GETTING STARTED WITH

TensorFlow

BY TIMOTHY SPANN

CONTENTS

INTRODUCTION TO TENSORFLOW

TensorFlow is a deep learning library from Google that is open-source and available on GitHub. TensorFlow excels at numerical computing, which is critical for deep learning. It has a rich set of application programming interfaces in most major languages and environments needed for deep learning projects: Python, C, C++, Rust, Haskell, Go, Java, Android, IoS, Mac OS, Windows, Linux, and Raspberry Pi. The primary unit in TensorFlow is a tensor. A tensor consists of a set of primitive values shaped into an array of any number of dimensions. These massive numbers of large arrays are the reason that GPUs and other processors designed to do floating point mathematics excel at speeding up these algorithms.

TENSORS

A tensor is a set of primitives in a multidimensional array, ranked by number of dimensions

```
0:   5.0
1:   [5.0, 10.0, 15.0, 20.0, 25.0}
2:   [[5.0, 10.0], [15.0, 20.0], [25.0, 30.0], [35.0,
40.0, 45.0]]
```

In TensorFlow programs, you will need to use variables to hold tensors in memory. Some tensors are constants and not variables; for example, static numbers. An important note is that before you start using variables, you must initialize them to a value. TensorFlow first builds a graph of all the operation to be done, and then when a "session" is called, it "runs" the graph. It's built to be scalable, by changing internal data representation to tensors (AKA multi-dimensional arrays).

DOING SOME SIMPLE MATH

```
import tensorflow as tf

firstnumber = tf.constant([10, 20, 30, 35, 40, 50])
secondnumber = tf.Variable(firstnumber + 50)

with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    print(session.run(secondnumber))
```

The APIs are well-documented and well-designed, so it was easy, for example, for me to port the LabelImage Java example to work as a processor for Apache NiFi. Some people prefer a higher-level interface for neural network programming, such as Keras. Keras is available for Python and works not only for TensorFlow but also for CNTK and Theano. I recommend using Python 3.7 and starting with plain TensorFlow before you investigate Keras. TensorFlow programs are usually structured into a construction phase, which assembles a data graph, and an execution phase, which uses a session to execute operations in the graph.

Questioning whether TensorFlow is ready for production environments belies the fact that TensorFlow has been in the open-source community for almost two years and has been used by many companies. TensorFlow excels in production environments due to its distributed and parallel design and its ability to access data from Hadoop HDFS. Another key feature of TensorFlow is TensorFlow Serving, which is a high-performance production server for deploy and serve machine learning models using gRPC as its network protocol.

THREE LAYERS OF TENSORFLOW ARCHITECTURE

1. **High-level libraries**: Python, TensorBoard, Java, and more.
2. **TensorFlow core**: Neural net ops, graph execution engine, and more.
3. **Platforms**: CPUs, GPUs, TPUs, Android, and iOS across all major operating systems.

QUICK PYTHON EXAMPLE

```
import tensorflow as tf
five = tf.constant("00")
tf.Variable(0, name="variablename")
session = tf.Session()

file_writer = tf.summary.FileWriter('tflogs', session.
graph)

run_options = tf.RunOptions(trace_level=tf.RunOptions.
FULL_TRACE)
run_metadata = tf.RunMetadata()

summary, result = session.run(fetches=five, feed_dict=None,
options=run_options, run_metadata=run_metadata)
print('Value %s' % result)
file_writer.add_run_metadata(run_metadata, 'result %s' %
result)
session.close()
```

Neural networks have been around for a long time, but now they can be run in a reasonable amount of time due to more powerful computers, GPUs, and elastic cloud computing. Google has made neural networks mainstream through the public availability of TensorFlow on all major clouds and platforms to be used by researchers, data scientists, data engineers, cloud developers, mobile developers, and web application users.

An example implementation of a neural network would be to define an architecture, load data to the model, break up data into batches to preprocess, convert it, and use it for training. As the model gets trained in increments, the model is saved for reuse. Future runs test the model on new data to check performance. After that, you continue to train over many iterations and as much training data as you can obtain. This is helped by fast networks, fast CPUs, fast GPUs, large RAM, large hard drives, and generally fast computers.

