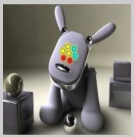



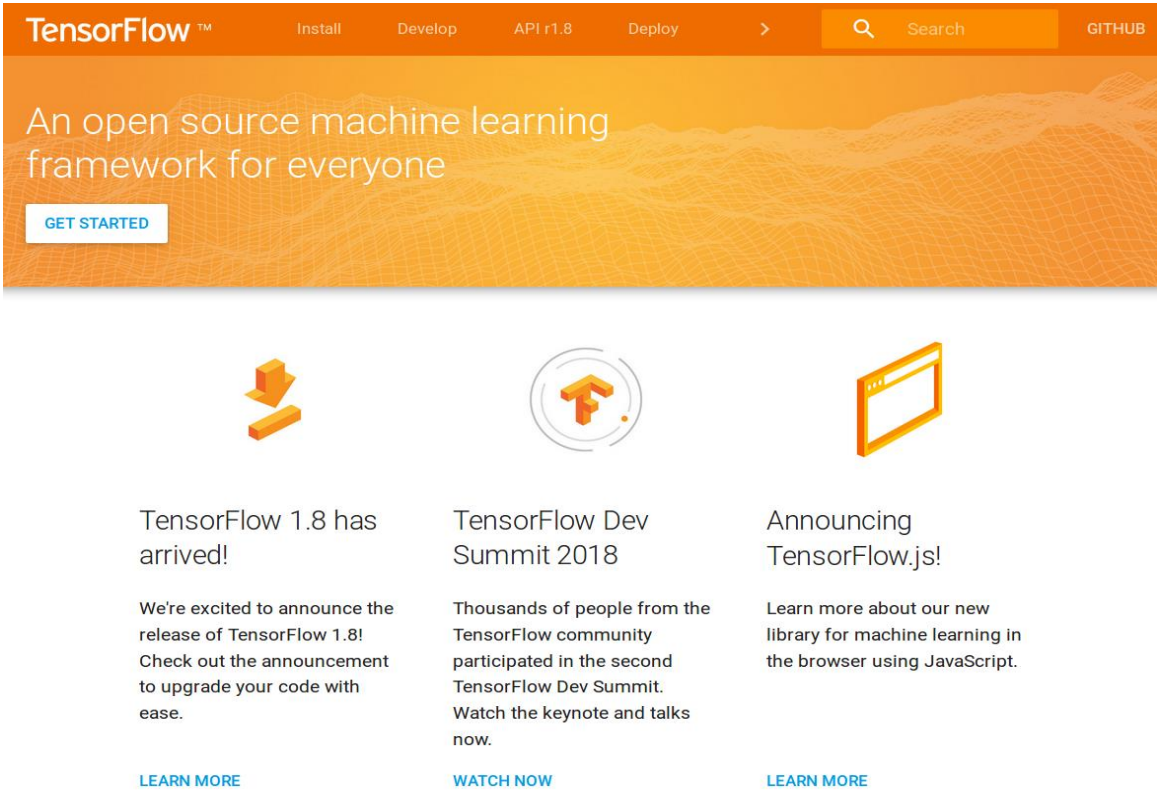

Lecture 12 – TensorFlow Basics

Machine Learning

TensorFlow Basics

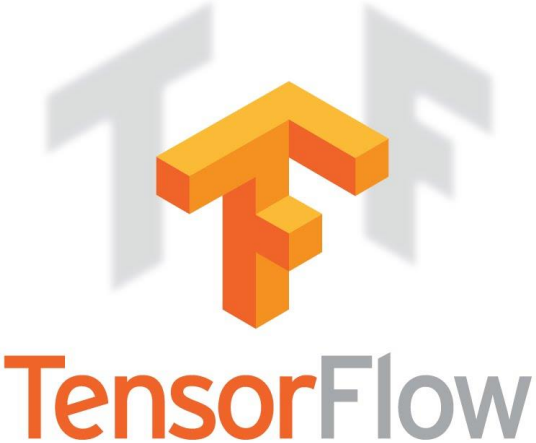
1. Why Tensorflow?
2. Tf Website
3. Data Flow Graph
4. Installation And Simple Validation
5. Say Hello For TF With Jupyter Notebook
6. TF Mechanics And Computational Graph
7. Tensor, Placeholder And Variable
8. MNIST Dataset and its TF Example

