实验一 手写数字识别

一、 实验目的

1. 掌握卷积神经网络基本原理；
2. 掌握 Tensorflow（或其他框架）的基本用法以及构建卷积网络的基本操作；
3. 了解 Tensorflow（或其他框架）在 GPU 上的使用方法。

二、 实验要求

1. 搭建 Tensorflow（或其他框架）环境；
2. 构建一个规范的卷积神经网络结构；
3. 在 MNIST 手写数字数据集上进行训练和评估，实现测试集准确率达到 98%及以上；
4. 按规定时间在课程网站提交实验报告、代码以及 PPT。

三、 实验原理（以 Tensorflow 为例）

1. TensorFlow 基本用法：

使用 TensorFlow, 必须了解 TensorFlow:

- 使用图(graph) 来表示计算任务。
- 在被称之为会话 (Session) 的上下文 (context) 中执行图。
- 使用 tensor 表示数据。
- 通过 变量 (Variable) 维护状态。
- 使用 feed 和 fetch 可以为任意的操作(arbitrary operation) 赋值或者从其中获取数据。

TensorFlow 是一个编程系统, 使用图来表示计算任务。图中的节点被称之为 op (operation 的缩写)。一个 op 获得 0 个或多个 Tensor, 执行计算, 产生 0 个或多个 Tensor。每个 Tensor 是一个类型化的多维数组。例如, 你可以将一小组图像集表示为一个四维浮点数数组, 这四个维度分别是 [batch, height, width, channels]。

一个 TensorFlow 图描述了计算的过程。为了进行计算, 图必须在“会话”

里被启动。 “会话“将图的 op 分发到诸如 CPU 或 GPU 之类的设备上, 同时提供执行 op 的方法。 这些方法执行后, 将产生的 tensor 返回。在 Python 语言中, 返回的 tensor 是 numpy ndarray 对象; 在 C 和 C++ 语言中, 返回的 tensor 是 tensorflow::Tensor 实例。

2. 卷积神经网络：

典型的卷积神经网络由卷积层、池化层、激活函数层交替组合构成，因此可将其视为一种层次模型，形象地体现了深度学习中“深度”之所在。

（1） 卷积操作

卷积运算是卷积神经网络的核心操作，给定二维的图像 I 作为输入，二维卷积核 K， 卷积运算可表示为：

$$S(i,j)=(I*K)(i,j)=\sum_m\sum_n I(i+m,j+n)K(m,n) \tag{1}$$

给定 5×5 输入矩阵、3×3 卷积核，相应的卷积操作如图 1 所示。

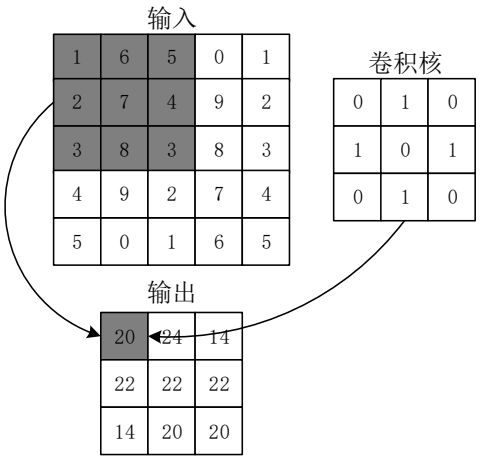

图 1 卷积运算

在使用 TensorFlow 等深度学习框架时，卷积层会有 padding 参数，常用的有两种选择，一个是“valid”，一个是“same”。前者是不进行填充，后者则是进行数据填充并保证输出与输入具有相同尺寸。

构建卷积或池化神经网络时，卷积步长也是一个很重要的基本参数。它控制了每个操作在特征图上的执行间隔。

（2） 池化操作

池化操作使用某位置相邻输出的总体统计特征作为该位置的输出，常用最大池化（max-pooling）和均值池化（average-pooling）。池化层不包含需要训练学习

的参数，仅需指定池化操作的核大小、操作步长以及池化类型。池化操作示意如图 2 所示。

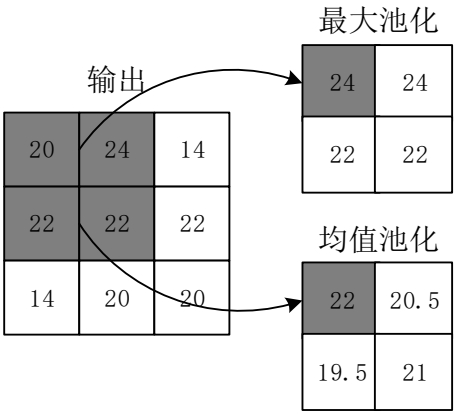

图 2 池化操作

（3） 激活函数层

卷积操作可视为对输入数值进行线性计算发挥线性映射的作用。激活函数的引入，则增强了深度网络的非线性表达能力，从而提高了模型的学习能力。常用的激活函数有 sigmoid、tanh 和 ReLU 函数。

