TensorFlow shines a light on deep learning

Google's open source framework for machine learning and neural networks is fast and flexible, rich in models, and easy to run on CPUs or GPUs

By Martin Heller , Contributing Editor, InfoWorld | Oct 5, 2016

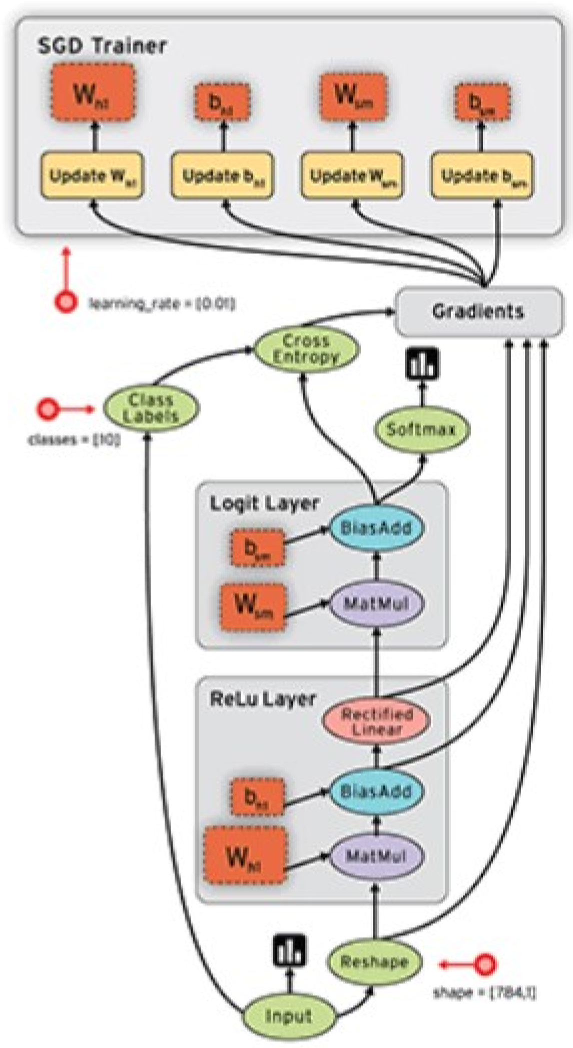
http://www.infoworld.com/article/3127397/artificial-intelligence/review-tensorflow-shines-a-light-on-deep-learning.html

What makes Google Google? Arguably it is machine intelligence, along with a vast sea of data to apply it to. While you may never have as much data to process as Google does, you can use the very same machine learning and neural network library as Google. That library, TensorFlow, was developed by the Google Brain team over the past several years and released to open source in November 2015.

TensorFlow does computation using data flow graphs. Google uses TensorFlow internally for many of its products, both in its datacenters and on mobile devices. For example, the Translate, Maps, and Google apps all use TensorFlow-based neural networks running on our smartphones. And TensorFlow underpins the applied machine learning APIs for Google Cloud Natural Language, Speech, Translate, and Vision.

Data flow graphs are directed acyclic graphs that describe a computation for TensorFlow. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. In addition to the library, TensorFlow includes an interactive program for displaying data flow graphs, TensorBoard.

The principal language for using TensorFlow is Python, although there is limited support for C++. The tutorials supplied with TensorFlow include applications for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation)-based simulations.



This is Google's animated illustration of how tensors flow from node to node through the edges of a data flow graph.

TensorFlow currently runs on Ubuntu Linux, MacOS, Android, iOS, and Raspberry Pi, using Python 2.7, 3.4, or 3.5. Nvidia CUDA GPUs are supported on Linux and MacOS but not required. Google supplies Docker images for TensorFlow both with and without GPU support.

Google offers TensorFlow as a service as part of the Cloud Machine Learning Platform, along with Cloud Dataflow. The Cloud Machine Learning Platform is currently in limited preview and considered to be in alpha test. TensorFlow is also available for use in Cloud Datalab, a beta cloud service and Docker image that is based on Jupyter (formerly IPython) notebooks.