TF Website: https://www.tensorflow.org



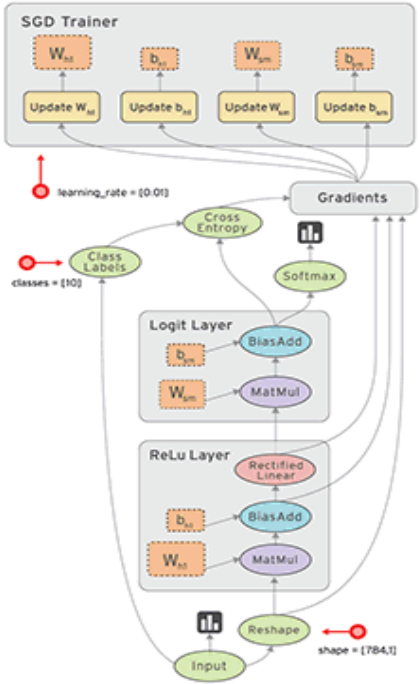
TensorFlow

- TensorFlow™ is an open source software library for numerical computation using data flow graphs.
- But what does it actually do?
  - TensorFlow provides primitives for defining functions on tensors and automatically computing their derivatives.
- With Python!



What is a Data Flow Graph?

- **Dataflow** is a common programming model for ***parallel computing***
  - the *nodes* represent units of ***computation***,
  - the *edges* represent the ***data*** consumed or produced by a computation.
- **What's a Tensor?**
  - Matrix, vector, scalar
  - Formally, tensors are multilinear maps from vector spaces to the real numbers
  - a tensor can be represented as a multidimensional array of numbers.



Everything is Tensor

Tensors

```
In [3]: 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]
```

- 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,
- while its **shape** is a tuple of integers specifying the array's length along each dimension
- TensorFlow uses numpy arrays to represent tensor **values**.

Tensor Ranks, Shapes, and Types

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

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

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

https://www.tensorflow.org/programmers_guide/dims_types

Jupyter notebook

- Jupyter is an web-based interactive development environment.
- It support multiple programming languages like Python.
  - When you install the Anaconda,
  - you get your own jupyter notebook at the same time!
- For more details
  - https://jupyter.readthedocs.io/en/latest/index.html
  - Video Tutorial: https://www.youtube.com/user/roshanRush

How to run?

$ jupyter notebook

http://nbviewer.jupyter.org/github/twistedhardware/mltutorial/blob/master/notebooks/jupyter/1.Introduction.ipynb

TensorFlow Hello World!

Hello TensorFlow!

In [2]:
```
# Create a constant op
# This op is added as a node to the default graph
hello = tf.constant("Hello, TensorFlow!")

# start a TF session
sess = tf.Session()

# run the op and get result
print(sess.run(hello))
```

```
b'Hello, TensorFlow!'
```

‘b’ indicates Bytes literals.

http://stackoverflow.com/questions/6269765/

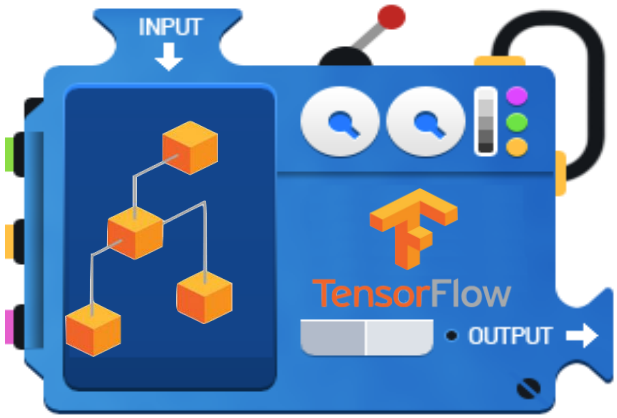
TensorFlow Mechanics

1. Build **graph** using TensorFlow **operations**

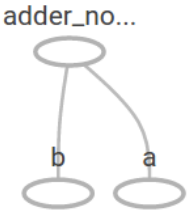
   tf.Graph

2. **feed data** and **run graph** (Session -> **op**erations**s**)
   sess*.run *(ops)

