Tensorflow 指南

2015 年 12 月 24 日

目录

第一章　起步

1.1 Introduction

Let's get you up and running with TensorFlow!

But before we even get started, let's peek at what TensorFlow code looks like in the Python API, so you have a sense of where we're headed.

Here's a little Python program that makes up some data in two dimensions, and then fits a line to it.

```
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype("float32")
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data +
    b
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this
    first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```

The first part of this code builds the data flow graph. TensorFlow does not actually run any computation until the session is created and the run function is called.

To whet your appetite further, we suggest you check out what a classical machine learning problem looks like in TensorFlow. In the land of neural networks the most "classic" classical problem is the MNIST handwritten digit classification. We offer two introductions here, one for machine learning newbies, and one for pros. If you've already trained dozens of MNIST models in other software packages, please take the red pill. If you've never even

heard of MNIST, definitely take the blue pill. If you're somewhere in between, we suggest skimming blue, then red.

If you're already sure you want to learn and install TensorFlow you can skip these and charge ahead. Don't worry, you'll still get to see MNIST – we'll also use MNIST as an example in our technical tutorial where we elaborate on TensorFlow features.

Recommended Next Steps

Download and Setup Basic Usage TensorFlow Mechanics 101

本章的目的是让你了解和运行 TensorFlow!

在开始之前, 让我们先看一段使用 Python API 撰写的 TensorFlow 示例代码, 让你对将要学习的内容有初步的印象.

这段很短的 Python 程序生成了一些三维数据, 然后用一个平面拟合它.

```
import tensorflow as tf
import numpy as np

# 使用 NumPy 生成假数据(phony data), 总共 100 个点.
x_data = np.float32(np.random.rand(2, 100)) # 随机输入
y_data = np.dot([0.100, 0.200], x_data) + 0.300

# 构造一个线性模型
#
b = tf.Variable(tf.zeros([1]))
W = tf.Variable(tf.random_uniform([1, 2], -1.0, 1.0))
y = tf.matmul(W, x_data) + b

# 最小化方差
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# 初始化变量
init = tf.initialize_all_variables()

# 启动图 (graph)
sess = tf.Session()
sess.run(init)

# 拟合平面
for step in xrange(0, 201):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(W), sess.run(b)

# 得到最佳拟合结果 W: [[0.100  0.200]], b: [0.300]
```

为了进一步激发你的学习欲望, 我们想让你先看一下 TensorFlow 是如何解决一个经典的机器学习问题的. 在神经网络领域, 最为经典的问题莫过于 MNIST 手写数字分类问题. 我们准备了两篇不同的教程, 分别面向机器学习领域的初学者和专家. 如果你已经使用其它软件训练过许多 MNIST 模型, 请阅读高级教程 (红色药丸链接). 如果你以前从未听说过 MNIST, 请阅读初级教程 (蓝色药丸链接). 如果你的水平介于这两类人之间, 我们

建议你先快速浏览初级教程, 然后再阅读高级教程.

如果你已经下定决心, 准备学习和安装 TensorFlow, 你可以略过这些文字, 直接阅读后面的章节. 不用担心, 你仍然会看到 MNIST – 在阐述 TensorFlow 的特性时, 我们还会使用 MNIST 作为一个样例.

1.2 下载与安装

你可以使用我们提供的二进制包, 或者使用源代码, 安装 TensorFlow.

1.2.1 安装需求

TensorFlow Python API 目前支持 Python 2.7 和 python 3.3 以上版本。

支持 GPU 运算的版本 (仅限 Linux) 需要 Cuda Toolkit 7.0 和 CUDNN 6.5 V2. 具体请参考 Cuda 安装。

1.2.2 安装总述

TensorFlow 支持通过以下不同的方式安装:

- Pip 安装: Install TensorFlow on your machine, possibly upgrading previously installed Python packages. May impact existing Python programs on your machine.
- Virtualenv 安装: Install TensorFlow in its own directory, not impacting any existing Python programs on your machine.
- Docker 安装: Run TensorFlow in a Docker container isolated from all other programs on your machine.

If you are familiar with Pip, Virtualenv, or Docker, please feel free to adapt the instructions to your particular needs. The names of the pip and Docker images are listed in the corresponding installation sections.

If you encounter installation errors, see common problems for some solutions.

1.2.3 Pip Installatioin

Pip is a package management system used to install and manage software packages written in Python.

The packages that will be installed or upgraded during the pip install are listed in the REQUIRED_PACKAGES section of setup.py

Install pip (or pip3 for python3) if it is not already installed:

```
# Ubuntu/Linux 64-bit
$ sudo apt-get install python-pip python-dev
```

```
# Mac OS X
$ sudo easy_install pip
```

Install TensorFlow:

```
# Ubuntu/Linux 64-bit, CPU only:
$ sudo pip install --upgrade https://storage.googleapis.com/
    tensorflow/linux/cpu/tensorflow-0.6.0-cp27-none-linux_x86_64.whl
```

```
# Ubuntu/Linux 64-bit, GPU enabled:
$ sudo pip install --upgrade https://storage.googleapis.com/
    tensorflow/linux/gpu/tensorflow-0.6.0-cp27-none-linux_x86_64.whl
```

```
# Mac OS X, CPU only:
$ sudo easy_install --upgrade six
$ sudo pip install --upgrade https://storage.googleapis.com/
    tensorflow/mac/tensorflow-0.6.0-py2-none-any.whl
```

For Python 3:

```
# Ubuntu/Linux 64-bit, CPU only:
$ sudo pip3 install --upgrade https://storage.googleapis.com/
    tensorflow/linux/cpu/tensorflow-0.6.0-cp34-none-linux_x86_64.whl
```

```
# Ubuntu/Linux 64-bit, GPU enabled:
$ sudo pip3 install --upgrade https://storage.googleapis.com/
    tensorflow/linux/gpu/tensorflow-0.6.0-cp34-none-linux_x86_64.whl
```

```
# Mac OS X, CPU only:
$ sudo easy_install --upgrade six
$ sudo pip3 install --upgrade https://storage.googleapis.com/
    tensorflow/mac/tensorflow-0.6.0-py3-none-any.whl
```

You can now test your installation.

1.2.4 Virtualenv Installation

Virtualenv is a tool to keep the dependencies required by different Python projects in separate places. The Virtualenv installation of TensorFlow will not override pre-existing version of the Python packages needed by TensorFlow.

With Virtualenv the installation is as follows:

- Install pip and Virtualenv.
- Create a Virtualenv environment.
- Activate the Virtualenv environment and install TensorFlow in it.
- After the install you will activate the Virtualenv environment each time you want to use TensorFlow.

Install pip and Virtualenv:

```
# Ubuntu/Linux 64-bit
$ sudo apt-get install python-pip python-dev python-virtualenv
```

```
# Mac OS X
$ sudo easy_install pip
$ sudo pip install --upgrade virtualenv
```

Create a Virtualenv environment in the directory ~/tensorflow:

```
$ virtualenv --system-site-packages ~/tensorflow
```

Activate the environment and use pip to install TensorFlow inside it:

```
$ source ~/tensorflow/bin/activate  # If using bash
$ source ~/tensorflow/bin/activate.csh  # If using csh
(tensorflow)$  # Your prompt should change

# Ubuntu/Linux 64-bit, CPU only:
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/
    tensorflow/linux/cpu/tensorflow-0.5.0-cp27-none-linux_x86_64.whl

# Ubuntu/Linux 64-bit, GPU enabled:
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/
    tensorflow/linux/gpu/tensorflow-0.5.0-cp27-none-linux_x86_64.whl

# Mac OS X, CPU only:
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/
    tensorflow/mac/tensorflow-0.5.0-py2-none-any.whl
```

and again for python3:

```
$ source ~/tensorflow/bin/activate  # If using bash
$ source ~/tensorflow/bin/activate.csh  # If using csh
(tensorflow)$  # Your prompt should change

# Ubuntu/Linux 64-bit, CPU only:
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/
    tensorflow/linux/cpu/tensorflow-0.6.0-cp34-none-linux_x86_64.whl

# Ubuntu/Linux 64-bit, GPU enabled:
(tensorflow)$ pip install --upgrade https://storage.googleapis.com/
    tensorflow/linux/gpu/tensorflow-0.6.0-cp34-none-linux_x86_64.whl

# Mac OS X, CPU only:
(tensorflow)$ pip3 install --upgrade https://storage.googleapis.com/
    tensorflow/mac/tensorflow-0.6.0-py3-none-any.whl
```

With the Virtualenv environment activated, you can now test your installation.

When you are done using TensorFlow, deactivate the environment.

```
(tensorflow)$ deactivate
$  # Your prompt should change back
```

To use TensorFlow later you will have to activate the Virtualenv environment again:

```
$ source ~/tensorflow/bin/activate  # If using bash.
$ source ~/tensorflow/bin/activate.csh  # If using csh.
(tensorflow)$  # Your prompt should change.
# Run Python programs that use TensorFlow.
...
# When you are done using TensorFlow, deactivate the environment.
(tensorflow)$ deactivate
```

1.2.5 Docker Installation

Docker is a system to build self contained versions of a Linux operating system running on your machine. When you install and run TensorFlow via Docker it completely isolates the installation from pre-existing packages on your machine.

We provide 4 Docker images:

- `b.gcr.io/tensorflow/tensorflow`: TensorFlow CPU binary image.
- `b.gcr.io/tensorflow/tensorflow:latest-devel`:CPU Binary image plus source code.
- `b.gcr.io/tensorflow/tensorflow:latest-gpu`:TensorFlow GPU binary image.
- `b.gcr.io/tensorflow/tensorflow:latest-devel-gpu`:GPU Binary image plus source code.

We also have tags with latest replaced by a released version (eg `0.6.0-gpu`).

With Docker the installation is as follows:

- Install Docker on your machine.
- Create a Docker group to allow launching containers without sudo.
- Launch a Docker container with the TensorFlow image. The image gets downloaded automatically on first launch.

See installing Docker for instructions on installing Docker on your machine.

After Docker is installed, launch a Docker container with the TensorFlow binary image as follows.

```
$ docker run -it b.gcr.io/tensorflow/tensorflow
```

If you're using a container with GPU support, some additional flags must be passed to expose the GPU device to the container. For the default config, we include a script in the repo with these flags, so the command-line would look like:

```
$ path/to/repo/tensorflow/tools/docker/docker_run_gpu.sh b.gcr.io/
    tensorflow/tensorflow:gpu
```

You can now test your installation within the Docker container.

1.2.6 Test the TensorFlow Installation

(Optional, Linux) Enable GPU Support

If you installed the GPU version of TensorFlow, you must also install the Cuda Toolkit 7.0 and CUDNN 6.5 V2. Please see Cuda installation.

You also need to set the `LD_LIBRARY_PATH` and `CUDA_HOME` environment variables. Consider adding the commands below to your `~/.bash_profile`. These assume your CUDA installation is in `/usr/local/cuda`:

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/lib64"
export CUDA_HOME=/usr/local/cuda
```

Run TensorFlow from the Command Line

See common problems if an error happens.