TensorFlow features

Google touts six key advantages for TensorFlow: deep flexibility, true portability, the ability to connect research and production, autodifferentiation of variables, language options, and the ability to maximize performance by prioritizing GPUs over CPUs. Let's try to understand what these mean and why they are important.

Deep flexibility. The major point in this area is that a data flow graph isn't limited to representing neural networks. While there *is* a lot of support for neural networks in TensorFlow, you are free to use libraries on top of TensorFlow or write your own, in Python or C++. Google provides a good example in the form of a partial differential equation (PDE) solver. This flexibility comes from the carefully layered architecture of the library.

True portability. Because TensorFlow supports CPUs or GPUs and desktop, server, container, and mobile computing platforms, you have a lot of options about where you run it. There are limits, however. Some of the training tasks you might want to perform with TensorFlow require significant hardware performance if you want them to converge in a reasonable amount of time, such as a multi-CPU machine with CUDA GPUs or a Google Tensor Processing Unit (TPU), which is a custom ASIC. (No, mere mortals can't use TPUs -- yet. But Google is exploring what would make sense.)

For example, a relatively simple convolutional model for classifying handwritten digits from the MNIST data set takes half an hour to train on a single 2.6GHz Intel Core i7. If you train the same model with one or more recent Nvidia GPUs and the latest CUDA SDKs and deep neural network libraries, it will go much faster -- how much faster depends on the number and level of the GPUs and the efficiency of the libraries. If you train the same model on a TPU, it will take a few minutes. By the same token, it is reasonable to run *predictions* from a TensorFlow model on an Android or iOS device or a Raspberry Pi, but not to *train* the model on a mobile device.

Connect research and production. This point flows from portability, but it's an improvement on previous practice. Not long ago, researchers designed machine learning models in programs intended for individual use (Matlab and Mathematica come immediately to mind, but there are many), then the models had to be rewritten, trained, and deployed using scalable technologies (for example, Python and C++ programs). By contrast, TensorFlow supports researchers developing algorithms, as well as product teams training models and deploying them for customers at scale, all with the same code.

Autodifferentiation. Many training algorithms rely on gradient descent in one form or another, often stochastic gradient descent (SGD). A gradient is a vector of partial derivatives, so before you can implement a gradient-based algorithm, you need a way to differentiate one variable with respect to all the other variables of interest. TensorFlow does that for you, letting you move on to more interesting problems. The derivative computation extends your graph, and you can see that when you view your graph in TensorBoard. This capability is not unique to TensorFlow, but it's very nice to have.

Language options. TensorFlow is currently at the Model T stage of its language options: You can work in any language you want, as long as it's Python. OK, that's an exaggeration. There's a documented limited subset of the API that can be used from C++ and an undocumented Go language directory in the repository that appears to implement a Go API on top of the C++ API.

Maximize performance. Google says, "TensorFlow allows you to make the most of your available hardware. Freely assign compute elements of your TensorFlow graph to different devices, and let TensorFlow handle the copies." I haven't done this yet myself; it's covered in the Distributed TensorFlow and Using GPUs tutorials. Unless you have your own datacenter, you might want to consider deploying TensorFlow servers and clusters in one of the public clouds.

Installing TensorFlow

I installed TensorFlow and all its dependencies on a MacBook Pro running OS X El Capitan, using the Python 2.7 `pip` and `easy_install` utilities, as well as the Homebrew installation manager. I also checked out a few TensorFlow source code repositories from GitHub. Though my MacBook has a supported Nvidia GPU, I ran into trouble trying to get the GPU version of TensorFlow to work and wound up installing the slower CPU version. Because of conflicts in dependencies, it took me about seven tries; I eventually used the `--ignore-installed` flag to get `pip` to complete the process. (My woes related to an incompatibility between the Xcode 8.0 beta I was running and the CUDA compiler. Your sailing will likely be smoother.)

