Tensorflow介绍

Tensorflow安装

- Linux CPU-only: Python 2 (build history) / Python 3.4 (build history) / Python 3.5 (build history)
- Linux GPU: Python 2 (build history) / Python 3.4 (build history) / Python 3.5 (build history)
- Mac CPU-only: Python 2 (build history) / Python 3 (build history)
- Mac GPU: Python 2 (build history) / Python 3 (build history)
- Windows CPU-only: Python 3.5 64-bit (build history)
- Windows GPU: Python 3.5 64-bit (build history)
- Android: demo APK, native libs (build history)

https://github.com/tensorflow/tensorflow

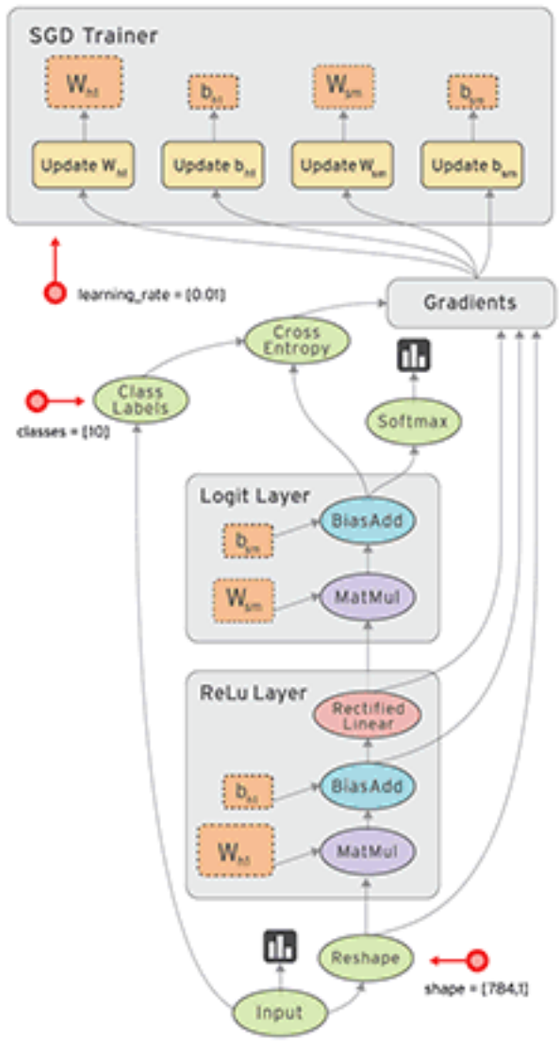
例如：（CPU，Linux下）

pip install https://ci.tensorflow.org/view/Nightly/job/nightly-matrix-cpu/TF_BUILD_IS_OPT=OPT,TF_BUILD_IS_PIP=PIP,TF_BUILD_PYTHON_VERSION=PYTHON2,label=cpu-slave/lastSuccessfulBuild/artifact/pip_test/whl/tensorflow-1.1.0rc1-cp27-none-linux_x86_64.whl

Tensorflow简介

一个使用数据流图(data flow graphs)技术来进行数值计算的开源软件库。数据流图中的节点，代表数值运算；节点节点之间的边，代表多维数据(tensors)之间的某种联系。你可以在多种设备（含有CPU或GPU）上通过简单的API调用来使用该系统的功能。TensorFlow是由Google Brain团队的研发人员负责的项目。

数据流图(Data Flow Graph)



- 数据流图是描述有向图中的数值计算过程。有向图中的节点通常代表数学运算，但也可以表示数据的输入、输出和读写等操作；有向图中的边表示节点之间的某种联系，它负责传输多维数据(Tensors)。图中这些tensors的flow也就是TensorFlow的命名来源。

- **1 灵活性**

- TensorFlow不是一个严格的神经网络工具包，只要你可以使用数据流图来描述你的计算过程，你可以使用TensorFlow做任何事情。你还可以方便地根据需要来构建数据流图，用简单的Python语言来实现高层次的功能。

- **2 可移植性**

- TensorFlow可以在任意具备CPU或者GPU的设备上运行，你可以专注于实现你的想法，而不用去考虑硬件环境问题，你甚至可以利用Docker技术来实现相关的云服务。