3. **update variables** in the **graph** (and **return** values)



Computational Graph



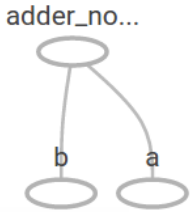

```
In [4]: node1 = tf.constant(3.0, tf.float32)
        node2 = tf.constant(4.0) # also tf.float32 implicitly
        node3 = tf.add(node1, node2)
```

```
In [5]: print("node1: ", node1)
        print("node2: ", node2)
        print("node3: ", node3)
```

```
node1:  Tensor("Const_1:0", shape=(), dtype=float32)
node2:  Tensor("Const_2:0", shape=(), dtype=float32)
node3:  Tensor("Add:0", shape=(), dtype=float32)
```

- A **computational graph** is a series of TensorFlow **operations** arranged into a **graph**.
- The **graph** is composed of two types of objects.
  - **Operations** (or "**ops**"): The nodes of the graph. Operations describe calculations that *consume* and *produce* **tensors**.
  - **Tensors**: The edges in the graph. These represent the values that will **flow** through the graph. Most TensorFlow functions return tf.Tensors.

Computational Graph



- (1) Build graph (tensors) using TensorFlow operations

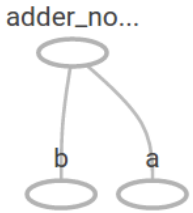
```
In [4]: node1 = tf.constant(3.0, tf.float32)
        node2 = tf.constant(4.0) # also tf.float32 implicitly
        node3 = tf.add(node1, node2)
```

- (2) feed data and run graph (operation)

  sess.run (op)

- (3) update variables in the graph (and return values)

Computational Graph [Session]



- **(2) feed data and run graph (operation)**

  sess.run (op)
- **(3) update variables in the graph (and return values)**

```
In [6]: sess = tf.Session()
        print("sess.run(node1, node2): ", sess.run([node1, node2]))
        print("sess.run(node3): ", sess.run(node3))

        sess.run(node1, node2):  [3.0, 4.0]
        sess.run(node3):  7.0
```

```
# create a simple session
sess = tf.Session()
# ...
sess.close()

# Or Create a default in-process session for the current default graph.
with tf.Session() as sess:
  # ...
```

Computational Graph

- **Session**
  - To evaluate tensors, instantiate a tf.Session object, informally known as a sess(ion).
  - A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations.
    - ✓ If a tf.Graph is like a .py file, a tf.Session is like the python executable.
  - You can pass *multiple tensors* to tf.Session.run.

    The run method transparently handles any combination of tuples or dictionaries
  - During a call to tf.Session.run any tf.Tensor only has a single value

```
In [5]: vec = tf.random_uniform(shape=(3,))
        out1 = vec + 1
        out2 = vec + 2
        print(sess.run(vec))
        print(sess.run(vec))
        print(sess.run(out1))
        print(sess.run(out2))
        print(sess.run((out1, out2)))

        [0.18464649 0.14886129 0.5047021 ]
        [0.3270713  0.4014722  0.22448409]
        [1.3578813 1.7317374 1.8207817]
        [2.3646321 2.3802166 2.8418117]
        (array([1.3722222, 1.2407451, 1.8812546], dtype=float32),
        array([2.3722222, 2.240745 , 2.8812547], dtype=float32))
```