I installed Jupyter Notebooks as well. Jupyter is not a dependency of TensorFlow but is used by several of the TensorFlow samples. I ran into errors doing that with `pip`, but was able to succeed using `easy_install`.

I suspect that I'll be able to upgrade to the GPU version in the future once I (and the three vendors) resolve version conflicts among Xcode, the CUDA SDK and CUDA neural network libraries, and TensorFlow.

There are four other methods to install TensorFlow: Virtualenv, Anaconda, Docker, and from source code. Virtualenv gives you an isolated Python environment. Once you are at an active Virtualenv command line, you can install TensorFlow with `pip`. That extra step might be worth the effort if you have version conflicts between TensorFlow dependencies and dependencies of other Python programs you have installed.

Anaconda is a large Python distribution with lots of numerical and scientific packages and its own environment system. Once you have Anaconda installed, you can either install TensorFlow with `pip` or Conda. Again, the extra step might be worth the effort -- and if you do it, you should install the latest Python 3 versions of Anaconda and TensorFlow.

Installing TensorFlow in Docker is simple and quick, and you get an isolated, virtualized environment. You also get a Jupyter Notebook server that you can access from a browser, along with a few preconfigured notebooks. The disadvantage of using Docker on your own computer is that it will only get part of your CPU and RAM, which means that iterating a neural network model or something else complex can take a *long* time. On the other hand, you can run a Docker image in the cloud and get as much in the way of computing resources as you are willing to rent.

Building TensorFlow from sources creates a current `pip` *wheel*; you then install *that* instead of pulling a prebuilt wheel downloaded from Google Cloud Storage. Using the most current source code from the repository may fix some bugs for you; it may also introduce new bugs. Caveat coder.

Running TensorFlow

Assuming that you are using Python from the command line, a very basic interactive TensorFlow session looks like this:

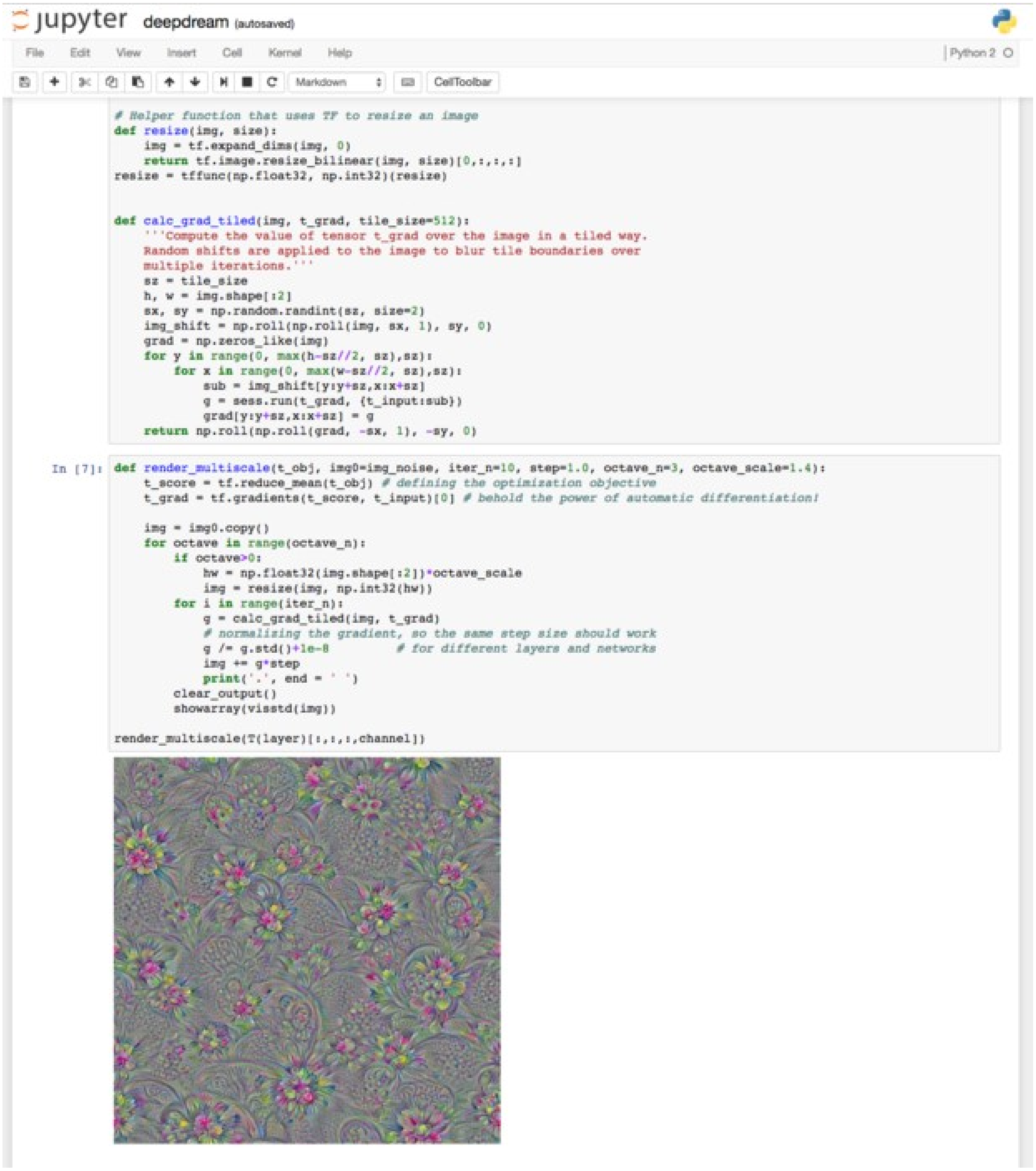
```
$ python
...
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
Hello, TensorFlow!
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> print(sess.run(a + b))
42
>>> exit()
```

To run a command-line sample, you might do the following:

```
$ python -m tensorflow.models.image.mnist.convolutional
```

The `-m` switch in the line above means that Python should load and run its argument as a library module script. You could also navigate to the correct directory and omit the switch. By the way, this script is the one that took about a half-hour to complete on my MacBook Pro (CPU-only).

If you prefer to run a Jupyter notebook, navigate to the directory where your notebook resides, and from the command line type `jupyter notebook`. You should see a directory come up in your browser, from which you can pick an existing notebook, such as the DeepDream notebook in the figure below, or you can create a new notebook for your own work.



```
# Helper function that uses TF to resize an image
def resize(img, size):
    img = tf.expand_dims(img, 0)
    return tf.image.resize_bilinear(img, size)[0,:,:,:]
resize = tffunc(np.float32, np.int32)(resize)


def calc_grad_tiled(img, t_grad, tile_size=512):
    '''Compute the value of tensor t_grad over the image in a tiled way.
    Random shifts are applied to the image to blur tile boundaries over
    multiple iterations.'''
    sz = tile_size
    h, w = img.shape[:2]
    sx, sy = np.random.randint(sz, size=2)
    img_shift = np.roll(np.roll(img, sx, 1), sy, 0)
    grad = np.zeros_like(img)
    for y in range(0, max(h-sz//2, sz),sz):
        for x in range(0, max(w-sz//2, sz),sz):
            sub = img_shift[y:y+sz,x:x+sz]
            g = sess.run(t_grad, {t_input:sub})
            grad[y:y+sz,x:x+sz] = g
    return np.roll(np.roll(grad, -sx, 1), -sy, 0)
```

```
In [7]: def render_multiscale(t_obj, img0=img_noise, iter_n=10, step=1.0, octave_n=3, octave_scale=1.4):
            t_score = tf.reduce_mean(t_obj) # defining the optimization objective
            t_grad = tf.gradients(t_score, t_input)[0] # behold the power of automatic differentiation!

            img = img0.copy()
            for octave in range(octave_n):
                if octave>0:
                    hw = np.float32(img.shape[:2])*octave_scale
                    img = resize(img, np.int32(hw))
                for i in range(iter_n):
                    g = calc_grad_tiled(img, t_grad)
                    # normalizing the gradient, so the same step size should work
                    g /= g.std()+1e-8         # for different layers and networks
                    img += g*step
                    print('.', end = ' ')
                clear_output()
                showarray(visstd(img))

        render_multiscale(T(layer)[:,:,:,channel])
```

