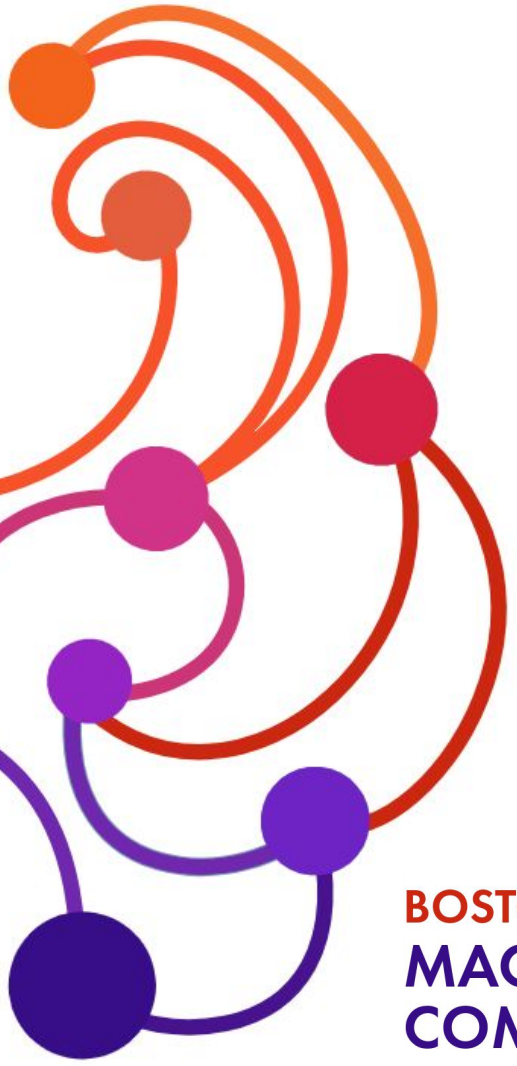


Getting Started with TensorFlow™

BOSTON UNIVERSITY
MACHINE INTELLIGENCE COMMUNITY

Rachel Manzelli
2/21/18

https://goo.gl/8n4GbX

What is TensorFlow?

- Python framework focusing on data flow graphs, used for machine learning applications
- Builds static computation graphs
  - Supports graph-structured/tree-structured data
- Schedules operations automatically when run
- We will be using TensorFlow Core (higher level APIs are also available, i.e. tf.contrib.learn)

Why TensorFlow?

What You Need for Today

- ➔ A computer
- ➔ An installation of Python 2 or 3
- ➔ A text editor & means of compiling code

Installing TensorFlow

If you have Python 2 or 3 installed, you can install TensorFlow via the command

```
>> pip install tensorflow
```

```
>> pip3 install tensorflow
```

in the command prompt.

You can also install via `virtualenv`.

(If these options cause issues, there is always the option of building from source...)

Start programming in TensorFlow

Start programming in your favorite editor or IDE (Sublime, XCode, …).
TensorFlow is an environment in **Python**, so we will be using Python syntax.

To use TensorFlow, you will need to include

```
import tensorflow as tf
```

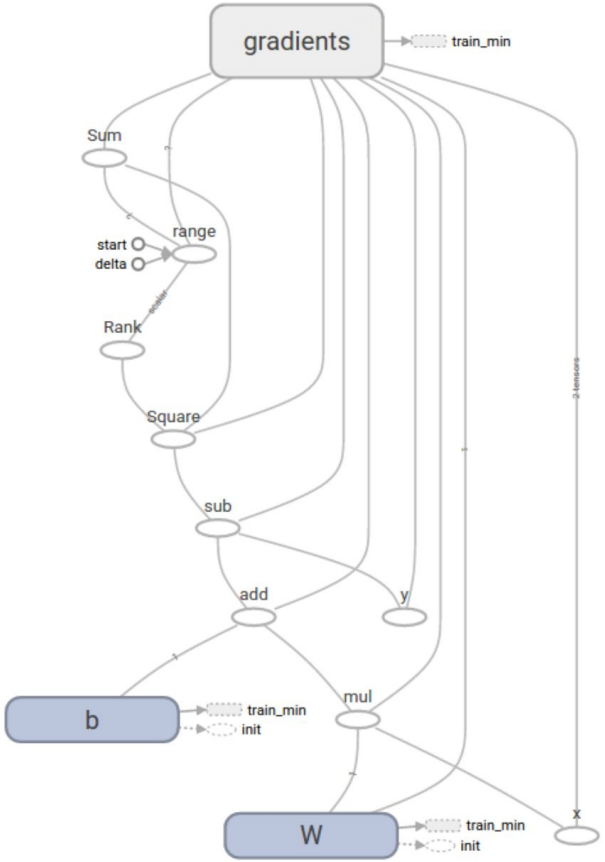
at the top of your .py file.

