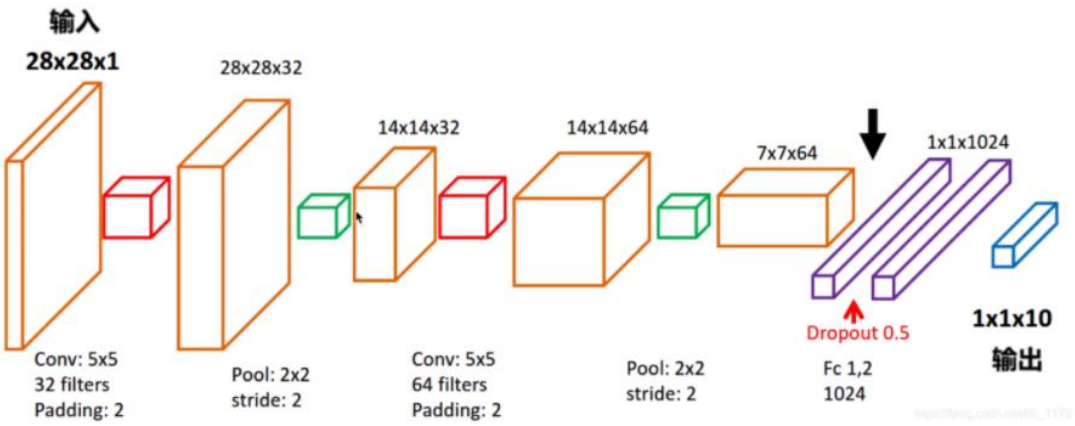
Tensorflow 中文手册
https://www.w3cschool.cn/tensorflow_python/tensorflow_python-bm7y28si.html
模型结构图:



首先明确模型的输入及输出（先不考虑 batch）
输入：一张手写数字图（28x28x1 像素矩阵） 1 是通道数
输出：预测的数字（1x10 的 one-hot 向量）
one hot 编码是将类别变量转换为机器学习算法易于利用的一种形式的过程，比如输出[0,0,0,0,0,0,0,0,1,0]代表数字“8”

各层的维度说明（先不考虑 batch）
输入层（28 x28 x1）
卷积层 1 的输出（28x28x32）(32 filters)
pooling 层 1 的输出（14x14x32）
卷积层 2 的输出（14x14x64）（64 filters)
pooling 层 2 的输出（7x7x64）
全连接层 1 的输出（1x1024）
全连接层 2 含 softmax 的输出（1x10）

注意，训练时采用 batch，只是加了一个维度而已，比如(28x28x1)→(100x28x28x1) batch=100

详细代码讲解
下载 mnist 手写数字图片数据集:

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

若报错可自行前往 http://yann.lecun.com/exdb/mnist/下载（或者其他地址），只要将四个压缩文件都放进 MNIST_data 文件夹即可，包含了四个部分:

- **训练数据集**：train-images-idx3-ubyte.gz （9.45 MB，包含60,000个样本）。
- **训练数据集标签**：train-labels-idx1-ubyte.gz（28.2 KB，包含60,000个标签）。
- **测试数据集**：t10k-images-idx3-ubyte.gz（1.57 MB ，包含10,000个样本）。
- **测试数据集标签**：t10k-labels-idx1-ubyte.gz（4.43 KB，包含10,000个样本的标签）。

名称	修改日期	类型	大小
t10k-images-idx3-ubyte	2020/12/16 0:07	WinRAR 压缩文...	1,611 KB
t10k-labels-idx1-ubyte	2020/12/16 0:07	WinRAR 压缩文...	5 KB
train-images-idx3-ubyte	2020/12/16 0:05	WinRAR 压缩文...	9,681 KB
train-labels-idx1-ubyte	2020/12/16 0:06	WinRAR 压缩文...	29 KB

Tensorflow 读取的 mnist 的数据形式（Datasets）
原训练集分出了 5000 作为验证集（实验中未使用）
训练集（train\0）的数量：55000
验证集（validation\1）的数量：5000
测试集（test\2）的数量：10000