- **3 提高开发效率**

- TensorFlow可以提升你所研究的东西产品化的效率，并且可以方便与同行们共享代码。

- **4 支持语言选项**

- 目前TensorFlow支持Python和C++语言。（但是你可以自己编写喜爱语言的SWIG接口）

- **5 充分利用硬件资源，最大化计算性能**

- 你需要理解在TensorFlow中，是如何:
  - 将计算流程表示成图;
  - 通过Sessions来执行图计算;
  - 将数据表示为tensors;
  - 使用Variables来保持状态信息;
  - 分别使用feeds和fetches来填充数据和抓取任意的操作结果;

Ops(operator)

- TensorFlow是一种将计算表示为图的编程系统。图中的节点称为ops(operation的简称)。一个ops使用0个或以上的Tensors，通过执行某些运算，产生0个或以上的Tensors。一个Tensor是一个多维数组，例如，你可以将一批图像表示为一个四维的数组[batch, height, width, channels]，数组中的值均为浮点数。

Session

- TensorFlow中的图描述了计算过程，图通过Session的运行而执行计算。Session将图的节点们(即ops)放置到计算设备(如CPUs和GPUs)上，然后通过方法执行它们；这些方法执行完成后，将返回tensors。在Python中的tensor的形式是numpy ndarray对象，而在C/C++中则是tensorflow::Tensor.

图计算

- TnsorFlow程序中图的创建类似于一个 [施工阶段]，而在 [执行阶段] 则利用一个session来执行图中的节点。很常见的情况是，在 [施工阶段] 创建一个图来表示和训练神经网络，而在 [执行阶段] 在图中重复执行一系列的训练操作。

创建图

- 在TensorFlow中，Constant是一种没有输入的ops，但是你可以将它作为其他ops的输入。Python库中的ops构造器将返回构造器的输出。TensorFlow的Python库中有一个默认的图，将ops构造器作为节点（Graph class）。

```
import tensorflow as tf

# Create a Constant op that produces a 1x2 matrix.  The op is
# added as a node to the default graph.
#
# The value returned by the constructor represents the output
# of the Constant op.
matrix1 = tf.constant([[3., 3.]])

# Create another Constant that produces a 2x1 matrix.
matrix2 = tf.constant([[2.],[2.]])

# Create a Matmul op that takes 'matrix1' and 'matrix2' as inputs.
# The returned value, 'product', represents the result of the matrix
# multiplication.
product = tf.matmul(matrix1, matrix2)
```

默认的图(Default Graph)现在有了三个节点：两个Constant()ops和一个matmul()op。为了得到这两个矩阵的乘积结果，还需要在一个session中启动图计算。

执行图

```
# Launch the default graph.
sess = tf.Session()

# To run the matmul op we call the session 'run()' method, passing 'product'
# which represents the output of the matmul op.  This indicates to the call
# that we want to get the output of the matmul op back.
#
# All inputs needed by the op are run automatically by the session.  They
# typically are run in parallel.
#
# The call 'run(product)' thus causes the execution of threes ops in the
# graph: the two constants and matmul.
#
# The output of the op is returned in 'result' as a numpy `ndarray` object.
result = sess.run(product)
print(result)
# ==> [[ 12.]]

# Close the Session when we're done.
sess.close()
```

```
with tf.Session() as sess:
  result = sess.run([product])
  print(result)
```

自动释放Session资源

Tensors

- TensorFlow中使用tensor数据结构（实际上就是一个多维数据）表示所有的数据，并在图计算中的节点之间传递数据。一个tensor具有固定的类型、级别和大小，更加深入理解这些概念可参考Rank, Shape, and Type。

变量(Variables)

- 变量在图执行时候保持着自家的状态。

- 图右图，简单的计数器。

```python
# Create a Variable, that will be initialized to the scalar value 0.
state = tf.Variable(0, name="counter")

# Create an Op to add one to `state`.

one = tf.constant(1)
new_value = tf.add(state, one)
update = tf.assign(state, new_value)

# Variables must be initialized by running an `init` Op after having
# launched the graph.  We first have to add the `init` Op to the graph.
init_op = tf.initialize_all_variables()

# Launch the graph and run the ops.
with tf.Session() as sess:
  # Run the 'init' op
  sess.run(init_op)
  # Print the initial value of 'state'
  print(sess.run(state))
  # Run the op that updates 'state' and print 'state'.
  for _ in range(3):
    sess.run(update)
    print(sess.run(state))

# output:

# 0
# 1
# 2
# 3
```

抓取(Fetches)

- 抓取OP的输出

```
input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)

with tf.Session() as sess:
  result = sess.run([mul, intermed])
  print(result)

# output:
# [array([ 21.], dtype=float32), array([ 7.], dtype=float32)]
```

填充(Feeds)

