TensorFlow

Introduction and Python Tutorial

Outlines

- Introduction To TensorFlow
- Installing TensorFlow
- TensorFlow Basic Operations
- Basic Training with TensorFlow

Part One

TensorFlow Basic Usage

What is TensorFlow?

- TensorFlow is a deep learning library was open-sourced by Google in Nov 2015.
- It is a multipurpose open source library for numerical computation using data flow graphs (networks).

What is TensorFlow?

- It has been designed with deep learning in mind it is written in C++ but can be used from many other programming language.
- It runs on many platforms (not windows friendly)

What is TensorFlow?

- The core of TensorFlow is the dataflow graph representing computations.
- The entire dataflow graph is a complete description of computations, and are executed on *devices* (CPUs or GPUs).

What is TensorFlow?

- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes.
- TensorFlow provides primitives for defining functions on tensors and automatically computing their derivatives.

What is a Tensor

- A mathematical object analogous to but more general than a vector, represented by an array of components that are functions of the coordinates of a space
- A scalar is a tensor, a vector is a tensor, a matrix is a tensor.
- Simply a multidimensional array of numbers

TensorFlow vs Numpy

- TensorFlow and Numpy are very similar. They are N-d array libraries!
- Numpy has Ndarray support, but does not offer methods to create tensor functions and automatically compute derivatives.
- Numpy does not have no GPU support.

TensorFlow vs Numpy

- TensorFlow and Numpy are very similar. They are N-d array libraries!
- Numpy has Ndarray support, but does not offer methods to create tensor functions and automatically compute derivatives.
- Numpy does not have no GPU support.

TensorFlow vs Theano

- The Grand-daddy of deep-learning frameworks, which is written in Python
- Numerous open-source deep-libraries have been built on top of Theano, including Keras, Lasagne and Blocks. These libs attempt to layer an easier to use API on top of Theano's occasionally non-intuitive interface.

TensorFlow vs Theano

- Theano is an open source project primarily developed by a machine learning group at the Université de Montréal.
- Tensorflow was inspired by Theano. Tensorflow has better support for distributed systems and more developer friendly.

Installing TesnsorFlow

- You must choose one of the following types of TensorFlow to install:
  - **TensorFlow with CPU support only**.
  - **TensorFlow with GPU support.**
- Platform options are Ububtu, Mac OS X, and Windows

Installing TesnsorFlow

- On Ubuntu and Mac OS X, you have the following options:
  - Virtualenv
  - Native using pip
  - Docker
  - Anaconda
- On Windows you have native "using pip: or Anaconda. For other languages (Java, C++ and Go) check documentations

Installing TensorFlow

- On Ubuntu 16.04 LTS using the virtualenv option
- Virtualenv is a virtual Python environment isolated from other Python development, incapable of interfering with or being affected by other Python programs on the same machine

Installing TensorFlow

- Installing pip, python dev and virtualenv

```
sudo apt-get install python3-pip python3-dev python-virtualenv
```

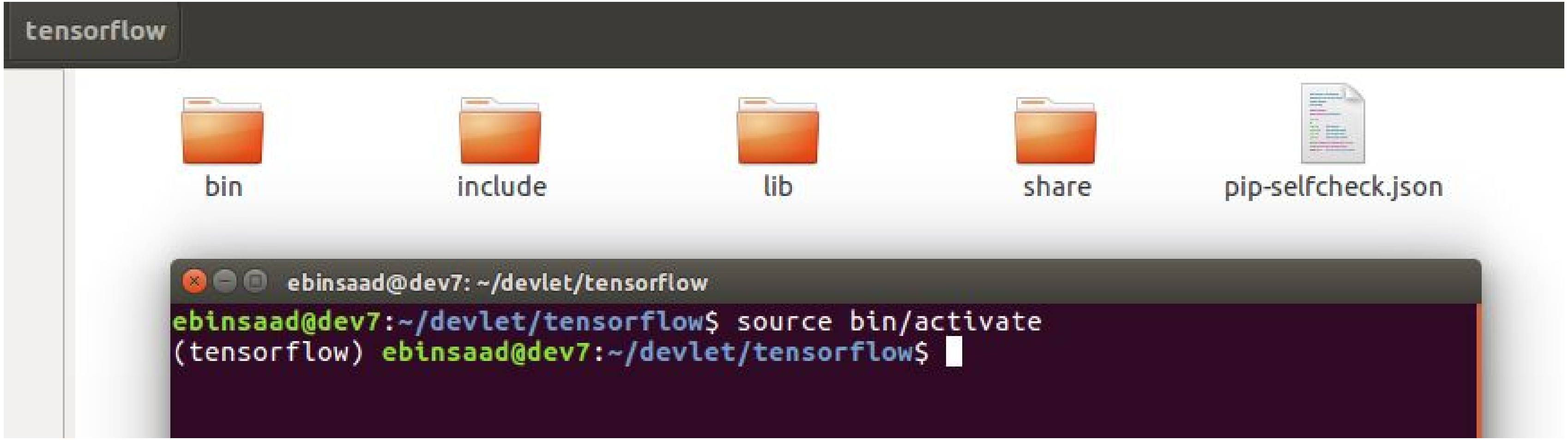
- Create a virtualenv environment

```
virtualenv --system-site-packages -p python3 tensorflow
```

