第2章

搭建TensorFlow环境

前面介绍了深度学习与人工智能的基础知识，以及TensorFlow框架的相关特性。本章将带领大家一步一步地将TensorFlow环境搭建起来。

2.1 基于pip安装

2.1.1 基于Windows环境安装TensorFlow

TensorFlow从0.12.0版本开始支持在Windows操作系统下安装，在Windows操作系统下安装TensorFlow的必备环境有：

1．Windows 7版本以上的64位操作系统；

2．64位Python 3.6.X的发行版本；

3．TensorFlow 0.12.0以上的版本；

4．GPU版本需要CUDA8和CUDNN5.1以上版本。

第一步 安装Python

TensorFlow在Windows上只支持64位3.5.X以上版本的Python。在选择Python发行版的时候，可以选择Python Releases for Windows版本或者Anaconda for Python版本。由于Anaconda for Python版本中包含了深度学习所需要的绝大多数的第三方库，所以推荐使用Anaconda for Python作为Windows下的开发环境。下面开始一步一步地安装Anaconda。

（1）下载Anaconda

Anaconda的下载地址为https://www.anaconda.com/download/，打开浏览器，输入网址后进入Anaconda的下载界面，如图2-1所示。

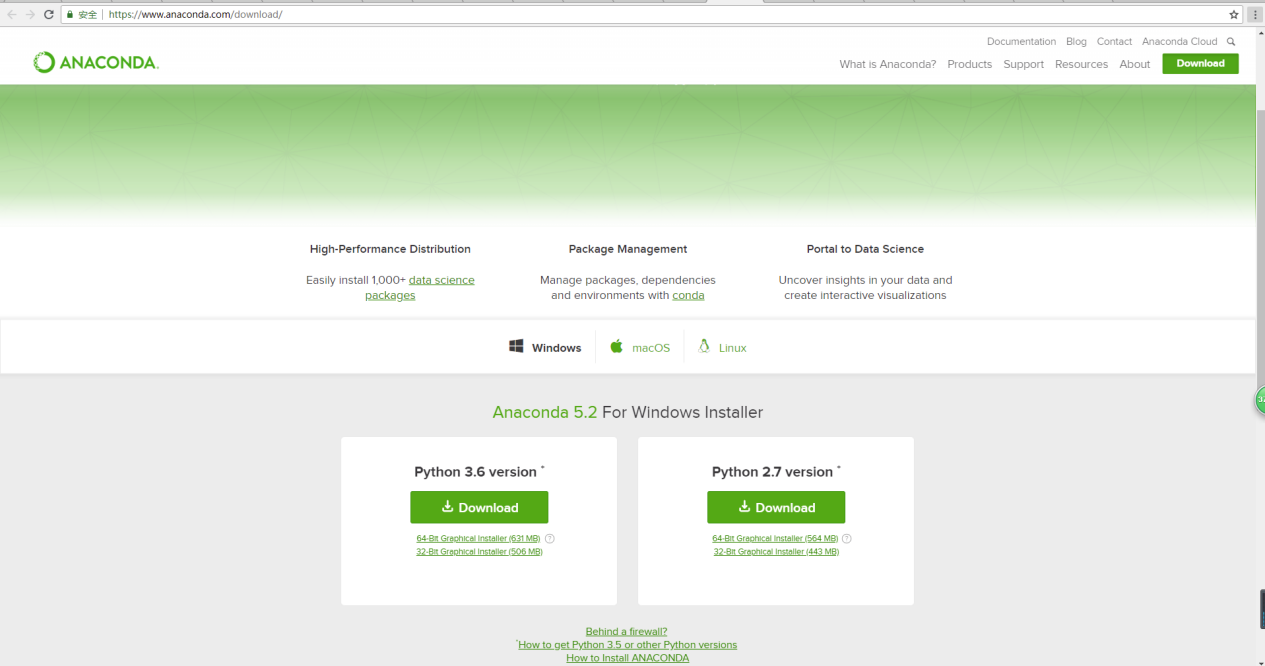


图2-1

由于TensorFlow只支持在64位Python下进行开发，所以一定要选择64-bit版本，千万不要选择32-bit版本，如图2-2所示。

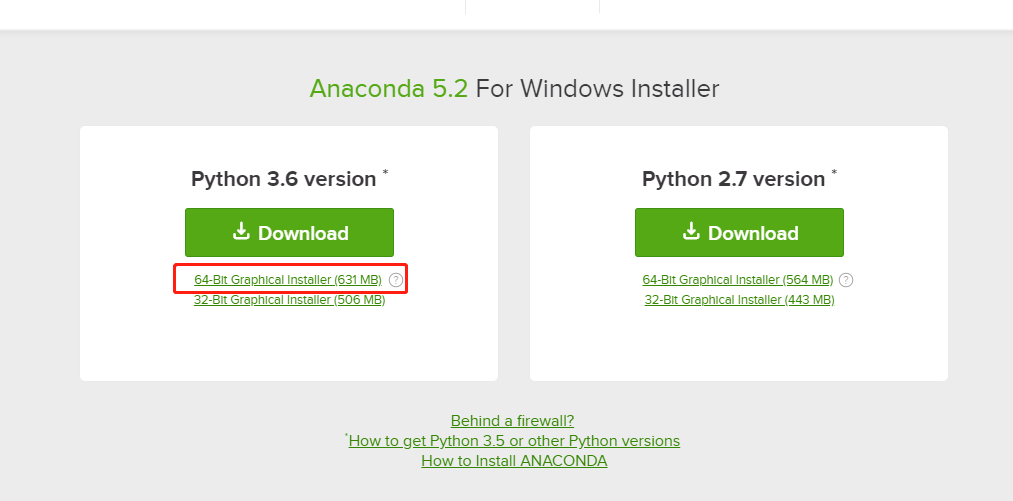


图2-2

即进入下载列表页面，选择Anaconda3-5.2.0-Windows-x86\_64.zip，等待下载完成即可，如果由于网络原因导致下载多次失败，可以尝试使用清华大学的源地址进行下载，下载地址为https://mirrors.tuna.tsinghua.edu.cn/ anaconda/archive/，如图2-3所示。