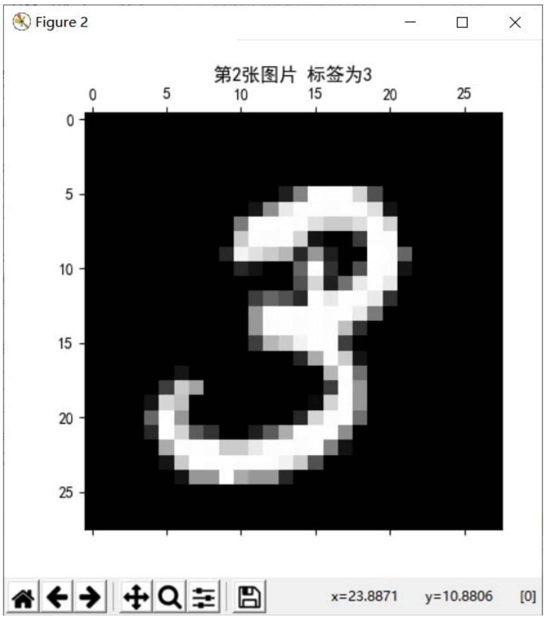
```
mnist = {Datasets: 3} Datasets(train=<tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet ob
> test = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x000002
> train = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x000002
> validation = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x0
> 0 = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x00000238
> 1 = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x00000238
> 2 = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x00000238
  __len__ = {int} 3
```

```
v mnist = {Datasets: 3} Datasets(train=<tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x000002
  v test = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x0000023842E76320>
      epochs_completed = {int} 0
    > images = {ndarray: (10000, 784)} [[0. 0. 0. ... 0. 0. 0.], [0. 0. 0. ... 0. 0. 0.], [0. 0. 0. ... 0. 0. 0.], ..., [0. 0. 0. ... 0. 0. 0.], [0.
    > labels = {ndarray: (10000, 10)} [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.], [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.], [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.], [1. 0.
      num_examples = {int} 10000
    > Protected Attributes
  v train = {DataSet} <tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet object at 0x0000023842E76390>
      epochs_completed = {int} 3
    > images = {ndarray: (55000, 784)} [[0. 0. 0. ... 0. 0. 0.], [0. 0. 0. ... 0. 0. 0.], [0. 0. 0. ... 0. 0. 0.], ..., [0. 0. 0. ... 0. 0. 0.], [0.
    > labels = {ndarray: (55000, 10)} [[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.], [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.], [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.], [1. 0.
      num_examples = {int} 55000
    > Protected Attributes
```

补充：可视化 train 数据集图片

```python
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
train_img = mnist.train.images
train_label = mnist.train.labels

for i in range(5):
    img = np.reshape(train_img[i, :], (28, 28))
    label = np.argmax(train_label[i, :])
```

```
    plt.matshow(img, cmap = plt.get_cmap('gray'))
    plt.title('第%d 张图片 标签为%d' %(i+1,label))
    plt.show()
```



卷积层 1 代码:

```
## conv1 layer 含 pool ##
W_conv1 = weight_variable([5, 5, 1, 32])
# 初始化 W_conv1 为[5,5,1,32]的张量 tensor, 表示卷积核大小为 5*5, 1 表示图像通道数（输入）, 32 表示卷积核个数即输出 32 个特征图（即下一层的输入通道数）
# 张量说明:
# 3                                   这个 0 阶张量就是标量, shape=[]
# [1., 2., 3.]                         这个 1 阶张量就是向量, shape=[3]
# [[1., 2., 3.], [4., 5., 6.]]           这个 2 阶张量就是二维数组, shape=[2, 3]
# [[[1., 2., 3.]], [[7., 8., 9.]]]       这个 3 阶张量就是三维数组, shape=[2, 1, 3]
# 即有几层中括号
b_conv1 = bias_variable([32])
# 偏置项, 参与 conv2d 中的加法, 维度会自动扩展到 28x28x32（广播）
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# output size 28x28x32
h_pool1 = max_pool_2x2(h_conv1)  # output size 14x14x32 卷积操作使用 padding 保持维度不变, 只靠 pool 降维
```

其中:

```
xs = tf.placeholder(tf.float32, [None, 784], name='x_input')
ys = tf.placeholder(tf.float32, [None, 10], name='y_input')
x_image = tf.reshape(xs, [-1, 28, 28, 1])
# 创建两个占位符, xs 为输入网络的图像, ys 为输入网络的图像标签
# 输入 xs(二维张量,shape 为[batch, 784])变成 4d 的 x_image, x_image 的 shape 应该是[batch,28,28,1], 第四维是通道数 1
# -1 表示自动推测这个维度的 size
# reshape 成了 conv2d 需要的输入形式; 若是直接进入全连接层, 则没必要 reshape
```

