minibatch Loss= 2336.761719, Training Accuracy= 0.93750
Iter 101120, Minibatch Loss= 3269.202637, Training Accuracy= 0.95312
Iter 102400, Minibatch Loss= 2005.108765, Training Accuracy= 0.89062
Iter 103680, Minibatch Loss= 3477.653320, Training Accuracy= 0.92188
Iter 104960, Minibatch Loss= 1546.365845, Training Accuracy= 0.92188
Iter 106240, Minibatch Loss= 1934.516724, Training Accuracy= 0.92188
Iter 107520, Minibatch Loss= 7250.362793, Training Accuracy= 0.89062
Iter 108800, Minibatch Loss= 1528.731445, Training Accuracy= 0.92188
Iter 110080, Minibatch Loss= 3144.792725, Training Accuracy= 0.89062
Iter 111360, Minibatch Loss= 3670.719727, Training Accuracy= 0.89062
Iter 112640, Minibatch Loss= 629.889160, Training Accuracy= 0.96875
Iter 113920, Minibatch Loss= 2885.683594, Training Accuracy= 0.90625
Iter 115200, Minibatch Loss= 2216.474609, Training Accuracy= 0.95312
Iter 116480, Minibatch Loss= 1167.999023, Training Accuracy= 0.92188
Iter 117760, Minibatch Loss= 1641.414185, Training Accuracy= 0.93750
Iter 119040, Minibatch Loss= 1943.491577, Training Accuracy= 0.92188
Iter 120320, Minibatch Loss= 1152.667969, Training Accuracy= 0.95312
Iter 121600, Minibatch Loss= 3437.177734, Training Accuracy= 0.90625
Iter 122880, Minibatch Loss= 4800.007812, Training Accuracy= 0.89062
Iter 124160, Minibatch Loss= 1889.345703, Training Accuracy= 0.93750
Iter 125440, Minibatch Loss= 1891.308472, Training Accuracy= 0.93750
Iter 126720, Minibatch Loss= 1241.198364, Training Accuracy= 0.96875
Iter 128000, Minibatch Loss= 4797.520508, Training Accuracy= 0.92188
Iter 129280, Minibatch Loss= 853.034058, Training Accuracy= 0.93750
Iter 130560, Minibatch Loss= 2700.815186, Training Accuracy= 0.89062

精度 在 增加。。。

```
Iter 106240, Minibatch Loss= 1934.516724, Training Accuracy= 0.92188
Iter 107520, Minibatch Loss= 7250.362793, Training Accuracy= 0.89062
Iter 108800, Minibatch Loss= 1528.731445, Training Accuracy= 0.92188
Iter 110080, Minibatch Loss= 3144.792725, Training Accuracy= 0.89062
Iter 111360, Minibatch Loss= 3670.719727, Training Accuracy= 0.89062
Iter 112640, Minibatch Loss= 629.889160, Training Accuracy= 0.96875
Iter 113920, Minibatch Loss= 2885.683594, Training Accuracy= 0.90625
Iter 115200, Minibatch Loss= 2216.474609, Training Accuracy= 0.95312
Iter 116480, Minibatch Loss= 1167.999023, Training Accuracy= 0.92188
Iter 117760, Minibatch Loss= 1641.414185, Training Accuracy= 0.93750
Iter 119040, Minibatch Loss= 1943.491577, Training Accuracy= 0.92188
Iter 120320, Minibatch Loss= 1152.667969, Training Accuracy= 0.95312
Iter 121600, Minibatch Loss= 3437.177734, Training Accuracy= 0.90625
Iter 122880, Minibatch Loss= 4800.007812, Training Accuracy= 0.89062
Iter 124160, Minibatch Loss= 1889.345703, Training Accuracy= 0.93750
Iter 125440, Minibatch Loss= 1891.308472, Training Accuracy= 0.93750
Iter 126720, Minibatch Loss= 1241.198364, Training Accuracy= 0.96875
Iter 128000, Minibatch Loss= 4797.520508, Training Accuracy= 0.92188
Iter 129280, Minibatch Loss= 853.034058, Training Accuracy= 0.93750
Iter 130560, Minibatch Loss= 2700.815186, Training Accuracy= 0.89062
Iter 131840, Minibatch Loss= 2522.647461, Training Accuracy= 0.92188
Iter 133120, Minibatch Loss= 1931.804199, Training Accuracy= 0.95312
Iter 134400, Minibatch Loss= 1303.242920, Training Accuracy= 0.95312
Iter 135680, Minibatch Loss= 3302.960938, Training Accuracy= 0.92188
Iter 136960, Minibatch Loss= 1135.277832, Training Accuracy= 0.95312
Iter 138240, Minibatch Loss= 356.843719, Training Accuracy= 0.96875
Iter 139520, Minibatch Loss= 682.124084, Training Accuracy= 0.95312
```