图2-3

（2）安装Anaconda

下载完成之后，双击下载后的安装文件，进行安装，如图2-4所示。

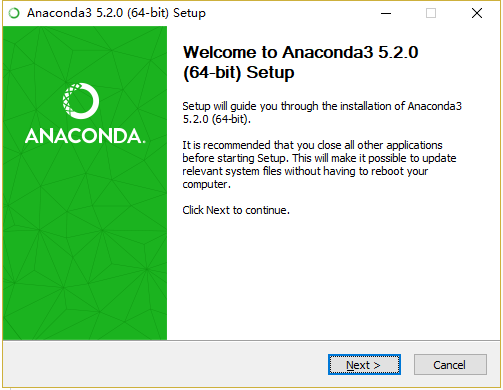


图2-4

单击Next>按钮，弹出协议界面，单击I Agree，进入选择安装类型的界面，如图2-5所示。

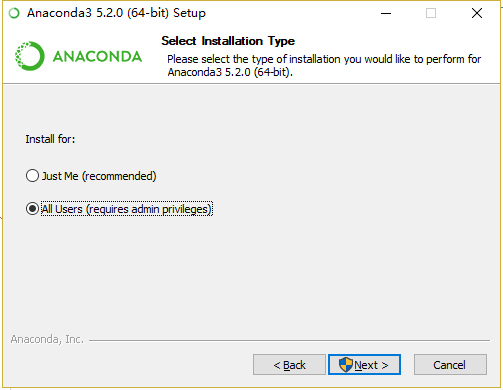


图2-5

在选择安装类型的界面中，建议选择All Users(requires admin privileges)，此项说明在使用计算机的所有用户都可以使用Anaconda开发环境，然后单击Next>按钮进入安装位置选择界面，选择路径后，单击Next>按钮继续安装，如图2-6所示。

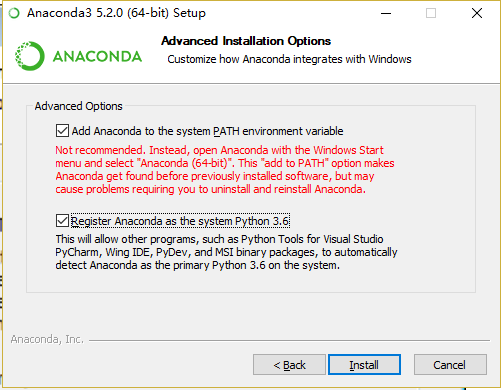


图2-6

在安装选项中，第一个选项是告诉我们将Anaconda加入到系统的环境变量里，第二个选项是告诉我们使用Python 3.6注册Anaconda到系统中，建议两个选项全部勾选，然后单击Install按钮进行安装，等待大约5分钟，整个程序安装完成。

接下来，进入控制台，输入conda --version，如果出现如图2-7所示的界面，则说明Anaconda安装成功，接下来就可以进行TensorFlow的安装了。



图2-7

第二步 安装TensorFlow

刚刚已经成功安装了Anaconda环境，接下来就要在Anaconda环境下安装TensorFlow了。

TensorFlow无论在哪个环境下都分为CPU和GPU两个版本，其中GPU只支持NVIDIA（英伟达）的芯片，也就是俗称的“N卡”，并且需要支持CUDA和cuDNN；CPU版本并无其他限制，一般推荐使用Intel系列的CPU。

在确保以上信息之后，下面将开始安装TensorFlow。首先通过conda create -n tensorflow-t python=3.6命令在Anaconda下创建一个虚拟的TensorFlow环境，如果第一次使用，则会出现如图2-8所示的提示，输入Y继续安装：