——————————————————以上使用到的函数的定义——————————————————

注意：tensorflow 的变量必须定义为 tf.Variable 类型

```
def weight_variable(shape):
    # tf.truncated_normal 从截断的正态分布中输出随机值.
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
    # 卷积核移动步长为 1,填充 padding 类型为 SAME,可以不丢弃任何像素点, VALID 丢弃边
缘像素点
    # 计算给定的 4-D input 和 filter 张量的 2-D 卷积
    # input shape [batch, in_height, in_width, in_channels]
    # filter shape [filter_height, filter_width, in_channels, out_channels]
    # stride 对应在这四维上的步长，默认[1,x,y,1]
def max_pool_2x2(x):
    # 采用最大池化，也就是取窗口中的最大值作为结果
    # x 是一个 4 维张量，shape 为[batch,height,width,channels]
    # ksize 表示 pool 窗口大小为 2x2,也就是高 2，宽 2
    # strides，表示在 height 和 width 维度上的步长都为 2
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')
```

——

卷积层 2 代码:

```
## conv2 layer 含 pool##
W_conv2 = weight_variable([5, 5, 32, 64])  # 同 conv1，不过卷积核数增为 64
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
# output size 14x14x64
h_pool2 = max_pool_2x2(h_conv2)
# output size 7x7x64
```

全连接层 1 代码:

```
## fc1 layer ##
# 含 1024 个神经元，初始化（3136，1024）的 tensor
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
# 将 conv2 的输出 reshape 成[batch, 7*7*16]的张量，方便全连接层处理
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

其中：

```
xs = tf.placeholder(tf.float32, [None, 784], name='x_input')
ys = tf.placeholder(tf.float32, [None, 10], name='y_input')
keep_prob = tf.placeholder(tf.float32)
x_image = tf.reshape(xs, [-1, 28, 28, 1])
keep_prob_rate = 0.5

# 在机器学习的模型中，如果模型的参数太多，而训练样本又太少，训练出来的模型很容易产生过拟合的现象。
# 在训练神经网络的时候经常会遇到过拟合的问题，过拟合具体表现在：模型在训练数据上损失函数较小，预测准确率较高；但是在测试数据上损失函数比较大，预测准确率较低。
# 神经元按 1-keep_prob 概率置 0，否则以 1/keep_prob 的比例缩放该元（并非保持不变）
# 这是为了保证神经元输出激活值的期望值与不使用 dropout 时一致，结合概率论的知识来具体看一下：假设一个神经元的输出激活值为 a，在不使用 dropout 的情况下，其输出期望值为 a，如果使用了 dropout，神经元就可能有保留和关闭两种状态，把它看作一个离散型随机变量，符合概率论中的 0-1 分布，其输出激活值的期望变为 p*a+(1-p)*0=pa，为了保持测试集与训练集神经元输出的分布一致，可以在训练时除以此系数或者测试时乘以此系数，或者在测试时乘以该系数
```

全连接层 2 代码：

```
## fc2 layer 含 softmax 层##
# 含 10 个神经元，初始化（1024，10）的 tensor
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
prediction = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# 交叉熵函数
```

$$loss = -\sum_{i=1}^{n} y_i \log(y_{i\_})$$

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                              reduction_indices=[1]))
```

补充 tf.reduce_mean

```
reduce_mean(
    input_tensor,
    axis=None,
    keep_dims=False,
    name=None,
    reduction_indices=None
)
```

计算张量的(各个维度上)元素的平均值，例如

x = tf.constant([[1., 1.], [2., 2.]])
tf.reduce_mean(x) # 1.5
tf.reduce_mean(x, 0) # [1.5, 1.5]
tf.reduce_mean(x, 1) # [1., 2.]T
0 代表输出是个行向量，那么就是各行每个维度取 mean

```python
# 使用 ADAM 优化器来做梯度下降,学习率 learning_rate=0.0001
learning_rate = 1e-4
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

# 模型训练后，计算测试集准确率
def compute_accuracy(v_xs, v_ys):
    global prediction
    # y_pre 将 v_xs(test)输入模型后得到的预测值 (10000,10)
    y_pre = sess.run(prediction, feed_dict={xs: v_xs, keep_prob: 1})
    # argmax(axis) axis = 1 返回结果为：数组中每一行最大值所在“列”索引值
    # tf.equal 返回布尔值，correct_prediction (10000，1)
    correct_prediction = tf.equal(tf.argmax(y_pre, 1), tf.argmax(v_ys, 1))
    # tf.cast 将 bool 转成 float32, tf.reduce_mean 求均值，作为 accuracy 值(0 到 1)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    result = sess.run(accuracy, feed_dict={xs: v_xs, ys: v_ys, keep_prob: 1})
    return result
```