Open a terminal and type the following:

```
$ python
...
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
Hello, TensorFlow!
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> print(sess.run(a + b))
42
>>>
```

Run a TensorFlow demo model

All TensorFlow packages, including the demo models, are installed in the Python library. The exact location of the Python library depends on your system, but is usually one of:

```
/usr/local/lib/python2.7/dist-packages/tensorflow
/usr/local/lib/python2.7/site-packages/tensorflow
```

You can find out the directory with the following command:

```
$ python -c 'import site; print("\n".join(site.getsitepackages()))'
```

The simple demo model for classifying handwritten digits from the MNIST dataset is in the sub-directory `models/image/mnist/convolutional.py`. You can run it from the command line as follows:

```
# Using 'python -m' to find the program in the python search path:
$ python -m tensorflow.models.image.mnist.convolutional
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
...etc...

# You can alternatively pass the path to the model program file to
    the python interpreter.
```

```
$ python /usr/local/lib/python2.7/dist-packages/tensorflow/models/
    image/mnist/convolutional.py
...
```

1.2.7 Installing from source

When installing from source you will build a pip wheel that you then install using pip. You'll need pip for that, so install it as described above.

Clone the TensorFlow repository

```
$ git clone --recurse-submodules https://github.com/tensorflow/
    tensorflow
```

`--recurse-submodules` is required to fetch the protobuf library that TensorFlow depends on.

Installation for Linux

Install Bazel Follow instructions here to install the dependencies for Bazel. Then download bazel version 0.1.1 using the installer for your system and run the installer as mentioned there:

```
$ chmod +x PATH_TO_INSTALL.SH
$ ./PATH_TO_INSTALL.SH --user
```

Remember to replace `PATH_TO_INSTALL.SH` with the location where you downloaded the installer.

Finally, follow the instructions in that script to place bazel into your binary path.

Install other dependencies

```
$ sudo apt-get install python-numpy swig python-dev
```

Configure the installation Run the configure script at the root of the tree. The configure script asks you for the path to your python interpreter and allows (optional) configuration of the CUDA libraries (see below).

This step is used to locate the python and numpy header files.

```
$ ./configure
Please specify the location of python. [Default is /usr/bin/python]:
```

Optional: Install CUDA (GPUs on Linux) In order to build or run TensorFlow with GPU support, both Cuda Toolkit 7.0 and CUDNN 6.5 V2 from NVIDIA need to be installed.

TensorFlow GPU support requires having a GPU card with NVidia Compute Capability >= 3.5. Supported cards include but are not limited to:

- NVidia Titan
- NVidia Titan X
- NVidia K20
- NVidia K40

Download and install Cuda Toolkit 7.0

https://developer.nvidia.com/cuda-toolkit-70

Install the toolkit into e.g. /usr/local/cuda

Download and install CUDNN Toolkit 6.5

https://developer.nvidia.com/rdp/cudnn-archive

Uncompress and copy the cudnn files into the toolkit directory. Assuming the toolkit is installed in /usr/local/cuda:

```
tar xvzf cudnn-6.5-linux-x64-v2.tgz
sudo cp cudnn-6.5-linux-x64-v2/cudnn.h /usr/local/cuda/include
sudo cp cudnn-6.5-linux-x64-v2/libcudnn* /usr/local/cuda/lib64
```

Configure TensorFlow's canonical view of Cuda libraries

When running the configure script from the root of your source tree, select the option Y when asked to build TensorFlow with GPU support.

```
$ ./configure
Please specify the location of python. [Default is /usr/bin/python]:
Do you wish to build TensorFlow with GPU support? [y/N] y
GPU support will be enabled for TensorFlow

Please specify the location where CUDA 7.0 toolkit is installed.
    Refer to
README.md for more details. [default is: /usr/local/cuda]: /usr/
    local/cuda

Please specify the location where CUDNN 6.5 V2 library is installed.
     Refer to
README.md for more details. [default is: /usr/local/cuda]: /usr/
    local/cuda

Setting up Cuda include
Setting up Cuda lib64
Setting up Cuda bin
Setting up Cuda nvvm
Configuration finished
```

This creates a canonical set of symbolic links to the Cuda libraries on your system. Every time you change the Cuda library paths you need to run this step again before you invoke the bazel build command.

Build your target with GPU support

From the root of your source tree, run:

```
$ bazel build -c opt --config=cuda //tensorflow/cc:
    tutorials_example_trainer

$ bazel-bin/tensorflow/cc/tutorials_example_trainer --use_gpu
# Lots of output. This tutorial iteratively calculates the major
    eigenvalue of
# a 2x2 matrix, on GPU. The last few lines look like this.
000009/000005 lambda = 2.000000 x = [0.894427 -0.447214] y =
    [1.788854 -0.894427]
000006/000001 lambda = 2.000000 x = [0.894427 -0.447214] y =
    [1.788854 -0.894427]
000009/000009 lambda = 2.000000 x = [0.894427 -0.447214] y =
    [1.788854 -0.894427]
```

Note that "–config=cuda" is needed to enable the GPU support.

Enabling Cuda 3.0

TensorFlow officially supports Cuda devices with 3.5 and 5.2 compute capabilities. In order to enable earlier Cuda devices such as Grid K520, you need to target Cuda 3.0. This can be done through TensorFlow unofficial settings with "configure".

```
$ TF_UNOFFICIAL_SETTING=1 ./configure

# Same as the official settings above

WARNING: You are configuring unofficial settings in TensorFlow.
    Because some
external libraries are not backward compatible, these settings are
    largely
untested and unsupported.

Please specify a list of comma-separated Cuda compute capabilities
    you want to
build with. You can find the compute capability of your device at:
https://developer.nvidia.com/cuda-gpus.
Please note that each additional compute capability significantly
    increases
your build time and binary size. [Default is: "3.5,5.2"]: 3.0

Setting up Cuda include
Setting up Cuda lib64
Setting up Cuda bin
Setting up Cuda nvvm
Configuration finished
```

Known issues

Although it is possible to build both Cuda and non-Cuda configs under the same source

tree, we recommend to run "bazel clean" when switching between these two configs in the same source tree.

You have to run configure before running bazel build. Otherwise, the build will fail with a clear error message. In the future, we might consider making this more conveninent by including the configure step in our build process, given necessary bazel new feature support.

Installation for Mac OS X

We recommend using homebrew to install the bazel and SWIG dependencies, and installing python dependencies using $easy_install or pip$.

Dependencies Follow instructions here to install the dependencies for Bazel. You can then use homebrew to install bazel and SWIG:

```
$ brew install bazel swig
```

You can install the python dependencies using $easy_install or pip. Using easy_install, run$

```
$ sudo easy_install -U six
$ sudo easy_install -U numpy
$ sudo easy_install wheel
```

We also recommend the ipython enhanced python shell, so best install that too:

```
$ sudo easy_install ipython
```

Configure the installation Run the `configure` script at the root of the tree. The configure script asks you for the path to your python interpreter.

This step is used to locate the python and numpy header files.

```
$ ./configure
Please specify the location of python. [Default is /usr/bin/python]:
Do you wish to build TensorFlow with GPU support? [y/N]
```

Create the pip package and install

```
$ bazel build -c opt //tensorflow/tools/pip_package:
    build_pip_package

# To build with GPU support:
$ bazel build -c opt --config=cuda //tensorflow/tools/pip_package:
    build_pip_package

$ bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/
    tensorflow_pkg

# The name of the .whl file will depend on your platform.
$ pip install /tmp/tensorflow_pkg/tensorflow-0.5.0-cp27-none-
    linux_x86_64.whl
```

1.2.8 Train your first TensorFlow neural net model

Starting from the root of your source tree, run:

```
$ cd tensorflow/models/image/mnist
$ python convolutional.py
Succesfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Succesfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Succesfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Succesfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Initialized!
Epoch 0.00
Minibatch loss: 12.054, learning rate: 0.010000
Minibatch error: 90.6%
Validation error: 84.6%
Epoch 0.12
Minibatch loss: 3.285, learning rate: 0.010000
Minibatch error: 6.2%
Validation error: 7.0%
...
...
```

1.2.9 Common Problems

GPU-related issues

If you encounter the following when trying to run a TensorFlow program:

```
ImportError: libcudart.so.7.0: cannot open shared object file: No
    such file or directory
```

Make sure you followed the the GPU installation instructions.

Pip installation issues

Can't find setup.py If, during pip install, you encounter an error like:

```
...
IOError: [Errno 2] No such file or directory: '/tmp/pip-o6Tpui-build
    /setup.py'
```

Solution: upgrade your version of pip:

```
pip install --upgrade pip
```

This may require sudo, depending on how pip is installed.

SSLError: SSL_VERIFY_FAILED If, during pip install from a URL, you encounter an error like:

```
...
SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed
```

Solution: Download the wheel manually via curl or wget, and pip install locally.

Linux issues

If you encounter:

```
...
 "__add__", "__radd__",
              ^
SyntaxError: invalid syntax
```

Solution: make sure you are using Python 2.7.

Mac OS X: ImportError: No module named copyreg

On Mac OS X, you may encounter the following when importing tensorflow.

```
>>> import tensorflow as tf
...
ImportError: No module named copyreg
```

Solution: TensorFlow depends on protobuf, which requires the Python package `six -1.10.0`. Apple's default Python installation only provides `six-1.4.1`.

You can resolve the issue in one of the following ways:

- pgrade the Python installation with the current version of `six`:

```
$ sudo easy_install -U six
```

- Install TensorFlow with a separate Python library:
  - Virtualenv
  - Docker

  Install a separate copy of Python via Homebrew or MacPorts and re-install TensorFlow in that copy of Python.