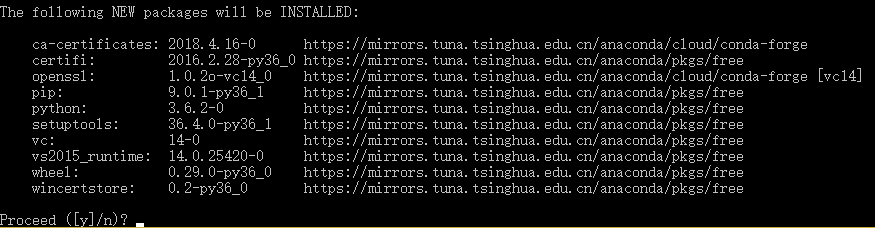


图2-8

当出现如图2-9所示的界面，则表示虚拟环境创建成功。

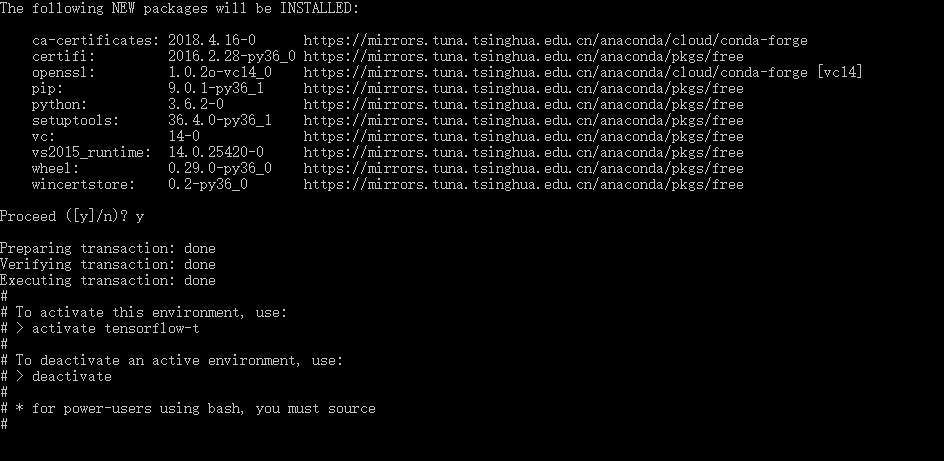


图2-9

输入activate tensorflow-t激活tensorFlow虚拟环境，如图2-10所示。



图2-10

执行pip install tensorflow --upgrade安装TensorFlow，此时，系统会自动安装TensorFlow以及所需的依赖。安装完成后的界面如图2-11所示。

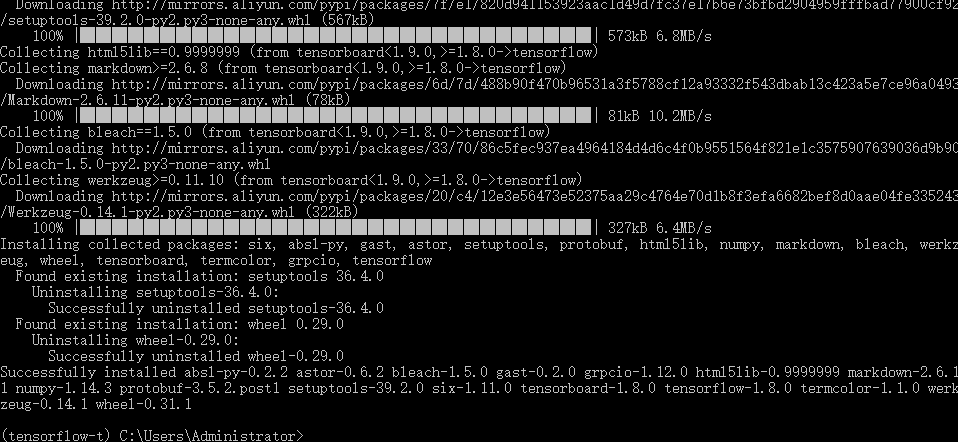


图2-11

如果要安装GPU版本的TensorFlow，只需要将上述的执行命令改为：pip install tensorflow-gpu --upgrade即可，关于GPU版本的安装和使用，我们会在后面的章节中讲解。

**第三步 运行TensorFlow**

在控制台（CMD）输入activate tensorflow-t进入tensorflow虚拟环境，然后输入python，进入python环境输入代码：

>>> import tensorflow as tf

>>> hello = tf.constant(‘hello Tensorflow!’)

>>> sess = tf.Session()

>>> print(sess.run(hello))

>>> b’hello Tensorflow!’

如果在控制台成功输出“hello Tensorflow!”，则说明Tensorflow运行成功。

小贴士

如果采用pip安装TensorFlow会出现如图2-12所示的警告信息，这是由于使用的是预编译版本进行安装的，不会影响使用，如果不想看到这个提示，可使用源码安装的形式进行安装。

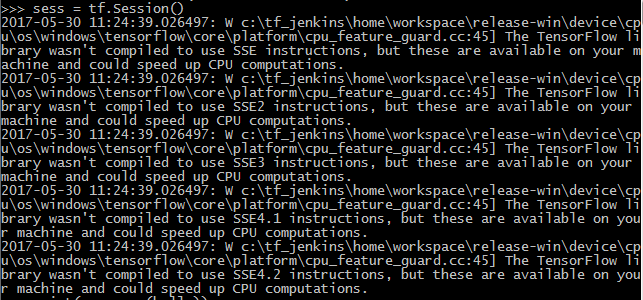


图2-12

2.1.2 基于Linux环境安装TensorFlow

首先通过python -V命令来查看Linux下Python的版本，如果版本小于3.X，建议升级到3.6.X版本的Python。

**第一步 安装或升级Python 3.6**

（1）输入：wget https://www.python.org/ftp/python/3.6.2/Python-3.6.2.tgz命令，下载python 3.5.2安装包。

（2）下载完成后，使用tar -zxvf Python-3.6.2.tgz命令进行解压缩。

（3）通过cd Python-3.6.2/命令进入python目录。

（4）执行./configure命令，安装Python，执行完成后如图2-13所示。

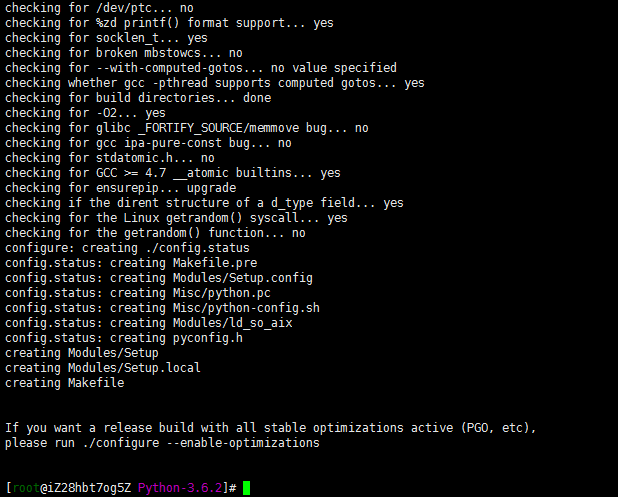


图2-13

（5）输入make命令，编译Python。

（6）编译完成后，输入make install进行安装，当出现如图2-14所示的界面，说明安装完成，不过此时我们的Python版本还是系统默认的版本（一般是2.7.X），此时需要通过ln -s /usr/local/bin/python3.6 /usr/bin/python命令进行一下版本替换后即可。

