TENSORFLOW 入门

模型名	AlexNet	VGG	GoogLeNet	ResNet
初入江湖	2012	2014	2014	2015
层数	8	19	22	152
Top-5错误	16.4%	7.3%	6.7%	3.57%
Data Augmentation	+	+	+	+
Inception(NIN)	-	-	+	-
卷积层数	5	16	21	151
卷积核大小	11,5,3	3	7,1,3,5	7,1,3,5
全连接层数	3	3	1	1
全连接层大小	4096,4096,1000	4096,4096,1000	1000	1000
Dropout	+	+	+	+
Local Response Normalization	+	-	+	-
Batch Normalization	-	-	-	+

表1 AlexNet、VGG、GoogLeNet、ResNet对比

TENSORFLOW介绍

TensorFlow™ 是一个使用数据流图进行数值计算的开源软件库。图中的节点代表数学运算， 而图中的边则代表在这些节点之间传递的多维数组（张量）。这种灵活的架构可让您使用一个 API 将计算工作部署到桌面设备、服务器或者移动设备中的一个或多个 CPU 或 GPU。

TensorFlow 最初是由 Google 机器智能研究部门的 Google Brain 团队中的研究人员和工程师开发的，用于进行机器学习和深度神经网络研究， 但它是一个非常基础的系统，因此也可以应用于众多其他领域。

TensorFlow 可以灵活适应不同的使用规模，既支持探索性研究，也支持大规模生产用途。 TensorFlow目前最新的版本是1.4

TENSORFLOW安装及编程基础

我们推荐使用GPU运行tensorflow，因为GPU比CPU更适合张量的运算，效率成倍数提升。但是并不是每台设备都配备GPU，没有GPU的设备可以使用CPU版本。

Tensorflow支持多种语言的，推荐使用python做为API，因为它代码简洁，容易上手，最重要的是能混合其他DL库一块使用。

Tensorflow需要python 3.5以上的运行环境，如果没有请下载安装。安装有python环境的执行pip install tensorflow或者 pip install tensorflow-gpu 即可进行安装，安装过程会自动下载安装所需要的依赖库。

有线性代数与矩阵，概率论，多重积分，级数变换，信息论等工科基础。

了解前馈神经网络（BP），卷积神经网络（CNN），循环神经网络（RNN）算法。

TENSOR（张量）

Tensor（张量）：张量是tensorflow核心的基本数据单元。 张量是一个多维数组。

rank：张量的秩,就是张量的维数。 shape：描述张量的形状。计算方法：去掉中括号，逗号+1就是数组的值。数组的的长度是rank。举例：

[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]

[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]

[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]

THE COMPUTATIONAL GRAPH
（计算图）

计算图是由一系列的tensorflow的操作组成，并且这些操作编配成计算图的节点。 站在计算图的角度，你可以认为tensorflow程序是有相对独立的两部分组成。 构建计算图、运行计算图。

```
node1 = tf.constant(3.0, dtype=tf.float32)

node2 = tf.constant(4.0)

print(node1, node2)
```

打印结果：

```
Tensor("Const:0", shape=(), dtype=float32)

Tensor("Const_1:0", shape=(), dtype=float32)
```

SESSION（会话）

Session：囊括tensorflow运行时候的状态，运行控制。

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

结果：[3.0, 4.0]

我们当然可以写出更复杂的计算图。

```
from __future__ import print_function
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))
```

打印出两行：

```
node3: Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3): 7.0
```

PLACEHOLDERS（占位符）

placeholders：在计算图中，能够接受额外输入，通常情况下，提供的值晚于定义。

a = tf.placeholder(tf.float32)

b = tf.placeholder(tf.float32)

adder_node = a + b 以上三行代码，很像一个函数，或者是lambda表达式。我们可以通过feed_dict参数

去填充一些具体的值。

print(sess.run(adder_node, {a: 3, b: 4.5}))

print(sess.run(adder_node, {a: [1, 3], b: [2, 4]}))

输出结果：

7.5 [3. 7.]

