TensorFlow是个什么东东？

编写：土豆 MoreZheng SlyneD
校对：碧海听滔 Molly
总校对与审核：寒小阳

在前面的作业中你已经写了很多代码来实现很多的神经网络功能。Dropout, Batch Norm 和 2D卷积是深度学习在计算机视觉中的一些重活。你已经很努力地让你的代码有效率以及向量化。

对于这份作业的最后一个部分，我们不会继续探讨之前的代码，而是转到两个流行的深度学习框架之一。在这份Notebook中，主要是Tensorflow（在其他的notebook中，还会有PyTorch代码）.

TensorFlow是什么？

Tensorflow是基于Tensor来执行计算图的系统，对于变量(Variables)有原生的自动反向求导的功能。在它里面，我们用的n维数组的tensor相当于是numpy中的ndarray。

为什么用tensorflow?

- 我们的代码将会运行在GPU上，因此会在训练的时候快很多。不过，编写在GPU上运行的程序模块的方法不在这门课的范围内。
- 我们希望你为你的项目使用这些框架，这样比起你自己编写基础代码，要更加有效率。
- 我们希望你们可以站在巨人的肩膀上！TensorFlow和PyTorch都是优秀的框架，可以让你的生活更轻松，既然你已经明白了他们的原理，你就可以随意地使用它们了。
- 我们希望你可以编写一些能在学术或工业界可以使用的深度学习代码。

我该怎么学习TensorFlow？

TensorFlow已经有许多优秀的教程，包括来自google自己的那些。

另外，这个notebook也会带领你过一遍在TensorFlow中，训练模型所需要用到的许多东西。如果你需要学习更多内容，或者了解更多细节，可以去看本Notebook的结尾部分，那里可以找到一些有用的教程链接。

加载数据

```
import tensorflow as tf
import numpy as np
import math
import timeit
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from cs231n.data_utils import load_CIFAR10


def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=10000):
```

```
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the two-layer neural net classifier. These are the same steps as
    we used for the SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = range(num_training, num_training + num_validation)
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = range(num_training)
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = range(num_test)
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    return X_train, y_train, X_val, y_val, X_test, y_test


# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 32, 32, 3)
Train labels shape:  (49000,)
Validation data shape:  (1000, 32, 32, 3)
Validation labels shape:  (1000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

案例模型

一些实用的建议

我们的图像数据格式是:N x H x W x C, 其中

- N 是数据点的个数

- H 是每张图片的高度(单位：像素)
- W 是每张图片的宽度(单位: 像素)
- C 是通道的数量 (通常是3：R, G, B)

这是一种正确的表示数据的方式，比如当我们做一些像是2D卷积这样的操作，需要理解空间上相邻的像素点。但是，当我们把图像数据放到全连接的仿射层(affine layers)中时，我们希望一个数据样本可以用一个向量来表示，这个时候，把数据分成不同的通道、行和列就不再有用了。

案例模型本尊

训练你自己模型的第一步就是要定义它的结构。 这里有一个定义在TensorFlow中的卷积神经网络的例子 -- 试着搞清楚每一行都在做什么，要记住，每一行都建立在前一行之上。 目前我们还没有训练什么东西 -- 这后面会讲到 -- 现在， 我们希望你能够明白这些东西都是怎么建立起来的。

在这个例子里面，你们会看到2D的卷积层， ReLU激活层，和全连接层（线性的）。 你们也会看到Hinge loss损失函数， 以及Adam优化器是如何使用的。

确保要明白为什么线性层的参数是5408和10。

TensorFlow细节

在TensorFlow中，像我们前面的Notebook一样，我们首先要初始化我们的变量，然后是我们的模型。

```
# clear old variables
tf.reset_default_graph()

# setup input (e.g. the data that changes every batch)
# The first dim is None, and gets sets automatically based on batch size fed in
# 设置输入，比如每个batch要输入的数据
# 第一维是None，可以根据输入的batch size自动改变。

X = tf.placeholder(tf.float32, [None, 32, 32, 3])
y = tf.placeholder(tf.int64, [None])
is_training = tf.placeholder(tf.bool)

def simple_model(X,y):
    # define our weights (e.g. init_two_layer_convnet)
    #  定义权重W
    # setup variables
    # 设置变量
    Wconv1 = tf.get_variable("Wconv1", shape=[7, 7, 3, 32])
    bconv1 = tf.get_variable("bconv1", shape=[32])
    W1 = tf.get_variable("W1", shape=[5408, 10])
    b1 = tf.get_variable("b1", shape=[10])

    # define our graph (e.g. two_layer_convnet)
    # 定义我们的图


    # 这里我们需要用到conv2d函数，建议大家仔细阅读官方文档
    # tf.nn.conv2d()  https://www.tensorflow.org/api_docs/python/tf/nn/conv2d
    # conv2d(input,filter,strides,padding,use_cudnn_on_gpu=None,data_format=None,name=None)

    # input ： [batch, in_height, in_width, in_channels]
```

```
    # filter/kernel: [filter_height, filter_width, in_channels, out_channels]
    # strides: 长度为4的1维tensor，用来指定在每一个维度上的滑动的窗口滑动的步长
    # 水平或者垂直滑动通常会指定strides = [1,stride,,stride,1]
    # padding: 'VALID' 或者是 'SAME'
    # data_format: 数据的输入格式，默认是‘NHWC’


    # 根据输出的大小的公式：(W-F+2P)/S + 1
    # W: 图像宽度   32
    # F：Filter的宽度  7
    # P: padding了多少  0
    # padding='valid' 就是不padding  padding='same' 自动padding若干个行列使得输出的feature map和原输入feature map的尺寸一致
    # S: stride 步长  2

    a1 = tf.nn.conv2d(X, Wconv1, strides=[1,2,2,1], padding='VALID') + bconv1
    # (W-F+2P)/S + 1 = (32 - 7 + 2*0)/2 + 1 = 13
    # 那么输出的feature map的尺寸就是 13 * 13 * 32 = 5408   （Wconv1 有32个out channels，也就是说有32个filters）

    h1 = tf.nn.relu(a1) # 对a1中的每个神经元加上激活函数relu
    h1_flat = tf.reshape(h1,[-1,5408])  # reshape h1，把feature map展开成 batchsize * 5408
    y_out = tf.matmul(h1_flat,W1) + b1  # 得到输出的logits: y_out
    return y_out

y_out = simple_model(X,y)