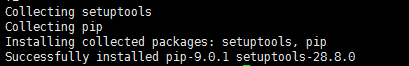


图2-14

（7）输入python -V命令，显示版本为3.6.2，说明安装Python 3.6.2成功，如图2-15所示。



图2-15

**第二步 安装TensorFlow**

接下来，输入pip install tensorflow --upgrade进行安装；如果看到如图2-16所示的界面，则说明安装成功。

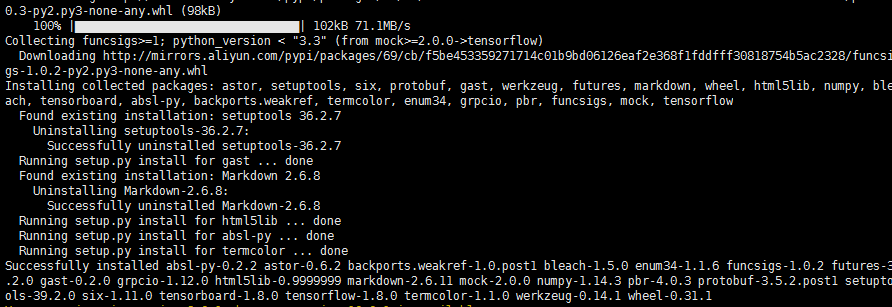


图2-16

如果要安装GPU版本的TensorFlow，只需要将命令改为pip install tensorflow-gpu --upgrade即可。

**第三步 运行TensorFlow**

首先输入python，进入python环境，然后在控制台输入以下代码：

>>> import tensorflow as tf

>>> hello = tf.constant(‘hello Tensorflow!’)

>>> sess = tf.Session()

>>> print(sess.run(hello))

>>> b’hello Tensorflow!’

如果在控制台成功输出“hello Tensorflow!”，则说明Tensorflow运行成功。

2.2 基于Java安装TensorFlow

TensorFlow目前也支持使用Java语言进行开发，下面将一步步讲解如何基于Java语言安装TensorFlow。

（1）首先下载TensorFlow的jar包和jni所需要的dll文件，下载地址为：https://www.tensorflow.org/install/install\_java，向下拉动，下载jar包和jni文件，如图2-17所示。