Mac OS X: TypeError: `__init__()` got an unexpected keyword argument 'syntax'

On Mac OS X, you may encounter the following when importing tensorflow.

```
>>> import tensorflow as tf
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/tensorflow/__init__.
   py", line 4, in <module>
    from tensorflow.python import *
```

```
  File "/usr/local/lib/python2.7/site-packages/tensorflow/python/
    __init__.py", line 13, in <module>
     from tensorflow.core.framework.graph_pb2 import *
...
  File "/usr/local/lib/python2.7/site-packages/tensorflow/core/
    framework/tensor_shape_pb2.py", line 22, in <module>
     serialized_pb=_b('\n,tensorflow/core/framework/tensor_shape.
    proto\x12\ntensorflow\"d\n\x10TensorShapeProto\x12-\n\x03\x64im\
    x18\x02 \x03(\x0b\x32 .tensorflow.TensorShapeProto.Dim\x1a!\n\x03
    \x44im\x12\x0c\n\x04size\x18\x01 \x01(\x03\x12\x0c\n\x04name\x18\
    x02 \x01(\tb\x06proto3')
TypeError: __init__() got an unexpected keyword argument 'syntax'
```

This is due to a conflict between protobuf versions (we require protobuf 3.0.0). The best current solution is to make sure older versions of protobuf are not installed, such as:

```
$ pip install --upgrade protobuf
```

1.3 Basic Usage

To use TensorFlow you need to understand how TensorFlow:

- Represents computations as graphs.
- Executes graphs in the context of Sessions.
- Represents data as tensors.
- Maintains state with Variables.
- Uses feeds and fetches to get data into and out of arbitrary operations.

1.3.1 Overview

TensorFlow is a programming system in which you represent computations as graphs. Nodes in the graph are called ops (short for operations). An op takes zero or more `Tensors`, performs some computation, and produces zero or more `Tensors`. A Tensor is a typed multi-dimensional array. For example, you can represent a mini-batch of images as a 4-D array of floating point numbers with dimensions `[batch, height, width, channels]`.

A TensorFlow graph is a description of computations. To compute anything, a graph must be launched in a `Session`. A Session places the graph ops onto `Devices`, such as CPUs or GPUs, and provides methods to execute them. These methods return tensors produced by ops as numpy ndarray objects in Python, and as `tensorflow::Tensor` instances in C and C++.

1.3.2 The computation graph

TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph.

For example, it is common to create a graph to represent and train a neural network in the construction phase, and then repeatedly execute a set of training ops in the graph in the execution phase.

TensorFlow can be used from C, C++, and Python programs. It is presently much easier to use the Python library to assemble graphs, as it provides a large set of helper functions not available in the C and C++ libraries.

The session libraries have equivalent functionalities for the three languages.

Building the graph

To build a graph start with ops that do not need any input (source ops), such as Constant, and pass their output to other ops that do computation.

The ops constructors in the Python library return objects that stand for the output of the constructed ops. You can pass these to other ops constructors to use as inputs.

The TensorFlow Python library has a default graph to which ops constructors add nodes. The default graph is sufficient for many applications. See the Graph class documentation for how to explicitly manage multiple graphs.

```
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)
```

The default graph now has three nodes: two constant() ops and one matmul() op. To actually multiply the matrices, and get the result of the multiplication, you must launch the graph in a session.

Launching the graph in a session

Launching follows construction. To launch a graph, create a Session object. Without arguments the session constructor launches the default graph.

See the Session class for the complete session API.

```
# 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()
```

Sessions should be closed to release resources. You can also enter a Session with a "with" block. The Session closes automatically at the end of the with block.

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

The TensorFlow implementation translates the graph definition into executable operations distributed across available compute resources, such as the CPU or one of your computer's GPU cards. In general you do not have to specify CPUs or GPUs explicitly. TensorFlow uses your first GPU, if you have one, for as many operations as possible.

If you have more than one GPU available on your machine, to use a GPU beyond the first you must assign ops to it explicitly. Use with...Device statements to specify which CPU or GPU to use for operations:

```
with tf.Session() as sess:
  with tf.device("/gpu:1"):
    matrix1 = tf.constant([[3., 3.]])
    matrix2 = tf.constant([[2.],[2.]])
    product = tf.matmul(matrix1, matrix2)
    ...
```

Devices are specified with strings. The currently supported devices are:

"/cpu:0": The CPU of your machine. "/gpu:0": The GPU of your machine, if you have one. "/gpu:1": The second GPU of your machine, etc. See Using GPUs for more information about GPUs and TensorFlow.

1.3.3 Interactive Usage

The Python examples in the documentation launch the graph with a `Session` and use the `Session.run()` method to execute operations.

For ease of use in interactive Python environments, such as `IPython` you can instead use the `InteractiveSession` class, and the `Tensor.eval()` and `Operation.run()` methods. This avoids having to keep a variable holding the session.

```
# Enter an interactive TensorFlow Session.
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.Variable([1.0, 2.0])
a = tf.constant([3.0, 3.0])

```

```
# Initialize 'x' using the run() method of its initializer op.
x.initializer.run()

# Add an op to subtract 'a' from 'x'.  Run it and print the result
sub = tf.sub(x, a)
print(sub.eval())
# ==> [-2. -1.]

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

1.3.4 Tensors

TensorFlow programs use a tensor data structure to represent all data – only tensors are passed between operations in the computation graph. You can think of a TensorFlow tensor as an n-dimensional array or list. A tensor has a static type, a rank, and a shape. To learn more about how TensorFlow handles these concepts, see the Rank, Shape, and Type reference.

1.3.5 Variables

Variables maintain state across executions of the graph. The following example shows a variable serving as a simple counter. See Variables for more details.

```
# 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
```

The `assign()` operation in this code is a part of the expression graph just like the `add()` operation, so it does not actually perform the assignment until `run()` executes the expression.

You typically represent the parameters of a statistical model as a set of Variables. For example, you would store the weights for a neural network as a tensor in a Variable. During training you update this tensor by running a training graph repeatedly.

1.3.6 Fetches

To fetch the outputs of operations, execute the graph with a `run()` call on the `Session` object and pass in the tensors to retrieve. In the previous example we fetched the single node `state`, but you can also fetch multiple tensors:

```
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)]
```

All the ops needed to produce the values of the requested tensors are run once (not once per requested tensor).

1.3.7 Feeds

The examples above introduce tensors into the computation graph by storing them in `Constants` and `Variables`. TensorFlow also provides a feed mechanism for patching a tensor directly into any operation in the graph.

A feed temporarily replaces the output of an operation with a tensor value. You supply feed data as an argument to a `run()` call. The feed is only used for the run call to which it is passed. The most common use case involves designating specific operations to be "feed" operations by using `tf.placeholder()` to create them:

```
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)]
```

A `placeholder()` operation generates an error if you do not supply a feed for it. See the MNIST fully-connected feed tutorial (source code) for a larger-scale example of feeds.

第二章　基础教程

2.1 综述

MNIST For ML Beginners

If you're new to machine learning, we recommend starting here. You'll learn about a classic problem, handwritten digit classification (MNIST), and get a gentle introduction to multiclass classification.

View Tutorial

Deep MNIST for Experts

If you're already familiar with other deep learning software packages, and are already familiar with MNIST, this tutorial with give you a very brief primer on TensorFlow.

View Tutorial

TensorFlow Mechanics 101

This is a technical tutorial, where we walk you through the details of using TensorFlow infrastructure to train models at scale. We use again MNIST as the example.

View Tutorial

Convolutional Neural Networks

An introduction to convolutional neural networks using the CIFAR-10 data set. Convolutional neural nets are particularly tailored to images, since they exploit translation invariance to yield more compact and effective representations of visual content.

View Tutorial

Vector Representations of Words

This tutorial motivates why it is useful to learn to represent words as vectors (called word embeddings). It introduces the word2vec model as an efficient method for learning embeddings. It also covers the high-level details behind noise-contrastive training methods (the biggest recent advance in training embeddings).

View Tutorial

Recurrent Neural Networks

An introduction to RNNs, wherein we train an LSTM network to predict the next word in an English sentence. (A task sometimes called language modeling.)

View Tutorial

Sequence-to-Sequence Models

A follow on to the RNN tutorial, where we assemble a sequence-to-sequence model for machine translation. You will learn to build your own English-to-French translator, entirely machine learned, end-to-end.

View Tutorial

Mandelbrot Set

TensorFlow can be used for computation that has nothing to do with machine learning. Here's a naive implementation of Mandelbrot set visualization.

View Tutorial

Partial Differential Equations

As another example of non-machine learning computation, we offer an example of a naive PDE simulation of raindrops landing on a pond.

View Tutorial

MNIST Data Download

Details about downloading the MNIST handwritten digits data set. Exciting stuff.

View Tutorial

Image Recognition

How to run object recognition using a convolutional neural network trained on ImageNet Challenge data and label set.

View Tutorial

We will soon be releasing code for training a state-of-the-art Inception model.

Deep Dream Visual Hallucinations

Building on the Inception recognition model, we will release a TensorFlow version of the Deep Dream neural network visual hallucination software.

COMING SOON

2.2 MNIST 之机器学习入门

这个教程的目标读者是对机器学习和 TensorFlow 都不太了解的新手。如果你已经了解 MNIST 和 softmax 回归 (softmax regression) 的相关知识，你可以阅读这个快速上手教程。

当我们开始学习编程的时候，第一件事往往是学习打印“Hello World”。就好比编程入门有 Hello World，机器学习入门有 MNIST。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：

它也包含每一张图片对应的标签，告诉我们这个是数字几。比如，上面这四张图片的标签分别是 5,0,4,1。

在此教程中，我们将训练一个机器学习模型用于预测图片里面的数字。我们的目的不是要设计一个世界一流的复杂模型 – 尽管我们会在之后给你源代码去实现一流的预测模型 – 而是要介绍下如何使用 TensorFlow。所以，我们这里会从一个很简单的数学模型开始，它叫做 Softmax Regression。

对应这个教程的实现代码很短，而且真正有意思的内容只包含在三行代码里面。但是，去理解包含在这些代码里面的设计思想是非常重要的：TensorFlow 工作流程和机器学习的基本概念。因此，这个教程会很详细地介绍这些代码的实现原理。

2.2.1 MNIST 数据集

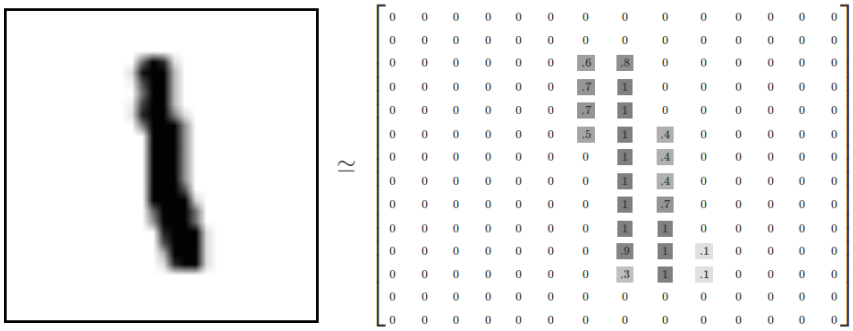
MNIST 数据集的官网是Yann LeCun's website。在这里，我们提供了一份 python 源代码用于自动下载和安装这个数据集。你可以下载这段代码，然后用下面的代码导入到你的项目里面，也可以直接复制粘贴到你的代码文件里面。