USE CASES

- **Speech recognition**: Recognize what people are saying and convert it to text.
- **Image recognition**: Describe, in text, what an image is.
- **Sentiment analysis**: Determine if a human sentence is positive, negative, or neutral.
- **Text summarization**: Summarize a longer article automatically.
- **Mobile image and video processing**: Process images and videos from a mobile device.
- **Language translation**: Translate between human languages.
- **Object detection in photos**: Determine what items are in photos and box them.
- **Image captioning**: Add captions and possible paragraphs describing an image.
- **Chatbot**: Intelligence response and routing of live people with TensorFlow-generated replies.
- **Cancer detection**: Compare facial and body photos with photos of known cancer patients.

SETUP AND INSTALLATION

Serious data scientists will want a powerful desktop computer that has 128GB of RAM or more, several NVidia high-end CUDA cards, fast SSD storage, and a recent version of Ubuntu. The main platform to run TensorFlow is Ubuntu-based Linux. If you have an NVidia GPU that supports CUDA, you need to install those NVidia libraries to make sure you take advantage of all the GPUs on the device. This is critical for performance, as the large number of matrix multiplication requires floating point capable processors like GPUS. If you do not have a GPU available, you will face difficulties in running data that's not pre-trained. Even a decent machine will stall your progress if it is not equipped with a CUDA-capable, many-core GPU. On several occasions, I have attempted to train IM2TXT a PowerBook and gave up after days of running with little progress.

If you are prepared to use pre-trained models or just try out a basic example, feel free to other a non-GPU enabled device.

Whatever platform you have picked, TensorFlow has a lot of means of installation: source build from GitHub, Python virtual env, Python pip, Docker, and Anaconda. Anaconda is a nice environment, but I like with keep it simple to Python 3.6 and pip3. For some environments, you may need to be root or have sudo access.

```
apt-get install python3-pip python3-dev
pip3 install tensorflow
pip3 install grpcio==1.4.0
```

For GPU

```
pip3 install tensorflow-gpu
```

Sometimes, that doesn't work. Look at GitHub or try:

```
sudo pip3 install --upgrade latesturlforyourarchitecture
from github.
```

I also recommend you have the TensorFlow source, models, and TensorBoard GitHub repositories downloaded. You should also install JDK 8+, Bazel, and all build tools required for your operating system. I recommend you only work with TensorFlow on Ubuntu or OSX. If you are interested in running on Raspberry PI, NVidia TX1, or other devices, make sure you follow specific guidelines for those builds. They will often have specialized instructions, utilities, versions, or even forks of source code to utilize.

On OSX, you should have homebrew installed:

```
brew upgrade bazel
```

With older versions, or if something goes wrong, you may need to try some other installation options.

```
sudo easy_install --upgrade six
sudo pip install --upgrade tensorflow
pip install grpc
```

I recommend checking that you have TensorFlow installed by running `pip show` and then running our example program above.

```
pip show tensorflow
Name: tensorflow
Version: 1.3.0
Summary: TensorFlow helps the tensors flow
Home-page: http://tensorflow.org/
Author: Google Inc.
Author-email: opensource@google.com
License: Apache 2.0
Location: /usr/local/lib/python2.7/site-packages
Requires: wheel, backports.weakref, protobuf, numpy,
mock, tensorflow-tensorboard, six
```

For those really interested in the cutting edge or facing issues, you can try the current nightly build for your platform here.

GETTING STARTED

Most people start with MNIST, which is a dataset of handwritten digits. This is a fun, well-documented example to start with, and is often thought of as the "Hello World" example. Before you start

with this, you must know Python and some basics on running shell programs in your environment.

As we documented in our simple example, start off by importing TensorFlow. From there, you can create constants and variables, create a session, and then run the session. A session is needed to execute your graph(s).

In the Java version, the key imports are:

```
import org.tensorflow.DataType;
import org.tensorflow.Graph;
import org.tensorflow.Output;
import org.tensorflow.Session;
import org.tensorflow.Tensor;
```

MAVEN BUILD FOR JAVA

```
<dependency>
  <groupId>org.tensorflow</groupId>
  <artifactId>tensorflow</artifactId>
  <version>1.3.0</version>
</dependency>
```

In the Java version, you start with a graph, import a graph definition, create a session, and return a tensor. I highly recommend looking at the Java and Android examples provided in the TensorFlow GitHub repository.