好像最后是99。2% 精确度 情况 ，手写识别 0-9 这里有98.4%了

```
Iter 171520, Minibatch Loss= 526.852173, Training Accuracy= 0.98438
Iter 172800, Minibatch Loss= 515.544922, Training Accuracy= 0.96875
Iter 174080, Minibatch Loss= 1972.009277, Training Accuracy= 0.95312
```

好吧。这次运行 精度大概97.6左右 下一章节 我修改一下系数以及 加几层层网络试试效果
，

```
Iter 168960, Minibatch Loss= 1515.357178, Training Accuracy= 0.93750
Iter 170240, Minibatch Loss= 2387.783203, Training Accuracy= 0.89062
Iter 171520, Minibatch Loss= 526.852173, Training Accuracy= 0.98438
Iter 172800, Minibatch Loss= 515.544922, Training Accuracy= 0.96875
Iter 174080, Minibatch Loss= 1972.009277, Training Accuracy= 0.95312
Iter 175360, Minibatch Loss= 1027.833252, Training Accuracy= 0.98438
Iter 176640, Minibatch Loss= 1787.946655, Training Accuracy= 0.96875
Iter 177920, Minibatch Loss= 1316.361084, Training Accuracy= 0.96875
Iter 179200, Minibatch Loss= 4223.639160, Training Accuracy= 0.89062
Iter 180480, Minibatch Loss= 579.320557, Training Accuracy= 0.93750
Iter 181760, Minibatch Loss= 119.243103, Training Accuracy= 0.98438
Iter 183040, Minibatch Loss= 2415.408691, Training Accuracy= 0.93750
Iter 184320, Minibatch Loss= 419.208099, Training Accuracy= 0.96875
Iter 185600, Minibatch Loss= 2027.504150, Training Accuracy= 0.93750
Iter 186880, Minibatch Loss= 979.027954, Training Accuracy= 0.90625
Iter 188160, Minibatch Loss= 1028.229858, Training Accuracy= 0.95312
Iter 189440, Minibatch Loss= 550.274719, Training Accuracy= 0.95312
Iter 190720, Minibatch Loss= 132.578918, Training Accuracy= 0.96875
Iter 192000, Minibatch Loss= 2373.851074, Training Accuracy= 0.92188
Iter 193280, Minibatch Loss= 419.340149, Training Accuracy= 0.98438
Iter 194560, Minibatch Loss= 1912.506592, Training Accuracy= 0.93750
Iter 195840, Minibatch Loss= 655.547241, Training Accuracy= 0.95312
Iter 197120, Minibatch Loss= 1848.252319, Training Accuracy= 0.93750
Iter 198400, Minibatch Loss= 1250.863770, Training Accuracy= 0.95312
Iter 199680, Minibatch Loss= 803.159851, Training Accuracy= 0.92188
Optimization Finished!
Testing Accuracy: 0.976562
root@iZu1cdurunpZ:~/tensorflowtest#
```


上一篇：TensorFlow 人工智能引擎 入门教程之一 基本概念以及理解

下一篇：原 TensorFlow人工智能引擎入门教程之三 实现一个自创的CNN卷积神经网络