```
import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

下载下来的数据集被分成两部分：60000 行的训练数据集（‘mnist.train‘）和 10000 行的测试数据集（‘mnist.test‘）。这样的切分很重要，在机器学习模型设计时必须有一个单独的测试数据集不用于训练而是用来评估这个模型的性能，从而更加容易把设计的模型推广到其他数据集上（泛化）。

正如前面提到的一样，每一个 MNIST 数据单元有两部分组成：一张包含手写数字的图片和一个对应的标签。我们把这些图片设为“xs”，把这些标签设为“ys”。训练数据集和测试数据集都包含 xs 和 ys，比如训练数据集的图片是`mnist.train.images`，训练数据集的标签是`mnist.train.labels`。

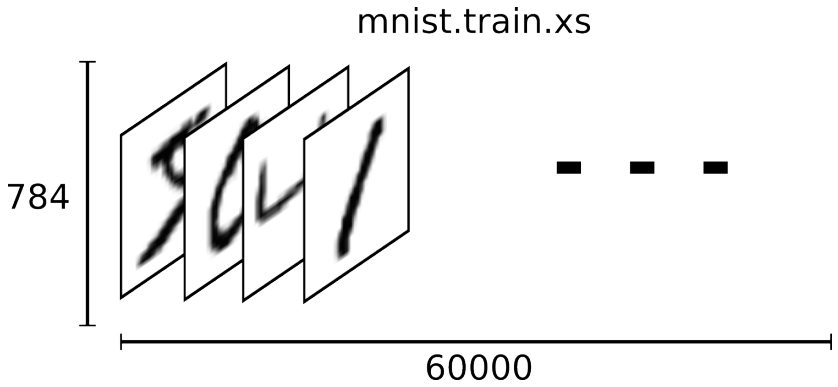
每一张图片包含 28×28 像素。我们可以用一个数字数组来表示这张图片：



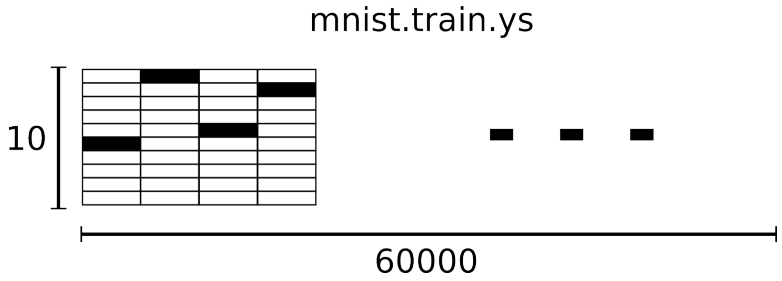
我们把这个数组展开成一个向量，长度是 $28 \times 28 = 784$。如何展开这个数组（数字间的顺序）不重要，只要保持各个图片采用相同的方式展开。从这个角度来看，MNIST 数据集的图片就是在 784 维向量空间里面的点, 并且拥有比较复杂的结构 (提醒: 此类数据的可视化是计算密集型的)。

展平图片的数字数组会丢失图片的二维结构信息。这显然是不理想的，最优秀的计算机视觉方法会挖掘并利用这些结构信息，我们会在后续教程中介绍。但是在这个教程中我们忽略这些结构，所介绍的简单数学模型，softmax 回归 (softmax regression)，不会利用这些结构信息。

因此，在 MNIST 训练数据集中，`mnist.train.images`是一个形状为 [60000, 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。



相对应的 MNIST 数据集的标签是介于 0 到 9 的数字，用来描述给定图片里表示的数字。为了用于这个教程，我们使标签数据是"one-hot vectors"。一个 one-hot 向量除了某一位的数字是 1 以外其余各维度数字都是 0。所以在此教程中，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])。因此，`mnist.train.labels`是一个 [60000, 10] 的数字矩阵。



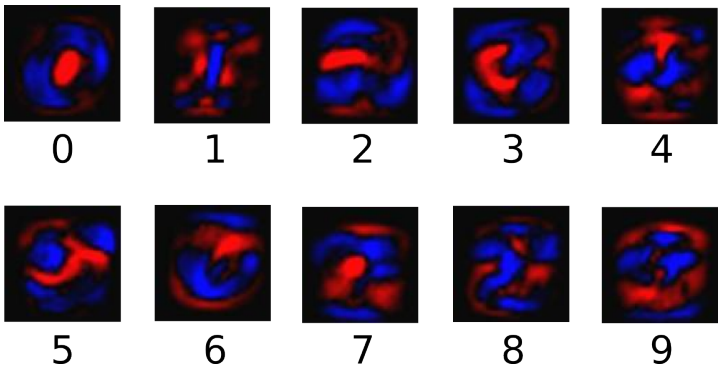
现在，我们准备开始真正的建模之旅啦！

2.2.2 Softmax 回归介绍

我们知道 MNIST 的每一张图片都表示一个数字，从 0 到 9。我们希望得到给定图片代表每个数字的概率。比如说，我们的模型可能推测一张包含 9 的图片代表数字 9 的概率是 80% 但是判断它是 8 的概率是 5%（因为 8 和 9 都有上半部分的小圆），然后给予它代表其他数字的概率更小的值。

这是一个使用 softmax 回归（softmax regression）模型的经典案例。softmax 模型可以用来给不同的对象分配概率。即使在之后，我们训练更加精细的模型时，最后一步也需要用 softmax 来分配概率。

softmax 回归（softmax regression）分两步：首先，为了得到一张给定图片属于某个特定数字类的证据（evidence），我们对图片像素值进行加权求和。如果这个像素具有很强的证据说明这张图片不属于该类，那么相应的权值为负数，相反如果这个像素拥有有利的证据支持这张图片属于这个类，那么权值是正数。下面的图片显示了一个模型学习到的图片上每个像素对于特定数字类的权值。红色代表负数权值，蓝色代表正数权值。



我们也需要加入一个额外的偏置量（bias），因为输入往往会带有一些无关的干扰量。因此对于给定的输入图片 x 它代表的是数字 x 的证据可以表示为

$$evidence_i = \sum_j W_{i,j} x_j + b_i \tag{2.1}$$

其中，W_i 代表权重，b_i 代表第 i 类的偏置量，j 代表给定图片 x 的像素索引用于像素求

和。然后用 softmax 函数可以把这些证据转换成概率 y:

$$y = softmax(evidence) \tag{2.2}$$

这里的 softmax 可以看成是一个激励（activation）函数或是链接（link）函数，把我们定义的线性函数的输出转换成我们想要的格式，也就是关于 10 个数字类的概率分布。因此，给定一张图片，它对于每一个数字的吻合度可以被 softmax 函数转换成为一个概率值。softmax 函数可以定义为：

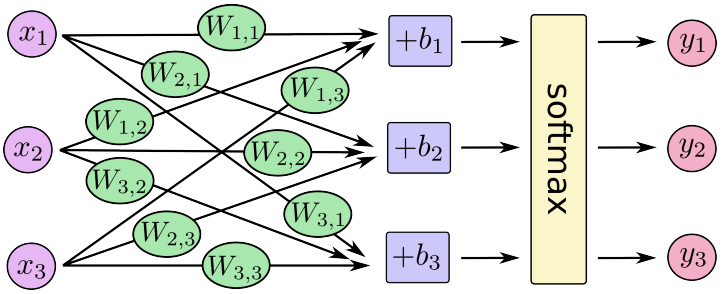
$$softmax(x) = normalize(exp(x)) \tag{2.3}$$

展开等式右边的子式，可以得到：

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.4}$$

但是更多的时候把 softmax 模型函数定义为前一种形式：把输入值当成幂指数求值，再正则化这些结果值。这个幂运算表示，更大的证据对应更大的假设模型（hypothesis）里面的乘数权重值。反之，拥有更少的证据意味着在假设模型里面拥有更小的乘数系数。假设模型里的权值不可以是 0 值或者负值。Softmax 然后会正则化这些权重值，使它们的总和等于 1，以此构造一个有效的概率分布。（更多的关于 Softmax 函数的信息，可以参考 Michael Nieslen 的书里面的这个部分，其中有关于 softmax 的可交互式的可视化解释。）

对于 softmax 回归模型可以用下面的图解释，对于输入的 xs 加权求和，再分别加上一个偏置量，最后再输入到 softmax 函数中：



如果把它写成一个方程，可以得到：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_1 + W_{1,3}x_1 + b_1 \\ W_{2,1}x_2 + W_{2,2}x_2 + W_{2,3}x_2 + b_2 \\ W_{3,1}x_3 + W_{3,2}x_3 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

我们也可以用向量表示这个计算过程：用矩阵乘法和向量相加。这有助于提高计算效率（也是一种更有效的思考方式）。

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

更进一步，可以写成更加紧凑的方式：

$$y = softmax(Wx + b) \tag{2.5}$$

2.2.3 实现回归模型

为了用 python 实现高效的数值计算，我们通常会使用函数库，比如 NumPy，会把类似矩阵乘法这样的复杂运算使用其他外部语言实现。不幸的是，从外部计算切换回 Python 的每一个操作，仍然是一个很大的开销。如果你用 GPU 来进行外部计算，这样的开销会更大。用分布式的计算方式，也会花费更多的资源用来传输数据。

TensorFlow 也把复杂的计算放在 python 之外完成，但是为了避免前面说的那些开销，它做了进一步完善。TensorFlow 不单独地运行单一的复杂计算，而是让我们可以先用图描述一系列可交互的计算操作，然后全部一起在 Python 之外运行。（这样类似的运行方式，可以在不少的机器学习库中看到。）

使用 TensorFlow 之前，首先导入它：

```
import tensorflow as tf
```

我们通过操作符号变量来描述这些可交互的操作单元，可以用下面的方式创建一个：

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

x 不是一个特定的值，而是一个占位符`placeholder`，我们在 TensorFlow 运行计算时输入这个值。我们希望能够输入任意数量的 MNIST 图像，每一张图展平成 784 维的向量。我们用 2 维的浮点数张量来表示这些图，这个张量的形状是 [None，784]。（这里的`None`表示此张量的第一个维度可以是任何长度的。）

我们的模型也需要权重值和偏置量，当然我们可以把它们当做是另外的输入（使用占位符），但 TensorFlow 有一个更好的方法来表示它们：`Variable`。一个`Variable`代表一个可修改的张量，存在在 TensorFlow 的用于描述交互性操作的图中。它们可以用于计算输入值，也可以在计算中被修改。对于各种机器学习应用，一般都会有模型参数，可以用`Variable`表示。