VARIABLES（变量）

Variables ：是在模型训练中，允许修改的量。相同的输入，通过修改变量得到不同输出值。在线性模型中用来描述权重和偏移量。

```
W = tf.Variable([.3], dtype=tf.float32)

b = tf.Variable([-.3], dtype=tf.float32)

x = tf.placeholder(tf.float32)

linear_model = W*x + b
```

变量必须显示申明，使用：

```
init = tf.global_variables_initializer()//初始化全局变量

sess.run(init)

print(sess.run(linear_model, {x: [1, 2, 3, 4]}))
```

TF.TRAIN

tf.train是用于训练模型的低级API。

最简单的优化方式，梯度下降。就是求损失函数的导数，每一步都会逼近损失函数最小值。当然tensorflow提供tf.gradients自动求导，自动优化损失函数。

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
sess.run(init) # reset values to incorrect defaults.
for i in range(1000):
    sess.run(train, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]})
print(sess.run([W, b]))
```

```
[array([-0.9999969], dtype=float32), array([ 0.99999082], dtype=float32)]
```

TF.ESTIMATOR

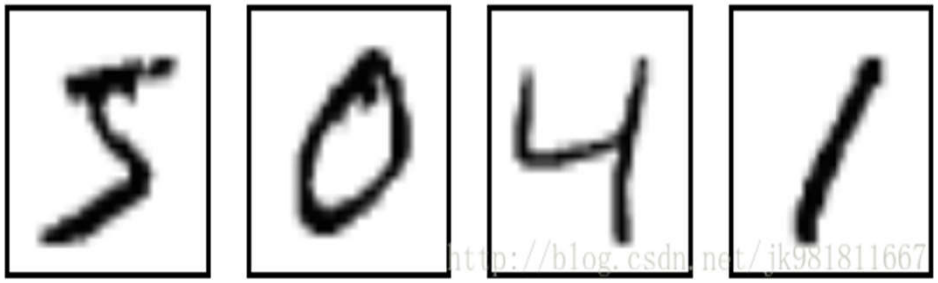
tf.estimator是一个高级API，定义了许多通用型模型。包括运行训练循环、运行评估循环、管理数据集等功能。

