python_numpy_tensorflow_tutorial

September 11, 2016

1 What is Python?

From Wikipedia: - Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. - Design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.

2 Python Function

```
In [1]: def main():
            print "Hello World"
        main()
```

```
Hello World
```

2.1 Python String Manipulation

```
In [2]: line1 = "Hello"
        line2 = "ECE544"
        class_num = 544
        print line1 + ' ' + line2

        print "%s %s" % (line1,line2)

        print "%s ECE%d" % (line1, class_num)
```

```
Hello ECE544
Hello ECE544
Hello ECE544
```

2.2 Python Containers

2.2.1 Python Dictionary

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Let's say we have data containing an student i.d. and their favorite classifier.
Student 1 : SVM
Student 2 : Perceptron
Studnet 3 : Linear Regression

```
In [3]: student_classifier_dict = {}
        student_classifier_dict['1'] = 'SVM'
        student_classifier_dict['2'] = 'Perceptron'
        student_classifier_dict['3'] = 'Linear Regression'
        print student_classifier_dict
```

```
        student_classifier_dict = {'1':'SVM', '2':'Perceptron', '3':'Linear Regression'}
        print student_classifier_dict

        print "Student %s's favorite classifier is %s" % ('1', student_classifier_dict['1'])
```

```
{'1': 'SVM', '3': 'Linear Regression', '2': 'Perceptron'}
{'1': 'SVM', '3': 'Linear Regression', '2': 'Perceptron'}
Student 1's favorite classifier is SVM
```

2.2.2 Python List

https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

Note: Python index starts with 0

Same example as above

```
In [4]: student_classifier_list = []
        student_classifier_list.append('SVM')
        student_classifier_list.append('Perceptron')
        student_classifier_list.append('Linear Regression')

        print student_classifier_list

        print "Student %d's favorite classifier is %s" % (2, student_classifier_list[1])

        # Indexing with list
        print student_classifier_list[0:2]
        print student_classifier_list[0:-1]
        print student_classifier_list[1:2]
```

```
['SVM', 'Perceptron', 'Linear Regression']
Student 2's favorite classifier is Perceptron
['SVM', 'Perceptron']
['SVM', 'Perceptron']
['Perceptron']
```

2.3 Python File IO

```
In [5]: # Write to a file
        f = open("foo.txt", "wb")
        f.write("1: SVM\n")
        f.write("2: Perceptron\n")
        f.write("3: Linear Regression\n")
        f.close()
```

```
In [6]: # Read the file back
        f = open("foo.txt", "rb")
        student_classifier_dict_2 = {}
        for line in f:
            print line
            student_id, classifer = line.rstrip().split(':')
            student_classifier_dict_2[student_id] = classifer

        print student_classifier_dict_2
```

```
1: SVM

2: Perceptron

3: Linear Regression

{'1': ' SVM', '3': ' Linear Regression', '2': ' Perceptron'}
```

2.4 Python Class

https://docs.python.org/3/tutorial/classes.html

```
In [7]: class Student(object):
            def __init__(self, student_id, classifier):
                self.student_id = student_id
                self.classifier = classifier
            def print_classifer(self):
                print "Student %s's favorite classifier is %s" % (self.student_id, self.classifier)


In [8]: student1 = Student('1','SVM')
        student1.print_classifer()

Student 1's favorite classifier is SVM
```

3 What is Numpy?

http://www.numpy.org/

NumPy is the fundamental package for scientific computing with Python. Main functionalities: - a powerful N-dimensional array object - useful linear algebra, Fourier transform, and random number capabilities

3.1 Numpy Container (np.array)

Numpy's array is very similar to MATLAB's matrix.

There are many similarites between MATLAB and numpy, a few examples are shown below.

