Introduction of TensorFlow
(Basic Concept)

Ko, Youngjoong

Dept. of Computer Engineering,
Dong-A University
Contents

1. Start Up

2. Overview of TensorFlow

3. Two Computation Phrases

4. Tensors

5. Variables

6. Operations

7. References
Google Machine Learning Tools

1st Generation: DistBelief

- Dean et al. 2011
- Major Output Products
  - Inception (Image Categorization)
  - Google Search
  - Google Translate
  - Google Photos

2nd Generation: TensorFlow

- Dean et al. 2015 (November, 1st)
- Most of DistBelief users at Google have already switched to TensorFlow
Main Developers of *DistBelief* and *TensorFlow*

Jeffrey Adgate "Jeff" Dean (born 1968) is an American computer scientist and software engineer. He is currently a Google Senior Fellow in the Systems and Infrastructure Group.

- Advertising / Crawling / Indexing / Query Systems
- ...

- **BigTable** a large-scale semi-structured storage system.
- **MapReduce** a system for large-scale data processing applications.
- **Google Brain** a system for large-scale artificial neural networks
- **LevelDB** an open source on-disk key-value store.
- **TensorFlow** an open source machine learning software library.
- ...

→ Google Core

→ Hadoop

→ Large ML
## Features

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Dataflow-like model</th>
<th>A directed Computational Graph를 통해 모든 계산을 표현</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Python, C++</td>
<td>현재는 Python, C++ 만 지원되나 FrontEnd 언어를 각 개발자가 만들면 되는 형태</td>
</tr>
<tr>
<td>Deployment</td>
<td>Code once, Run everywhere</td>
<td>하나의 코드를 구현하면, 단-복수의 기계에서 똑같이 작동함</td>
</tr>
<tr>
<td>Computing Resource</td>
<td>CPU, GPU</td>
<td>CPU와 GPU를 동시에 활용가능한 형태의 계산 Infra</td>
</tr>
<tr>
<td>Distribution Process</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local Implementation, Distributed Implementation</td>
<td>Local, Distribution 2가지 모드에 대해 구현되어 있으며 옵션만 바뀌면 TensorFlow가 알아서 작동됨</td>
</tr>
<tr>
<td>Math Expressions</td>
<td>Math Graph Expression, Auto Differentiation</td>
<td>수식 및 계산 방식을 graph 형태로 표현. 따라서 자동으로 미분가능, 특히 Gradient 계산에 최적화해서 개발</td>
</tr>
<tr>
<td>Optimization</td>
<td>Auto Elimination, Kernel Optimization, Communication Optimization, Support model, data parallelism, ...</td>
<td>다양한 형태의 최적화를 TF가 알아서 진행함</td>
</tr>
</tbody>
</table>
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))  # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784, 100], -1, 1))  # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")  # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)  # Relu(Wx+b)
C = [...]  # Cost computed as a function # of Relu

\[ C = \text{ReLU}(W \cdot x + b) \]
Let's peek at what TensorFlow code looks like

The first part of this code builds the data flow graph.

```python
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)
```
Let's peek at what TensorFlow code looks like

TensorFlow does not actually run any computation until the session is created and the *run* function is called.

```python
# 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]
```

Download and set up Tensorflow

- [https://www.tensorflow.org/versions/0.6.0/get_started/os_setup.html](https://www.tensorflow.org/versions/0.6.0/get_started/os_setup.html)
Overview of TensorFlow

- 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.

- Overview of TensorFlow
  - A programming system in which you represent computations as graphs
  - Nodes in the graph
    - Operation (op): to perform some computations
    - Input: one or more tensor, Output: one or more tensor
    - Tensor: a typed multi-dimensional array
      - EX) a mini-batch of images as a 4-D array of floating,
        \[ [\text{batch, height, width, channels}] \]
To compute anything in TensorFlow

- A graph must be launched in a **Session**.
- A **Session**
  - Place the graph ops onto Devices, such as **CPUs** or **GPUs**
  - Provide methods to execute them
  - Return tensors produced by ops as **numpy ndarray objects** in Python, and as **tensorflow::Tensor** instances in C and C++.