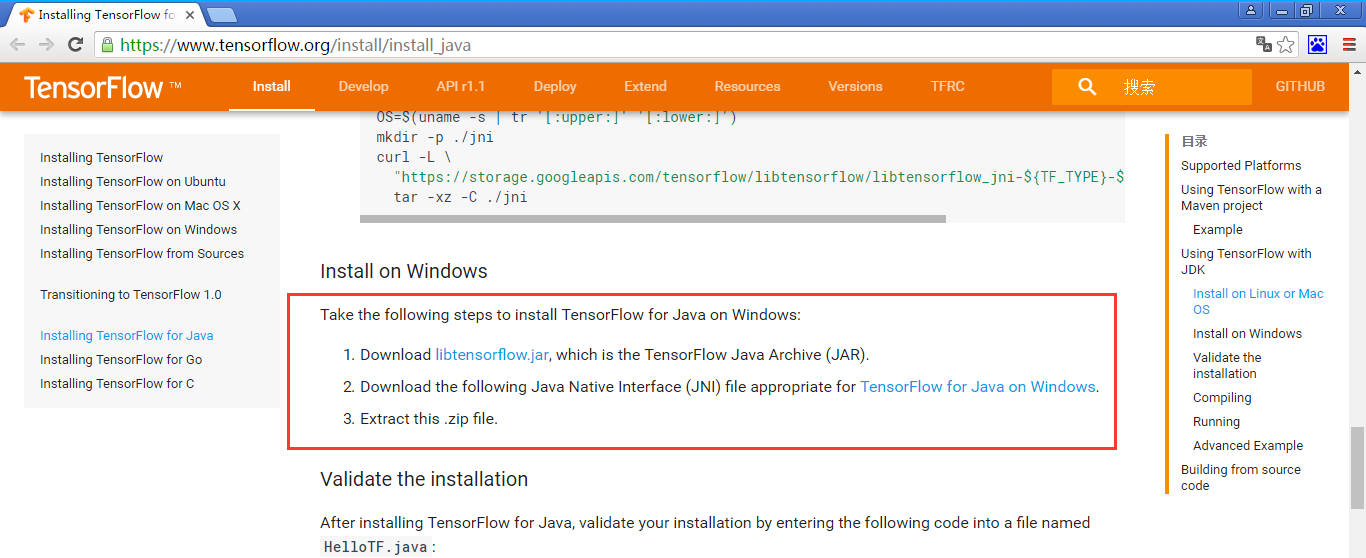


图2-17

接下来，在本地使用Eclipse新建一个java项目，并将刚刚下载好的jar包引入进来，如图2-18所示。

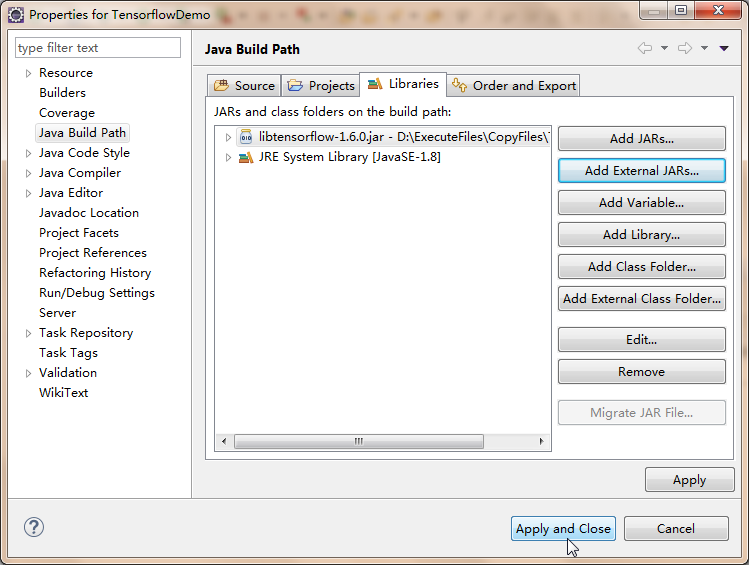


图2-18

将刚刚下载的jni压缩包解压，将解压后的.dll文件先放到libs目录下（需要自己建立），如图2-19所示。

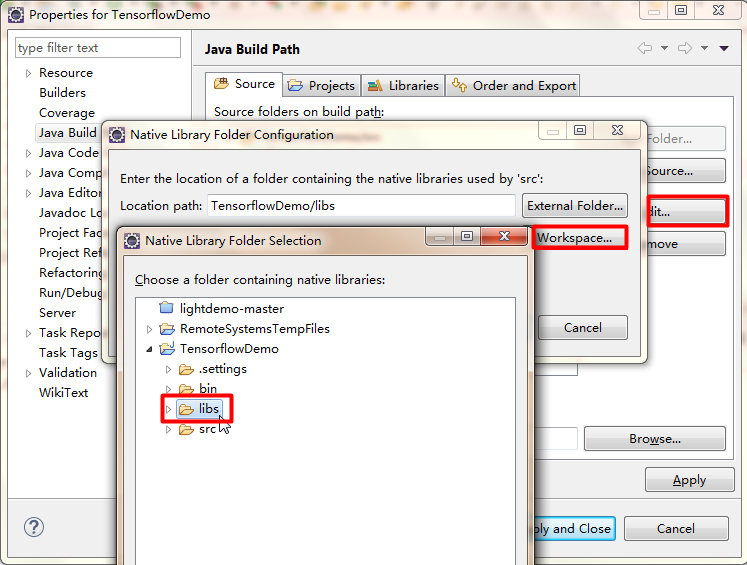


图2-19

然后引用到工程下，如图2-20所示。

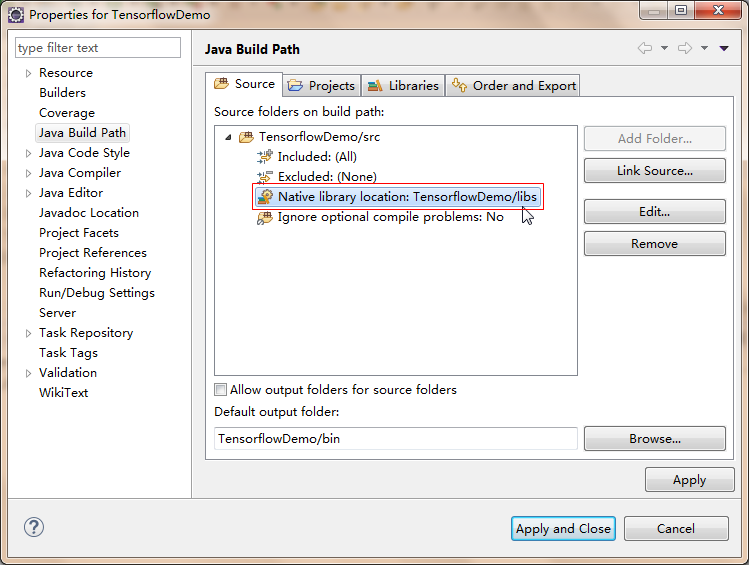


图2-20

最后完整的目录如图2-21所示。

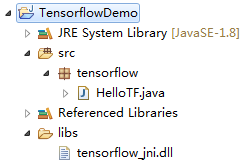


图2-21

（2）编写代码：

**package** testTensorflow;

**import** org.tensorflow.TensorFlow;

**public** **class** HelloTensorflow {

**public** **static** **void** main(String[] args) {

System.*out*.println(TensorFlow.*version*());

}

}

（3）运行java文件，如果打印出版本名称，则说明运行成功，如图2-22所示。

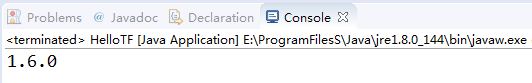


图2-22

此时，我们成功将TensorFlow使用Java运行。

2.3 安装TensorFlow的常用依赖模块

在使用TensorFlow进行开发时会经常使用一些第三方的依赖库来提高开发效率，在本书后面的实战部分也会经常用到这些扩展依赖库，下面来介绍并安装TensorFlow所需要的模块。