# define our loss
# 定义我们的loss

total_loss = tf.losses.hinge_loss(tf.one_hot(y,10),logits=y_out)
mean_loss = tf.reduce_mean(total_loss) # loss求平均

# define our optimizer
# 定义优化器，设置学习率
optimizer = tf.train.AdamOptimizer(5e-4) # select optimizer and set learning rate
train_step = optimizer.minimize(mean_loss)
```

TensorFlow支持许多其他层的类型，损失函数和优化器 - 你将在后面的实验中遇到。 这里是官方的API文档（如果上面有任何参数搞不懂，这些资源就会非常有用）

- 各种层，激活函数，损失函数:https://www.tensorflow.org/api_guides/python/nn
- 优化器: https://www.tensorflow.org/api_guides/python/train#Optimizers
- BatchNorm: https://www.tensorflow.org/api_docs/python/tf/layers/batch_normalization

训练一轮

我们在上面已经定义了图所需要的操作，为了能够执行TensorFlow图中定义的计算，我们首先需要创建一个tf.Session对象。一个session中包含了TensorFlow运行时的状态。更多内容请参考TensorFlow指南 Getting started

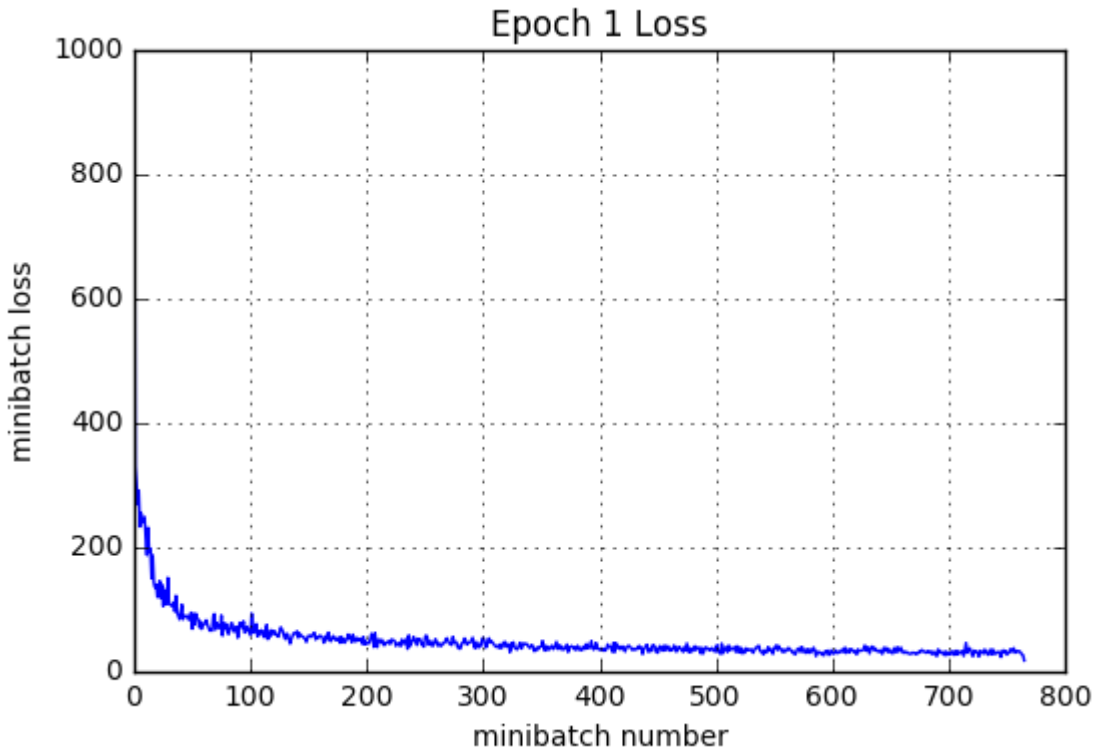
我们也可以指定一个设备，比如/cpu:0 或者 /gpu:0。 这种类型的操作可以参考this TensorFlow guide

下面你应该可以看到验证集上的loss在0.4到0.6之间，准确率在0.3到0.35。

```
def run_model(session, predict, loss_val, Xd, yd,
              epochs=1, batch_size=64, print_every=100,
              training=None, plot_losses=False):

    '''
    run model函数主要是控制整个训练的流程，需要传入session，调用session.run(variables)会得到variables里面各个变量的值。
    这里当训练模式的时候，也就是training!=None，我们传入的training是之前定义的train_op，调用session.run(train_op)会自动完成反向求导，
    整个模型的参数会发生更新。
    当training==None时,是我们需要对验证集合做一次预测的时候(或者是测试阶段)，这时我们不需要反向求导，所以variables里面并没有加入train_op
    '''
    # have tensorflow compute accuracy
    # 计算准确度（ACC值）
    correct_prediction = tf.equal(tf.argmax(predict,1), y)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    # shuffle indicies
    # 对训练样本进行混洗
    train_indicies = np.arange(Xd.shape[0])
    np.random.shuffle(train_indicies)

    training_now = training is not None

    # setting up variables we want to compute (and optimizing)
    # if we have a training function, add that to things we compute
    # 设置需要计算的变量
    # 如果需要进行训练，将训练过程(training)也加进来
    variables = [mean_loss,correct_prediction,accuracy]
    if training_now:
        variables[-1] = training

    # counter
    # 进行迭代
    iter_cnt = 0
    for e in range(epochs):
        # keep track of losses and accuracy
        # 记录损失函数和准确度的变化
        correct = 0
        losses = []
        # make sure we iterate over the dataset once
        # 确保每个训练样本都被遍历
        for i in range(int(math.ceil(Xd.shape[0]/batch_size))):
            # generate indicies for the batch
            # 产生一个minibatch的样本
            start_idx = (i*batch_size)%Xd.shape[0]
            idx = train_indicies[start_idx:start_idx+batch_size]

