TensorFlow 卷积层

让我们看下如何在 TensorFlow 里面实现 CNN。

TensorFlow 提供了 tf.nn.conv2d() 和 tf.nn.bias_add() 函数来创建你自己的卷积层。

```
# Output depth
k_output = 64

# Image Properties
image_width = 10
image_height = 10
color_channels = 3

# Convolution filter
filter_size_width = 5
filter_size_height = 5

# Input/Image
input = tf.placeholder(
    tf.float32,
    shape=[None, image_height, image_width, color_channels])

# Weight and bias
weight = tf.Variable(tf.truncated_normal(
    [filter_size_height, filter_size_width, color_channels, k_output]))
bias = tf.Variable(tf.zeros(k_output))

# Apply Convolution
conv_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')
# Add bias
conv_layer = tf.nn.bias_add(conv_layer, bias)
# Apply activation function
conv_layer = tf.nn.relu(conv_layer)
```

上述代码用了 tf.nn.conv2d() 函数来计算卷积，weights 作为滤波器，[1, 2, 2, 1] 作为 strides。TensorFlow 对每一个 input 维度使用一个单独的 `stride` 参数， `[batch, input_height, input_width, input_channels]` 。我们通常把 batch 和 input_channels（strides 序列中的第一个第四个）的 stride 设为 1。

你可以专注于修改 input_height 和 input_width， batch 和 input_channels 都设置成 1。input_height 和 input_width strides 表示滤波器在input 上移动的步长。上述例子中，在 input 之后，设置了一个 5x5 ，stride 为 2 的滤波器。

tf.nn.bias_add() 函数对矩阵的最后一维加了偏置项。

- 注意 padding 的取值可以有"SAME"和"VALID",两者的区别在于如果输入为2行3列且stride=2时，VALID只会关注前两行两列，而SAME模式会自动在最后补上一列0.

TensorFlow 最大池化

由 Aphex34 (自己的作品) CC BY-SA 4.0， 通过 Wikimedia Commons 共享

这是一个最大池化的例子max pooling 用了 2x2 的滤波器 stride 为 2。四个 2x2 的颜色代表滤波器移动每个步长所产出的最大值。

例如 1, 0], [4, 6 生成 6，因为 6 是这4个数字中最大的。同理 2, 3], [6, 8 生成 8。 理论上，最大池化操作的好处是减小输入大小，使得神经网络能够专注于最重要的元素。最大池化只取覆盖区域中的最大值，其它的值都丢弃。

TensorFlow 提供了 tf.nn.max_pool() 函数，用于对卷积层实现 最大池化 。

```
...
conv_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')
conv_layer = tf.nn.bias_add(conv_layer, bias)
conv_layer = tf.nn.relu(conv_layer)
# Apply Max Pooling
conv_layer = tf.nn.max_pool(
    conv_layer,
    ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1],
    padding='SAME')
```

`tf.nn.max_pool()` 函数实现最大池化时， **ksize**参数是滤波器大小，**strides**参数是步长。**2x2** 的滤波器配合 **2x2** 的步长是常用设定。

ksize 和 strides 参数也被构建为四个元素的列表，每个元素对应 input tensor 的一个维度 ([batch, height, width, channels])，对 ksize 和 strides 来说，batch 和 channel 通常都设置成 1。

```
import tensorflow as tf

a=tf.constant([
        [[1.0,2.0,3.0,4.0],
        [5.0,6.0,7.0,8.0],
        [8.0,7.0,6.0,5.0],
        [4.0,3.0,2.0,1.0]],
        [[4.0,3.0,2.0,1.0],
         [8.0,7.0,6.0,5.0],
         [1.0,2.0,3.0,4.0],
         [5.0,6.0,7.0,8.0]]
    ])

a=tf.reshape(a,[1,4,4,2])

pooling=tf.nn.max_pool(a,[1,2,2,1],[1,1,1,1],padding='VALID')
with tf.Session() as sess:
    print("image:")
    image=sess.run(a)
    print (image)
    print("reslut:")
    result=sess.run(pooling)
    print (result)

import numpy as np

a= np.array([[[1.0,2.0,3.0,4.0],
        [5.0,6.0,7.0,8.0],
        [8.0,7.0,6.0,5.0],
        [4.0,3.0,2.0,1.0]],
        [[4.0,3.0,2.0,1.0],
         [8.0,7.0,6.0,5.0],
         [1.0,2.0,3.0,4.0],
         [5.0,6.0,7.0,8.0]]
    ])
print(a)
a.shape
a.reshape(4,4,2)
# reshape,后两位描述的是最小单元矩阵的长、宽
```