But before we jump into that…

TensorFlow Concepts

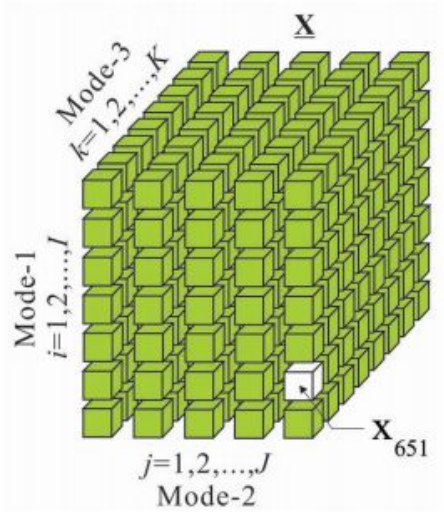
- **Computational graph**: a series of operations arranged into a graph of nodes
  - **nodes**: take tensors as inputs and outputs a tensor
  - **session** objects: tf.Session.run()
- We create graphs in 2 steps:
  1. **Build (initialize)**
  2. **Run (evaluate)**
- (Image taken from TensorBoard)

```
>> tensorboard --logdir=/your_log_dir
```



TensorFlow Concepts

- Tensors: representations of high-dimensional arrays
  - **rank:** number of dimensions of a tensor



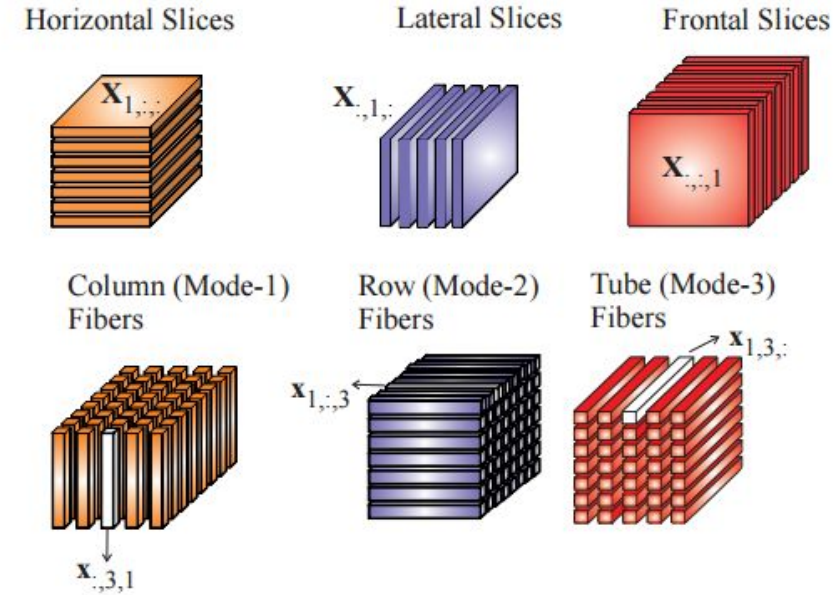


Figure 2: A 3rd-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$, with entries $x_{i,j,k} =$

How does TensorFlow manipulate data?

- ➔ TensorFlow < 1.4
  - ◆ **Placeholders (tf.placeholder)**: promise to provide a value later, but are initialized for the purpose of creating a graph
  - ◆ **Feed dictionaries (feed_dict)**: allow for the node to be evaluated with multiple values
  - ◆ **Variables (tf.Variable)**: trainable parameters, i.e. take in arbitrary values
- ➔ TensorFlow 1.4 and 1.5
  - ◆ Estimators and Datasets APIs

Let's build a graph with TensorFlow!