            # create a feed dictionary for this batch

            # 生成一个输入字典(feed dictionary)
```

```
            feed_dict = {X: Xd[idx,:],
                         y: yd[idx],
                         is_training: training_now }
            # get batch size
            # 获取minibatch的大小
            actual_batch_size = yd[idx].shape[0]

            # have tensorflow compute loss and correct predictions
            # and (if given) perform a training step
            # 计算损失函数和准确率
            # 如果是训练模式的话，执行训练过程
            loss, corr, _ = session.run(variables,feed_dict=feed_dict)

            # aggregate performance stats
            # 记录本轮的训练表现
            losses.append(loss*actual_batch_size)
            correct += np.sum(corr)

            # print every now and then
            # 定期输出模型表现
            if training_now and (iter_cnt % print_every) == 0:
                print("Iteration {0}: with minibatch training loss = {1:.3g} and accuracy of
{2:.2g}"\
                      .format(iter_cnt,loss,np.sum(corr)/actual_batch_size))
            iter_cnt += 1
        total_correct = correct/Xd.shape[0]
        total_loss = np.sum(losses)/Xd.shape[0]
        print("Epoch {2}, Overall loss = {0:.3g} and accuracy of {1:.3g}"\
              .format(total_loss,total_correct,e+1))
        if plot_losses:
            plt.plot(losses)
            plt.grid(True)
            plt.title('Epoch {} Loss'.format(e+1))
            plt.xlabel('minibatch number')
            plt.ylabel('minibatch loss')
            plt.show()
    return total_loss,total_correct

with tf.Session() as sess:
    with tf.device("/cpu:0"): #"/cpu:0" or "/gpu:0"
        sess.run(tf.global_variables_initializer())
        print('Training')
        run_model(sess,y_out,mean_loss,X_train,y_train,1,64,100,train_step,True)
        print('Validation')
        run_model(sess,y_out,mean_loss,X_val,y_val,1,64)
```

```
Training
Iteration 0: with minibatch training loss = 14.5 and accuracy of 0.078
Iteration 100: with minibatch training loss = 0.89 and accuracy of 0.34
Iteration 200: with minibatch training loss = 0.678 and accuracy of 0.33
Iteration 300: with minibatch training loss = 0.832 and accuracy of 0.16
Iteration 400: with minibatch training loss = 0.524 and accuracy of 0.33
Iteration 500: with minibatch training loss = 0.487 and accuracy of 0.44
Iteration 600: with minibatch training loss = 0.467 and accuracy of 0.33
Iteration 700: with minibatch training loss = 0.399 and accuracy of 0.41
Epoch 1, Overall loss = 0.771 and accuracy of 0.31
```




```
Validation
Epoch 1, Overall loss = 0.472 and accuracy of 0.373
```

训练一个特定的模型

在这部分，我们会指定一个模型需要你来构建。这里的目标并不是为了得到好的性能(后面会需要)，只是为了让你适应理解TensorFlow的文档以及配置你自己的模型。 用上面的代码作为指导，用相应的TensorFlow文档构建一个下面这样结构的模型:

- 7x7的卷积窗口，32个卷积核，步长为1
- ReLU激活层
- BatchNorm层（可训练变量，包含中心(centering)和范围(scale)）
- 2x2 的Max Pooling层，步长为2
- 包含1024个神经元的仿射层(affine layer)
- ReLu激活层

- 1024个输入单元，10个输出单元的仿射层

这里的卷积，激活函数，全连接层都跟之前的代码相似。

下面的batch normalization部分，笔者借鉴了https://github.com/ry/tensorflow-resnet/blob/master/resnet.py下面的batch normalization部分

这里的bath_normalization主要用到两个函数: tf.nn.moments() 用来计算mean, variance
tf.nn.batchnormalization() 根据预先算好的mean和variance对数据进行batch norm.

另外，我们在课件中看到的beta和gamma，在tf.nn.batchnormalization中对应的分别是offset和scale，这点在文档中都有详细的说明。 值得注意的是，在测试中，我们用到的mean和variance并不是当前测试集batch的mean和variance，而应该是对训练集训练过程中逐步迭代获得的。我这里的逐步迭代是加入了decay，来用每次新的batch的mean和variance，更新一点全局的mean，variance。 另外，我们更新了全局的mean和variance，需要添加

```
tf.add_to_collection(tf.GraphKeys.UPDATE_OPS, update_moving_mean)
tf.add_to_collection(tf.GraphKeys.UPDATE_OPS, update_moving_variance)
```

这两个操作， 并且我们的train_step需要稍作修改:

```
# batch normalization in tensorflow requires this extra dependency
extra_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(extra_update_ops):
    train_step = optimizer.minimize(mean_loss)
```

```
from tensorflow.python.training import moving_averages
from tensorflow.python.ops import control_flow_ops
# clear old variables
# 清除旧变量
tf.reset_default_graph()

# define our input (e.g. the data that changes every batch)
# The first dim is None, and gets sets automatically based on batch size fed in
# 定义输入数据（如每轮迭代中都会改变的数据）
# 第一维是None，每次迭代时都会根据输入数据自动设定
X = tf.placeholder(tf.float32, [None, 32, 32, 3])
y = tf.placeholder(tf.int64, [None])
is_training = tf.placeholder(tf.bool)

# define model
# 定义模型
def complex_model(X,y,is_training):
    # parameters
    # 定义一些常量
    MOVING_AVERAGE_DECAY = 0.9997
    BN_DECAY = MOVING_AVERAGE_DECAY
    BN_EPSILON = 0.001

    # 7x7 Convolutional Layer with 32 filters and stride of 1
    # 7x7的卷积窗口，32个卷积核，步长为1
    Wconv1 = tf.get_variable("Wconv1", shape=[7, 7, 3, 32])
```