The result shows a **different** random value **on each call** to run,

but a **consistent** value during a **single run**

(out1, out2) receive the same random input

Placeholders and Feeding

```
In [7]: 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, feed_dict={a: 3, b: 4.5}))
        print(sess.run(adder_node, feed_dict={a: [1,3], b: [2, 4]}))
```

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

- A graph can be parameterized to accept external inputs, known as **placeholders**
- A **placeholder** is a promise to provide a value later, like a function argument.
- We can evaluate this graph with *multiple inputs* by using the **feed_dict** argument of the ***run*** *method* to feed concrete values to the **placeholders**

Placeholders and Feeding (without feed_dict and add closed session)

In [10]:
```
import tensorflow as tf
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)
sess = tf.Session()
print(sess.run(adder_node, {a: 3, b: 4.5}))
sess.close()
print(sess.run(adder_node, {a: 3, b: 4.5}))
```

```
7.5

---------------------------------------------------------------------------
----
RuntimeError                                 Traceback (most recent call l
ast)
<ipython-input-10-2ecc9be8ffc4> in <module>()
      6 print(sess.run(adder_node, {a: 3, b: 4.5}))
      7 sess.close()
----> 8 print(sess.run(adder_node, {a: 3, b: 4.5}))

~/anaconda3/envs/tensorflow/lib/python3.6/site-packages/tensorflow/pyth
on/client/session.py in _run(self, handle, fetches, feed_dict, options,
run_metadata)
   1056     # Check session.
   1057     if self._closed:
-> 1058       raise RuntimeError('Attempted to use a closed Session.')
   1059     if self.graph.version == 0:
   1060       raise RuntimeError('The Session graph is empty.  Add oper
ations to the '

RuntimeError: Attempted to use a closed Session.
```

Variables

- A TensorFlow **variable** is the best way to represent shared, persistent state manipulated by your program.
- *The best way* to create a variable is to call the **tf.get_variable()** function.

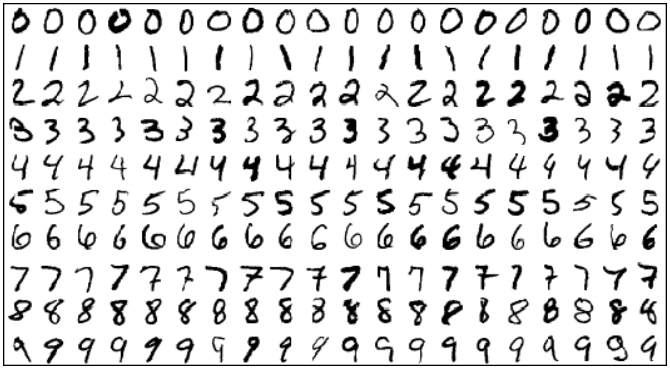
```
In [1]: import tensorflow as tf

In [2]: # simply provide the name and shape
        my_variable = tf.get_variable("my_variable", [1, 2, 3])
```

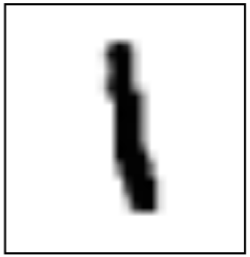
This variable will, by **default**, have the **dtype** *tf.float32* and its initial value will be **randomized** via *tf.glorot_uniform_initializer*.

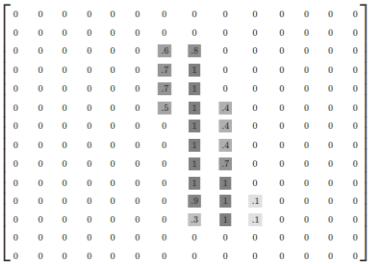
```
In [3]: # specify the dtype and initializer
        my_int_variable = tf.get_variable("my_int_variable", [1, 2, 3],
                          dtype=tf.int32, initializer=tf.zeros_initializer)

        # initialize a tf.Variable to have the value of a tf.Tensor.
        other_variable = tf.get_variable("other_variable",
                             dtype=tf.int32, initializer=tf.constant([23, 42]))
```


