TensorFlow
Agenda

- Introduction
- Machine Learning
- Artificial Neural Networks
- TensorFlow Basics
- Image Recognition Example
- Conclusion
Introduction

• Open source library by Google for Machine Learning
• Successor to DistBelief made by Google Brain
• Scalability and portability core feature.
• Main uses are pattern recognition using Artificial Neural Networks
• Large community with wide range of applications internally and outside Google
Machine Learning (ML)

- Application of Artificial Intelligence (AI)
  AI - Broader concept, Machines perform “intelligent” tasks
  ML - Give machines data and make them learn by themselves.

- **ML**: "A Field of study that gives computers the ability to learn without being explicitly programmed."
  - Arthur Samuel, 1959

- Requires a large amount of data and computational power
Different ML tasks

SUPERVISED LEARNING

UNSUPERVISED LEARNING

REINFORCEMENT LEARNING
Artificial Neural Networks

- Network Architecture
- Forward propagation
- Node activation / how they work
- Mathematical model of Neural network
- Gradient computation
Network Architecture
Forward propagation

\[ a_n^1 = w_{1,1}a_1^0 + w_{1,2}a_2^0 + w_{1,3}a_3^0 + \ldots + w_{1,n}a_n^0 \]
Mathematical model

\[
\begin{bmatrix}
a_1^2 \\
a_2^2 \\
a_3^2
\end{bmatrix}
= \sigma \left( \begin{bmatrix}
w_{1,1} & w_{1,2} & w_{1,3} \\
w_{2,1} & w_{2,2} & w_{2,3} \\
w_{3,1} & w_{3,2} & w_{3,3}
\end{bmatrix}
\begin{bmatrix}
a_1^1 \\
a_2^1 \\
a_3^1
\end{bmatrix}
- \begin{bmatrix}
b_1^1 \\
b_2^1 \\
b_3^1
\end{bmatrix} \right)
\]

Which can be rewritten for a more general case as

\[
a^{n+1} = \sigma(Wa^n - b^n)
\]
Gradient computation

\[ J = \frac{1}{2} \sum (y - \hat{y})^2 \]

\[ \nabla J = \left( \frac{\delta J}{\delta w_{1,1}}, \frac{\delta J}{\delta b_1}, ..., \frac{\delta J}{\delta w_{j,k}}, \frac{\delta J}{\delta b_j} \right) \]
TensorFlow Basics

- Tensors
- Computational graph and Sessions
- TensorBoard
- Placeholders
- Variables
- Training and Optimization
TensorFlow Basics

What is a Tensor?

A *multidimensional array*

Different
* ranks
* types

<table>
<thead>
<tr>
<th>Rank</th>
<th>Math entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Scalar (magnitude only)</td>
</tr>
<tr>
<td>1</td>
<td>Vector (magnitude and direction)</td>
</tr>
<tr>
<td>2</td>
<td>Matrix (table of numbers)</td>
</tr>
<tr>
<td>3</td>
<td>3-Tensor (cube of numbers)</td>
</tr>
<tr>
<td>n</td>
<td>n-Tensor (you get the idea)</td>
</tr>
</tbody>
</table>

```python
mystr = tf.Variable(['Hello'], tf.string)
cool_numbers = tf.Variable([3.14159, 2.71828], tf.float32)
first_primes = tf.Variable([2, 3, 5, 7, 11], tf.int32)
its_very_complicated = tf.Variable([[12.3, -4.85], [7.5, -6.23]], tf.complex64)
```
Computational graph and Sessions

Computational graph
- a series of TensorFlow opts / nodes arranged into a graph

Session
- for graph evaluation

```python
import tensorflow as tf

# Build graph
a = tf.constant([[-1.0, -1.0, -1.0], [-1.0, -1.0, -1.0]])
b = tf.constant(1.0, shape=[3, 2]) # an other way of defining a tensor
c = tf.matmul(a, b)

# Create a session object
sess = tf.Session()

# Run the graph
print(sess.run(c))
```