```
    bconv1 = tf.get_variable("bconv1", shape=[32])
    h1 = tf.nn.conv2d(X, Wconv1, strides=[1,1,1,1], padding='VALID') + bconv1
    # ReLU Activation Layer
    # ReLU激活层
    a1 = tf.nn.relu(h1)  # a1的形状是 [batch_size, 26, 26, 32]
    # Spatial Batch Normalization Layer (trainable parameters, with scale and centering)
    # for so-called "global normalization", used with convolutional filters with shape [batch,
height, width, depth],
    # 与全局标准化(global normalization)对应，这里的标准化过程我们称之为局部标准化(Spatial Batch
Normalization)。记住，我们的卷积窗口大小是[batch, height, width, depth]
    # pass axes=[0,1,2]
    # 需要标准化的轴的索引是 axes = [0, 1, 2]
    axis = list(range(len(a1.get_shape()) - 1))  # axis = [0,1,2]
    mean, variance = tf.nn.moments(a1, axis) # mean, variance for each feature map 求出每个卷积结
果(feature map)的平均值，方差

    params_shape = a1.get_shape()[-1:]   # channel or depth 取出最后一维，即通道(channel)或叫深度
(depth)
    # each feature map should have one beta and one gamma
    # 每一片卷积结果(feature map)都有一个beta值和一个gamma值
    beta = tf.get_variable('beta',
                           params_shape,
                           initializer=tf.zeros_initializer)

    gamma = tf.get_variable('gamma',
                            params_shape,
                            initializer=tf.ones_initializer)

    # mean and variance during trianing are recorded and saved as moving_mean and
moving_variance
    # moving_mean and moving variance are used as mean and variance in testing.
    # 训练过程中得出的平均值和方差都被记录下来，并被用来计算移动平均值(moving_mean)和移动方差
(moving_variance)
    # 移动平均值(moving_mean)和移动方差(moving_variance)将在预测阶段被使用
    moving_mean = tf.get_variable('moving_mean',
                                  params_shape,
                                  initializer=tf.zeros_initializer,
                                  trainable=False)
    moving_variance = tf.get_variable('moving_variance',
                                      params_shape,
                                      initializer=tf.ones_initializer,
                                      trainable=False)

    # update variable by variable * decay + value * (1 - decay)
    # 更新移动平均值和移动方差，更新方式是 variable * decay + value * (1 - decay)
    update_moving_mean = moving_averages.assign_moving_average(moving_mean,
                                                               mean, BN_DECAY)
    update_moving_variance = moving_averages.assign_moving_average(
        moving_variance, variance, BN_DECAY)
    tf.add_to_collection(tf.GraphKeys.UPDATE_OPS, update_moving_mean)
    tf.add_to_collection(tf.GraphKeys.UPDATE_OPS, update_moving_variance)


    mean, variance = control_flow_ops.cond(
```

```
        is_training, lambda: (mean, variance),
        lambda: (moving_mean, moving_variance))


    a1_b = tf.nn.batch_normalization(a1, mean, variance, beta, gamma, BN_EPSILON)
    # 2x2 Max Pooling layer with a stride of 2
    # 2x2 的池化层，步长为2
    m1 = tf.nn.max_pool(a1_b, ksize=[1,2,2,1], strides = [1,2,2,1], padding='VALID')
    # shape of m1 should be batchsize * 26/2 * 26/2 * 32 = batchsize * 5408
    # Affine layer with 1024 output units
    # 池化后的结果m1的大小应为 batchsize * 26/2 * 26/2 * 32 = batchsize * 5408
    # 仿射层共输出2014个值
    m1_flat = tf.reshape(m1, [-1, 5408])
    W1 = tf.get_variable("W1", shape=[5408, 1024])
    b1 = tf.get_variable("b1", shape=[1024])
    h2 = tf.matmul(m1_flat,W1) + b1
    # ReLU Activation Layer
    # ReLU激活层
    a2 = tf.nn.relu(h2)
    # Affine layer from 1024 input units to 10 outputs
    # 仿射层有1024个输入和10个输出
    W2 = tf.get_variable("W2", shape=[1024, 10])
    b2 = tf.get_variable("b2", shape=[10])
    y_out = tf.matmul(a2,W2) + b2
    return y_out


y_out = complex_model(X,y,is_training)
```

为了确保你做对了，用下面的工具来检查你的输出维度，应该是64 x 10。因为我们的batch size是64，仿射层的最后一个输出是10个神经元对应10个类。

```
# Now we're going to feed a random batch into the model
# and make sure the output is the right size
# 现在我们随机输入一个batch进入模型，来验证一下输出的大小是否如预期
x = np.random.randn(64, 32, 32,3)
with tf.Session() as sess:
    with tf.device("/cpu:0"): #"/cpu:0" or "/gpu:0"
        tf.global_variables_initializer().run()

        ans = sess.run(y_out,feed_dict={X:x,is_training:True})
        %timeit sess.run(y_out,feed_dict={X:x,is_training:True})
        print(ans.shape)
        print(np.array_equal(ans.shape, np.array([64, 10])))
```

Out：

```
10 loops, best of 3: 118 ms per loop
(64, 10)
True
```

You should see the following from the run above

```
(64, 10)
```

```
True
```

GPU!

现在我们要在GPU设备下试一下我们的模型，剩下的代码都保持不变，我们的变量和操作都会用加速的代码路径来执行。然而如果没有GPU，我们会有Python exception然后不得不重建我们的图。在一个双核的CPU上，你大概可以看到50-80毫秒一个batch, 如果用Google Cloud GPUs 应该在2-5毫秒每个batch。

笔者注: 以下代码笔者用了CPU实现，得到的结果也是CPU的，如果读者使用了GPU，可以忽略下面每一个batch得到的计算时间结果。

```
try:
    with tf.Session() as sess:
        with tf.device("/cpu:0") as dev: # 可以是"/cpu:0" 或 "/gpu:0"
            tf.global_variables_initializer().run()

            ans = sess.run(y_out,feed_dict={X:x,is_training:True})
            %timeit sess.run(y_out,feed_dict={X:x,is_training:True})
except tf.errors.InvalidArgumentError:
    print("no gpu found, please use Google Cloud if you want GPU acceleration")
    # rebuild the graph
    # trying to start a GPU throws an exception
    # and also trashes the original graph
    tf.reset_default_graph()
    X = tf.placeholder(tf.float32, [None, 32, 32, 3])
    y = tf.placeholder(tf.int64, [None])
    is_training = tf.placeholder(tf.bool)
    y_out = complex_model(X,y,is_training)
```

Out：