- ➔ Open your editor again :)
- ➔ Import TensorFlow and initiate a session (using `tf.Session`)
- ➔ Create a simple adder and multiplier by creating two nodes that will hold numbers, and nodes to store their output:

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)
add_and_triple = adder_node * 3.
```

MIC

Let's build a graph with TensorFlow!

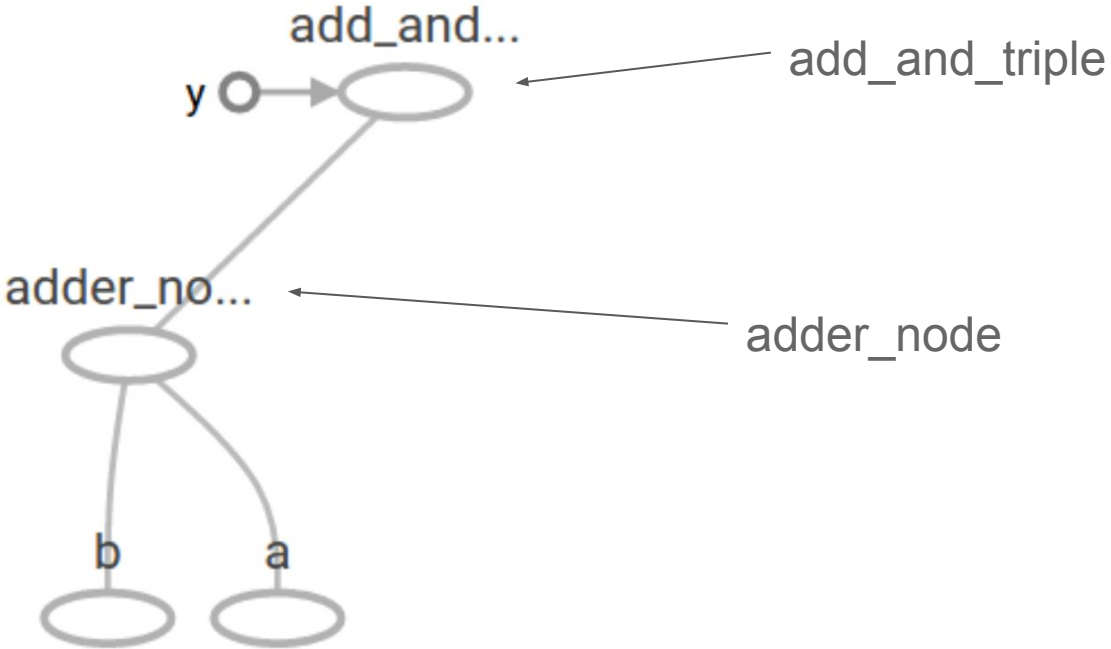
- ➔ Let's compute some outputs using tf.sess.run().
- ➔ Evaluate these operations for multiple values using a feed dictionary:

```
print(sess.run(adder_node, {a: 3, b:4.5}))
print(sess.run(adder_node, {a: [1,3], b: [2, 4]}))
print(sess.run(add_and_triple, {a: 3, b:4.5}))
```