```
feature_columns = [tf.feature_column.numeric_column("x", shape=[1])]
estimator = tf.estimator.LinearRegressor(feature_columns=feature_columns)
x_train = np.array([1., 2., 3., 4.])
y_train = np.array([0., -1., -2., -3.])
x_eval = np.array([2., 5., 8., 1.])
y_eval = np.array([-1.01, -4.1, -7, 0.])
input_fn = tf.estimator.inputs.numpy_input_fn(
    {"x": x_train}, y_train, batch_size=4, num_epochs=None, shuffle=True)
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    {"x": x_train}, y_train, batch_size=4, num_epochs=1000, shuffle=False)
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    {"x": x_eval}, y_eval, batch_size=4, num_epochs=1000, shuffle=False)
estimator.train(input_fn=input_fn, steps=1000)
train_metrics = estimator.evaluate(input_fn=train_input_fn)
eval_metrics = estimator.evaluate(input_fn=eval_input_fn)
print("train metrics: %r"% train_metrics)
print("eval metrics: %r"% eval_metrics)
```

MNIST实例分析

MNIST：是一个入门级的计算机视觉数据集。

它包含各种手写数字图片：

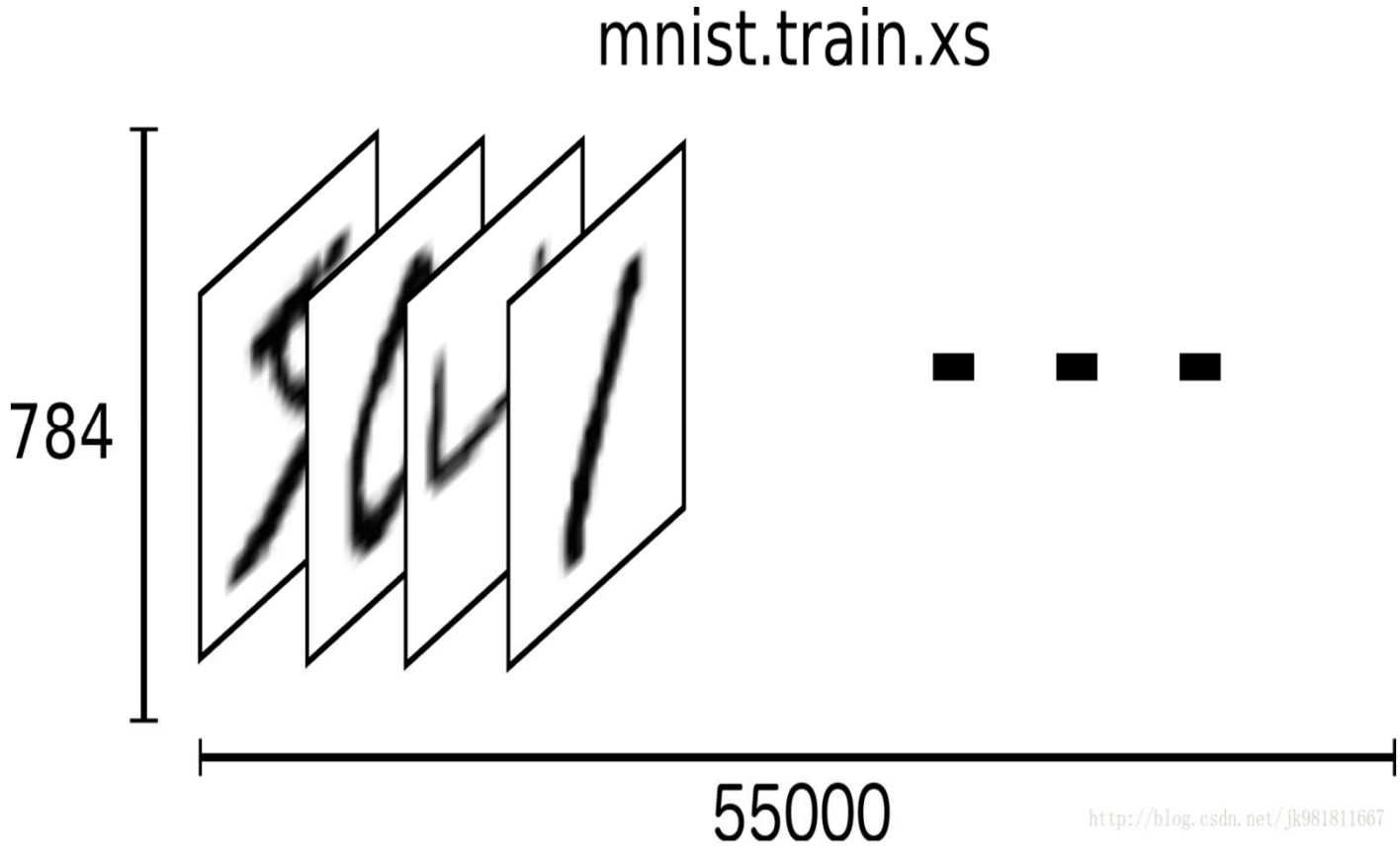


上面这4张图片的标签是5，0，4，1。

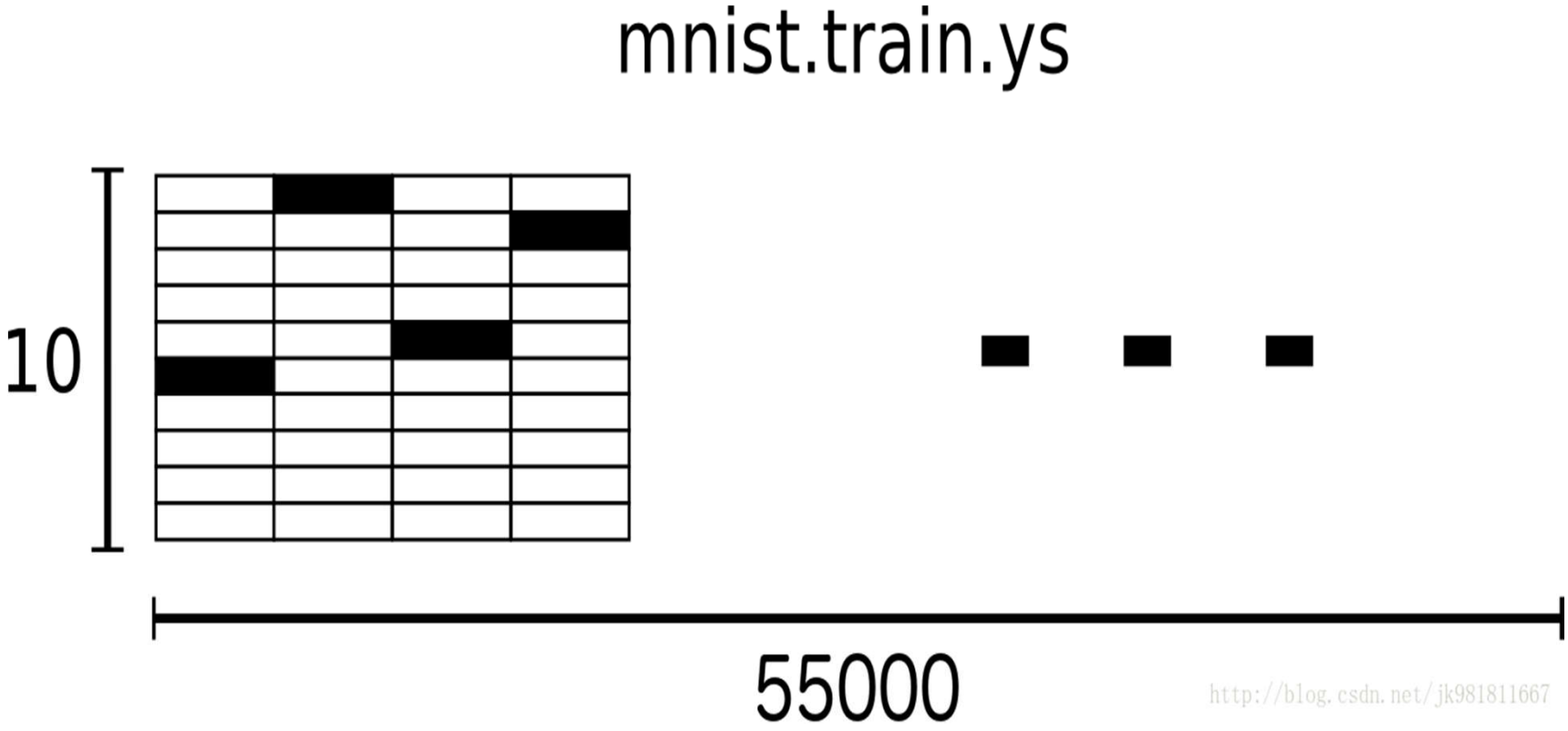
实例目的：使用tensorflow训练一个手写体图像识别模型，识别一张手写数字照片。