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

我们赋予`tf.Variable`不同的初值来创建不同的`Variable`：在这里，我们都用全为零的张量来初始化`W`和`b`。因为我们要学习`W`和`b`的值，它们的初值可以随意设置。

注意，`W`的维度是`[784，10]`，因为我们想要用 784 维的图片向量乘以它以得到一个 10 维的证据值向量，每一位对应不同数字类。`b`的形状是`[10]`，所以我们可以直接把它加到输出上面。

现在，可以实现我们的模型了，只需以下一行代码：

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

首先，我们用tf.matmul(X，W)表示 x 乘以 W，对应之前等式里面的 W_x，这里 x 是一个 2 维张量拥有多个输入。然后再加上 b，把和输入到tf.nn.softmax函数里面。

至此，我们先用了几行简短的代码来设置变量，然后只用了一行代码来定义我们的模型。TensorFlow 不仅仅可以使 softmax 回归模型计算变得特别简单，它也用这种非常灵活的方式来描述其他各种数值计算，从机器学习模型对物理学模拟仿真模型。一旦被定义好之后，我们的模型就可以在不同的设备上运行：计算机的 CPU，GPU，甚至是手机！

2.2.4 训练模型

为了训练我们的模型，我们首先需要定义一个指标来评估这个模型是好的。其实，在机器学习，我们通常定义指标来表示一个模型是坏的，这个指标称为成本（cost）或损失（loss），然后尽量最小化这个指标。但是，这两种方式是相同的。

一个非常常见的，非常漂亮的成本函数是“交叉熵”（cross-entropy）。交叉熵产生于信息论里面的信息压缩编码技术，但是它后来演变成为从博弈论到机器学习等其他领域里的重要技术手段。它的定义如下：

$$H_{y'}(u) = -\sum_i y_{i'} log(y_i) \tag{2.6}$$

y 是我们预测的概率分布,y' 是实际的分布（我们输入的 one-hot vector)。比较粗糙的理解是，交叉熵是用来衡量我们的预测用于描述真相的低效性。更详细的关于交叉熵的解释超出本教程的范畴，但是你很有必要好好理解它。

为了计算交叉熵，我们首先需要添加一个新的占位符用于输入正确值：

```
y = tf.placeholder("float", [None,10])
```

然后我们可以用

$$-\sum y' log(y) \tag{2.7}$$

计算交叉熵:

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```

首先，用 tf.log 计算 y 的每个元素的对数。接下来，我们把 y_ 的每一个元素和 tf.log(y_) 的对应元素相乘。最后，用 tf.reduce_sum 计算张量的所有元素的总和。（注意，这里的交叉熵不仅仅用来衡量单一的一对预测和真实值，而是所有 100 幅图片的交叉熵的总和。对于 100 个数据点的预测表现比单一数据点的表现能更好地描述我们的模型的性能。

现在我们知道我们需要我们的模型做什么啦，用 TensorFlow 来训练它是非常容易的。因为 TensorFlow 拥有一张描述你各个计算单元的图，它可以自动地使用反向传播

算法 (backpropagation algorithm) 来有效地确定你的变量是如何影响你想要最小化的那个成本值的。然后，TensorFlow 会用你选择的优化算法来不断地修改变量以降低成本。

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

在这里，我们要求 TensorFlow 用梯度下降算法（gradient descent algorithm）以 0.01 的学习速率最小化交叉熵。梯度下降算法（gradient descent algorithm）是一个简单的学习过程，TensorFlow 只需将每个变量一点点地往使成本不断降低的方向移动。当然 TensorFlow 也提供了其他许多优化算法：只要简单地调整一行代码就可以使用其他的算法。

TensorFlow 在这里实际上所做的是，它会在后台给描述你的计算的那张图里面增加一系列新的计算操作单元用于实现反向传播算法和梯度下降算法。然后，它返回给你的只是一个单一的操作，当运行这个操作时，它用梯度下降算法训练你的模型，微调你的变量，不断减少成本。

现在，我们已经设置好了我们的模型。在运行计算之前，我们需要添加一个操作来初始化我们创建的变量：

```
init = tf.initialize_all_variables()
```

现在我们可以在一个 Session 里面启动我们的模型，并且初始化变量：

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

然后开始训练模型，这里我们让模型循环训练 1000 次！

```
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})
```

该循环的每个步骤中，我们都会随机抓取训练数据中的 100 个批处理数据点，然后我们用这些数据点作为参数替换之前的占位符来运行 train_step。

使用一小部分的随机数据来进行训练被称为随机训练（stochastic training）- 在这里更确切的说是随机梯度下降训练。在理想情况下，我们希望用我们所有的数据来进行每一步的训练，因为这能给我们更好的训练结果，但显然这需要很大的计算开销。所以，每一次训练我们可以使用不同的数据子集，这样做既可以减少计算开销，又可以最大化地学习到数据集的总体特性。

2.2.5 评估我们的模型

那么我们的模型性能如何呢？

首先让我们找出那些预测正确的标签。`tf.argmax()`是一个非常有用的函数，它能给你在一个张量里沿着某条轴的最高条目的索引值。比如，`tf.argmax(y,1)`是模型认为每个输入最有可能对应的那些标签，而`tf.argmax(y_,1)`代表正确的标签。我们可以用`tf.equal`来检测我们的预测是否真实标签匹配。

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

这行代码会给我们一组布尔值。为了确定正确预测项的比例，我们可以把布尔值转换成浮点数，然后取平均值。例如，[True, False, True, True]会变成[1,0,1,1]，取平均值后得到0.75.

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

最后，我们计算所学习到的模型在测试数据集上面的正确率。

```
print sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.
    test.labels})
```

最终结果值应该大约是91%。

这个结果好吗？嗯，并不太好。事实上，这个结果是很差的。这是因为我们仅仅使用了一个非常简单的模型。不过，做一些小小的改进，我们就可以得到97%的正确率。最好的模型甚至可以获得超过99.7%的准确率！（想了解更多信息，可以看看这个关于各种模型的性能对比列表。）

比结果更重要的是，我们从这个模型中学习到的设计思想。不过，如果你仍然对这里的结果有点失望，可以查看下一个教程，在那里你将学到如何用FensorFlow构建更加复杂的模型以获得更好的性能！

原文地址：MNIST For ML Beginners 翻译：linbojin 校对：

2.3 深入 MNIST

TensorFlow 是一个做大规模数值计算的强大库。其中一个特点就是它能够实现和训练深度神经网络。在这一小节里，我们将会学习在 MNIST 上构建深度卷积分类器的基本步骤。

这个教程假设你已经熟悉神经网络和 *MNIST* 数据集。如果你尚未了解，请查看新手指南.

2.3.1 安装

在创建模型之前，我们会先加载 MNIST 数据集，然后启动一个 TensorFlow 的 session。

加载 MINIST 数据

为了方便起见，我们已经准备了一个脚本来自动下载和导入 MNIST 数据集。它会自动创建一个'MNIST_data' 的目录来存储数据。

```
import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

开始 TensorFlow 的交互会话

Tensorflow 基于一个高效的 C++ 模块进行运算。与这个模块的连接叫做 session。一般而言，使用 TensorFlow 程序的流程是先创建一个图，然后在 session 中加载它。

这里，我们使用更加方便的 InteractiveSession 类。通过它，你可以更加灵活地构建你的代码。它能让你在运行图的时候，插入一些构建计算图的操作。这能给使用交互式文本 shell 如 iPython 带来便利。如果你没有使用 InteractiveSession 的话，你需要在开始 session 和加载图之前，构建整个计算图。

```
import tensorflow as tf
sess = tf.InteractiveSession()
```

计算图

传统的计算行为中，为了更高效地在 Python 里进行数值计算，我们一般会使用像 NumPy 这样用其他语言编写的 lib，在 Python 外完成这些费时的操作（例如矩阵运算）。可是，每一步操作依然会经常在 Python 和第三方 lib 之间切换。这些操作很糟糕，特别是当你想在 GPU 上进行计算，又或者想使用分布式的做法的时候。这些情况下数据传输代价高昂。

在 TensorFlow 中，也有 Python 与外界的频繁操作。但是它在这一方面，做了进一步的改良。TensorFlow 构建一个交互操作的图，作为一个整体在 Python 外运行，而不

是以代价高昂的单个交互操为单位在 Python 外运行。这与 Theano、Torch 的做法很相似。

所以，这部分 Python 代码，目的是构建这个在外部运行的计算图，并安排这个计算图的哪一部分应该被运行。详细请阅读计算图部分的基本用法。

2.3.2 构建 Softmax Regression 模型

在这小节里，我们将会构建一个一层线性的 softmax regression 模型。下一节里，我们会扩展到多层卷积网络。

占位符

我们先来创建计算图的输入（图片）和输出（类别）。

```
x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])
```

这里的x和y并不是具体值，他们是一个placeholder，是一个变量，在 TensorFlow 运行计算的时候使用。

输入图片 x 是浮点数 2 维张量。这里，定义它的shape为[None, 784]，其中 784 是单张展开的 MNIST 图片的维度数。shape的第一维输入指代一个 batch 的大小，None，可为任意值。输出值y_也是一个 2 维张量，其中每一行为一个 10 维向量代表对应 MNIST 图片的分类。

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

Variables

我们现在为模型定义权重W和偏置b。它们可以被视作是额外的输入量，但是 TensorFlow 有一个更好的方式来处理：Variable。一个Variable代表着在 TensorFlow 计算图中的一个值，它是能在计算过程中被读取和修改的。在机器学习的应用过程中，模型参数一般用Variable来表示。

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

我们在调用tf.Variable的时候传入初始值。在这个例子里，我们把W和b都初始化为零向量。W是一个 784×10 的矩阵（因为我们有 784 个特征和 10 个输出值）。b是一个 10 维的向量（因为我们有 10 个分类）。

Variable需要在session之前初始化，才能在session中使用。初始化需要初始值（本例当中是全为零）传入并赋值给每一个Variable。这个操作可以一次性完成。

```
sess.run(tf.initialize_all_variables())
```

预测分类与损失函数

现在我们可以实现我们的 regression 模型了。这只需要一行！我们把图片x和权重矩阵W相乘，加上偏置b，然后计算每个分类的 softmax 概率值。

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

在训练中最小化损失函数同样很简单。我们这里的损失函数用目标分类和模型预测分类之间的交叉熵。

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```

注意，tf.reduce_sum把minibatch里的每张图片的交叉熵值都加起来了。我们计算的交叉熵是指整个minibatch的。

2.3.3 训练模型

我们已经定义好了模型和训练的时候用的损失函数，接下来使用 TensorFlow 来训练。因为 TensorFlow 知道整个计算图，它会用自动微分法来找到损失函数对于各个变量的梯度。TensorFlow 有大量内置的优化算法这个例子中，我们用最速下降法让交叉熵下降，步长为 0.01。

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

这一行代码实际上是用来往计算图上添加一个新操作，其中包括计算梯度，计算每个参数的步长变化，并且计算出新的参数值。

train_step这个操作，用梯度下降来更新权值。因此，整个模型的训练可以通过反复地运行train_step来完成。