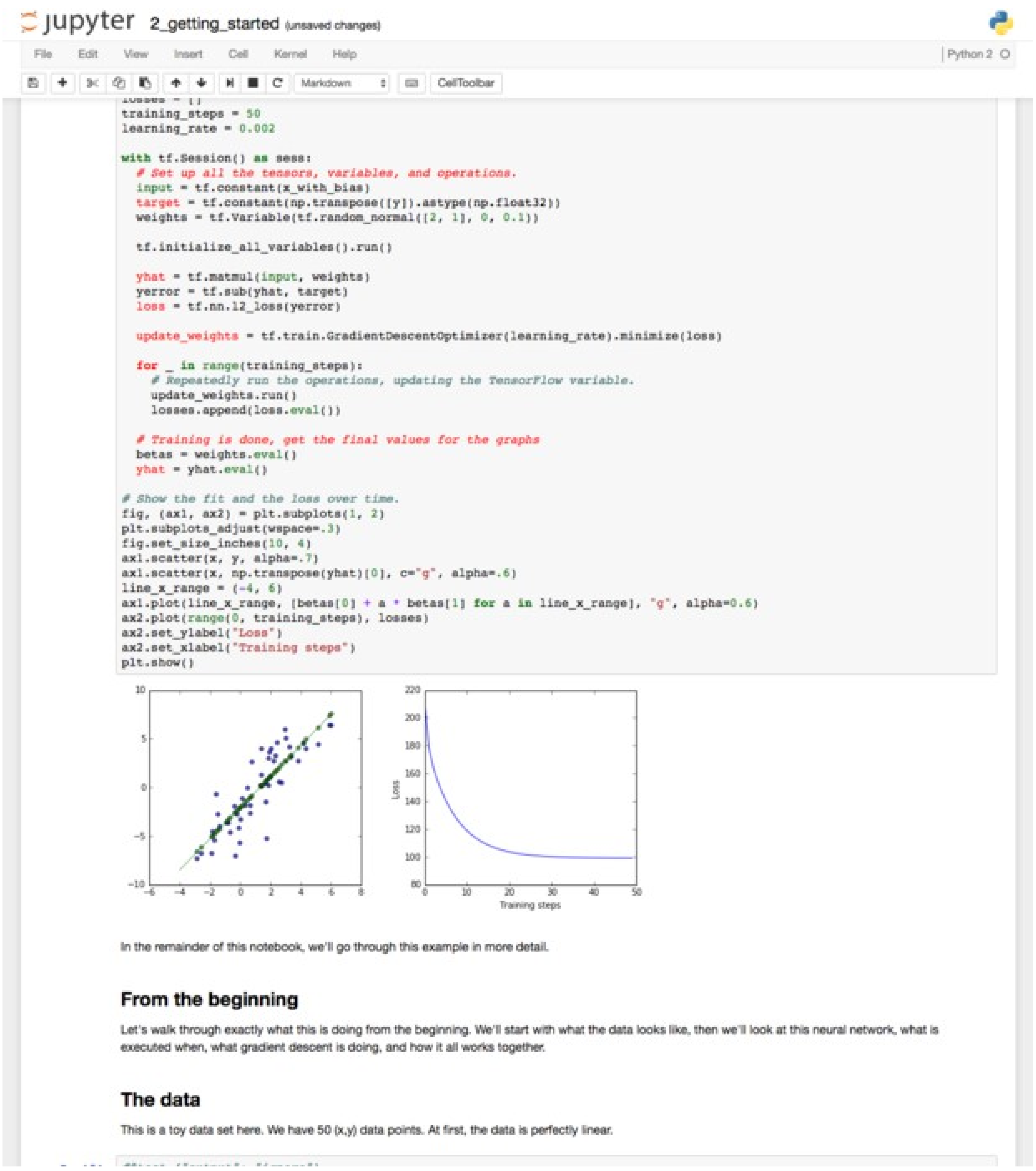
Jupyter Notebook displaying a TensorFlow DeepDream example. Functions with `tf` prefixes belong to TensorFlow, as do functions with `sess` prefixes. Functions with `np` prefixes belong to NumPy.