This produce the output:

```
[[  3.  -3.]
 [  3.  -3.]]
```
TensorBoard

- interactive visualization tool

```python
writer = tf.summary.FileWriter('output_folder', sess.graph)
```

- Run the command: `tensorboard --logdir=path/to/log-directory`
- In a web browser, navigate to: `localhost:6006`
Add name and name scopes for better readability

```python
import tensorflow as tf
a = tf.constant([-1.0, shape=[2, 3]], name = 'A')
b = tf.constant([1.0, shape=[3, 2]], name = 'B')
c = tf.matmul(a, b, name = 'C')

da = tf.matmul([1.0, shape=[2, 3]], name = 'A')
b = tf.constant([1.0, shape=[3, 2]], name = 'B')
c = tf.matmul(a, b, name = 'C')
```

```python
import tensorflow as tf
with tf.name_scope('Model1'):
    a = tf.constant([1.0, shape=[2, 3]], name = 'A')
b = tf.constant([1.0, shape=[3, 2]], name = 'B')
c = tf.matmul(a, b, name = 'C')

with tf.name_scope('Model2'):
    d = tf.matmul(c, c, name = 'D')

sess = tf.Session()
print(sess.run(c))
writer = tf.summary.FileWriter("output", sess.graph)
```
Placeholders

import tensorflow as tf

# create placeholder
x = tf.placeholder(dtype=tf.float32)

# define session object in order to evaluate
sess = tf.Session()

# run and print place holder
print(sess.run(x))  # will fail since x is not provided with values

# make random numbers with numpy, 4X4 tensor
rand_array = np.random.rand(4,4)

print(sess.run(x,feed_dict={x: rand_array}))  # will work

• Must be provided with values at a later stage
Variables

- trainable parameters
- initial value and explicitly initialized

```python
v = tf.Variable([1.2, 1.3])
sess = tf.Session()
initialize = tf.global_variables_initializer()
sess.run(initialize)
```
Training

- Adjust the Variables in our model to minimize a cost function
- tf.train choose optimization algorithm
- Base Class : Optimizer
  - provides methods to compute gradients
- GradientDecentOptimizer

\[
V_i = V_{i-1} - \alpha \frac{\partial J}{\partial V}_{i-1}
\]
Image recognition example

- Digit recognition example
- Use MNIST database for training and validation data
- Example code
- Showcase model
The MNIST database
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

with tf.name_scope('model'):
    #input
    x = tf.placeholder(tf.float32, [None, 784], name='x')

    #layer 1
    layer_size = 700
    W1 = tf.Variable(tf.random_normal([784, layer_size]), name='W1')
    b1 = tf.Variable(tf.random_normal([layer_size]), name='b1')
    y1 = tf.nn.sigmoid(tf.matmul(x, W1) + b1, name='y1')

    #layer 2
    W2 = tf.Variable(tf.random_normal([layer_size, 10]), name='W2')
    b2 = tf.Variable(tf.random_normal([10]), name='b2')
    y2 = tf.nn.softmax(tf.matmul(y1, W2) + b2, name='y2')

    #output
    y = y2
    y_ = tf.placeholder(tf.float32, [None, 10])

    with tf.name_scope('Training'):
        cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
                                                   reduction_indices=[1]), name='cross_entropy')
        learning_rate = 1
        train_step = tf.train.GradientDescentOptimizer(learning_rate, name='Train_step').minimize(cross_entropy)

    with tf.name_scope('Accuracy'):
        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1), name='Correct_prediction')
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='Accuracy')

sess = tf.InteractiveSession()  # create session object
tf.global_variables_initializer().run()  # initialize variables
tf.summary.FileWriter("graph", sess.graph)  # TensorBoard visualization

batch_size=100
for i in range(10000):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
First 500 learning steps
10 000 training steps
Final 100 training steps
10,000 training steps
Last 100 training steps

Accuracy plotted for different hidden layers
Final 100 training steps, for more nodes
Different batch sizes
Different numbers of hidden layers