四、 实验所用工具及数据集（以 Tensorflow 为例）

1. 工具

Anaconda、TensorFlow

（Tensorflow 安装教程参考：Tensorflow 官网、Tensorflow 中文社区、https://github.com/tensorflow/tensorflow）

2. 数据集

MNIST 手写数字数据集

（下载地址及相关介绍：http://yann.lecun.com/exdb/mnist/）

五、 实验步骤与方法（以 Tensorflow 为例）

1）安装实验环境，包括 Anaconda、TensorFlow，如果使用 GPU 版本还需要安装 cuda、cudnn；

2）下载 MNIST 手写数字数据集；

3）加载 MNIST 数据；

加载图像：

```
with gzip.open(filename) as bytestream:
    bytestream.read(16) #每个像素存储在文件中的大小为 16bits
    buf = bytestream.read(IMAGE_SIZE * IMAGE_SIZE * num_images * NUM_CHANNELS)
    data = numpy.frombuffer(buf, dtype=numpy.uint8).astype(numpy.float32)
    #像素值[0, 255]被调整到[-0.5, 0.5]
    data = (data - (PIXEL_DEPTH / 2.0)) / PIXEL_DEPTH
    #调整为 4 维张量[image index, y, x, channels]
    data = data.reshape(num_images, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS)
```

加载标签：

```
with gzip.open(filename) as bytestream:
    bytestream.read(8) #每个标签存储在文件中的大小为 8bits
    buf = bytestream.read(1 * num_images)
    labels = numpy.frombuffer(buf, dtype=numpy.uint8).astype(numpy.int64)
```

4）构建模型计算图；

创建输入占位符：

```
#这是训练样本和标签被送到图表的地方。
#这些占位符节点将在每个节点输入一批训练数据
#训练步骤使用{feed_dict}参数进行下面的 Run（）调用。
train_data_node = tf.placeholder( data_type(),
        shape=(BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS))
train_labels_node = tf.placeholder(tf.int64, shape=(BATCH_SIZE,))
eval_data = tf.placeholder(data_type(),
        shape=(EVAL_BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS))
```

初始化变量：

```
# 下面的变量包含所有可训练的权重。当我们调用时将分配它们时，它们被传递一个初始值：
# {tf.global_variables_initializer().run()}
conv1_weights = tf.Variable(
        tf.truncated_normal([5, 5, NUM_CHANNELS, 32],   # 5x5 filter, depth 32.
                            stddev=0.1,
                            seed=SEED, dtype=data_type()))
conv1_biases = tf.Variable(tf.zeros([32], dtype=data_type()))
conv2_weights = tf.Variable(tf.truncated_normal(
        [5, 5, 32, 64], stddev=0.1,
        seed=SEED, dtype=data_type()))
conv2_biases = tf.Variable(tf.constant(0.1, shape=[64], dtype=data_type()))
fc1_weights = tf.Variable(    # fully connected, depth 512.
        tf.truncated_normal([IMAGE_SIZE // 4 * IMAGE_SIZE // 4 * 64, 512],
                            stddev=0.1,
                            seed=SEED,
```

```
                                dtype=data_type()))
fc1_biases = tf.Variable(tf.constant(0.1, shape=[512], dtype=data_type()))
fc2_weights = tf.Variable(tf.truncated_normal([512, NUM_LABELS],
                                              stddev=0.1,
                                              seed=SEED,
                                              dtype=data_type()))
fc2_biases = tf.Variable(tf.constant(
    0.1, shape=[NUM_LABELS], dtype=data_type()))
```

CNN 模型构建：

```
# 2D 卷积，带有“SAME”填充（即输出要素图与输入的大小相同）。
# 请注意，{strides}是一个 4D 数组，其形状与数据布局匹配：[image index，y，x，depth]。
conv = tf.nn.conv2d(data,
                    conv1_weights,
                    strides=[1, 1, 1, 1],
                    padding='SAME')
# 偏置和 ReLU 非线性激活。
relu = tf.nn.relu(tf.nn.bias_add(conv, conv1_biases))
# 最大池化。
# 内核大小规范{ksize}也遵循数据布局。 这里我们有一个 2 的池化窗口和 2 的步幅。
pool = tf.nn.max_pool(relu,
                      ksize=[1, 2, 2, 1],
                      strides=[1, 2, 2, 1],
                      padding='SAME')
conv = tf.nn.conv2d(pool,
                    conv2_weights,
                    strides=[1, 1, 1, 1],
                    padding='SAME')
relu = tf.nn.relu(tf.nn.bias_add(conv, conv2_biases))
pool = tf.nn.max_pool(relu,
                      ksize=[1, 2, 2, 1],
                      strides=[1, 2, 2, 1],
                      padding='SAME')
# 将特征图变换为 2D 矩阵，以将其提供给完全连接的图层。
pool_shape = pool.get_shape().as_list()
reshape = tf.reshape(
    pool,
    [pool_shape[0], pool_shape[1] * pool_shape[2] * pool_shape[3]])
# 全连接层。
hidden = tf.nn.relu(tf.matmul(reshape, fc1_weights) + fc1_biases)
```