实例数据：28像素X28像素预处理之后的照片。训练集：55000行 ；预测集：10000行 ；

交叉验证集：5000行



mnist.train.images 是一个形状为 [55000, 784] 的张量，第一个维度数字用来索引图片，第二个维度数字用来索引每张图片中的像素点。在此张量里的每一个元素，都表示某张图片里的某个像素的强度值，值介于0和1之间。



one-hot vectors数字n将表示成一个只有在第n维度（从0开始）数字为1的10维向量。
比如，标签0将表示成([1,0,0,0,0,0,0,0,0,0,0])。mnist.train.labels是一个形状 [55000, 10] 的张量

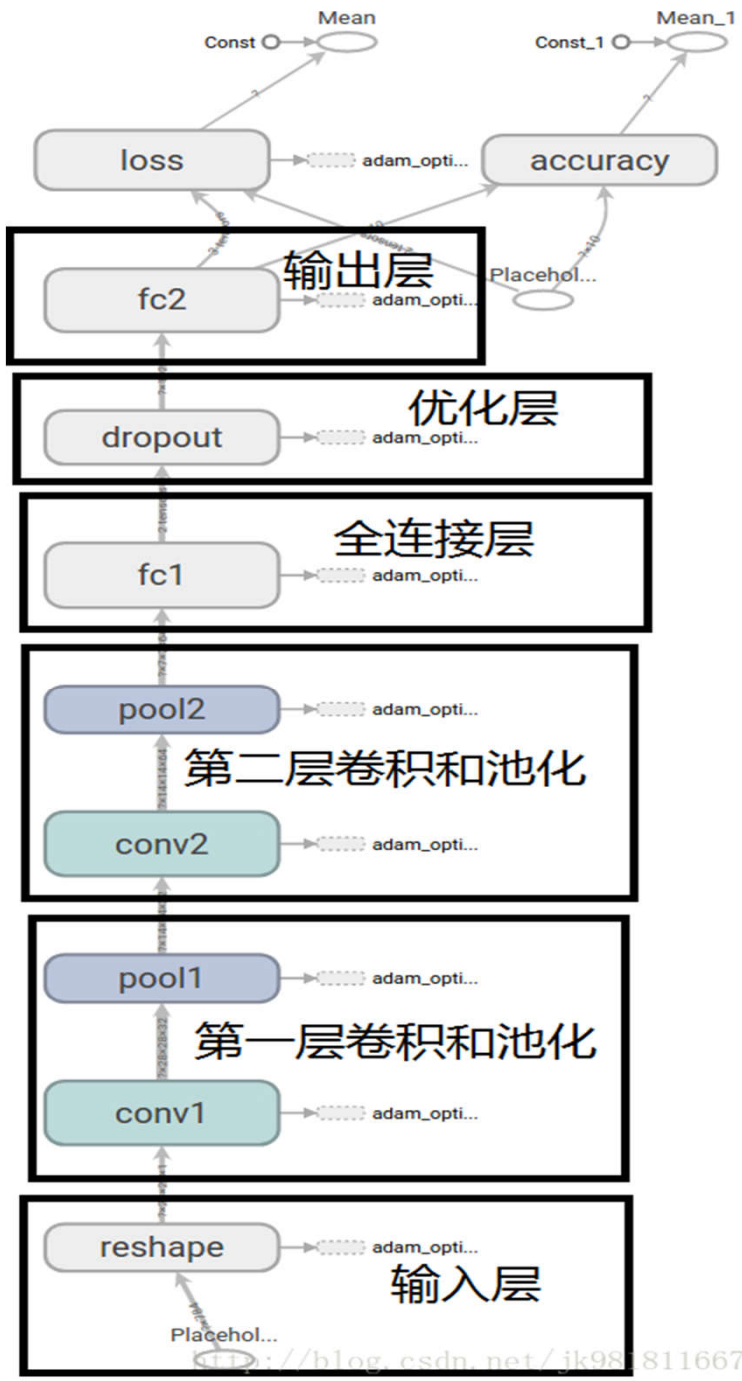
数据下载地址

文件	内容
train-images-idx3-ubyte.gz	训练集图片 - 55000 张 训练图片, 5000 张 验证图片
train-labels-idx1-ubyte.gz	训练集图片对应的数字标签
t10k-images-idx3-ubyte.gz	测试集图片 - 10000 张 图片
t10k-labels-idx1-ubyte.gz	测试集图片对应的数字标签

MNIST_data百度网盘地址：https://pan.baidu.com/s/1cjm7tc

input_data.py百度网盘地址：https://pan.baidu.com/s/1c21zu4C

input_data.py用于下载训练和测试的MNIST数据集的python源码。



权重初始化
WEIGHT INITIALIZATION

在神经网络中会创建大量的权重和偏置值。如何初始化这些variables。

为避免零梯度，我们使用tf.truncated_normal(shape, mean, stddev)生成正态分布的值。shape表示生成张量的维度，mean是均值，stddev是标准差。这样聚能保证随机初始化的权重，偏置值不同。

正态分布：统计样本常见的一种数值分布。自然情况下，人的身高是属于正态分布的。

```
def weight_variable(shape):
  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)
```

卷积和池化
CONVOLUTION AND POOLING

tensorflow提供非常灵活的卷积，池化操作。

TensorFlow also gives us a lot of flexibility in convolution and pooling operations.

卷积使用1步长（stride size），0边距（padding size）的模板，保证输出和输入是同一个大小。

池化用简单传统的2x2大小的模板做max pooling。

```
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

step1，输入层

```
x_image = tf.reshape(x, [-1,28,28,1])
```

tf.reshape(tensor, shape, name=None)调整tensor形状。

step2，第一层卷积和池化

```
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)
h_pool1 = max_pool_2x2(h_conv1)
```

step3，第二层卷积和池化

```
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

step4，全连接层

```
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

step5，优化层

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

step6，输出层

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

step7，评估和训练

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
sess.run(tf.initialize_all_variables())
for i in range(20000):
  batch = mnist.train.next_batch(50)
  if i%100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
        x:batch[0], y_: batch[1], keep_prob: 1.0})
    print "step %d, training accuracy %g"%(i, train_accuracy)
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print "test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
```