TensorFlow 程序通常被组织成一个构建阶段（graph）和一个执行阶段.
上述阶段就是构建阶段，现在进入执行阶段，反复执行图中的训练操作，首先需要创建一个 Session 对象，如
sess = tf.Session() ****** sess.close()
Session 对象在使用完后需要关闭以释放资源. 除了显式调用 close 外，也可以使用 "with" 代码块 来自动完成关闭动作,如下

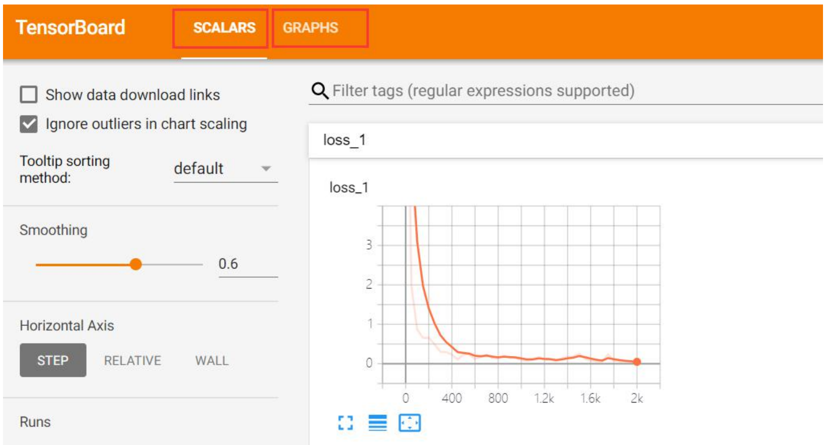
```python
with tf.Session() as sess:
    # 初始化图中所有 Variables
    init = tf.global_variables_initializer()
    sess.run(init)
    # 总迭代次数(batch)为 max_epoch=1000,每次取 100 张图做 batch 梯度下降

    for i in range(max_epoch):
        # mnist.train.next_batch 默认 shuffle=True，随机读取，batch 大小为 100
        batch_xs, batch_ys = mnist.train.next_batch(100)
        # 此 batch 是个 2 维 tuple，batch[0]是(100，784)的样本数据数组，batch[1]是
(100，10)的样本标签数组，分别赋值给 batch_xs, batch_ys
        sess.run(train_step, feed_dict={xs: batch_xs, ys: batch_ys, keep_prob:
keep_prob})
    # 暂时不进行赋值的元素叫占位符（如 xs、ys），run 需要它们时得赋值，feed_dict 就
是用来赋值的，格式为字典型
        if (i+1) % 50 == 0:
            print("step %d, test accuracy %g" % (i+1, compute_accuracy(
                mnist.test.images, mnist.test.labels)))
```