EXAMPLE RUN

```
import tensorflow as tf

firstnumber = tf.constant([10, 20, 30, 35, 40, 50])
secondnumber = tf.Variable(firstnumber + 50)

with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    print(session.run(secondnumber))

python3 testtf2.py
2017-10-02 14:40:33.009719: W tensorflow/core/platform/
cpu_feature_guard.cc:45] The TensorFlow library wasn't
compiled to use SSE4.2 instructions, but these are
available on your machine and could speed up CPU
computations.
2017-10-02 14:40:33.009742: W tensorflow/core/platform/
cpu_feature_guard.cc:45] The TensorFlow library wasn't
compiled to use AVX instructions, but these are available
on your machine and could speed up CPU computations.
2017-10-02 14:40:33.009747: W tensorflow/core/platform/
cpu_feature_guard.cc:45] The TensorFlow library
wasn't compiled to use AVX2 instructions, but these
are available on your machine and could speed up CPU
computations.
2017-10-02 14:40:33.009751: W tensorflow/core/platform/
cpu_feature_guard.cc:45] The TensorFlow library wasn't
compiled to use FMA instructions, but these are available
on your machine and could speed up CPU computations.
[ 60  70  80  85  90 100]
```

COMMON TENSORFLOW OPERATIONS

- **tf.rank**
  Returns the rank of a tensor.
- **tf.constant**
  Creates a constant from a value, type, and shape.
- **tf.Variable**
  Creates a variable from a tensor.
- **tf.decode_csv**
  Converts CSV files to tensors, with each column mapping to a tensor.
- **tf.decode_base64**
  Decodes base-64-encoded tensor strings.
- **tf.subtract**
  Subtracts one tensor from another (must be the same type).
- **tf.add**
  Adds two tensors of the same type together, for numbers and strings.
- **tf.to_int32**
  Converts tensor primitives to int32.
- **tf.multiply**
  Multiplies two tensors of the same type together, for numbers.
- **tf.div**
  Divides numerator by denominator of real numeric types and return quotient.
- **tf.abs**
  The absolute value of the tensor.
- **tf.negative**
  Returns negative value of numeric tensor.
- **tf.maximum**
  Returns the maximum of two tensors for a subset of numeric types.
- **tf.minimum**
  Returns the minimum of two tensor for a subset of numeric types.
- **tf.string_to_number**
  Converts a string tensor to a specified numeric type; specify `out_type=tf.DType`, where `DType` is a subset of numeric types.
- **tf.concat**
  Concatenates a tensor, along one specified dimension.
- **tf.fill**
  Creates a tensor filled with a specified scalar value.
- **tf.tuple**
  Groups tensors together.
- **tf.zeros**
  Creates a tensor populated with zeros of a certain numeric type shape.
- **tf. convert_to_tensor**
  Converts the value to a tensor of specified type.
- **tf.while_loop**
  The while loop.
- **tf.case**
  The case operations on bools.

- **tf.count_up_to**
  Increments a mutable tensor of int32 or int64 to an int limit.
- **tf.Print**
  Prints out a debug message of tensors and messages.
- **tf.is_nan**
  Returns elements of tensors that are not a number.
- **tf.is_finite**
  Returns elements of tensors that are finite, for half, float32, and float64.
- **tf.logical_and**
  Returns truth value of each boolean element of two tensors.
- **tf.logical_not**
  Returns truth value of not each boolean element of a tensor.
- **tf.logical_or**
  Returns truth value of each boolean element of two tensors.
- **tf.logical_xor**
  Returns truth value of each of boolean elements, exclusive or.
- **tf.equal**
  Returns truth value of first tensor == second tensor, element-wise, on two tensors.
- **tf.not_equal**
  Returns truth value of first tensor != second tensor, element-wise, on two tensors.
- **tf.greater_equal**
  Returns truth value of first tensor >= second tensor, element-wise, on two tensors.
- **tf.greater**
  Returns truth value of first tensor > second tensor, element-wise, on two tensors.
- **tf.less_equal**
  Returns truth value of first tensor <= second tensor, element-wise, on two tensors.
- **tf.less**
  Returns truth value of first tensor < second tensor, element-wise, on two tensors.
- **tf.where**
  Returns elements from either of two tensors based on condition.
- **tf.image.decode_jpeg**
  Decodes a JPEG image to a uint8 tensor.
- **tf.image.decode_png**
  Decodes a PNG image to a uint8 or uint16 tensor.
- **tf.image.decode_gif**
  Decodes a GIF image to a uint8 tensor.
- **tf.image_decode_bmp**
  Decodes a BMP image to a uint8 tensor.
- **tf.image.resize_images**
  Resizes images to sizes using one of four predefined methods.
- **tf.image.rot90**
  Rotates an image counter-clockwise 90 degrees.
- **tf.image.transpose_image**
  Transposes an image by swapping first two dimensions.
- **tf.image.adjust_brightness**
  Adjusts image brightness.
- **tf.image.adjust_contrast**
  Adjusts contrast of images.
- **tf.image.adjust_hue**
  Adjusts hue of images.
- **tf.image.adjust_gamma**
  Runs gamma correction on input image.
- **tf.image.adjust_saturation**
  Adjusts saturation channel in image.
- **tf.image.flip_left_right**
  Flips an image horizontally.
- **tf.image.flip_up_down**
  Flips an image vertically.
- **tf.image.draw_bounding_boxes**
  Draws bounding boxes on a batch of images.
- **tf.nn**
  A large collection of neural network ops.
- **tf.contrib**
  Contributed libraries around keras, audio, video, TFLearn, and more.
- **tf.Session()**
  An instance of a session that is the environment to execute operations in a graph to compute tensors.