在TensorFlow开发过程中，我们会经常用到以下扩展包。

1．NumPy

NumPy系统是Python开源的数值计算扩展，用来存储和处理大型矩阵，比Python自身的嵌套列表（nested list structure）结构要高效得多（该结构也可以用来表示矩阵（matrix））。据说Python因NumPy而可变身成免费的更强大的MATLAB系统。

NumPy扩展的pip安装方法为：

pip/pip3 install numpy --upgrade

2．SciPy

SciPy是一款方便、易于使用、专为科学和工程设计的Python工具包。它包括统计、优化、整合、线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解器，等等。

SciPy扩展的pip安装方法为：

pip/pip3 install scipy --upgrade

如果在Windows中无法安装，则建议进入官方网站下载安装包，然后执行以下命令进行安装。

>python setup.py build

>python setup.py install

3．Matplotlib

Matplotlib 是一个 Python 的 2D绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形。通过 Matplotlib，开发者仅需要几行代码，便可以生成绘图、直方图、功率谱、条形图、错误图、散点图等。

Matplotlib扩展的pip安装方法为：

pip/pip3 install matplotlib --upgrade

4．TFLearn

TFLearn是一个构建在TensorFlow之上的模块化和透明的深度学习库，它为TensorFlow提供高层次API，可快速搭建实验环境，同时保持对TensorFlow的完全透明和兼容性。后面会详细地讲解TFLearn框架。