- Activate the virtualenv

```
source ~/tensorflow/bin/activate
```

Installing TensorFlow



- The preceding source command should change your prompt to **(tensorflow)$**
- Note that you must activate the virtualenv environment each time you use TensorFlow.

Installing TensorFlow

- Install TensorFlow with pip

```
pip3 install --upgrade tensorflow
```

- Run the following code to test your installation

```
import tensorflow as tf

hello = tf.constant('Hello, TensorFlow!')

sess = tf.Session()

print(sess.run(hello))
```

Installing TensorFlow

- You should see the following if your installation was successful

```
Time  Line #  Log Message
3.6s       0  2017-07-24 22:03:46.086291: W tensorflow/core/platform/cpu_feature_guard.cc:45] The
              TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on
              your machine and could speed up CPU computations.
              2017-07-24 22:03:46.086339: 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.
              2017-07-24 22:03:46.086350: 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.
              2017-07-24 22:03:46.086366: 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.
              2017-07-24 22:03:46.086375: 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.
3.7s       1  b'Hello, TensorFlow!'
```

Tensors

- The central unit of data in TensorFlow is the **tensor**.
- A tensor consists of a set of primitive values shaped into an array of any number of dimensions. A tensor's **rank** is its number of dimensions

```
3 'a rank 0 tensor; this is a scalar with shape []'

[1. ,2., 3.] 'a rank 1 tensor; this is a vector with shape [3]'

[[1., 2., 3.], [4., 5., 6.]] 'a rank 2 tensor; a matrix with shape [2, 3]'

[[[1., 2., 3.]], [[7., 8., 9.]]] 'a rank 3 tensor with shape [2, 1, 3]'
```

Examples

Using TensorFlow

- Think of TensorFlow Core programs as consisting of :
  1. Building the computational graph.
  2. Running the computational graph.

Using TensorFlow

- What is a computational graph?
    - A series of TensorFlow operations arranged into a graph of nodes.
    - Each node takes zero or more tensors as inputs and produces a tensor as an output. One type of node is a constant. Like all TensorFlow constants, it takes no inputs, and it outputs a value it stores internally.

Using TensorFlow

```python

import tensorflow as tf

# create two floating point Tensors node1 and node2

node1 = tf.constant(3.0, dtype=tf.float32)

node2 = tf.constant(4.0) # also tf.float32 implicitly

print(node1, node2)
```

```
Time  Line #  Log Message
2.5s       0  Tensor("Const:0", shape=(), dtype=float32)
              Tensor("Const_1:0", shape=(), dtype=float32)
```

Using TensorFlow

The previous code only prints the node object not the values. To actually evaluate the nodes, we must run the computational graph within a **session**

```
# Create a session object to execute the  computational graph
sess = tf.Session()

print(sess.run([node1, node2]))
```

Using TensorFlow

```python
import tensorflow as tf

# Create two floating point Tensors node1 and node2
node1 = tf.constant(3.0, dtype=tf.float32)

node2 = tf.constant(4.0) # also tf.float32 implicitly

print(node1, node2)

# Create a session object to execute the  computational graph
sess = tf.Session()

print(sess.run([node1, node2]))
```

```
Time Line #  Log Message
2.8s      0  Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
             [3.0, 4.0]
```

What is a TensorFlow Session?