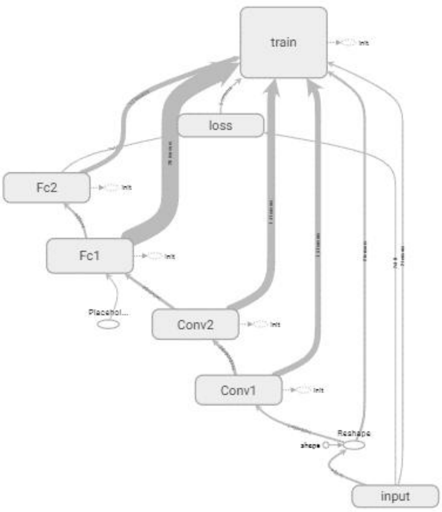
利用自带的 tensorboard 可视化模型（深入理解图的概念）

tensorboard 支持 8 种可视化，也就是上图中的 8 个选项卡，它们分别是：

- SCALARS：标量曲线。例如准确率、损失率、权重和偏置等变化。在代码中tf.summary.scalar()定义需要在这个选项卡下展示的标量。
- IMAGES：数据图像。对于图像分类问题，可以将输入或者训练过程中的图片展示出来。在代码中用tf.summary.image()定义需要在这个选项卡中展示的数据（直接绘制成图像展示），默认是绘制3张图片。
- AUDIO：音频数据。还没有做过语音分析的例子，估计同上。
- GRAPHS：tensorflow的数据流图。数据流图的完善性，都需要在程序中详细定义。用with tf.name_scope()或者with tf.name_scope() as scope来定义。
- DISTRIBUTIONS：数据分布图。可以用来显示激活前和激活后数据的分布情况，辅助设计分析。
- HISTOGRAMS：数据柱状图。在代码中用tf.summary.histogram()定义。
- EMBEDDINGS：用于文本分析，展示词向量的投影分布（例如word2vec）

tensorboard 通过运行一个本地服务器，监听 6006 端口，在浏览器发出请求时，分析训练时记录的数据，绘制训练过程中的数据曲线、图像。





以可视化 loss（scalars）、graphs 为例：

为了在 graphs 中展示节点名称，在设计网络时可用 with tf.name_scope()限定命名空间

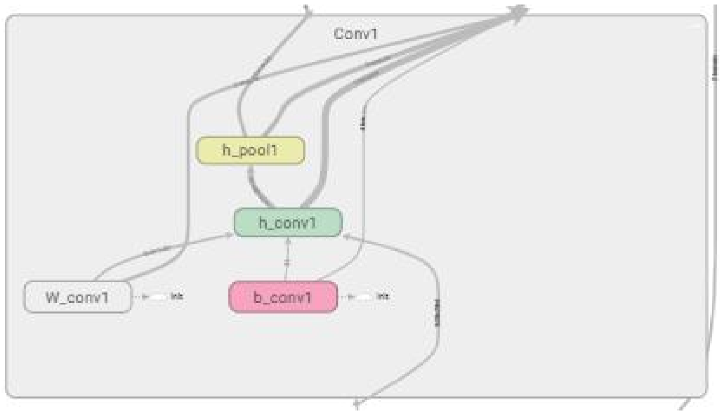
以第一个卷积层为例：

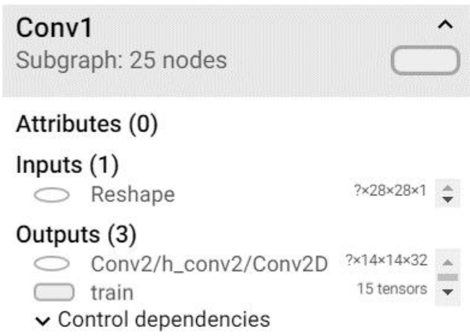
```
with tf.name_scope('Conv1'):
    with tf.name_scope('W_conv1'):
```

```
        W_conv1 = weight_variable([5, 5, 1, 32])
    with tf.name_scope('b_conv1'):
        b_conv1 = bias_variable([32])
    with tf.name_scope('h_conv1'):
        h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
    with tf.name_scope('h_pool1'):
        h_pool1 = max_pool_2x2(h_conv1)
```

同样地，对所有节点进行命名

如下，Conv1 中的名称即命名结果






```
with tf.name_scope('loss'):
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                                  reduction_indices=[1]))
```

在 with tf.Session() as sess 中添加

```
losssum = tf.summary.scalar('loss', cross_entropy)
# loss 计入 summary 中，可以被统计
writer = tf.summary.FileWriter("", graph=sess.graph)
# tf.summary.FileWriter 指定一个文件用来保存图
```

在 if i % 50 == 0 中添加

```
summery= sess.run(losssum, feed_dict={xs: batch_xs, ys: batch_ys, keep_prob:
keep_prob_rate})
writer.add_summary(summery, i)
# add_summary () 方法将训练过程数据保存在 filewriter 指定的文件中
```