You can also get a Jupyter notebook by running a TensorFlow Docker image. The command line is as follows:

```
docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow
```

The `-p` switch exports the first port listed from Docker to the second port listed on the host. Then you can open the notebooks that reside inside the Docker image from a browser tab on your host by browsing to localhost:8888 if the image is running locally, or port 8888 of the correct IP or DNS address if it is running remotely.

In a Jupyter notebook or other interactive session, you may use the `tf.InteractiveSession()` constructor instead of `tf.Session()`. The subsequent code is a little less verbose.



```
losses = []
training_steps = 50
learning_rate = 0.002

with tf.Session() as sess:
  # Set up all the tensors, variables, and operations.
  input = tf.constant(x_with_bias)
  target = tf.constant(np.transpose([y]).astype(np.float32))
  weights = tf.Variable(tf.random_normal([2, 1], 0, 0.1))

  tf.initialize_all_variables().run()

  yhat = tf.matmul(input, weights)
  yerror = tf.sub(yhat, target)
  loss = tf.nn.l2_loss(yerror)

  update_weights = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

  for _ in range(training_steps):
    # Repeatedly run the operations, updating the TensorFlow variable.
    update_weights.run()
    losses.append(loss.eval())

  # Training is done, get the final values for the graphs
  betas = weights.eval()
  yhat = yhat.eval()

# Show the fit and the loss over time.
fig, (ax1, ax2) = plt.subplots(1, 2)
plt.subplots_adjust(wspace=.3)
fig.set_size_inches(10, 4)
ax1.scatter(x, y, alpha=.7)
ax1.scatter(x, np.transpose(yhat)[0], c="g", alpha=.6)
line_x_range = (-4, 6)
ax1.plot(line_x_range, [betas[0] + a * betas[1] for a in line_x_range], "g", alpha=0.6)
ax2.plot(range(0, training_steps), losses)
ax2.set_ylabel("Loss")
ax2.set_xlabel("Training steps")
plt.show()
```

In the remainder of this notebook, we'll go through this example in more detail.