TFLearn框架的pip安装方法为：

pip/pip3 install git+https://github.com/tflearn/tflearn.git

5．Librosa

Librosa库可以很方便地提取音频元素，对音频进行各种处理。

Librosa库的pip安装方法为：

pip install librosa --upgrade

以上库可使用Anaconda进行安装，安装步骤如下所示。

（1）打开Anaconda Navigator，切换到Environments选项卡，在右侧搜索要安装的扩展，如图2-23所示。

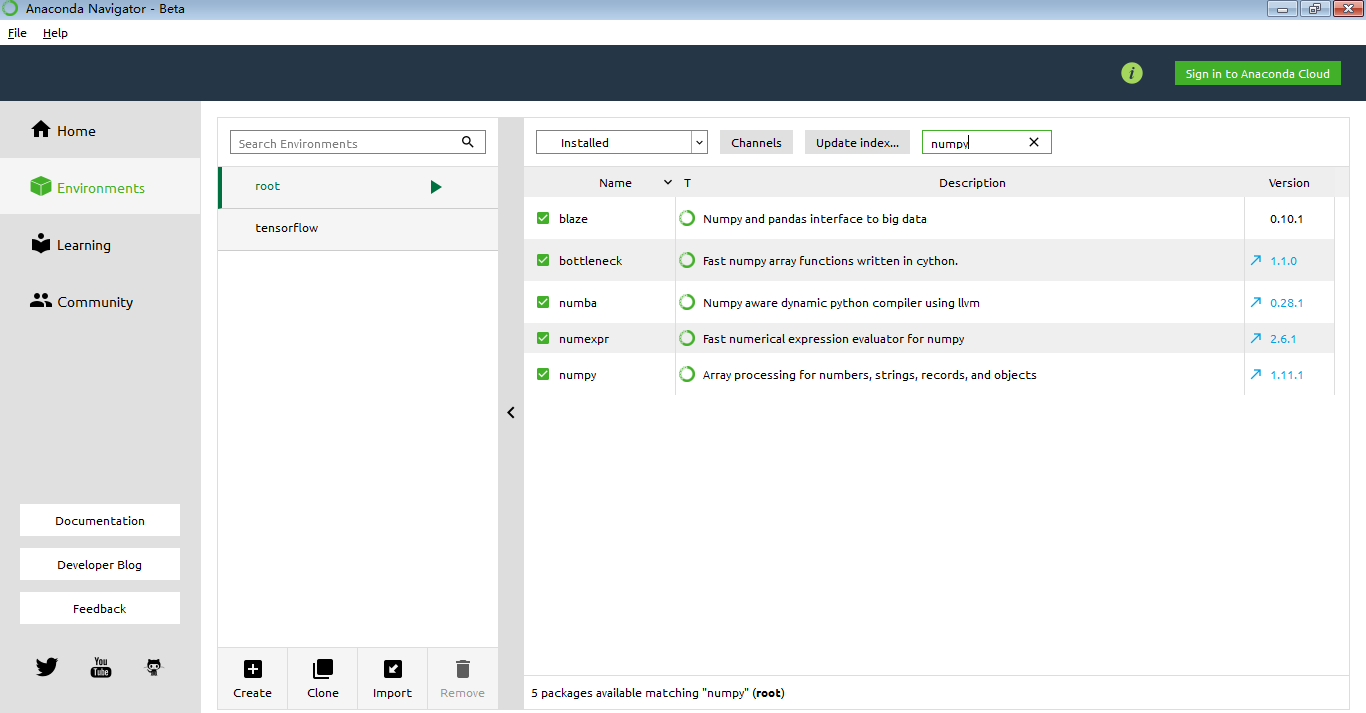


图2-23

（2）选择要安装的扩展，单击Apply按钮进行安装，如图2-24所示。

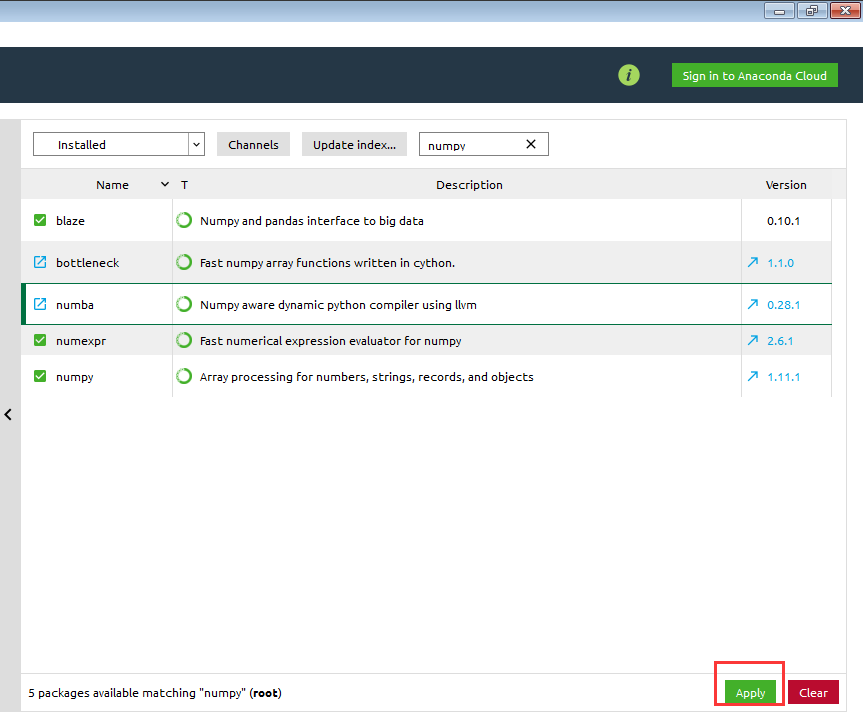


图2-24

2.4 Hello TensorFlow

TensorFlow安装好之后就带大家走进TensorFlow的代码世界，一步步地建立起自己的深度学习模型。这里将会使用TensorFlow官方最为推荐的MNIST数据集实现一个手写数字识别程序。

2.4.1 MNIST数据集

MNIST数据集是Google实验室和纽约大学柯朗研究所共同建立的一个手写数字的数据库（一般习惯称为手写数字），该数据库中存有60000张手写数字的训练图像和10000张手写数字的测试图像。我们可以通过<http://yann.lecun.com/exdb/mnist/> 网站直接下载数据集，下载之后会包含train-images-idx3-ubyte.gz、train-images-idx3- ubyte.gz、[t10k-images-idx3-ubyte. gz](http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz)、t10k-labels-idx1-ubyte.gz四个文件，前两个文件是训练数据集的图像和标签，后两个文件是测试数据集的图像和标签。我们所下载的这些图像并不是标准的图像格式（例如jpg，png等），而是将它们都保存在二进制文件中，并被转化成一个28×28的矩阵。如果通过程序将二进制文件转换成图片，将会得到类似如图2-25所示的集合。

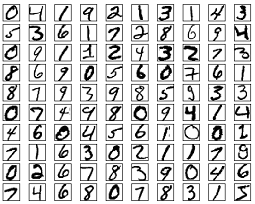


图2-25

其中每一个小的数字如图2-26所示。

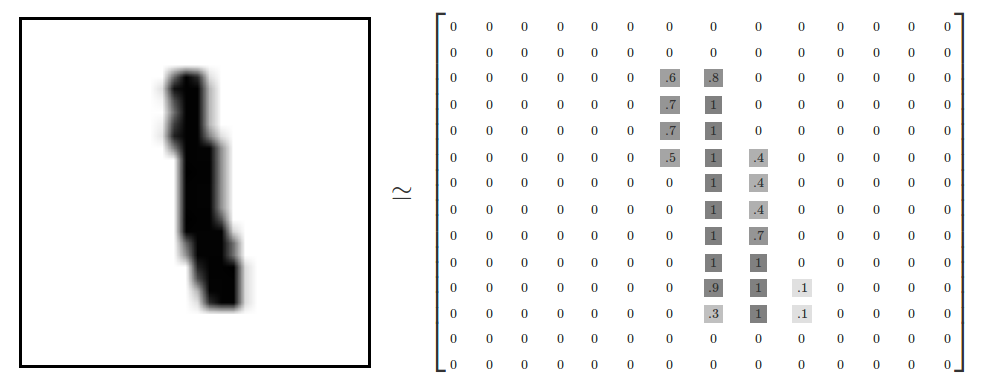


图2-26

图2-26右侧为左侧数字展开的图像，图像总是在矩阵的正中央。

还可以通过脚本文件下载MNIST数据集，脚本文件可以在TensorFlow目录\examples\tutorials\mnist下找到，为了使用方便，一般建议将其复制一份放在自己工程的目录下。

文件下载完成后，可以编写一个脚本test\_size.py来查看下载的数据是否正确：

import input\_data  
mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

print("训练数据的大小为：%d" % (mnist.train.num\_examples))

print("测试数据集的大小为：%d" % (mnist.test.num\_examples))

如果能够正确地输出数据集数量，则说明我们下载的数据集无误：

Extracting MNIST\_data/train-images-idx3-ubyte.gz

Extracting MNIST\_data/train-labels-idx1-ubyte.gz

Extracting MNIST\_data/t10k-images-idx3-ubyte.gz

Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz

训练数据集的大小为：55000。

测试数据集的大小为：10000。

2.4.2 编写训练程序

首先，新建一个项目目录，将官方的input\_data.py文件放入项目根目录下，并新建一个mnist.py文件，然后导入tensorflow包和input\_data.py文件，并使用变量mnist来读取数据，在读取数据的过程中，如果发现本地没有相关文件，则会自动下载。

import tensorflow as tf  
import input\_data  
mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