https://www.tensorflow.org/programmers_guide/variables

MNIST Dataset







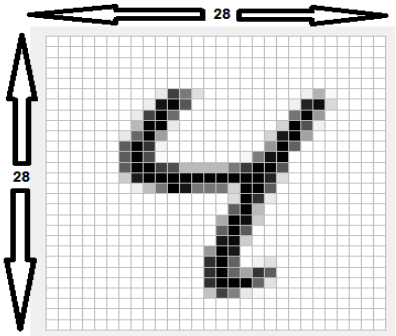
[train-images-idx3-ubyte.gz](): training set images (9912422 bytes)
[train-labels-idx1-ubyte.gz](): training set labels (28881 bytes)
[t10k-images-idx3-ubyte.gz](): test set images (1648877 bytes)
[t10k-labels-idx1-ubyte.gz](): test set labels (4542 bytes)

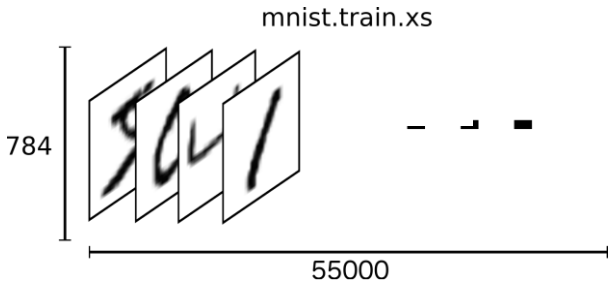
http://yann.lecun.com/exdb/mnist/

MNIST Dataset

- *# Check out https://www.tensorflow.org/get_started/mnist/beginners for more information about the mnist dataset*

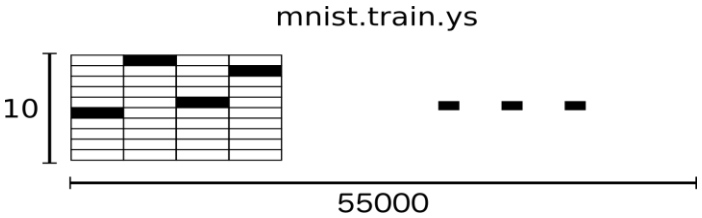
- *# MNIST data image of shape 28 * 28 = 784*
- X = tf.placeholder(tf.float32, [**None**, 784])
- *# 0 - 9 digits recognition = 10 classes*
- Y = tf.placeholder(tf.float32, [**None**, nb_classes])





http://derindelimavi.blogspot.hk/2015/04/mnist-el-yazs-rakam-veri-seti.html

MNIST Dataset & EXAMPLE



import tensorflow

```
In [1]: import tensorflow as tf
        import numpy as np
```

```
In [2]: from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets("MNIST_data/",one_hot=True)
```

Dataset	Purpose
input_data.train	55000 images and labels, for primary training.
input_data.validation	5000 images and labels, for iterative validation of training accuracy.
input_data.test	10000 images and labels, for final testing of trained accuracy.

MNIST Dataset

get the data

```
In [2]: # if there is no "MNIST_data/" then download
        from tensorflow.examples.tutorials.mnist import input_data
        # extract and read the mnist data
        mnist = input_data.read_data_sets("MNIST_data/",one_hot=True)
```

```
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

one-hot

```
In [48]: one_hot =tf.one_hot([[0],[1],[2],[0]],depth=3)

In [49]: sess.run(tf.one_hot([[0],[1],[2],[0]],depth=3))
Out[49]: array([[[ 1.,  0.,  0.]],

                [[ 0.,  1.,  0.]],

                [[ 0.,  0.,  1.]],

                [[ 1.,  0.,  0.]]], dtype=float32)

In [50]: sess.run(tf.one_hot([[0],[1],[2],[0]],depth=4))
Out[50]: array([[[ 1.,  0.,  0.,  0.]],