```
10 loops, best of 3: 115 ms per loop
```

你应该可以看到即使是一个简单的前向传播过程在GPU上面也极大的加快了速度。所以对于下面剩下的作业(构建assignment3以及你的project的模型的时候)，你应该用GPU设备。 然而，对于tensorflow，默认的设备是GPU(如果有的话)，没有GPU的情况下会自动使用CPU。所以从现在开始我们都可以跳过设备的指定部分。

训练模型

既然你已经看到怎么定义一个模型并进行前向传播，下面，我们来用你上面创建的复杂模型，在训练集上训练一轮(epoch)。

确保你明白下面的每一个TensorFlow函数(对应于你自定义的神经网络)是怎么用的。

首先，传建一个RMSprop优化器(用学习率为1e-3)和一个交叉熵损失函数。可以参考TensorFlow文档来找到更多的信息。

```
# Inputs 输入
```

```
#     y_out: is what your model computes 模型输出
#     y: is your TensorFlow variable with label information 数据的真实标签
# Outputs 输出
#    mean_loss: a TensorFlow variable (scalar) with numerical loss 损失函数均值
#    optimizer: a TensorFlow optimizer 优化器
# This should be ~3 lines of code! 大概需要约3行代码
total_loss = tf.nn.softmax_cross_entropy_with_logits(logits=y_out, labels=tf.one_hot(y,10))
mean_loss = tf.reduce_mean(total_loss)

# define our optimizer 定义优化器
optimizer = tf.train.RMSPropOptimizer(1e-3) # select optimizer and set learning rate 定义优化器和学习率
```

```
# batch normalization in tensorflow requires this extra dependency
# tensorflow中执行batchNorm需要这些额外的依赖
extra_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(extra_update_ops):
    train_step = optimizer.minimize(mean_loss)
```

训练模型

下面我们创建一个session，并且在一个epoch上训练模型。你应该可以看到loss在1.4到2.0之间，准确率在0.4-0.5之间。由于初始化和随机种子的不同，具体值可能会有一些变化。

```
sess = tf.Session()

sess.run(tf.global_variables_initializer())
print('Training')
run_model(sess,y_out,mean_loss,X_train,y_train,1,64,100,train_step)
```

Out：

```
Training
Iteration 0: with minibatch training loss = 3.39 and accuracy of 0.078
Iteration 100: with minibatch training loss = 3.18 and accuracy of 0.14
Iteration 200: with minibatch training loss = 1.78 and accuracy of 0.41
Iteration 300: with minibatch training loss = 1.86 and accuracy of 0.39
Iteration 400: with minibatch training loss = 1.32 and accuracy of 0.48
Iteration 500: with minibatch training loss = 1.2 and accuracy of 0.66
Iteration 600: with minibatch training loss = 1.27 and accuracy of 0.59
Iteration 700: with minibatch training loss = 1.32 and accuracy of 0.48
Epoch 1, Overall loss = 1.67 and accuracy of 0.452
```

Out：

```
(1.6708081902873759, 0.45230612244897961)
```

查看模型的精确度

让我们看一下训练和测试代码 -- 在下面你自己建的模型中，可以随意使用这些代码来评估模型。你应该可以看到loss在1.3-2.0之间，准确率是0.45到0.55之间。

```
print('Validation')
run_model(sess,y_out,mean_loss,X_val,y_val,1,64)
```

Out：

```
Validation
Epoch 1, Overall loss = 1.44 and accuracy of 0.538
```

Out：

```
(1.4403997488021851, 0.53800000000000003)
```

现在你可以实验不同的结构，高参，损失函数和优化器来训练一个模型，能够在CIFAR-10上得到大于等于70%的准确率，你可以用上面的run_model函数。

你可以尝试的

- **Filter size**: 上面我们用了7 x 7的大小；用小一点的filter也许会更加的有效率。
- **Number of filters**: 上面我们用32个filter，用少一点会不会更好?
- **Pooling vs Strided Convolution**: 你有没有用max pooling? 还是只用了卷积（strided convolution）?
- **Batch normalization**: 尝试在卷积层后面加上局部bath norm(spatial batch normalization)， 在全连接层后面加上普通标准化(vanilla batch normalization)，你的网络有没有训练的更快一点？
- **Network architecture**: 上面的网络有两层可以训练的参数，你是不是可以用一个深度的网络训练的更好，下面这些架构你可以尝试一下：
  - [conv-relu-pool]xN -> [affine]xM -> [softmax or SVM]
  - [conv-relu-conv-relu-pool]xN -> [affine]xM -> [softmax or SVM]
  - [batchnorm-relu-conv]xN -> [affine]xM -> [softmax or SVM]
- **Use TensorFlow Scope**: 用TensorFlow scope 和/或 tf.layers来使得写更深的网络更方便。关于怎么用tf.layers详见这个教程
- **Use Learning Rate Decay**:正如这篇笔记所指出的，衰减学习率也许会帮助模型收敛。当loss不再随着epoch改变或者任何其他你觉得合适的启发式规则，可以试着衰减每一轮中的学习率。学习率衰减详见Tensorflow documentation
- **Global Average Pooling**: 除了展开然后构建多个全连接层，你可以用卷积操作直到你的图片足够的小(比如7x7)，然后在上面加一个average poooling层，来得到一个1x1(1,1,filter个数)的图片，然后再reshape成一个向量(filter个数)。这在Google Inception Network中有使用，详见表1，是他们的网络架构。
- **Regularization**: 加l2权重正则化，或者用在TensorFlow MNIST教程中的dropout

训练建议

对于每个你尝试的网络架构，你应该调整学习率和正则化强度，这么做的话有一些很重要的东西需要记住: -如果参数设置的很好，你应该可以在几百个迭代中就看到提升 -对于超参的选择，要记住由粗到精的方法，从一个很大范围的超参开始，通过迭代来找到那些表现不错的参数组合。 -一旦你发现了几组似乎有效的参数，在这些参数附近再进一步搜索。这时你也许会需要训练更多的轮数(epochs). -你应该用验证集来找超参，我们会用你在验证集上找到的最好的参数来测试测试集，从而来评估你的模型表现。

除此以外

如果你比较爱冒险，还有很多其他特征你可以尝试来提升你的模型性能。你并不一定需要实现下面的全部内容，不过尝试实现它们可以获得额外的加分。

- 其他的更新方法，这个作业中我们用了SGD+momentum, RMSprop 以及Adam；你可以试试其他的比如AdaGrad或者AdaDelta.
- 其他的激活函数像是leaky ReLU, parametric ReLU, ELU, 或者 MaxOut.
- 集成模型
- 数据扩增
- 新的结构，比如：
    - ResNets 前一层的输入直接连接到下一层的输出
    - DenseNets 前一层的输入全部级联起来。（译者注：就是原始数据会和每一层相连）
    - This blog has an in-depth overview

如果你决定实现一些其他的东西，请在下面的"Extra Credict Description"中叙述一下。

我们期望的

最起码，你应该可以训练出一个ConNet在验证集上得到至少70%的准确率，这只是一个最低界限。

- 如果你够细心，应该是可以得到一个远远高于这个结果的准确率！额外的分数会加给得分特别高的模型或者独特的方法。

你应该用下面的空间来做实验并训练你的模型。这个Notebook中的最后一个cell应该包含了你的模型在训练集和验证集的准确率。

开开心心地训练吧！