然后定义一个变量*x*，并给变量*x*设置一个placeholder，在TensorFlow中，placeholder一般用来描述一个等待输入的节点，将这个节点设置为float类型，并设定其形状（shape）为[None,784]，这里的None意味着可以是任意的数，一般可以把placeholder理解为一个占位符，先将这个位置给预先留出，等到我们后面需要使用的时候，再对其进行赋值操作。

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

接着使用tf.Variable()方法定义两个变量，tf.Variable()在前面的章节中讲过，主要为了定义变量，这里使用tf.zeros()方法来定义一个形状为[784,10]的全0张量，也可以理解为一个全0矩阵，这里的10一般可以理解为有10类，将其定义为*w*，也就是weight，即权重。

w = tf.Variable(tf.zeros([784,10]))

再定义一个偏移量bias，将其设置为一个10维的向量。

b = tf.Variable(tf.zeros([10]))

下面把向量化后的图片*x*和权重矩阵***w***相乘，加上偏置量*b*，然后计算每个分类的Softmax值。

y = tf.nn.softmax(tf.matmul(x,w) +b)

接下来定义一个变量*y*\_ ，*y*\_也是一个二维张量，其中的每一行为一个10维的向量，用于代表对应某一MNIST图片等类别。

y\_ = tf.placeholder(“float”, [NONE, 10])

虽然placeholder的shape参数是可选的，但是有了它，TensorFlow能够自动捕捉因数据维度不一致而导致的错误。

接着，我们再来定义一个交叉熵，一般通过交叉熵来描述一个模型的准确程度。

cross\_entropy = -tf.reduce\_sum(y\_ \* tf.log(y))

在这里，我们通过tf.reduce\_sum函数把minibatch里的每张图片的交叉熵值进行相加，计算出的交叉熵就是整个minibatch的。

下面开始进行模型的训练，用最基础的梯度下降算法来创建一个优化器train\_step。

train\_step = tf.train\_GradientDescentOptimizer(0.01).minimize(cross\_entropy)

在上面的代码中，我们将步长设置为0.01，这行代码实际上用来往计算图添加一个新的操作，其中包括了计算的梯度，以及计算每个参数的步长变化，并且计算出新的参数值，最后返回train\_step操作对象，在运行时会使用梯度下降来更新参数。

接下来调用tf.global\_variables\_initializer()方法来初始化创建的变量。

init = tf.global\_variables\_initializer()

在有些教材中会使用init = tf.initialize\_all\_variables()方法来初始化参数，目前这个方法已经被废弃，以后统一使用tf.global\_variables\_initializer()初始化变量。

目前为止，已经完成了训练TensorFlow模型前的所有准备工作，接下来我们需要做以下两件事：

（1）训练TensorFlow模型；

（2）利用测试集来验证训练结果。

在训练TensorFlow模型的过程中，一般会使用循环将训练的数据进行多次的训练，以此来保证训练的强度和精度，提高训练的准确度。这里以训练3000次为例，编写训练代码。

for i in range(3000):  
 batch\_xs, batch\_ys = mnist.train.next\_batch(100)  
 feed\_dict = {x: batch\_xs, y\_: batch\_ys}  
 sess.run(train\_step, feed\_dict)

首先定义了一个3000次的循环，表示训练的次数是3000次，mnist.train.next\_batch(100)表示每次迭代都会增加100个训练样本数据，然后执行train\_step函数，并通过feed\_dict将*x*和*y*\_张量的占位符用训练数据代替。

下一步需要评估训练的精度到底如何。使用tf.argmax(*y*,1)的返回值来表示模型对于任一输入*x*预测到的标签值，使用tf.argmax(*y*\_,1)代表正确的标签，使用tf.equal来检测预测是否与真实的标签匹配，如果索引位置一样则表示匹配。

correct\_prediction = tf.equal(tf.argmax(y, 1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,”float"))

最后，我们计算一下所学习到的模型中测试数据集的正确率：

print(sess.run(accuracy,feed\_dict={x: mnist.test.images, y\_:mnist.test.labels}))

运行以上程序，等待半分钟左右，控制台打印出其预测结果为0.9148，这个结果每次都是变化的。

除了数据外，在进行深度学习模型的构建中，其步长和训练次数也决定了预测的准确率，下面来更改一下程序代码，将步长从100调整为50：

batch\_xs, batch\_ys = mnist.train.next\_batch(50)

再一次运行程序，查看其预测的正确率，这里打印的结果为0.9202，那是不是步长越短越好呢？我们来继续做一个实验，将步长再从50调整到10，来看看结果：

batch\_xs, batch\_ys = mnist.train.next\_batch(10)

运行后，得到的结果为0.9088，事实证明，并不是步长越小越好。这说明，在深度学习的过程中，任何一个值都是有临界点的，超过了这个临界点，预测精度就会越来越低，因此，找到临界点（在深度学习过程中我们一般将这个过程称之为调参或参数调优），才能使我们的深度学习网络的精确度更高。

本例当中运用的概念和过程在后续章节当中还会有更详细的阐述，如果有不懂的地方不用急，照着做出结果就是胜利。

2.5 小结

本章着重介绍了如何通过pip方式、Java方式和源码方式在Windows系统、Linux系统上安装CPU和GPU版本的TensorFlow，以及对在安装过程中可能存在的问题加以说明和解释。接着又介绍了在TensorFlow使用过程中会经常用到的扩展模块的安装，这些扩展模块在今后学习的过程中会大量使用。