在 Terminal 中输入

tensorboard --logdir=E:\cnn_mnist

```
Terminal:   Local ×   +
libifcoremd.dll     00007FFE92E043E4  Unknown               Unknown  Unknown
KERNELBASE.dll      00007FFF3F1862A3  Unknown               Unknown  Unknown
KERNEL32.DLL        00007FFF419F7C24  Unknown               Unknown  Unknown
ntdll.dll           00007FFF41C8D4D1  Unknown               Unknown  Unknown

(tfgpu) E:\XMU_SPEECH\lab_mnist_cnn\lab\exercise\chap5_CNN>tensorboard --logdir=E:\XMU_SPEECH\lab_mnist_cnn\lab\exercise\chap5_CNN
2021-01-08 03:09:25.188160: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cu
TensorBoard 1.15.0 at http://LAPTOP-R9006LH5:6006/ (Press CTRL+C to quit)
```

将网址中的 LAPTOP-R9006LH5 改为 localhost，复制在浏览器中打开即可

附录（完整代码 1）：

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
def weight_variable(shape):
    # tf.truncated_normal 从截断的正态分布中输出随机值.
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
# 偏置初始化
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
# 使用 tf.nn.conv2d 定义 2 维卷积
def conv2d(x, W):
    # 卷积核移动步长为 1,填充 padding 类型为 SAME,简单地理解为以 0 填充边缘, VALID 采用不填充的方式，多余地进行丢弃
    # 计算给定的 4-D input 和 filter 张量的 2-D 卷积
    # input shape [batch, in_height, in_width, in_channels]
    # filter shape [filter_height, filter_width, in_channels, out_channels]
    # stride 长度为 4 的 1-D 张量,input 的每个维度的滑动窗口的步幅
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    # 采用最大池化，也就是取窗口中的最大值作为结果
    # x 是一个 4 维张量，shape 为[batch,height,width,channels]
    # ksize 表示 pool 窗口大小为 2x2,也就是高 2，宽 2
    # strides，表示在 height 和 width 维度上的步长都为 2
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')
# 计算 test set 的 accuracy，v_xs (10000,784), y_ys (10000,10)
def compute_accuracy(v_xs, v_ys):
    global prediction
    # y_pre 将 v_xs 输入模型后得到的预测值 (10000,10)
    y_pre = sess.run(prediction, feed_dict={xs: v_xs, keep_prob: 1})
    # argmax(axis) axis = 1 返回结果为：数组中每一行最大值所在“列”索引值
    # tf.equal 返回布尔值，correct_prediction (10000，1)
    correct_prediction = tf.equal(tf.argmax(y_pre, 1), tf.argmax(v_ys, 1))
    # tf.cast 将 bool 转成 float32, tf.reduce_mean 求均值，作为 accuracy 值(0 到 1)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    result = sess.run(accuracy, feed_dict={xs: v_xs, ys: v_ys, keep_prob: 1})
    return result

xs = tf.placeholder(tf.float32, [None, 784], name='x_input')
ys = tf.placeholder(tf.float32, [None, 10], name='y_input')
max_epoch = 2000
```

```python
keep_prob = tf.placeholder(tf.float32)
x_image = tf.reshape(xs, [-1, 28, 28, 1])
keep_prob_rate = 0
# 卷积层 1
# input size 28x28x1 （以一个样本为例）batch=100 则 100x28x28x1
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# output size 28x28x32
h_pool1 = max_pool_2x2(h_conv1)  # output size 14x14x32 卷积操作使用 padding 保持维度不变，只靠 pool 降维

# 卷积层 2
W_conv2 = weight_variable([5, 5, 32, 64])  # 同 conv1，不过卷积核数增为 64
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
# output size 14x14x64
h_pool2 = max_pool_2x2(h_conv2)
# output size 7x7x64

h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
# 全连接层 1
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
# 将 conv2 的输出 reshape 成[batch, 7*7*16]的张量，方便全连接层处理
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# 全连接层 2
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
prediction = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                              reduction_indices=[1]))
learning_rate = 1e-4
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

with tf.Session() as sess:
    # 初始化图中所有 Variables
    init = tf.global_variables_initializer()
    sess.run(init)
    # 总迭代次数(batch)为 max_epoch=1000,每次取 100 张图做 batch 梯度下降
    print("step 0, test accuracy %g" % (compute_accuracy(
        mnist.test.images, mnist.test.labels)))
```