Two Computation Phrases of a Graph

- **Construction phrase**
  - Assemble a graph
  - *ex)* create a graph to represent and train a neural network
- **Execution phrase**
  - Use a session to execute ops in the graph
  - *ex)* repeatedly execute a set of training ops in the graph
Two Computation Phrases

- **Building the Graph**
  - Start with ops that do not need any input (source ops), `Constant`
  - Pass their output to other ops that do computation
  - Ops constructors return objects
    - Stand for the output of the constructed ops
    - Pass these to other ops constructors to use as inputs

- **Default Graph**
  - Ops constructors add node to that graph

```python
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.]])
```
**Two Computation Phrases**

- **Default Graph**
  - Has three nodes: two constant() ops and one matmul() op

  ```python
  # 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)
  ```

- **Launching the graph in a session**
  - Create a **Session** Object: should be closed to release resources
  - Without arguments, session constructor launches the default graph

  ```python
  # 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. 
  result = sess.run(product) 
  print result  
  # => [[ 12.]] 

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

- **Session** launches the graph, **Session.run()** method executes operations

- **A Session with Block**
  - Close automatically at the end of the with block

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

- **GPU Usage**
  - Translate the graph definition into executable operations distributed across available compute resources, such as CPU or GPU
  - If you have GPU, TensorFlow uses your first GPU
Interactive Usage

- In Python environments, such as Ipython, the `InteractiveSession` class is used.
- `Tensor.eval()` and `Operation.run()`
- This avoids having to keep a variable holding the session

```python
# 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()
```
Tensors

- Tensor data structure to represent all data
- Only tensors are passed between operations in the computation graph
- n-dimensional array or list
  - Static type, a rank, and a shape

<table>
<thead>
<tr>
<th>Rank</th>
<th>Math entity</th>
<th>Python example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Scalar (magnitude only)</td>
<td>s = 483</td>
</tr>
<tr>
<td>1</td>
<td>Vector (magnitude and direction)</td>
<td>v = [1.1, 2.2, 3.3]</td>
</tr>
<tr>
<td>2</td>
<td>Matrix (table of numbers)</td>
<td>m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]</td>
</tr>
<tr>
<td>3</td>
<td>3-Tensor (cube of numbers)</td>
<td>t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]</td>
</tr>
<tr>
<td>n</td>
<td>n-Tensor (you get the idea)</td>
<td>....</td>
</tr>
</tbody>
</table>
Tensors

- **Tensors**
  - **Shape**
    
    | Rank | Shape         | Dimension number | Example                                           |
    |------|---------------|------------------|---------------------------------------------------|
    | 0    | []            | 0-D              | A 0-D tensor. A scalar.                           |
    | 1    | [D0]          | 1-D              | A 1-D tensor with shape [D0].                     |
    | 2    | [D0, D1]      | 2-D              | A 2-D tensor with shape [D0, D1].                 |
    | 3    | [D0, D1, D2]  | 3-D              | A 3-D tensor with shape [D0, D1, D2].             |
    | n    | [D0, D1, ..., Dn] | n-D              | A tensor with shape [D0, D1, ..., Dn].            |

- **Data Types**

    | Data type   | Python type  | Description              |
    |-------------|--------------|--------------------------|
    | DT_FLOAT    | tf.float32  | 32 bits floating point.  |
    | DT_DOUBLE   | tf.float64  | 64 bits floating point.  |
    | DT_INT64    | tf.int64    | 64 bits signed integer.  |
    | DT_INT32    | tf.int32    | 32 bits signed integer.  |
    | DT_INT16    | tf.int16    | 16 bits signed integer.  |
Tensors

Data Types

<table>
<thead>
<tr>
<th>Tensors</th>
<th>Data Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_INT8</td>
<td>tf.int8</td>
<td>8 bits signed integer.</td>
</tr>
<tr>
<td>DT_UINT8</td>
<td>tf.uint8</td>
<td>8 bits unsigned integer.</td>
</tr>
<tr>
<td>DT_STRING</td>
<td>tf.string</td>
<td>Variable length byte arrays. Each element of a Tensor is a byte array.</td>
</tr>
<tr>
<td>DT_BOOL</td>
<td>tf.bool</td>
<td>Boolean.</td>
</tr>
<tr>
<td>DT_COMPLEX64</td>
<td>tf.complex64</td>
<td>Complex number made of two 32 bits floating points: real and imaginary parts.</td>
</tr>
<tr>
<td>DT_QINT32</td>
<td>tf.qint32</td>
<td>32 bits signed integer used in quantized Ops.</td>
</tr>
<tr>
<td>DT_QINT8</td>
<td>tf.qint8</td>
<td>8 bits signed integer used in quantized Ops.</td>
</tr>
<tr>
<td>DT_QUINT8</td>
<td>tf.quint8</td>
<td>8 bits unsigned integer used in quantized Ops.</td>
</tr>
</tbody>
</table>
Variables

**Variables: Creation, Initialization, Saving and Loading**

- To hold and update parameters, maintain state in the graph across calls to `run()`
- In-memory buffers containing tensors
- Must be explicitly initialized and can be saved to disk during and after training

**Class `tf.Variable`**

- Constructor: an initial value for the variable, a Tensor of any type and shape
- After construction, the type and shape are fixed
- `assign` Op with `validate_shape=False`
Variables

❖ Creation

➤ Pass a Tensor as its initial value to the Variable() constructor
➤ Initial value: constants, sequences and random values
  ▪ `tf.zeros()`, `tf.linspace()`, `tf.random_normal()`
➤ Fixed shape: the same shape as ops’ shape

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

➤ Calling `tf.Variable()` adds several ops to the graph

❖ Initialization

➤ Add an op and run it
➤ `tf.initialize_all_variables()`