```
for i in range(1000):
    batch = mnist.train.next_batch(50)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
```

每一步迭代，我们都会加载 50 个训练样本，然后执行一次 train_step，使用 feed_dict，用训练数据替换 placeholder 向量 x 和 y_。

注意，在计算图中，你可以用 feed_dict 来替代任何张量，并不仅限于替换 placeholder。

评估模型

我们的模型效果怎样？

首先，要先知道我们哪些 label 是预测正确了。tf.argmax 是一个非常有用的函数。它会返回一个张量某个维度中的最大值的索引。例如，tf.argmax(y,1) 表示我们模型对每个输入的最大概率分类的分类值。而 tf.argmax(y_,1) 表示真实分类值。我们可以用 tf.equal 来判断我们的预测是否与真实分类一致。

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

这里返回一个布尔数组。为了计算我们分类的准确率，我们将布尔值转换为浮点数来代表对、错，然后取平均值。例如：[True, False, True, True] 变为 [1,0,1,1]，计算出平均值为 0.75。

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

最后，我们可以计算出在测试数据上的准确率，大概是 91%。

```
print accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.
    labels})
```

2.3.4 构建多层卷积网络模型

在 MNIST 上只有 91% 正确率，实在太糟糕。在这个小节里，我们用一个稍微复杂的模型：卷积神经网络来改善效果。这会达到大概 99.2% 的准确率。虽然不是最高，但是还是比较让人满意。

权重初始化

在创建模型之前，我们先来创建权重和偏置。一般来说，初始化时应加入轻微噪声，来打破对称性，防止零梯度的问题。因为我们用的是 ReLU，所以用稍大于 0 的值来初始化偏置能够避免节点输出恒为 0 的问题（dead neurons）。为了不在建立模型的时候反复做初始化操作，我们定义两个函数用于初始化。

```
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

TensorFlow 在卷积和池化上有很强的灵活性。我们怎么处理边界？步长应该设多大？在这个实例里，我们会一直使用 vanilla 版本。我们的卷积使用 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。我们的池化用简单传统的 2x2 大小的模板做 max pooling。为了代码更简洁，我们把这部分抽象成一个函数。

```
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')
```

第一层卷积

现在我们可以开始实现第一层了。它由一个卷积接一个 max pooling 完成。卷积在每个 5x5 的 patch 中算出 32 个特征。权重是一个 [5, 5, 1, 32] 的张量，前两个维度是 patch 的大小，接着是输入的通道数目，最后是输出的通道数目。输出对应一个同样大小的偏置向量。

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
```

为了用这一层，我们把 x 变成一个 4d 向量，第 2、3 维对应图片的宽高，最后一维代表颜色通道。

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

我们把 x_image 和权值向量进行卷积相乘，加上偏置，使用 ReLU 激活函数，最后 max pooling。

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

第二层卷积

为了构建一个更深的网络，我们会把几个类似的层堆叠起来。第二层中，每个 5x5 的 patch 会得到 64 个特征。

```
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)
```

密集连接层

现在，图片降维到 7x7，我们加入一个有 1024 个神经元的全连接层，用于处理整个图片。我们把池化层输出的张量 reshape 成一些向量，乘上权重矩阵，加上偏置，使用 ReLU 激活。

```
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)
```

Dropout

为了减少过拟合，我们在输出层之前加入 dropout。我们用一个 placeholder 来代表一个神经元在 dropout 中被保留的概率。这样我们可以在训练过程中启用 dropout，在测试过程中关闭 dropout。TensorFlow 的 tf.nn.dropout 操作会自动处理神经元输出值的 scale。所以用 dropout 的时候可以不用考虑 scale。

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

输出层

最后，我们添加一个 softmax 层，就像前面的单层 softmax regression 一样。

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

训练和评估模型

这次效果又有多好呢？我们用前面几乎一样的代码来测测看。只是我们会用更加复杂的 ADAM 优化器来做梯度最速下降，在 feed_dict 中加入额外的参数 keep_prob 来控制 dropout 比例。然后每 100 次迭代输出一次日志。

```
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})
```

以上代码，在最终测试集上的准确率大概是 99.2

目前为止，我们已经学会了用 TensorFlow 来快速和简易地搭建、训练和评估一个复杂一点儿的深度学习模型。

原文地址：Deep MNIST for Experts 翻译：chenweican 校对：HongyangWang

2.4 TensorFlow Mechanics 101

代码地址: tensorflow/g3doc/tutorials/mnist/

本篇教程的目的，是向大家展示如何利用 TensorFlow 使用（经典）MNIST 数据集训练并评估一个用于识别手写数字的简易前馈神经网络（feed-forward neural network）。我们的目标读者是有兴趣使用 TensorFlow 的机器学习资深人士。

因此，撰写该系列教程并不是为了教大家机器学习领域的基础知识。

在学习本教程之前，请确保您已按照安装 TensorFlow 教程中的要求，完成了安装。

2.4.1 教程使用的文件

本教程引用如下文件：

只需要直接运行 fully_connected_feed.py 文件，就可以开始训练：

python fully_connected_feed.py

2.4.2 准备数据

MNIST 是机器学习领域的一个经典问题，指的是让机器查看一系列大小为 28x28 像素的手写数字灰度图像，并判断这些图像代表 0-9 中的哪一个数字。

更多相关信息，请查阅 Yann LeCun 网站中关于 MNIST 的介绍或者 Chris Olah 对 MNIST 的可视化探索。

下载

在 ‘run_training()‘方法的一开始，‘input_data.read_data_sets()‘函数会确保你的本地训练文件夹中，已经下载了正确的数据，然后将这些数据解压并返回一个含有 ‘DataSet‘实例的字典。

```
data_sets = input_data.read_data_sets(FLAGS.train_dir, FLAGS.
    fake_data)
```

1

Edit table here[2]

[1] ‘fake_data‘标记是用于单元测试的，读者可以不必理会。

[2] 了解更多数据有关信息，请查阅此系列教程的 [数据下载](mnist/download/index.md) 部分.

输入与占位符

`placeholder_inputs()`函数将生成两个`tf.placeholder`操作，定义传入图表中的 shape 参数，shape 参数中包括`batch_size`值，后续还会将实际的训练用例传入图表。

```
images_placeholder = tf.placeholder(tf.float32, shape=(batch_size,
    IMAGE_PIXELS))
labels_placeholder = tf.placeholder(tf.int32, shape=(batch_size))
```

在训练循环（training loop）的后续步骤中，传入的整个图像和标签数据集会被切片，以符合每一个操作所设置的`batch_size`值，占位符操作将会填补以符合这个`batch_size`值。然后使用`feed_dict`参数，将数据传入`sess.run()`函数。

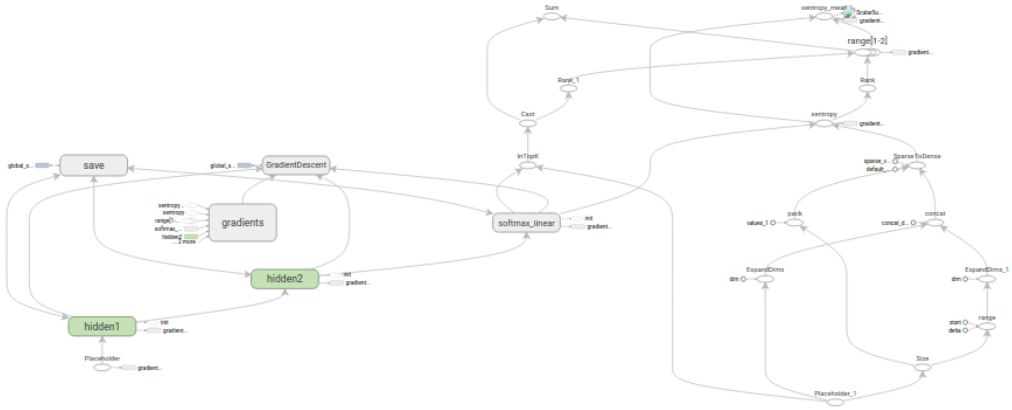
2.4.3 构建图表（Build the Graph）

在为数据创建占位符之后，就可以运行`mnist.py`文件，经过三阶段的模式函数操作：`inference()`，`loss()\lstinline`，和`training()`。图表就构建完成了。

1. `inference()` ——尽可能地构建好图表，满足促使神经网络向前反馈并做出预测的要求。

2. `loss()` ——往 inference 图表中添加生成损失（loss）所需要的操作（ops）。

3. `training()` ——往损失图表中添加计算并应用梯度（gradients）所需的操作。



推理（Inference）

`inference()`函数会尽可能地构建图表，做到返回包含了预测结果（output prediction）的 Tensor。

它接受图像占位符为输入，在此基础上借助 ReLu(Rectified Linear Units) 激活函数，构建一对完全连接层（layers），以及一个有着十个节点（node）、指明了输出 logtis 模型的线性层。

每一层都创建于一个唯一的 [‘tf.name$_s cope$‘](../$api_d ocs/python/framework.mdname_s cope$)

```
with tf.name_scope('hidden1') as scope:
```

在定义的作用域中,每一层所使用的权重和偏差都在 [‘tf.Variable‘](../api_docs/python/state_ops.md#Vari

```
weights = tf.Variable(tf.truncated_normal([IMAGE_PIXELS,
    hidden1_units], stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))),
    name='weights')
biases = tf.Variable(tf.zeros([hidden1_units]), name='biases')
```

例如,当这些层是在hidden1作用域下生成时,赋予权重变量的独特名称将会是"hidden1/weights"。

每个变量在构建时，都会获得初始化操作（initializer ops）。

在这种最常见的情况下,通过 [‘tf.truncated_normal‘](../api_docs/python/constant_op.md#truncated_norm

然后,通过 [‘tf.zeros‘](../api_docs/python/constant_op.md#zeros)biases0shapeconnectto

图表的三个主要操作,分别是两个 [‘tf.nn.relu‘](../api_docs/python/nn.md#relu)[‘tf.matmul‘](../api_docs/