- TensorFlow也提供这样的机制：先创建特定数据类型的占位符(placeholder)，之后再进行数据的填充。

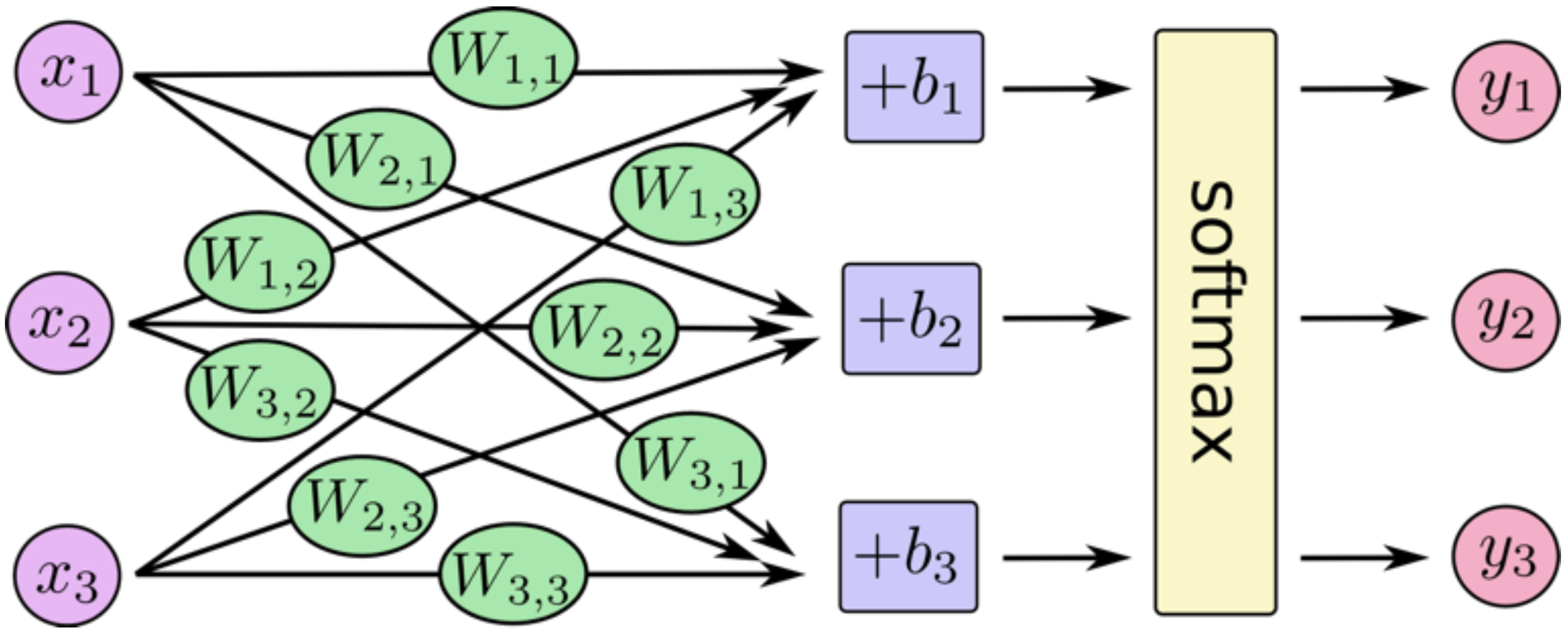
```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
  print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))

# output:
# [array([ 14.], dtype=float32)]
```