```
    for i in range(max_epoch):
        # mnist.train.next_batch 默认 shuffle=True，随机读取，batch 大小为 100
        batch_xs, batch_ys = mnist.train.next_batch(100)
        # 此 batch 是个 2 维 tuple，batch[0]是(100，784)的样本数据数组，batch[1]是(100，10)的样本标签数组，分别赋值给 batch_xs，batch_ys
        sess.run(train_step, feed_dict={xs: batch_xs, ys: batch_ys, keep_prob: keep_prob_rate})
        # 暂时不进行赋值的元素叫占位符（如 xs、ys），run 需要它们时得赋值，feed_dict 就是用来赋值的，格式为字典型
        if (i + 1) % 50 == 0:
            print("step %d, test accuracy %g" % (i + 1, compute_accuracy(
                mnist.test.images, mnist.test.labels)))
```

附录（完整代码 2 带 tensorboad 可视化）：

```
import tensorflow.compat.v1 as tf
# import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import os

# 导入 input_data 用于自动下载和安装 MNIST 数据集
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
learning_rate = 1e-4
keep_prob_rate = 0.7  # drop out 比例（补偿系数）
# 为了保证神经元输出激活值的期望值与不使用 dropout 时一致，我们结合概率论的知识来具体看一下：假设一个神经元的输出激活值为 a，
# 在不使用 dropout 的情况下，其输出期望值为 a，如果使用了 dropout，神经元就可能有保留和关闭两种状态，把它看作一个离散型随机变量，
# 它就符合概率论中的 0-1 分布，其输出激活值的期望变为 p*a+(1-p)*0=pa，为了保持测试集与训练集神经元输出的分布一致，可以训练时除以此系数或者测试时乘以此系数
# 即输出节点按照 keep_prob 概率置 0，否则以 1/keep_prob 的比例缩放该节点（而并非保持不变）
max_epoch = 2000