```
[7.5]
[3. 7.]
22.5
```

MIC

What does our graph look like?

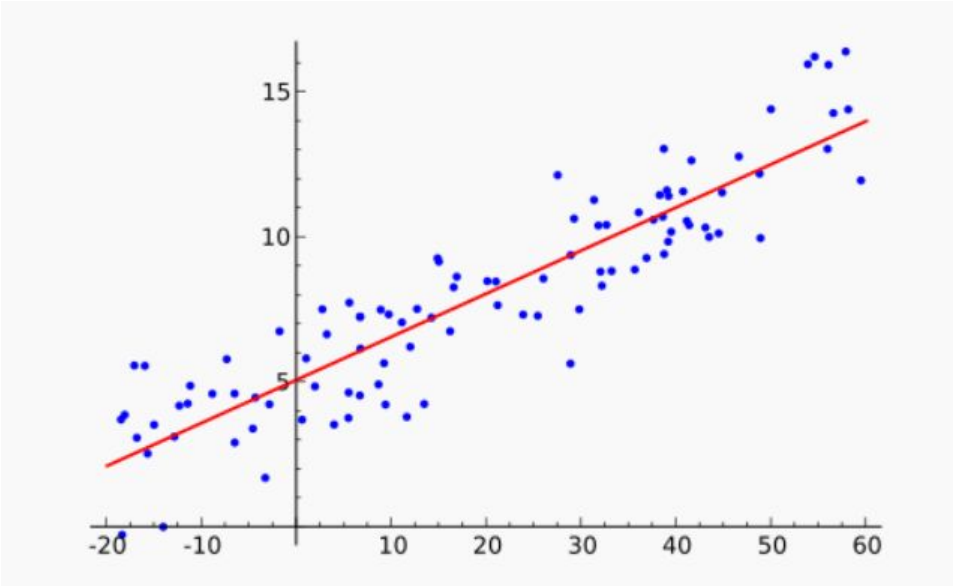


Building a Trainable Graph

- ➔ Most applications in machine learning involve training, and updating parameters, which tf.constant cannot do. The values of constants in TensorFlow cannot change while the graph is evaluated.
- ➔ **tf.Variable** represents a trainable parameter in the graph (no concrete value)

Building a Linear Model

- Let's start by building a very simple linear model.
- These models are based on a statistical process called **linear regression**: assuming the relationship between the data and parameters fits linearly.



MIC

Building a Linear Model

- ➔ The only thing we know in this case is x. (We want to estimate y, but don't know the fit, which is represented by W and b).
- ➔ So, we can create trainable parameters for the other parameters using **tf.Variable**:

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b
```

MIC

Building a Linear Model

- When variables are created in TensorFlow, they require a special initializer.
- It is important to realize that we are not running the model (there are no parameters yet!) We just initialize the variables like this:

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

- Feed values into our model like before (x is the placeholder):

```
print(sess.run(linear_model, {x:[1,2,3,4]}))
```

MIC

Evaluation of the Model

```
[0.          0.3         0.6         0.90000004]
```

- ➔ What does this mean? How can we evaluate the performance of this model?
- ➔ A common **loss function** for linear regression is **least squares**, which looks like this:

$$\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.$$

… which squares the distance of the prediction from the ground truth value, and sums them all. We aim to minimize this sum (as close to zero as possible).

MIC

Translating Loss to TensorFlow

➔ To evaluate the loss, we can translate our cost function to TensorFlow as follows (with y as a placeholder for the ground truth values):

```
y = tf.placeholder(tf.float32)
squared_deltas = tf.square(linear_model - y)
loss = tf.reduce_sum(squared_deltas)
print(sess.run(loss, {x:[1,2,3,4], y:[0,-1,-2,-3]}))
```

Ground truth values of y for this linear model

➔ Our loss ends up being `23.66`

MIC

Improving the Model

➔ Manually assign the tensors W and b to -1 and 1, respectively.

```
W = tf.Variable([-1], dtype=tf.float32)
b = tf.Variable([1], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b
```

➔ These are the “perfect” values of our trainable parameters: the model will always predict y perfectly (with 0.0 loss) with these parameters (try it!)

➔ https://github.com/pkmital/tensorflow_tutorials/blob/master/python/02_linear_regression.py

Improving the Model

- ➔ Here, we know the values of W and b that will minimize the loss.
- ➔ The point of machine learning is for the model to figure out the perfect parameters through training and minimizing the cost function!

A preview into how awesome TF is!

- Into your command shell of choice, clone TensorFlow's GitHub repository called "models":

```
>> git clone https://github.com/tensorflow/models
```

(This will take a few minutes - there's a lot in there to play around with!)

- For now, change into the getting_started directory:

```
>> cd models/samples/core/get_started
```

Install dependencies

➔ To run this example network, we need a few things:

`>> pip install pandas:` Data structure framework.