                [[ 0.,  1.,  0.,  0.]],

                [[ 0.,  0.,  1.,  0.]],

                [[ 1.,  0.,  0.,  0.]]], dtype=float32)

In [51]: sess.run(tf.reshape(one_hot,shape=[-1,3]))
Out[51]: array([[ 1.,  0.,  0.],
                [ 0.,  1.,  0.],
                [ 0.,  0.,  1.],
                [ 1.,  0.,  0.]], dtype=float32)
```

If you have download the MNIST dataset

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

MNIST EXAMPLE

set variables

In [4]:
```
nb_class = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32,[None,784])
# 0-9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32,[None, nb_class])

W = tf.get_variable("w", [784,nb_class],
                    initializer=tf.truncated_normal_initializer(stddev=0.1))
b = tf.get_variable("biases", [nb_class],
                    initializer=tf.constant_initializer(0.1))
```

The *None* means, the value will be got by *X / 784*

MNIST EXAMPLE

softmax for the linear regression

In [4]:
```
# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis),axis = 1))
optimizer = tf.train.GradientDescentOptimizer(
    learning_rate=0.1).minimize(cost)

# test model
is_correct = tf.equal(tf.argmax(hypothesis,1),tf.argmax(Y,1))
# Caculate accruracy
accuracy = tf.reduce_mean(tf.cast(is_correct,tf.float32))
```

nn **is a module and** ***softmax*** **is an operation(function)**

MNIST EXAMPLE

training epoch/batch

In [5]:
```
# parameters
training_epoches = 15
batch_size = 100

with tf.Session() as sess:
    # initialize Tensorflow Variables
    sess.run(tf.global_variables_initializer())
    # training cycles
    for epoch in range(training_epoches):
        avg_cast = 0
        total_batch = int(mnist.train.num_examples/batch_size)

        for i in range(total_batch):
            batch_xs,batch_ys = mnist.train.next_batch(batch_size)
            c,_ = sess.run([cost,optimizer]
                           ,feed_dict={X:batch_xs,Y:batch_ys})
            avg_cast += c / total_batch

        print("Epoch:",'%04d'%(epoch+1)
              ,'cost=','{:.9f}'.format(avg_cast))

    print("Learning finished")

    # Test the model using test sets
    print("old_Accuracy: ", accuracy.eval(session=sess, feed_dict={
          X: mnist.test.images, Y: mnist.test.labels}))
    print('new_Accuracy:',sess.run(accracy,feed_dict={
        X: mnist.test.images, Y: mnist.test.labels}))
```

MNIST EXAMPLE

```
# Sample image show and prediction
import matplotlib.pyplot as plt
import random


# Get one and predict
r = random.randint(0,mnist.test.num_examples -1 )
print("Label:",sess.run(tf.arg_max(mnist.test.labels[r:r+1],1)))
print("Prediction:",sess.run(tf.arg_max(hypothesis,1),
      feed_dict={X:mnist.test.images[r:r+1]}))

plt.imshow(mnist.test.images[r:r+1].reshape(28,28),
           cmap='Greys',interpolation='nearest')
plt.show()
```

MNIST EXAMPLE

```
Epoch: 0001 cost= 2.804654121
Epoch: 0002 cost= 1.179020686
Epoch: 0003 cost= 0.921789622
Epoch: 0004 cost= 0.797202877
Epoch: 0005 cost= 0.720808540
Epoch: 0006 cost= 0.666504125
Epoch: 0007 cost= 0.625679149
Epoch: 0008 cost= 0.593117082
Epoch: 0009 cost= 0.566464136
Epoch: 0010 cost= 0.544340848
Epoch: 0011 cost= 0.525419921
Epoch: 0012 cost= 0.508744514
Epoch: 0013 cost= 0.494571513
Epoch: 0014 cost= 0.481707098
Epoch: 0015 cost= 0.470132457
Learning finished
Accuracy:  0.8881
Label: [7]
Prediction: [7]
```