```
# Feel free to play with this cell
# 这里的代码可以随意把玩

def my_model(X,y,is_training):
    def conv_relu_pool(X, num_filter=32, conv_strides = 1, kernel_size=[3,3], pool_size=[2,2],
pool_strides = 2):
        conv1 = tf.layers.conv2d(inputs=X, filters=num_filter, kernel_size=kernel_size, stides =
conv_strides, padding="same", activation=tf.nn.relu)
        pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=pool_size, strides =
pool_strides)
        return pool1



    def conv_relu_conv_relu_pool(X, num_filter1=32, num_filter2=32, conv_strides = 1,
```

```
kernel_size=[5,5],pool_size=[2,2], pool_strides = 2):
        conv1 = tf.layers.conv2d(inputs=X,filters=num_filter1,kernel_size=kernel_size,
strides=conv_strides, padding="same",activation=tf.nn.relu)
        conv2 = tf.layers.conv2d(inputs=conv1,filters=num_filter2,kernel_size=kernel_size,
strides=conv_strides, padding="same",activation=tf.nn.relu)
        # Pooling Layer #1
        pool1 = tf.layers.max_pooling2d(inputs=conv2, pool_size=pool_size, strides=pool_strides)
        return pool1

    def affline(X, num_units, act):
        return tf.layers.dense(inputs=X, units=num_units, activation=act)

    def batchnorm_relu_conv(X, num_filters=32, conv_strides = 2, kernel_size=[5,5],
is_training=True):
        bat1 = tf.layers.batch_normalization(X, training=is_training)
        act1 = tf.nn.relu(bat1)
        #conv1 = tf.layers.conv2d(inputs=act1, filters=num_filters,
        #                         kernel_size = kernel_size, strides = 2, padding="same",
activation=None,
        #
kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=0.1),
        #                         bias_regularizer=tf.contrib.layers.l2_regularizer(scale=0.1))
        conv1 = tf.layers.conv2d(inputs=act1, filters=num_filters,
                                 kernel_size = kernel_size, strides = 2, padding="same",
activation=None) # without regularization

        return conv1

    N = 3 # num of conv blocks
    M = 1 # num of affine
    conv = tf.layers.conv2d(inputs = X, filters=64, kernel_size=[5,5], strides=1,
padding="same", activation=None)


    for i in range(N):
        print(conv.get_shape())
        conv = batchnorm_relu_conv(conv, is_training=is_training)
        #conv = conv_relu_conv_relu_pool(conv)

    print(conv.get_shape())
    global_average_shape = conv.get_shape()[1:3] # 4,4

    # just flatten the output
    #avg_layer = tf.reshape(conv,(-1,512))

    # global average pooling method 1
    #avg_layer = tf.layers.average_pooling2d(conv,
(global_average_shape,global_average_shape),padding='valid')
    #avg_layer = tf.squeeze(avg_layer, axis=[1,2]) # remove all 1 axis

    # global average  pooling method 2
    avg_layer = tf.reduce_mean(conv, [1,2]) # the same as global max pooling
```

```
    print(avg_layer.get_shape())

    fc = avg_layer
    #keep_prob = tf.constant(0.5)
    for i in range(M):
        fc = affline(fc,100,tf.nn.relu)
        #fc = tf.nn.dropout(fc, keep_prob)

    fc = affline(fc, 10, None)

    return fc


tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, 32, 32, 3])
y = tf.placeholder(tf.int64, [None])
is_training = tf.placeholder(tf.bool)

y_out = my_model(X,y,is_training)
total_loss = tf.nn.softmax_cross_entropy_with_logits(logits=y_out, labels=tf.one_hot(y,10))
mean_loss = tf.reduce_mean(total_loss)

global_step = tf.Variable(0, trainable=False, name="Global_Step")
starter_learning_rate = 1e-2
learning_rate = tf.train.exponential_decay(starter_learning_rate, global_step,
                                           750, 0.96, staircase=True)

#learning_rate = starter_learning_rate
# define our optimizer
optimizer = tf.train.AdamOptimizer(learning_rate) # select optimizer and set learning rate