```
# 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)`
Variables

- **Saving and Restoring**
  - `tf.train.saver`
  - Checkpoint Files: Variables are saved in binary files that contain a map from variable names to tensor values

```python
# 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: %s" % save_path)
```
Variables

- Saving and Restoring
  - Restore

```python
# 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
...```
Choosing which Variables to Save and Restore

- No arguments to `tf.train.Saver()` ➔ handle all variables in the graph
  - Each one of them is saved under the name

- Save and restore a subset of the variables
  - Ex) trained neural net with 5 layers ➔ want to train a new model with 6 layers, restoring the parameters from the 5 layers

- Passing to the `tf.train.Saver()` constructor a Python dictionary: keys

```python
# 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.
...```
# Example code serving a simple counter

```python
# 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
```
Fetched

- Execute the graph with a `run()` call on the `Session` object and pass in the tensors to retrieve

```python
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])]
Variables

- **Feeds**
  - Patching a tensor directly into any operation in the graph
  - Temporarily replaces the output of an operation with a tensor value
  - Feed data as an argument to a `run()` call
  - Only used for the run call to which it is passed
  - `tf.placeholder()`

```python
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)]
```
## Operations

### Operations

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element-wise mathematical operations</td>
<td>Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...</td>
</tr>
<tr>
<td>Array operations</td>
<td>Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...</td>
</tr>
<tr>
<td>Matrix operations</td>
<td>MatMul, MatrixInverse, MatrixDeterminant, ...</td>
</tr>
<tr>
<td>Stateful operations</td>
<td>Variable, Assign, AssignAdd, ...</td>
</tr>
<tr>
<td>Neural-net building blocks</td>
<td>SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...</td>
</tr>
<tr>
<td>Checkpointing operations</td>
<td>Save, Restore</td>
</tr>
<tr>
<td>Queue and synchronization operations</td>
<td>Enqueue, Dequeue, MutexAcquire, MutexRelease, ...</td>
</tr>
<tr>
<td>Control flow operations</td>
<td>Merge, Switch, Enter, Leave, NextIteration</td>
</tr>
</tbody>
</table>
References


- [www.tensorflow.org](http://www.tensorflow.org)

- [http://www.slideshare.net/mikeranderson/2013-1114-enter-thematrix](http://www.slideshare.net/mikeranderson/2013-1114-enter-thematrix)
Thank you for your attention!

http://web.donga.ac.kr/yjko/

고 영 중