设置

H = height, W = width, D = depth

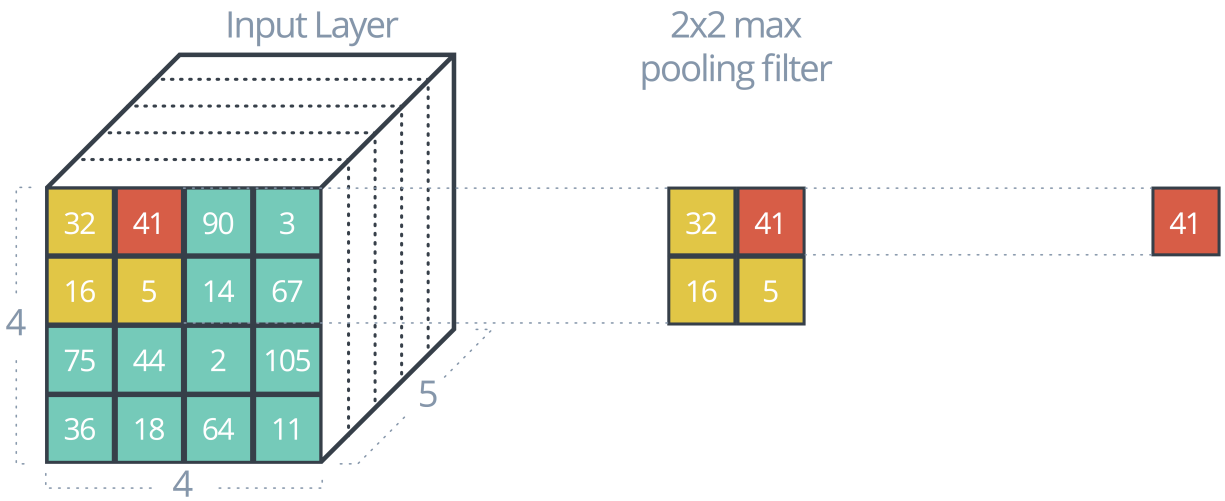
- 输入维度是 4x4x5 (HxWxD)
- 滤波器大小 2x2 (HxW)
- stride 的高和宽都是 2 (S)
  新的高和宽的公式是：

```
new_height = (input_height - filter_height)/S + 1
new_width = (input_width - filter_width)/S + 1
```

注意：池化层的输出深度与输入的深度相同。另外池化操作是分别应用到每一个深度切片层。

下图给你一个最大池化层如何工作的示例。这里，最大池化滤波器的大小是 2x2。当最大池化层在输入层滑动时，输出是这个 2x2 方块的最大值。

Pooling Mechanics Quiz



输出的维度大小是什么？格式写成 HxWxD。

答案

答案是 2x2x5。计算公式如下：

(4 - 2)/2 + 1 = 2
(4 - 2)/2 + 1 = 2
深度保持不变
这是对应的代码：

```
input = tf.placeholder(tf.float32, (None, 4, 4, 5))
filter_shape = [1, 2, 2, 1]
strides = [1, 2, 2, 1]
padding = 'VALID'
pool = tf.nn.max_pool(input, filter_shape, strides, padding)
```

pool 的输出维度是 [1, 2, 2, 5]，即使把 padding 改成 'SAME' 也是一样。

练习

很好！现在让我们练习一些池化的操作。

- 最大池化

下列输入最大池化的结果是？

```
[[[0, 1, 0.5, 10],
   [2, 2.5, 1, -8],
   [4, 0, 5, 6],
   [15, 1, 2, 3]]]
```

滤波器大小 2x2，stride 高和宽都是 2。输出维度是 2x2x1。

答案是4个数字，用（英文）逗号隔开。示例：1,2,3,4。

从左上到右下

答案

正确答案是 2.5,10,15,6。我们从左上角开始，然后从左到右，从上到下每次移动 2 个单位。

```
max(0, 1, 2, 2.5) = 2.5
max(0.5, 10, 1, -8) = 10
max(4, 0, 15, 1) = 15
max(5, 6, 2, 3) = 6
```

平均池化
下列输入的平均池化结果是？

```
[[[0, 1, 0.5, 10],
   [2, 2.5, 1, -8],
   [4, 0, 5, 6],
   [15, 1, 2, 3]]]
```

滤波器大小 2x2，stride 高和宽都是 2。输出维度是 2x2x1。

答案是4个数字，用（英文）逗号隔开。示例：1,2,3,4。

保留三位小数，顺序是左上到右下

- 注：把 2x2 滤波器按照 stride 在输入上移动，计算覆盖数字的最大值。stride 是 2 意味着我们每次移动两个单位。

TensorFlow 中的卷积网络

是时候看一下 TensorFlow 中的卷积神经网络的例子了。

网络的结构跟经典的 CNNs 结构一样，是卷积层，最大池化层和全链接层的混合。

这里你看到的代码与你在 TensorFlow 深度神经网络的代码类似，我们按 CNN 重新组织了结构。

如那一节一样，这里你将会学习如何分解一行一行的代码。你还可以下载代码自己运行。

感谢 Aymeric Damien 提供了这节课的原始 TensorFlow 模型。

现在开看下！

数据集

你从之前的课程中见过这节课的代码。这里我们导入 MNIST 数据集，用一个方便的函数完成对数据集的 batch，缩放和独热编码。

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets(".", one_hot=True, reshape=False)

import tensorflow as tf

# Parameters
# 参数
learning_rate = 0.00001
epochs = 10
batch_size = 128

# Number of samples to calculate validation and accuracy
# Decrease this if you're running out of memory to calculate accuracy
# 用来验证和计算准确率的样本数
# 如果内存不够，可以调小这个数字
test_valid_size = 256