简单例子1：曲线拟合

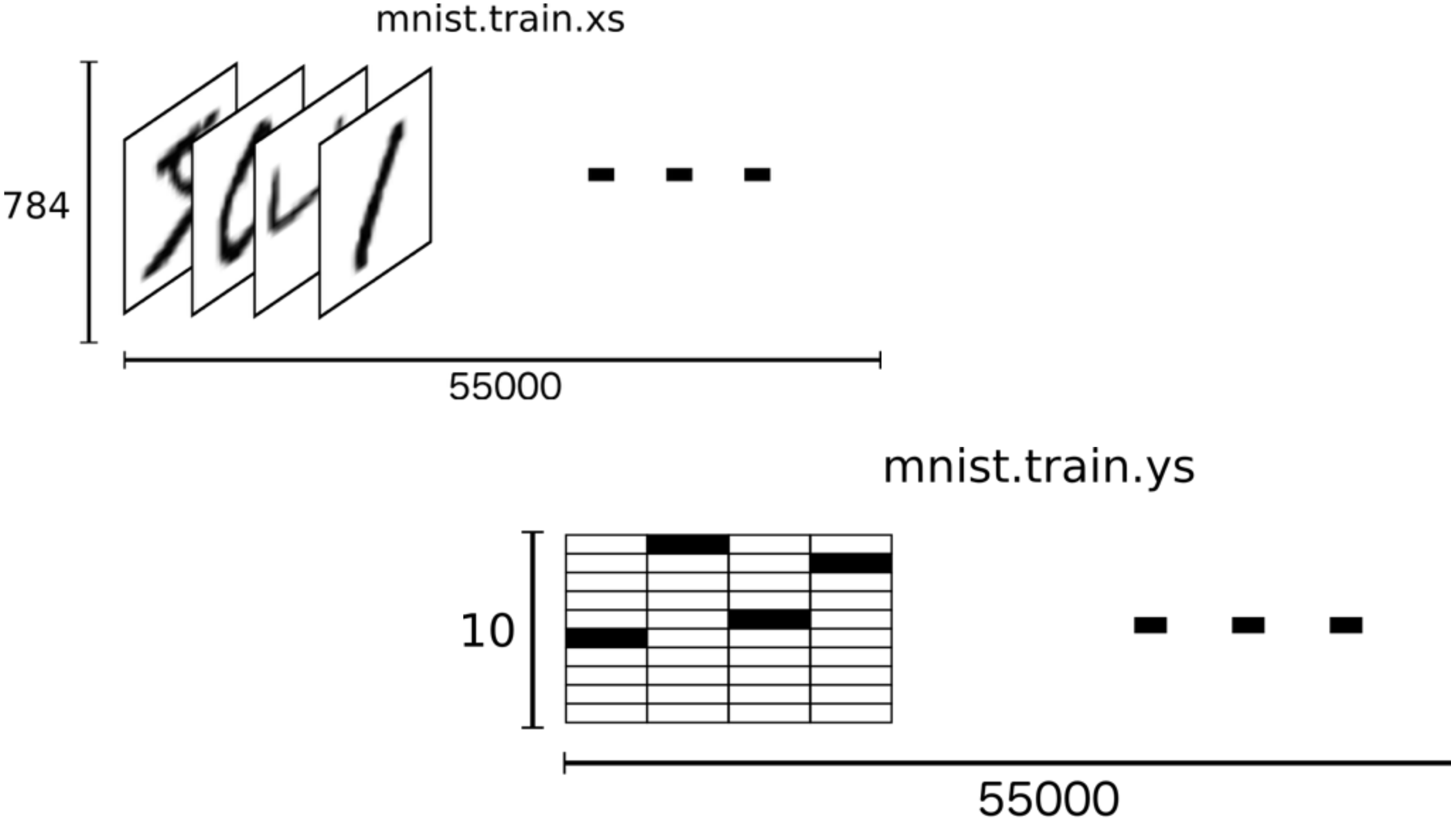
```
# 简化调用库名
import tensorflow as tf
import numpy as np

# 模拟生成100对数据对，对应的函数为y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype("float32")
y_data = x_data * 0.1 + 0.3

# 指定w和b变量的取值范围（注意我们要利用TensorFlow来得到w和b的值）
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# 最小化均方误差
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```
译
```
# 初始化TensorFlow参数
init = tf.initialize_all_variables()

# 运行数据流图（注意在这一步才开始执行计算过程）
sess = tf.Session()
sess.run(init)

# 观察多次迭代计算时，w和b的拟合值
for step in xrange(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# 最好的情况是w和b分别接近甚至等于0.1和0.3
```

简单例子2：MNIST手写体识别任务

模型图


x_1
x_2
x_3
$W_{1,1}$
$W_{2,1}$
$W_{1,2}$
$W_{3,2}$
$W_{1,3}$
$W_{2,2}$
$W_{3,1}$
$W_{2,3}$
$W_{3,3}$
$+b_1$
$+b_2$
$+b_3$
softmax
y_1
y_2
y_3

数据



```python
# -*- coding: utf-8 -*-

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
print("Download Done!")

x = tf.placeholder(tf.float32, [None, 784])

# paras
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])

# loss func
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))

train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

# init
init = tf.initialize_all_variables()

sess = tf.Session()
sess.run(init)

# train
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

correct_prediction = tf.equal(tf.arg_max(y, 1), tf.arg_max(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

print("Accuarcy on Test-dataset: ", sess.run(accuracy, feed_dict={x: mnist.test.
    images, y_: mnist.test.labels}))
```