From the beginning

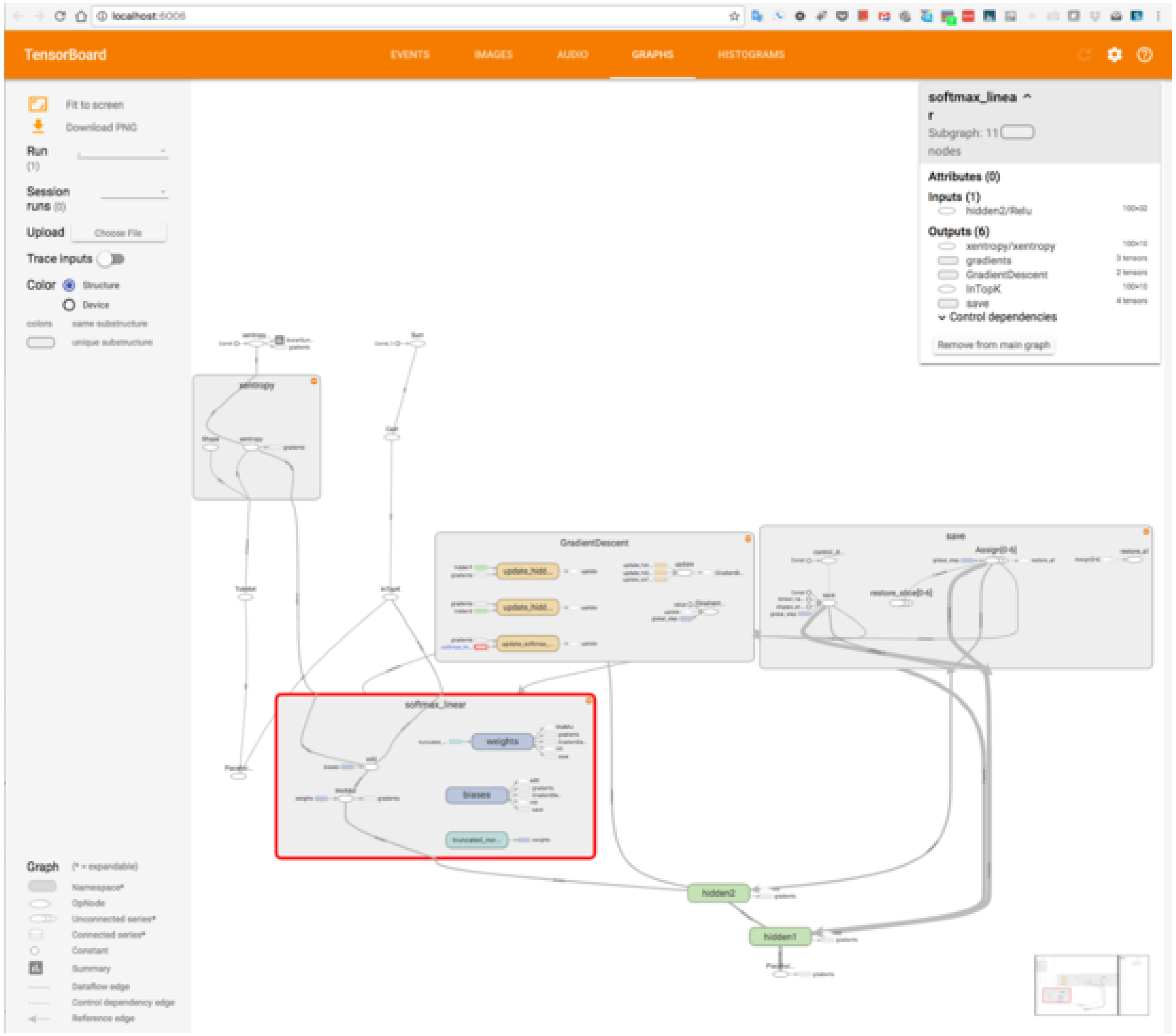
Let's walk through exactly what this is doing from the beginning. We'll start with what the data looks like, then we'll look at this neural network, what is executed when, what gradient descent is doing, and how it all works together.

The data

This is a toy data set here. We have 50 (x,y) data points. At first, the data is perfectly linear.

This Jupyter notebook is one of the three supplied in the TensorFlow Docker image. This code is doing a basic gradient descent optimization of a neural network learning from a small dataset.

Besides the library and samples, TensorFlow includes a suite of visualization tools called [TensorBoard](), as shown in the figure above. You can use TensorBoard to visualize your TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it. I've only recently begun with TensorBoard, but it looks to be a nice tool once your TensorFlow sessions are configured to log the necessary information. You can [try it out online]() even without having TensorFlow installed.



TensorBoard displaying the graph of a TensorFlow calculation. We have zoomed in on several sections to examine the details of the graph.