TRAINING

As we mentioned in an earlier section, you should not try your own training unless your problem space is small enough or you have sufficient hardware to run your training. Training examples like CIFAR-10 will also require downloading anywhere from hundreds of megabytes of data to terabytes of training data, especially when dealing with images and videos. You will be working on datasets of thousands of items and thousands of steps. These numbers can increase astronomically depending on your problem set and data. A cool dataset to check out is SVHN, which contains real-world street house numbers.

IMAGE RECOGNITION WITH INCEPTION V3

A very popular application of TensorFlow is image recognition. Whether it's from a Mac, a Raspberry Pi, or a mobile phone, this is a great use of TensorFlow. If you use Inception V3, you will get a pretty good set of data, which will result in good results. One thing to note is that you want a clear picture without

too much noise in it in order for recognition to have a high matching percentage.

For image recognition with the Inception v3 application, I recommend you download the pre-trained model here.

The example has a pre-trained 80-megabyte model from the TensorFlow models repo, which lets you quickly classify images against that model. This will run on a smartphone, a Raspberry Pi, or your laptop easily in a few seconds.

EXAMPLE

```
python classify_image.py --image_file /opt/demo/
SolarPanelsOnRoof.jpg
solar dish, solar collector, solar furnace (score =
0.98316)
window screen (score = 0.00196)
manhole cover (score = 0.00070)
radiator (score = 0.00041)
doormat, welcome mat (score = 0.00041)
```

USING TENSORBOARD FOR VISUALIZATION AND DEBUGGING

You will need to clone the TensorBoard GitHub repo. Then, you can run TensorBoard pointing to a recent log. We have generated one from the example given in this Refcard.

```
bazel run tensorboard -- -- logdir /opt/demo/tflogs
```

The summary data is logged and sent to the board for visualization. It is a great tool for learning what is going on and learning how to improve your results as you are training and tweaking your algorithms.

SUMMARY

TensorFlow is a really useful library with many practical uses. The Python binding is very mature and usable. When you decide to run in production consider using TensorFlow Serving, TensorFlow on Hadoop, or TensorFlow on Spark.

If you are just beginning to think about deep learning, neural networks, artificial intelligence, or just something practical like image analysis, then evaluate TensorFlow for your needs. I highly recommend you start with the basics and make sure you have a stable Python 3.7 environment set up with all the modules that you will need and that you have basic Python applications that run at a reasonable rate. At a minimum, you will need NumPy installed on your TensorFlow machine.

Step one is to get everything installed. Then, try the basic example we have included in this Refcard. The next step is to determine which of your use cases matches the ones I have listed. If you don't have one matching, search GitHub for TensorFlow projects. If you find nothing, perhaps it's time to look at other machine learning projects or other use cases to try.

ABOUT THE AUTHOR

TIMOTHY SPANN is a Big Data Solutions Engineer. He helps educate and disseminate performant open source solutions for Big Data initiatives to customers and the community. With over 15 years of experience in various technical leadership, architecture, sales engineering, and development roles, he is well-experienced in all facets of Big Data, cloud, IoT, and microservices. As part of his community efforts, he also runs the Future of Data Meetup in Princeton.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com