`>> pip install keras:` A TensorFlow wrapper.

```
>> pip install tensorflow --upgrade
```

Support for Keras was added in TensorFlow 1.4, so we need to make sure TensorFlow is updated.

Before we run anything...

- ➔ What does this network do? Image classification!
  - Trains on images
  - Tests on images it hasn't seen before, and tries to classify the image as something it has learned
  - **Supervised learning problem**

What does this network do?

- Iris setosa
- Iris virginica
- Iris versicolor



From left to right, *Iris setosa* (by Radomil, CC BY-SA 3.0), *Iris versicolor* (by Dlanglois, CC BY-SA 3.0), and *Iris virginica* (by Frank Mayfield, CC BY-SA 2.0).

MIC

Training and testing a network

- Into the command window, run the Python file `premade_estimator.py` in the current directory.

```
>> python premade_estimator.py
```

- This will train the network on the images for 1,000 iterations
- Then, it will test the network on test images
- Outputs loss and test accuracy by checking the predicted labels against the ground truth labels

```
INFO:tensorflow:Saving checkpoints for 1 into C:\Users\raman\AppData\Local\Temp\tmphrdx3y6c\model.ckpt.
INFO:tensorflow:loss = 279.39398, step = 1
INFO:tensorflow:global_step/sec: 852.418
INFO:tensorflow:loss = 16.906694, step = 101 (0.118 sec)
INFO:tensorflow:global_step/sec: 1061.02
INFO:tensorflow:loss = 10.27897, step = 201 (0.094 sec)
INFO:tensorflow:global_step/sec: 1108.16
INFO:tensorflow:loss = 9.171523, step = 301 (0.090 sec)
INFO:tensorflow:global_step/sec: 1108.16
INFO:tensorflow:loss = 7.0105577, step = 401 (0.090 sec)
INFO:tensorflow:global_step/sec: 1129.62
INFO:tensorflow:loss = 7.708337, step = 501 (0.089 sec)
INFO:tensorflow:global_step/sec: 1095.99
INFO:tensorflow:loss = 5.770026, step = 601 (0.091 sec)
INFO:tensorflow:global_step/sec: 1095.97
INFO:tensorflow:loss = 5.647806, step = 701 (0.091 sec)
INFO:tensorflow:global_step/sec: 1108.17
INFO:tensorflow:loss = 5.294636, step = 801 (0.090 sec)
INFO:tensorflow:global_step/sec: 1159.7
INFO:tensorflow:loss = 5.9896655, step = 901 (0.086 sec)
INFO:tensorflow:Saving checkpoints for 1000 into C:\Users\raman\AppData\Local\Temp\tmphrdx3y6c\model.ckpt.
INFO:tensorflow:Loss for final step: 3.7188513.
INFO:tensorflow:Starting evaluation at 2018-02-21-12:57:43
INFO:tensorflow:Restoring parameters from C:\Users\raman\AppData\Local\Temp\tmphrdx3y6c\model.ckpt-1000
INFO:tensorflow:Finished evaluation at 2018-02-21-12:57:43
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.96666664, average_loss = 0.058762528, global_step = 1000, loss = 1.7628758

Test set accuracy: 0.967

INFO:tensorflow:Restoring parameters from C:\Users\raman\AppData\Local\Temp\tmphrdx3y6c\model.ckpt-1000

Prediction is "Setosa" (99.6%), expected "Setosa"

Prediction is "Versicolor" (99.5%), expected "Versicolor"

Prediction is "Virginica" (97.3%), expected "Virginica"
PS C:\Users\raman\Documents\BUMIC\models\samples\core\get_started>
```

Next time

- ➔ **More concepts: Data and models!**
  - How do we translate data into TensorFlow structures we saw today?
  - What is a model? What types of models can we run with TensorFlow?
- ➔ **More exercises/projects**
  - Leave feedback and suggestions for what to do next time!

MIC

Upcoming Events with BUMIC

- ➔ Impact of AI discussion group: 2/27/18
- ➔ Deep learning security lecture: 2/25/18
- ➔ Next TensorFlow session: 3/7/18 @ 7pm

MIC

Thank you to our sponsors!