# Network Parameters
# 神经网络参数
n_classes = 10  # MNIST total classes (0-9 digits)
dropout = 0.75  # Dropout, probability to keep units
```

Weights and Biases

```
# Store layers weight & bias
weights = {
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    'out': tf.Variable(tf.random_normal([1024, n_classes]))}

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]))}
```

卷积

3×3 卷积滤波器。来源：
http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution
这是一个 3x3 的卷积滤波器的示例。以 stride 为 1 应用到一个范围在 0 到 1 之间的数据上。每一个 3x3 的部分与权值 1, 0, 1], [0, 1, 0], [1, 0, 1 做卷积，把偏置加上后得到右边的卷积特征。这里偏置是 0 。TensorFlow 中这是通过 tf.nn.conv2d() 和 tf.nn.bias_add() 来完成的。

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

tf.nn.conv2d() 函数与权值 W 做卷积。

在 TensorFlow 中，strides 是一个4个元素的序列；第一个位置表示 stride 的 batch 参数，最后一个位置表示 stride 的特征(feature)参数。最好的移除 batch 和特征(feature)的方法是你直接在数据集中把他们忽略，而不是使用 stride。要使用所有的 batch 和特征(feature)，你可以把第一个和最后一个元素设成1。

中间两个元素指纵向(height)和横向(width)的 stride，之前也提到过 stride 通常是正方形，height = width。当别人说 stride 是 3 的时候，他们意思是 tf.nn.conv2d(x, W, strides=[1, 3, 3, 1])。

为了更简洁，这里的代码用了tf.nn.bias_add() 来添加偏置。 tf.add() 这里不能使用，因为 tensors 的维度不同。

最大池化

带有 2x2 滤波器 和 stride 为 2 的最大池化。来源:
上面是一个最大池化的示例。滤波器大小是 2x2，stride 是 2。左边是输入，右边是输出。 四个 2x2 的

颜色代表每一次滤波器应用在左侧来构建右侧的最大结果。例如。1, 1], [5, 6 变成 6，3, 2], [1, 2 变成 3。

```
def maxpool2d(x, k=2):
    return tf.nn.max_pool(
        x,
        ksize=[1, k, k, 1],
        strides=[1, k, k, 1],
        padding='SAME')
```

tf.nn.max_pool() 函数做的与你期望的一样，它通过设定 ksize 参数来设定滤波器大小，从而实现最大池化。

模型

Image from Explore The Design Space video
在下面的代码中，我们创建了 3 层来实现卷积，最大池化以及全链接层和输出层。每一层对维度的改变都写在注释里。例如第一层在卷积部分把图片从 28x28x1 变成了 28x28x32。后面应用了最大池化，每个样本变成了 14x14x32。从 conv1 经过多层网络，最后到 output 生成 10 个分类。

```
def conv_net(x, weights, biases, dropout):
    # Layer 1 - 28*28*1 to 14*14*32
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    conv1 = maxpool2d(conv1, k=2)

    # Layer 2 - 14*14*32 to 7*7*64
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer - 7*7*64 to 1024
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    fc1 = tf.nn.dropout(fc1, dropout)

    # Output Layer - class prediction - 1024 to 10
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out
```

Session

现在让我们开始运行神经网络！

```
# tf Graph input
x = tf.placeholder(tf.float32, [None, 28, 28, 1])
y = tf.placeholder(tf.float32, [None, n_classes])
keep_prob = tf.placeholder(tf.float32)

# Model
logits = conv_net(x, weights, biases, keep_prob)

# Define loss and optimizer
cost = tf.reduce_mean(\
    tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)\
    .minimize(cost)

# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf. global_variables_initializer()

# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(epochs):
        for batch in range(mnist.train.num_examples//batch_size):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            sess.run(optimizer, feed_dict={
                x: batch_x,
                y: batch_y,
                keep_prob: dropout})

            # Calculate batch loss and accuracy
            loss = sess.run(cost, feed_dict={
                x: batch_x,
                y: batch_y,
                keep_prob: 1.})
            valid_acc = sess.run(accuracy, feed_dict={
                x: mnist.validation.images[:test_valid_size],
                y: mnist.validation.labels[:test_valid_size],
                keep_prob: 1.})

            print('Epoch {:>2}, Batch {:>3} -'
                  'Loss: {:>10.4f} Validation Accuracy: {:.6f}'.format(
                epoch + 1,
                batch + 1,
                loss,
                valid_acc))

    # Calculate Test Accuracy
```

```
    test_acc = sess.run(accuracy, feed_dict={
        x: mnist.test.images[:test_valid_size],
        y: mnist.test.labels[:test_valid_size],
        keep_prob: 1.})
    print('Testing Accuracy: {}'.format(test_acc))
```

这就是 TensorFlow 中的 CNN。接下来你亲手实践一下。