TensorFlow demos, tutorials, and models

There’s so much to learn about TensorFlow that, for the purposes of this article, all I can do is briefly discuss the materials that Google presents and outline a path through them. By the way, Google is actively working on adding more tutorial material and structuring it better.

You can get a feel for neural networks on the web, without installing anything, at the [TensorFlow Playground](). This example doesn’t use TensorFlow, however. Consider it to be background to build your intuition about neural networks.

Most developers should start learning TensorFlow by checking out its code repository and model repository. The next step is to install TensorFlow and validate your installation, while reading the introductory materials and going through at least one MNIST tutorial, which shows you ways to recognize handwritten characters. Then by all means go through the other tutorials, which show you the basic mechanics of TensorFlow and `tf.contrib.learn`, a high-level API for TensorFlow. Then, depending on your interests, you can dive into image processing, language and sequence processing, and non-machine-learning applications (Mandelbrot set and a PDE simulation).

Then you can work through all the how-to articles, which will get you to the point where you should be able to understand the contributed models in the "model zoo" and the TensorFlow articles in the Google Research blog. Finally, you can start writing your own TensorFlow Python code, using the API for reference, and copying code freely from the various samples and models. You might find that working interactively in Jupyter notebooks helps, or you might prefer working in a programming editor or IDE with good Python language support.

If at any time you discover that you don't quite have the background for the TensorFlow material, there are a number of free online courses on machine learning and related topics, five of which are discussed in an article by my colleague Serdar Yegulalp.

After living and breathing TensorFlow for about a week, I'm *almost* ready to write my own TensorFlow code -- although I'll probably adapt someone else's model, rather than start from scratch. TensorFlow is not an easy nut to crack, requiring knowledge of statistics, optimization, machine learning, and neural networks, as well as fluency in Python, all before you even start learning the framework. The architecture of TensorFlow is quite flexible once you grasp it, though I freely admit that I didn't quite get it for the first couple of days.

I've said elsewhere that TensorFlow ranks about 9 out of 10 on the software geekiness scale. I still think that's true, but that doesn't mean it's out of reach. You don't *need* to be a grad student in machine learning to learn and use TensorFlow, although I'm sure that would help. There's plenty of documentation, and it's pretty good, if perhaps more than you'll want to go through right away, and arranged in a manner that requires you to jump back and forth among different articles.

I rate TensorFlow excellent for its wide selection of algorithms and models. If you *are* a machine learning researcher, you'll appreciate the smorgasbord. If you simply want to solve a practical machine learning problem, you should be able to find at least one model you can use or adapt.

TensorFlow seems to perform as well as anything out there for neural network and deep learning training, despite an early benchmark that falsely indicated otherwise because of differing GPU libraries. If you need to do a lot of machine learning on large data sets, however, you'll want to use computing resources with GPUs or TPUs, and perhaps a server or cluster.

If you're serious about doing machine learning programming and you like to write Python, TensorFlow is a very good choice, although it has several competitors, including Caffe, CNTK, Theano, and Torch. On the other hand, if you simply need to process general text, speech, or images, or to perform language translations, the Google, HPE, and Microsoft clouds all offer applied machine learning services that may already be trained to do what you need.

InfoWorld Scorecard	**Models and algorithms (25%)**	**Ease of development (25%)**	**Documentation (20%)**	**Performance (20%)**	**Ease of deployment (10%)**	**Overall Score (100%)**
TensorFlow	9	8	9	10	8	8.9

r0.10
At a Glance

- **TensorFlow r0.10**

  InfoWorld Rating

  TensorFlow is a flexible and scalable open source framework for machine learning and neural networks.

  Pros

  - Wide variety of models and algorithms
  - Excellent performance on hardware with GPUs or TPUs
  - Excellent support for Python
  - Good documentation
  - Good software for displaying graphs

  Cons

  - Difficult to learn
  - Bare-bones support for C++