```
hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
```

```
hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
```

```
logits = tf.matmul(hidden2, weights) + biases
```

最后，程序会返回包含了输出结果的 ‘logits‘Tensor。

损失（Loss）

loss()函数通过添加所需的损失操作，进一步构建图表。

首先，labels_placeholer中的值，将被编码为一个含有 1-hot values 的 Tensor。例如，如果类标识符为 “3”，那么该值就会被转换为：[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

```
batch_size = tf.size(labels)
labels = tf.expand_dims(labels, 1)
indices = tf.expand_dims(tf.range(0, batch_size, 1), 1)
concated = tf.concat(1, [indices, labels])
onehot_labels = tf.sparse_to_dense(
    concated, tf.pack([batch_size, NUM_CLASSES]), 1.0, 0.0)
```

之后,又添加一个tf.nn.softmax_cross_entropy_with_logits 操作[3],用来比较inference()函数与 1-hot 标签所输出的 logits Tensor。

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits,
    onehot_labels, name='xentropy')
```

然后，使用tf.reduce_mean 函数，计算 batch 维度（第一维度）下交叉熵（cross entropy）的平均值，将将该值作为总损失。

[3]交叉熵是信息理论中的概念，可以让我们描述如果基于已有事实，相信神经网络所做的推测最坏会导致什么结果。更多详情，请查阅博文《可视化信息理论》(http://colah.github.io/posts/2015-09-Visual-Information/)

```
loss = tf.reduce_mean(cross_entropy, name='xentropy_mean')
```

最后，程序会返回包含了损失值的 Tensor。

训练

`training()`函数添加了通过梯度下降（gradient descent）将损失最小化所需的操作。

首先，该函数从`loss()`函数中获取损失 Tensor，将其交给`[tf.scalar_summary]`，后者在与`SummaryWriter`（见下文）配合使用时，可以向事件文件（events file）中生成汇总值（summary values）。在本篇教程中，每次写入汇总值时，它都会释放损失 Tensor 的当前值（snapshot value）。

```
tf.scalar_summary(loss.op.name, loss)
```

接下来，我们实例化一个`[tf.train.GradientDescentOptimizer]`，负责按照所要求的学习效率（learning rate）应用梯度下降法（gradients）。

```
optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)
```

之后，我们生成一个变量用于保存全局训练步骤（global training step）的数值，并使用`[`minimize()`]`函数更新系统中的三角权重（triangle weights）、增加全局步骤的操作。根据惯例，这个操作被称为`train_op`，是 TensorFlow 会话（session）诱发一个完整训练步骤所必须运行的操作（见下文）。

```
global_step = tf.Variable(0, name='global_step', trainable=False)
train_op = optimizer.minimize(loss, global_step=global_step)
```

最后，程序返回包含了训练操作（training op）输出结果的 Tensor。

2.4.4 训练模型

一旦图表构建完毕，就通过`fully_connected_feed.py`文件中的用户代码进行循环地迭代式训练和评估。

图表 (The Graph)

在`run_training()`这个函数的一开始，是一个 Python 语言中的`with`命令，这个命令表明所有已经构建的操作都要与默认的`[`tf.Graph`]`全局实例关联起来。

```
with tf.Graph().as_default():
```

`tf.Graph`实例是一系列可以作为整体执行的操作。TensorFlow 的大部分场景只需要依赖默认图表一个实例即可。

利用多个图表的更加复杂的使用场景也是可能的，但是超出了本教程的范围。

会话 (The Session)

完成全部的构建准备、生成全部所需的操作之后，我们就可以创建一个[`tf.Session`]，用于运行图表。

```
sess = tf.Session()
```

另外，也可以利用with代码块生成Session，限制作用域：

```
with tf.Session() as sess:
```

Session函数中没有传入参数，表明该代码将会依附于（如果还没有创建会话，则会创建新的会话）默认的本地会话。

生成会话之后，所有tf.Variable实例都会立即通过调用各自初始化操作中的[`sess.run()`]函数进行初始化。

```
init = tf.initialize_all_variables()
sess.run(init)
```

[`sess.run()`]方法将会运行图表中与作为参数传入的操作相对应的完整子集。在初次调用时，init操作只包含了变量初始化程序[`tf.group`]。图表的其他部分不会在这里，而是在下面的训练循环运行。

训练循环

完成会话中变量的初始化之后，就可以开始训练了。

训练的每一步都是通过用户代码控制，而能实现有效训练的最简单循环就是：

```
for step in xrange(max_steps):
    sess.run(train_op)
```

但是，本教程中的例子要更为复杂一点，原因是我们必须把输入的数据根据每一步的情况进行切分，以匹配之前生成的占位符。

向图表提

执行每一步时，我们的代码会生成一个反馈字典（feed dictionary），其中包含对应步骤中训练所要使用的例子，这些例子的哈希键就是其所代表的占位符操作。

fill_feed_dict函数会查询给定的DataSet，索要下一批次batch_size的图像和标签，与占位符相匹配的 Tensor 则会包含下一批次的图像和标签。

```
images_feed, labels_feed = data_set.next_batch(FLAGS.batch_size)
```

然后，以占位符为哈希键，创建一个 Python 字典对象，键值则是其代表的反馈 Tensor。

```
feed_dict = {
    images_placeholder: images_feed,
    labels_placeholder: labels_feed,
}
```

这个字典随后作为feed_dict参数，传入sess.run()函数中，为这一步的训练提供输入样例。

检查状态

在运行sess.run函数时，要在代码中明确其需要获取的两个值：`[train_op, loss]`。

```
for step in xrange(FLAGS.max_steps):
    feed_dict = fill_feed_dict(data_sets.train, images_placeholder,
    labels_placeholder)
    _, loss_value = sess.run([train_op, loss], feed_dict=feed_dict)
```

因为要获取这两个值，sess.run()会返回一个有两个元素的元组。其中每一个Tensor对象，对应了返回的元组中的 numpy 数组，而这些数组中包含了当前这步训练中对应Tensor 的值。由于train_op并不会产生输出，其在返回的元祖中的对应元素就是None，所以会被抛弃。但是，如果模型在训练中出现偏差，loss Tensor 的值可能会变成NaN，所以我们要获取它的值，并记录下来。

假设训练一切正常，没有出现NaN，训练循环会每隔 100 个训练步骤，就打印一行简单的状态文本，告知用户当前的训练状态。

```
if step % 100 == 0:
    print 'Step␣%d:␣loss␣=␣%.2f␣(%.3f␣sec)' % (step, loss_value,
    duration)
```

状态可视化 为了释放[TensorBoard]所使用的事件文件（events file），所有的即时数据（在这里只有一个）都要在图表构建阶段合并至一个操作（op）中。

```
summary_op = tf.merge_all_summaries()
```

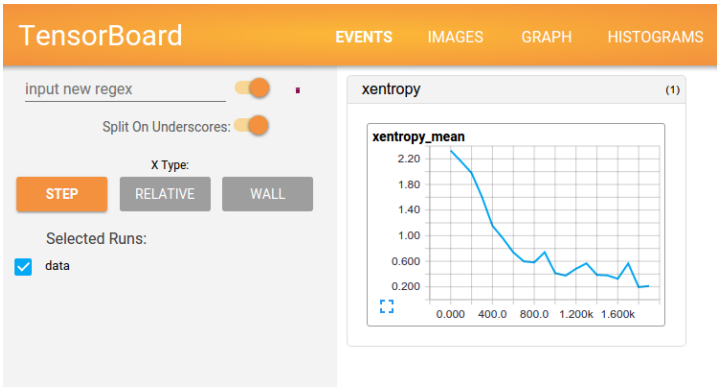
在创建好会话（session）之后，可以实例化一个[`tf.train.SummaryWriter`]，用于写入包含了图表本身和即时数据具体值的事件文件。

```
summary_writer = tf.train.SummaryWriter(FLAGS.train_dir, graph_def=
    sess.graph_def)
```

最后，每次运行summary_op时，都会往事件文件中写入最新的即时数据，函数的输出会传入事件文件读写器（writer）的add_summary()函数。。

```
summary_str = sess.run(summary_op, feed_dict=feed_dict)
summary_writer.add_summary(summary_str, step)
```

事件文件写入完毕之后，可以就训练文件夹打开一个 TensorBoard，查看即时数据的情况。



** 注意 **：了解更多如何构建并运行 TensorBoard 的信息，请查看相关教程 [Tensorboard：训练过程可视化](../how_tos/summaries_and_tensorboard.md)

保存检查点（checkpoint）

为了得到可以用来后续恢复模型以进一步训练或评估的检查点文件（checkpoint file），我们实例化一个[`tf.train.Saver`]

```
saver = tf.train.Saver()
```

在训练循环中，将定期调用[`saver.save()`]方法，向训练文件夹中写入包含了当前所有可训练变量值得检查点文件。

```
saver.save(sess, FLAGS.train_dir, global_step=step)
```

这样，我们以后就可以使用[`saver.restore()`]方法，重载模型的参数，继续训练。

```
saver.restore(sess, FLAGS.train_dir)
```

2.4.5 评估模型

每隔一千个训练步骤，我们的代码会尝试使用训练数据集与测试数据集，对模型进行评估。do_eval函数会被调用三次，分别使用训练数据集、验证数据集合测试数据集。

```
print 'Training Data Eval:'
do_eval(sess, eval_correct, images_placeholder, labels_placeholder,
    data_sets.train)
print 'Validation Data Eval:'
do_eval(sess, eval_correct, images_placeholder, labels_placeholder,
    data_sets.validation)
print 'Test Data Eval:'
do_eval(sess, eval_correct, images_placeholder, labels_placeholder,
    data_sets.test)
```

>注意,更复杂的使用场景通常是,先隔绝‘data_sets.test‘hyperparametertuningMNIST

构建评估图表（Eval Graph）

在打开默认图表（Graph）之前，我们应该先调用‘get_data(train = False)‘

```
test_all_images, test_all_labels = get_data(train=False)
```

在进入训练循环之前，我们应该先调用 'mnist.py'文件中的 'evaluation'函数，传入的 logits 和标签参数要与 'loss'函数的一致。这样做事为了先构建 Eval 操作。

```
eval_correct = mnist.evaluation(logits, labels_placeholder)
```

'evaluation'函数会生成 ['tf.nn.in$_t op_k$'](../$api_d ocs/python/nn.md in_t op_k$)$KK1$

```
eval_correct = tf.nn.in_top_k(logits, labels, 1)
```

评估图表的输出（Eval Output）

之后，我们可以创建一个循环，往其中添加 'feed$_d ict$''$sess.run()$''$eval_c orrect$'

```
for step in xrange(steps_per_epoch):
    feed_dict = fill_feed_dict(data_set,
                               images_placeholder,
                               labels_placeholder)
    true_count += sess.run(eval_correct, feed_dict=feed_dict)
```