# batch normalization in tensorflow requires this extra dependency
extra_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(extra_update_ops):
    train_step = optimizer.minimize(mean_loss, global_step=global_step)

print([x.name for x in tf.global_variables()])
```

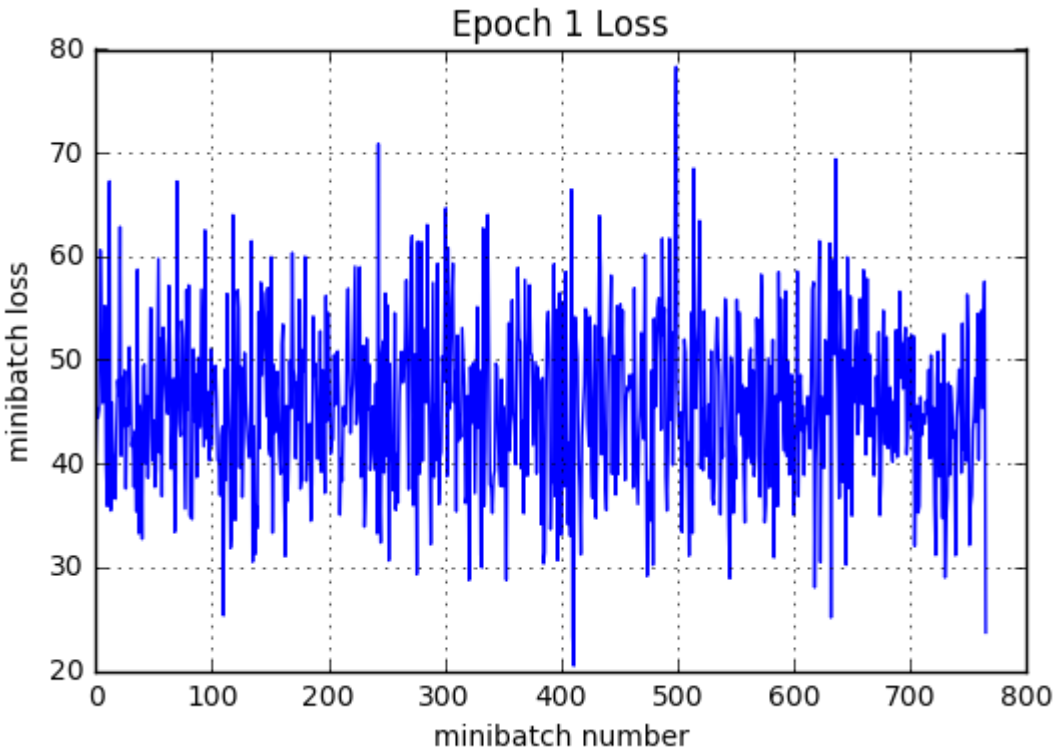
Out :

```
(?, 32, 32, 64)
(?, 16, 16, 32)
(?, 8, 8, 32)
(?, 4, 4, 32)
(?, 32)
['conv2d/kernel:0', 'conv2d/bias:0', 'batch_normalization/beta:0',
'batch_normalization/gamma:0', 'batch_normalization/moving_mean:0',
'batch_normalization/moving_variance:0', 'conv2d_1/kernel:0', 'conv2d_1/bias:0',
'batch_normalization_1/beta:0', 'batch_normalization_1/gamma:0',
'batch_normalization_1/moving_mean:0', 'batch_normalization_1/moving_variance:0',
'conv2d_2/kernel:0', 'conv2d_2/bias:0', 'batch_normalization_2/beta:0',
'batch_normalization_2/gamma:0', 'batch_normalization_2/moving_mean:0',
'batch_normalization_2/moving_variance:0', 'conv2d_3/kernel:0', 'conv2d_3/bias:0',
'dense/kernel:0', 'dense/bias:0', 'dense_1/kernel:0', 'dense_1/bias:0', 'Global_Step:0',
'beta1_power:0', 'beta2_power:0', 'conv2d/kernel/Adam:0', 'conv2d/kernel/Adam_1:0',
'conv2d/bias/Adam:0', 'conv2d/bias/Adam_1:0', 'batch_normalization/beta/Adam:0',
'batch_normalization/beta/Adam_1:0', 'batch_normalization/gamma/Adam:0',
'batch_normalization/gamma/Adam_1:0', 'conv2d_1/kernel/Adam:0', 'conv2d_1/kernel/Adam_1:0',
'conv2d_1/bias/Adam:0', 'conv2d_1/bias/Adam_1:0', 'batch_normalization_1/beta/Adam:0',
'batch_normalization_1/beta/Adam_1:0', 'batch_normalization_1/gamma/Adam:0',
'batch_normalization_1/gamma/Adam_1:0', 'conv2d_2/kernel/Adam:0', 'conv2d_2/kernel/Adam_1:0',
'conv2d_2/bias/Adam:0', 'conv2d_2/bias/Adam_1:0', 'batch_normalization_2/beta/Adam:0',
'batch_normalization_2/beta/Adam_1:0', 'batch_normalization_2/gamma/Adam:0',
'batch_normalization_2/gamma/Adam_1:0', 'conv2d_3/kernel/Adam:0', 'conv2d_3/kernel/Adam_1:0',
'conv2d_3/bias/Adam:0', 'conv2d_3/bias/Adam_1:0', 'dense/kernel/Adam:0',
'dense/kernel/Adam_1:0', 'dense/bias/Adam:0', 'dense/bias/Adam_1:0', 'dense_1/kernel/Adam:0',
'dense_1/kernel/Adam_1:0', 'dense_1/bias/Adam:0', 'dense_1/bias/Adam_1:0']
```

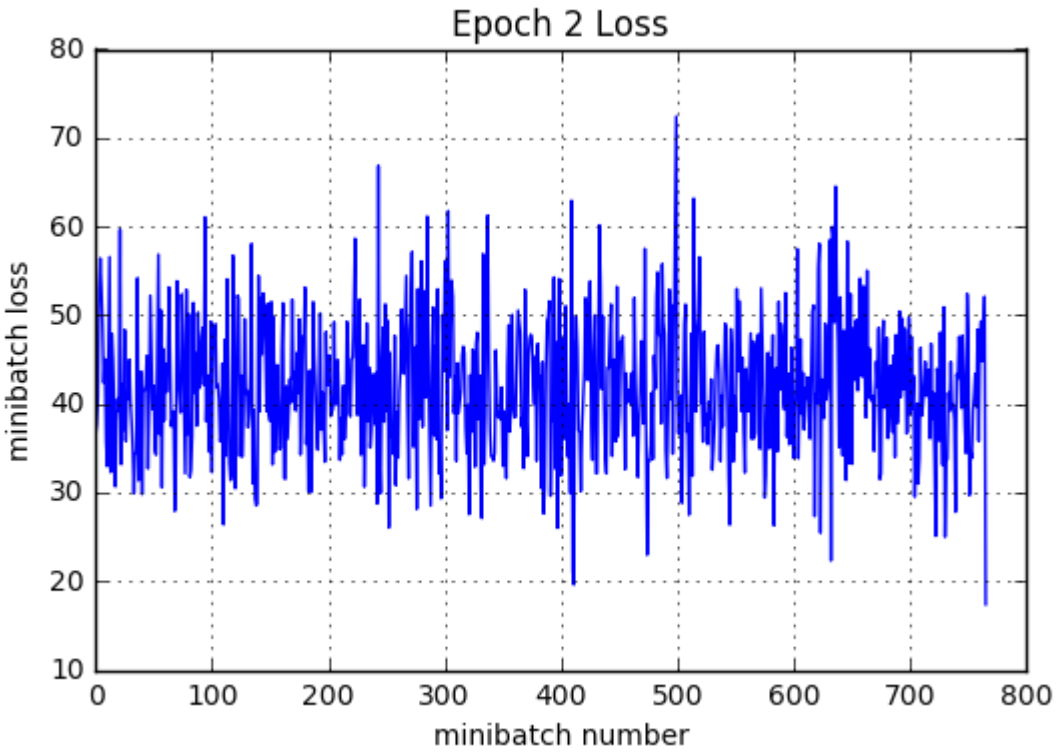
```python
# Feel free to play with this cell
# This default code creates a session
# and trains your model for 10 epochs
# then prints the validation set accuracy
#sess = tf.Session()
#sess.run(tf.global_variables_initializer())
print('Training')
run_model(sess,y_out,mean_loss,X_train,y_train,2,64
          ,100,train_step,True)
print('Validation')
run_model(sess,y_out,mean_loss,X_val,y_val,1,64)