More information at https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

```
In [9]: import numpy as np
```

3.2 Numpy Array Manipulation

More details at - http://docs.scipy.org/doc/numpy/reference/arrays.indexing.htm - https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

```
In [10]: # Declare an array
         X = np.array([[1,2,3],[4,5,6]])
         print X

         # Indexing an element
         print X[1,2]

         # indexing the last element of row 0
         print X[0,-1]

         # More advance indexing
```

```
        y = np.zeros(X.shape, dtype=np.bool)
        y[0,2] = True
        X[y] = -100
        print X

        # Get the 0th row
        print X[0, :]
```

```
[[1 2 3]
 [4 5 6]]
6
3
[[   1    2 -100]
 [   4    5    6]]
[   1    2 -100]
```

3.3 Numpy basic array operations

- "Matrix" Multiplication: https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html
- "Matrix" linear algebra: http://docs.scipy.org/doc/numpy/reference/routines.linalg.html

```
In [11]: # Ax = b
         A = np.random.randn(3,3)  # Creates a 3x3 random array
         x = np.random.randn(3,1)  # Creates a 3x1 vector
         b = np.dot(A,x)  # Compute Ax = b

         # Solve Ax = b
         x_hat = np.dot(np.linalg.inv(A),b)  # Solves for x

         print "This is x:"
         print x
         print "This is x_hat:"
         print x_hat
```

```
This is x:
[[ 0.01005125]
 [ 1.30976247]
 [ 1.31497944]]
This is x_hat:
[[ 0.01005125]
 [ 1.30976247]
 [ 1.31497944]]
```

3.4 Numpy array broadcasting

- http://docs.scipy.org/doc/numpy-1.10.0/user/basics.broadcasting.html
- http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

```
In [12]: # Performs element-wise multiplication (Example from links above)
         a = np.array([1.0, 2.0, 3.0])
         b = np.array([2.0, 2.0, 2.0])
         print a * b
         print('---------------------------------')
         # Performs element-wise multiplication per column.
```

```
A = np.ones((2,3))
b = np.array([[1],[2]])
print ("This is A:")
print A
print ("This is b:")
print b
print ("This is A*b:")
print A*b
print('----------------------------------')
# Performs element-wise multiplication per row.
A2 = np.ones((2,3))
b2 = np.array([[1,2,3]])
print ("This is A2:")
print A2
print ("This is b2:")
print b2
print ("This is A2*b2:")
print A2*b2

print('----------------------------------')
print ("This is A2+b2:")
print A2+b2
```

```
[ 2.  4.  6.]
----------------------------------
This is A:
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
This is b:
[[1]
 [2]]
This is A*b:
[[ 1.  1.  1.]
 [ 2.  2.  2.]]
----------------------------------
This is A2:
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
This is b2:
[[1 2 3]]
This is A2*b2:
[[ 1.  2.  3.]
 [ 1.  2.  3.]]
----------------------------------
This is A2+b2:
[[ 2.  3.  4.]
 [ 2.  3.  4.]]
```

3.5 Numpy File IO

http://docs.scipy.org/doc/numpy/reference/routines.io.html

4 What is TensorFlow?

https://www.tensorflow.org/ - "TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them." - The following examples follows the concepts covered in https://www.tensorflow.org/versions/r0.10/tutorials/index.html, and the key APIs in https://www.tensorflow.org/versions/r0.10/api_docs/python/client.html#session-management

4.1 TensorFlow Computational Graphs

- In TensorFlow, the compuation functions are used to define the computational graph, and does not have values until the computational graph is evaluted explicitly.

```
In [13]: import numpy as np
         import tensorflow as tf
         sess = tf.Session()

In [14]: A_tf = tf.random_uniform([2,3])
         print A_tf

         sess.run(tf.initialize_all_variables())
         A_tf_evaluted = sess.run(A_tf)
         print A_tf_evaluted
         print ("-----------------------------------------------")
         A_np = np.random.rand(2,3)
         print A_np
```

```
Tensor("random_uniform:0", shape=(2, 3), dtype=float32)
[[ 0.54514456  0.35032892  0.09330726]
 [ 0.35135043  0.72912657  0.6226058 ]]
-----------------------------------------------
[[ 0.94516267  0.12267702  0.07368606]
 [ 0.27118396  0.21015883  0.93899287]]
```