# 权重矩阵初始化
def weight_variable(shape):
    # tf.truncated_normal 从截断的正态分布中输出随机值.
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
# 偏置初始化
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

```
# 使用 tf.nn.conv2d 定义 2 维卷积
def conv2d(x, W):
    # 卷积核移动步长为 1,填充 padding 类型为 SAME,简单地理解为以 0 填充边缘, VALID 采用不填充的方式，多余地进行丢弃
    # 计算给定的 4-D input 和 filter 张量的 2-D 卷积
    # input shape [batch, in_height, in_width, in_channels]
    # filter shape [filter_height, filter_width, in_channels, out_channels]
    # stride 长度为 4 的 1-D 张量,input 的每个维度的滑动窗口的步幅
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    # 采用最大池化，也就是取窗口中的最大值作为结果
    # x 是一个 4 维张量，shape 为[batch,height,width,channels]
    # ksize 表示 pool 窗口大小为 2x2,也就是高 2，宽 2
    # strides，表示在 height 和 width 维度上的步长都为 2
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')
# 计算 test set 的 accuracy，v_xs (10000,784), y_ys (10000,10)
def compute_accuracy(v_xs, v_ys):
    global prediction
    # y_pre 将 v_xs 输入模型后得到的预测值 (10000,10)
    y_pre = sess.run(prediction, feed_dict={xs: v_xs, keep_prob: 1})
    # argmax(axis) axis = 1 返回结果为：数组中每一行最大值所在“列”索引值
    # tf.equal 返回布尔值，correct_prediction (10000，1)
    correct_prediction = tf.equal(tf.argmax(y_pre, 1), tf.argmax(v_ys, 1))
    # tf.cast 将 bool 转成 float32, tf.reduce_mean 求均值，作为 accuracy 值(0 到 1)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    result = sess.run(accuracy, feed_dict={xs: v_xs, ys: v_ys, keep_prob: 1})
    return result
with tf.name_scope('input'):
    xs = tf.placeholder(tf.float32, [None, 784], name='x_input')
    ys = tf.placeholder(tf.float32, [None, 10], name='y_input')
keep_prob = tf.placeholder(tf.float32)
x_image = tf.reshape(xs, [-1, 28, 28, 1])
# 输入转化为 4D 数据，便于 conv 操作
# 把输入 x(二维张量,shape 为[batch, 784])变成 4d 的 x_image，x_image 的 shape 应该是[batch,28,28,1]，第四维是通道数 1
# -1 表示自动推测这个维度的 size

## conv1 layer ##
with tf.name_scope('Conv1'):
    with tf.name_scope('W_conv1'):
        W_conv1 = weight_variable([5, 5, 1, 32])
        # 初始化 W_conv1 为[5,5,1,32]的张量 tensor，表示卷积核大小为 5*5，1 表示图像通道数，6 表示卷积核个数即输出 6 个特征图
```

```
        # 3                              这个 0 阶张量就是标量，shape=[]
        # [1., 2., 3.]                    这个 1 阶张量就是向量，shape=[3]
        # [[1., 2., 3.], [4., 5., 6.]]       这个 2 阶张量就是二维数组，shape=[2,
3]
        # [[[1., 2., 3.]], [[7., 8., 9.]]]     这个 3 阶张量就是三维数组，shape=[2,
1, 3]
        # 即有几层中括号
    with tf.name_scope('b_conv1'):
        b_conv1 = bias_variable([32])
    with tf.name_scope('h_conv1'):
        h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)  # output size
28x28x32 5x5x1 的卷积核作用在 28x28x1 的二维图上
    with tf.name_scope('h_pool1'):
        h_pool1 = max_pool_2x2(h_conv1)  # output size 14x14x32 卷积操作使用
padding 保持维度不变，只靠 pool 降维

## conv2 layer ##
with tf.name_scope('Conv2'):
    with tf.name_scope('W_conv2'):
        W_conv2 = weight_variable([5, 5, 32, 64])  # patch 5x5, in size 32, out
size 64
    with tf.name_scope('b_conv2'):
        b_conv2 = bias_variable([64])
    with tf.name_scope('h_conv2'):
        h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)  # output size
14x14x64
    with tf.name_scope('h_pool2'):
        h_pool2 = max_pool_2x2(h_conv2)  # output size 7x7x64
#  全连接层 1
## fc1 layer ##
# 1024 个神经元的全连接层
with tf.name_scope('Fc1'):
    with tf.name_scope('W_fc1'):
        W_fc1 = weight_variable([7 * 7 * 64, 1024])
    with tf.name_scope('b_fc1'):
        b_fc1 = bias_variable([1024])
    with tf.name_scope('h_pool2_flat'):
        h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
    with tf.name_scope('h_fc1'):
        h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
    with tf.name_scope('h_fc1_drop'):
        h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# 全连接层 2
```

```
## fc2 layer ##
with tf.name_scope('Fc2'):
    with tf.name_scope('W_fc2'):
        W_fc2 = weight_variable([1024, 10])
    with tf.name_scope('b_fc2'):
        b_fc2 = bias_variable([10])
    with tf.name_scope('prediction'):
        prediction = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# 交叉熵函数
with tf.name_scope('loss'):
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
                                                  reduction_indices=[1]))

# 使用 ADAM 优化器来做梯度下降,学习率为 learning_rate0.0001
with tf.name_scope('train'):
    train_step =
tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

with tf.Session() as sess:
    # 初始化图中所有 Variables
    init = tf.global_variables_initializer()
    sess.run(init)
    losssum = tf.summary.scalar('loss', cross_entropy)  # 若 placeholde 报错,
则 rerun
    # merged = tf.summary.merge_all() # 只有 loss 值需要统计, 故不需要 merge
    writer = tf.summary.FileWriter("", graph=sess.graph)
    # tf.summary.FileWriter 指定一个文件用来保存图
    # writer.close()
    #writer = tf.summary.FileWriter("", sess.graph) # 重新保存图时, 要在 console
里 rerun, 否则 graph 会累计 cmd 进入 tfgpu 环境 tensorboard --logdir=路径, 将网址中
的 laptop 替换为 localhost
    for i in range(max_epoch + 1):
        # mnist.train.next_batch 默认 shuffle=True, 随机读取
        batch_xs, batch_ys = mnist.train.next_batch(100)
        # 此 batch 是个 2 维 tuple, batch[0]是(100, 784)的样本数据数组, batch[1]是
(100, 10)的样本标签数组
        sess.run(train_step, feed_dict={xs: batch_xs, ys: batch_ys, keep_prob:
keep_prob_rate})
        if i % 50 == 0:
            summery= sess.run(losssum, feed_dict={xs: batch_xs, ys: batch_ys,
keep_prob: keep_prob_rate})
            # summary = sess.run(merged, feed_dict={xs: batch_xs, ys: batch_ys,
keep_prob: keep_prob_rate})
```

```
writer.add_summary(summery, i)
# add_summary（）方法将训练过程数据保存在 filewriter 指定的文件中
print("step %d, test accuracy %g" % (i, compute_accuracy(
    mnist.test.images, mnist.test.labels)))
```