# 下面的loss是我预先跑了5个epoch之后，又跑了两个epoch
```

Out：

```
Training
Iteration 0: with minibatch training loss = 0.614 and accuracy of 0.81
Iteration 100: with minibatch training loss = 0.653 and accuracy of 0.77
Iteration 200: with minibatch training loss = 0.852 and accuracy of 0.75
Iteration 300: with minibatch training loss = 0.868 and accuracy of 0.75
Iteration 400: with minibatch training loss = 0.517 and accuracy of 0.81
Iteration 500: with minibatch training loss = 0.744 and accuracy of 0.69
Iteration 600: with minibatch training loss = 0.547 and accuracy of 0.78
Iteration 700: with minibatch training loss = 0.692 and accuracy of 0.75
Epoch 1, Overall loss = 0.714 and accuracy of 0.745
```




```
Iteration 800: with minibatch training loss = 0.533 and accuracy of 0.8
Iteration 900: with minibatch training loss = 0.907 and accuracy of 0.69
Iteration 1000: with minibatch training loss = 0.595 and accuracy of 0.73
Iteration 1100: with minibatch training loss = 0.518 and accuracy of 0.83
Iteration 1200: with minibatch training loss = 0.837 and accuracy of 0.73
Iteration 1300: with minibatch training loss = 0.723 and accuracy of 0.73
Iteration 1400: with minibatch training loss = 0.923 and accuracy of 0.67
Iteration 1500: with minibatch training loss = 0.612 and accuracy of 0.8
Epoch 2, Overall loss = 0.656 and accuracy of 0.768
```




```
Validation
Epoch 1, Overall loss = 0.901 and accuracy of 0.708
```

Out：

```
(0.9011877479553223, 0.70799999999999996)
```

```
# Test your model here, and make sure
# the output of this cell is the accuracy
# of your best model on the training and val sets
# We're looking for >= 70% accuracy on Validation
# 在这里测试你的模型，确保本cell的输出是你的模型在训练集和验证集上最好的准确度
# 验证集的准确度应该在70%以上
print('Training')
run_model(sess,y_out,mean_loss,X_train,y_train,1,64)
print('Validation')

run_model(sess,y_out,mean_loss,X_val,y_val,1,64)
```

Out：

```
Training
Epoch 1, Overall loss = 0.607 and accuracy of 0.783
Validation
Epoch 1, Overall loss = 0.901 and accuracy of 0.708
```

Out：

```
(0.90118774318695072, 0.70799999999999996)
```

在这里写一下你都做了些什么吧

在这里讲述一下你做了神马，以及你实现的额外的特性，以及任何你用来训练和评估你的神经网络的可视化图

笔者简单的实现了上面要求中的几个块，分别试了一下效果，以及用了一下learning rate decay。建议读者可以尝试更多的组合，多查阅官方文档来加深对tensorflow的理解。另外在建模型的时候可以把每一步的结果的shape打印出来，从而对模型每一步的输出有个概念。如果训练的过程中遇到问题，可以先用tensorflow的官方文档上的cifar模型结构来运行一下，看看是否可以调通。

测试集-我们只测一次

既然我们已经有一个我们觉得还不错的结果，那我们需要把最后的模型放到测试集上。这就是我们最后会在比赛上得到的结果，根据这个结果，思考一下，这个结果和你的验证集准确率比起来如何。

```
print('Test')
run_model(sess,y_out,mean_loss,X_test,y_test,1,64)
```

Out：

```
Test
Epoch 1, Overall loss = 0.899 and accuracy of 0.696
```

Out：

```
(0.89940066184997558, 0.69550000000000001)
```

我们还会用TensorFlow做更多事情

后面的作业都会依赖Tensorflow，你也许会发现它对你的项目也很有帮助。

加分内容说明

如果你实现了额外的一些特性来获得加分，请在这里指明代码或者其它文件的位置。