- A session allows to execute graphs or part of graphs. It allocates resources (on one or more machines) for that and holds the actual values of intermediate results and variables.
- In the graph we define the structure and the operations.

What is TensorFlow Graph?

- A graph defines the computation. It doesn’t compute anything, it doesn’t hold any values, it just defines the operations that you specified in your code.
- The **TensorFlow** Python library has a **default graph** to which ops constructors add nodes

Using TensorFlow

We can build more complicated computations by combining Tensor nodes with operations (Operations are also nodes.). For example, we can add our two constant nodes and produce a new graph as follows:

```

node3 = tf.add(node1, node2)

print("node3: ", node3)
print("sess.run(node3): ",sess.run(node3))
```

Using TensorFlow

```python
import tensorflow as tf

# Create two floating point Tensors node1 and node2
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)

# Create a session object to execute the  computational graph
sess = tf.Session()
print(sess.run([node1, node2]))

node3 = tf.add(node1, node2)

print("node3: ", node3)
print("sess.run(node3): ",sess.run(node3))
```

```
3.3s        3   node3:  Tensor("Add:0", shape=(), dtype=float32)
                sess.run(node3):  7.0
```

Using TensorFlow

TensorFlow provides a utility called TensorBoard that can display a picture of the computational graph. Here is a screenshot showing how TensorBoard visualizes the graph:

How could we add input to the graph?

A graph can be parameterized to accept external inputs, known as **placeholders**. A **placeholder** is a promise to provide a value later.

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

adder_node = a + b  # + provides a shortcut for tf.add(a, b)
```

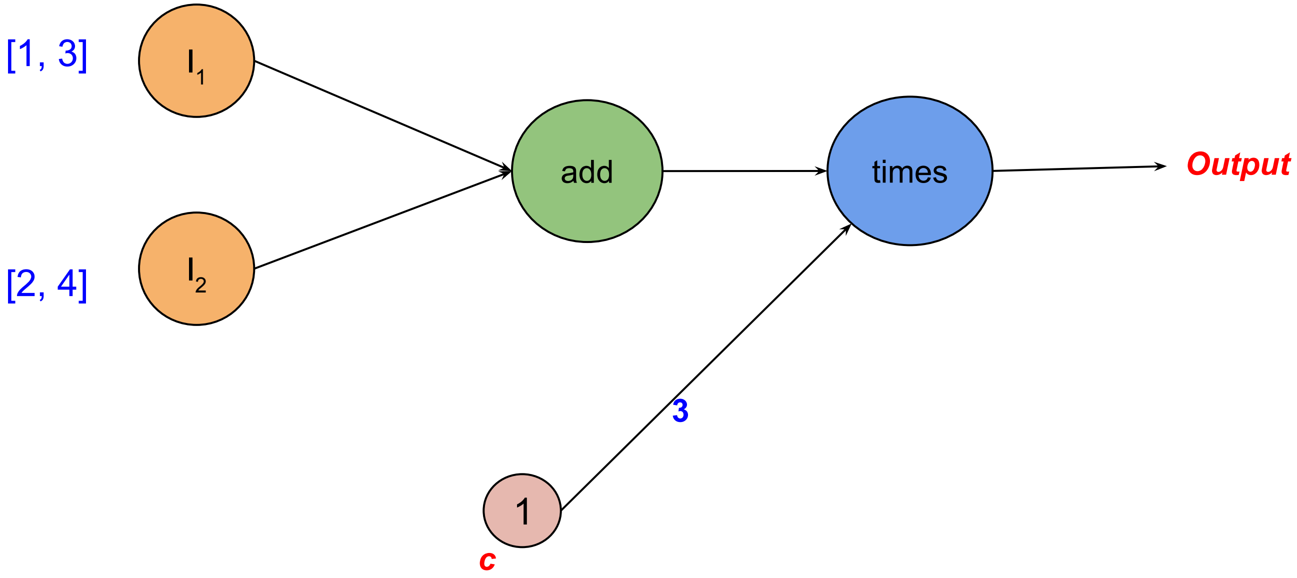
```
print(sess.run(adder_node, {a: 3, b:4.5}))
print(sess.run(adder_node, {a: [1,3], b: [2, 4]}))
```

How could we add input to the graph?