'true$_c ount$''$in_t op_k$'

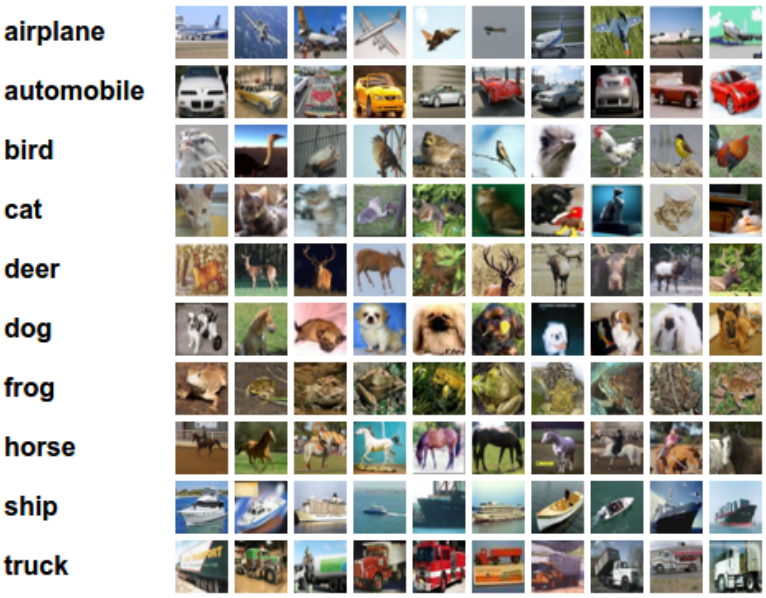
```
precision = float(true_count) / float(num_examples)
print '  Num examples: %d  Num correct: %d  Precision @ 1: %0.02f' %
    (
    num_examples, true_count, precision)
```

> 原文：[TensorFlow Mechanics 101](http://www.tensorflow.org/tutorials/mnist/tf/index.md) > 翻译：[bingjin](https://github.com/bingjin) > 校对：[LichAmnesia](https://github.com/LichAmnesia)

2.5 卷积神经网络

2.5.1 Overview

对 CIFAR-10 数据集的分类是机器学习中一个公开的基准测试问题，其任务是对一组 32x32RGB 的图像进行分类，这些图像涵盖了 10 个类别：`airplane`, `automobile`, `bird`, `cat`, `deer`, `dog`, `frog`, `horse`, `ship`, 和 `truck`.[4]



想了解更多信息请参考CIFAR-10 page，以及 Alex Krizhevsky 写的技术报告。

G 目标

本教程的目标是建立一个用于识别图像的相对较小的卷积神经网络，在这一过程中，本教程会：

- 着重于建立一个规范的网络组织结构，训练并进行评估；
- 为建立更大规模更加复杂的模型提供一个范例

选择 CIFAR-10 是因为它的复杂程度足以用来检验 TensorFlow 中的大部分功能，并可将其扩展为更大的模型。与此同时由于模型较小所以训练速度很快，比较适合用来测试新的想法，检验新的技术。

本教程的重点

CIFAR-10 教程演示了在 TensorFlow 上构建更大更复杂模型的几个种重要内容：

[4] This tutorial is intended for advanced users of TensorFlow and assumes expertise and experience in machine learning

- 相关核心数学对象，如卷积、修正线性激活、最大池化以及局部响应归一化；
- 训练过程中一些网络行为的可视化，这些行为包括输入图像、损失情况、网络行为的分布情况以及梯度；
- 算法学习参数的移动平均值的计算函数，以及在评估阶段使用这些平均值提高预测性能；
- 实现了一种机制，使得学习率随着时间的推移而递减；
- 为输入数据设计预存取队列，将磁盘延迟和高开销的图像预处理操作与模型分离开来处理；

我们也提供了模型的多 GUP 版本，用以表明：

- 可以配置模型后使其在多个 GPU 上并行的训练
- 可以在多个 GPU 之间共享和更新变量值

我们希望本教程给大家开了个头，使得在 Tensorflow 上可以为视觉相关工作建立更大型的 Cnns 模型

模型架构

本教程中的模型是一个多层架构，由卷积层和非线性层 (nonlinearities) 交替多次排列后构成。这些层最终通过全连通层对接到 softmax 分类器上。这一模型除了最顶部的几层外，基本跟 Alex Krizhevsky 提出的模型一致。

在一个 GPU 上经过几个小时的训练后，该模型达到了最高 86% 的精度。细节请查看下面的描述以及代码。模型中包含了 1,068,298 个学习参数，分类一副图像需要大概 19.5M 个乘加操作。

2.5.2 Code Organization

本教程的代码位于tensorflow/models/image/cifar10/.

2.5.3 CIFAR-10 模型

Model Inputs

Model Prediction

Model Training

2.5.4 Lauching and Training the Model

2.5.5 Evaluating a Model

2.5.6 Traning a Model Using Multiple GPU Cards

Placing Variables and Operations on Devices

Lauching and Training the Model on Multiple GPU cards

2.5.7 Next Steps

第三章　运作方式

3.1　变量: 创建、初始化、保存和加载

当训练模型时，用变量来存储和更新参数。变量包含张量 (Tensor) 存放于内存的缓存区。建模时它们需要被明确地初始化，模型训练后它们必须被存储到磁盘。这些变量的值可在之后模型训练和分析是被加载。

本文档描述以下两个 TensorFlow 类。点击以下链接可查看完整的 API 文档：

- tf.Variable 类
- tf.train.Saver 类

3.1.1　变量创建

当创建一个变量时，你将一个张量作为初始值传入构造函数 Variable()。TensorFlow 提供了一系列操作符来初始化张量，初始值是常量或是随机值。

```
# Create two variables.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
    name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
```

调用 tf.Variable() 添加一些操作 (Op, operation) 到 graph：

- 一个 Variable 操作存放变量的值。
- 一个初始化 op 将变量设置为初始值。这事实上是一个 tf.assign 操作。
- 初始值的操作，例如示例中对 biases 变量的 zeros 操作也被加入了 graph。

`tf.Variable`的返回值是 Python 的`tf.Variable`类的一个实例。

3.1.2　变量初始化

变量的初始化必须在模型的其它操作运行之前先明确地完成。最简单的方法就是添加一个给所有变量初始化的操作，并在使用模型之前首先运行那个操作。

你或者可以从检查点文件中重新获取变量值，详见下文。

使用`tf.initialize_all_variables()`添加一个操作对变量做初始化。记得在完全构建好模型并加载之后再运行那个操作。

```
# Create two variables.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                      name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
...
# Add an op to initialize the variables.
init_op = tf.initialize_all_variables()

# Later, when launching the model
with tf.Session() as sess:
  # Run the init operation.
  sess.run(init_op)
  ...
  # Use the model
  ...
```

由另一个变量初始化

你有时候会需要用另一个变量的初始化值给当前变量初始化。由于`tf.initialize_all_variables()`是并行地初始化所有变量，所以在有这种需求的情况下需要小心。

用其它变量的值初始化一个新的变量时，使用其它变量的`initialized_value()`属性。你可以直接把已初始化的值作为新变量的初始值，或者把它当做 tensor 计算得到一个值赋予新变量。

```
# Create a variable with a random value.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),name
    ="weights")
# Create another variable with the same value as 'weights'.
w2 = tf.Variable(weights.initialized_value(), name="w2")
# Create another variable with twice the value of 'weights'
w_twice = tf.Variable(weights.initialized_value() * 0.2, name="
    w_twice")
```

自定义初始化

`tf.initialize_all_variables()`函数便捷地添加一个 op 来初始化模型的所有变量。你也可以给它传入一组变量进行初始化。详情请见 Variables Documentation，包括检查变量是否被初始化。

3.1.3 保存和加载

最简单的保存和恢复模型的方法是使用 tf.train.Saver 对象。构造器给 graph 的所有变量，或是定义在列表里的变量，添加`save`和`restore`ops。`saver`对象提供了方法来运行这些 ops，定义检查点文件的读写路径。

Checkpoint Files

Variables are saved in binary files that, roughly, contain a map from variable names to tensor values.

When you create a Saver object, you can optionally choose names for the variables in the checkpoint files. By default, it uses the value of the `Variable.name` property for each variable.

保存变量

用`tf.train.Saver()`创建一个`Saver`来管理模型中的所有变量。

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add an op to initialize the variables.
init_op = tf.initialize_all_variables()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, initialize the variables, do some work, save the
# variables to disk.
with tf.Session() as sess:
  sess.run(init_op)
  # Do some work with the model.
  ..
  # Save the variables to disk.
  save_path = saver.save(sess, "/tmp/model.ckpt")
  print "Model saved in file: ", save_path
```

恢复变量

用同一个`Saver`对象来恢复变量。注意，当你从文件中恢复变量时，不需要事先对它们做初始化。

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
  # Restore variables from disk.
  saver.restore(sess, "/tmp/model.ckpt")
  print "Model restored."
```

```
  # Do some work with the model
  ...
```

选择存储和恢复哪些变量

如果你不给`tf.train.Saver()`传入任何参数，那么saver将处理graph中的所有变量。其中每一个变量都以变量创建时传入的名称被保存。

有时候在检查点文件中明确定义变量的名称很有用。举个例子，你也许已经训练得到了一个模型，其中有个变量命名为`"weights"`，你想把它的值恢复到一个新的变量`"params"`中。

有时候仅保存和恢复模型的一部分变量很有用。再举个例子，你也许训练得到了一个 5 层神经网络，现在想训练一个 6 层的新模型，可以将之前 5 层模型的参数导入到新模型的前 5 层中。

你可以通过给`tf.train.Saver()`构造函数传入 Python 字典，很容易地定义需要保持的变量及对应名称：键对应使用的名称，值对应被管理的变量。

注意：

> You can create as many saver objects as you want if you need to save and restore different subsets of the model variables. The same variable can be listed in multiple saver objects, its value is only changed when the saver restore() method is run.
>
> If you only restore a subset of the model variables at the start of a session, you have to run an initialize op for the other variables. See `tf.initialize_variables()` for more information.

如果需要保存和恢复模型变量的不同子集，可以创建任意多个 saver 对象。同一个变量可被列入多个 saver 对象中，只有当 saver 的`restore()`函数被运行时，它的值才会发生改变。

如果你仅在 session 开始时恢复模型变量的一个子集，你需要对剩下的变量执行初始化 op。详情请见`tf.initialize_variables()`。

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore only 'v2' using the name "my_v2"
saver = tf.train.Saver({"my_v2": v2})
# Use the saver object normally after that.
...
```

第四章　资源

第五章　其他