```python
import tensorflow as tf

# Create two placeholders to hold floating point Tensors

a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

adder_node = a + b  # + provides a shortcut for tf.add(a, b)

# Create a session object to execute the  computational graph
sess = tf.Session()

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

Using TensorFlow

Implement the following computational graph



Using TensorFlow

```
import tensorflow as tf

# Create two placeholders to hold floating point Tensors

a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

c = tf.constant(3.0, dtype=tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)

times_node = adder_node * c

# Create a session object to execute the  computational graph
sess = tf.Session()

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

Basic Training with TensorFlow

- In machine learning we will typically want a model that can take arbitrary inputs, such as the one above.
- To make the model trainable, we need to be able to modify the graph to get new outputs with the same input.
- **Variables** allow us to add trainable parameters to a graph. They are constructed with a type and initial value

Training a Linear Model

linear_model = W * x + b

```
W = tf.Variable([0.3], dtype=tf.float32)
b = tf.Variable([-0.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)

linear_model = W * x + b
```

Training a Linear Model

```
import tensorflow as tf

# linear_model = W * x + b

W = tf.Variable([0.3], dtype=tf.float32)
b = tf.Variable([-0.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)

linear_model = W * x + b

''' To initialize all the variables in a TensorFlow program,
    you must explicitly call a special operation as follows: '''

sess = tf.Session()

init = tf.global_variables_initializer()

sess.run(init)

# apply the mode with the inputs x
print(sess.run(linear_model, {x:[1,2,3,4]}))
```

Training a Linear Model

```
W = tf.Variable([0.3], dtype=tf.float32)
b = tf.Variable([-0.3], dtype=tf.float32)

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

linear_model = W * x + b

squared_deltas = tf.square(linear_model - y)

loss = tf.reduce_sum(squared_deltas)
```

```
# apply the mode with the inputs x with target y and print the sum square error
print(sess.run(loss, {x:[1,2,3,4], y:[0,-1,-2,-3]}))
```

Training a Linear Model

```python
import tensorflow as tf

# linear_model = W * x + b

W = tf.Variable([0.3], dtype=tf.float32)
b = tf.Variable([-0.3], dtype=tf.float32)

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

linear_model = W * x + b

squared_deltas = tf.square(linear_model - y)

loss = tf.reduce_sum(squared_deltas)

''' To initialize all the variables in a TensorFlow program,
    you must explicitly call a special operation as follows: '''

sess = tf.Session()

init = tf.global_variables_initializer()

sess.run(init)

# apply the mode with the inputs x with target y and print the sum square error
print(sess.run(loss, {x:[1,2,3,4], y:[0,-1,-2,-3]}))
```

The Result is

```
23.66
```

Training a Linear Model

- We want to learn the best values for W and b to minimize the loss.
- The simplest optimizer is **gradient descent**. It modifies each variable according to the magnitude of the derivative of loss with respect to that variable
- TensorFlow can automatically produce derivatives given only a description of the model using the function `tf.gradients`

Training a Linear Model

```
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares

# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

sess.run(init) # reset values to incorrect defaults.
for i in range(1000):
  sess.run(train, {x:[1,2,3,4], y:[0,-1,-2,-3]})

print(sess.run([W, b]))
```

Training a Linear Model

```python
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
# training data
x_train = [1,2,3,4]
y_train = [0,-1,-2,-3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
  sess.run(train, {x:x_train, y:y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

Training a Linear Model

The optimal parameters of the is linear model with respect to x and y are W = -1 and b = 1

```
TensorFlow library wasn't compiled to use SSE4.2 instructions,
your machine and could speed up CPU computations.
2017-07-25 21:37:02.915159: W tensorflow/core/platform/cpu_fea
TensorFlow library wasn't compiled to use AVX instructions, bu
your machine and could speed up CPU computations.
2017-07-25 21:37:02.915174: W tensorflow/core/platform/cpu_fea
TensorFlow library wasn't compiled to use AVX2 instructions, bu
your machine and could speed up CPU computations.
2017-07-25 21:37:02.915181: W tensorflow/core/platform/cpu_fea
TensorFlow library wasn't compiled to use FMA instructions, bu
your machine and could speed up CPU computations.
4.2s        1  W: [-0.9999969] b: [ 0.99999082] loss: 5.69997e-11
```

Questions