4.2 TensorFlow Sessions

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated - https://www.tensorflow.org/

```
In [15]: # Define a simple computation graph. b = Ax
         A = tf.constant(1.0, shape=[2,3])
         x = tf.constant([1.0,2.0,3.0], shape=[3,1])
         b = tf.matmul(A,x)

         sess.run(tf.initialize_all_variables())
         print("A:")
         print sess.run(A)
         print("x:")
         print sess.run(x)
         print("b:")
         print sess.run(b)
```

```
A:
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
x:
```

```
[[ 1.]
 [ 2.]
 [ 3.]]
b:
[[ 6.]
 [ 6.]]
```

```
In [16]: # Defining an update opteration
         x = tf.Variable(1.0)
         increment = x.assign_add(1.0)  # Adds 1 to the variable x.
         sess.run(tf.initialize_all_variables())
         for k in range(0,10):
             print sess.run(x)
             sess.run(increment)
```

```
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
```

4.3 Getting data into Tensorflow.

4.3.1 Feeding from numpy with placeholder

Revelent Tensorflow Pages
https://www.tensorflow.org/versions/r0.10/tutorials/mnist/tf/index.html#inputs-and-placeholders
https://github.com/tensorflow/tensorflow/blob/r0.10/tensorflow/examples/tutorials/mnist/fully_connected_feed.py

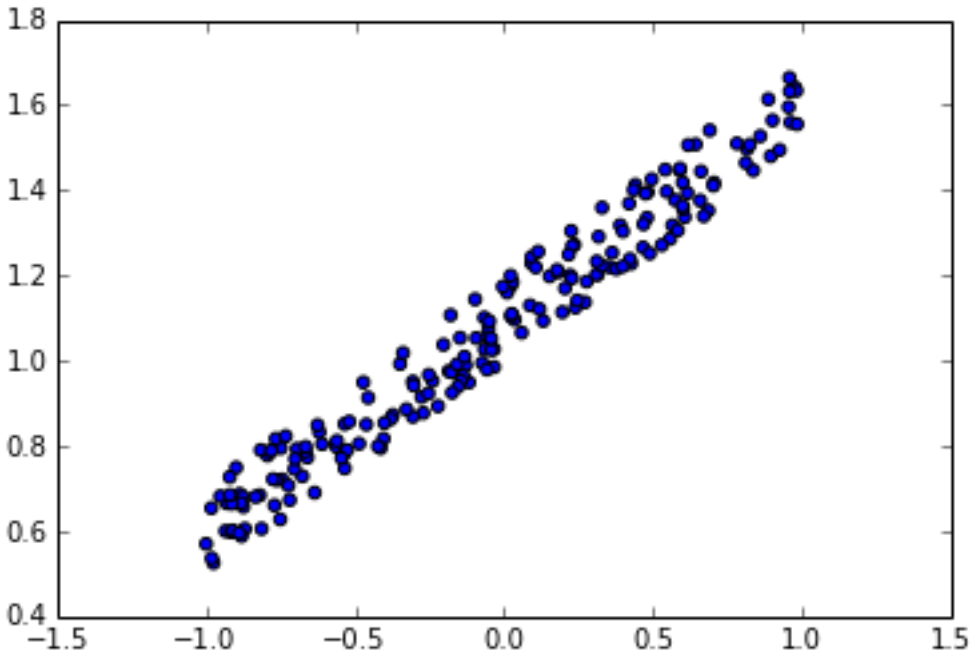
4.4 Linear Regression Tensorflow Example

This example will demonstrate the following concepts: - Matplotlib - Feeding Dictionary - Automatic differentiation

```
In [17]: %matplotlib inline
         import warnings
         warnings.filterwarnings("ignore")
         # Only need '%matplotlib inline' when running in ipytho notebook.
         import matplotlib.pyplot as plt
```

```
In [18]: # Create 200 data points, using a linear model with noise.
         x = (np.random.rand(200,1)-0.5)*2.0
         noise = 10*np.random.rand(200,1)
         b = 1
         y = 0.5*x+ b +0.02*noise
         plt.scatter(x,y)

         # Pad x with 1 for bias-term.
         x = np.concatenate((x,np.ones(x.shape)),axis=1)
```



In [19]:
```
# Define computational graph using placeholders
X_placeholder = tf.placeholder(tf.float32, shape=[None,2])  # Define a placeholder for the inp
y_placeholder = tf.placeholder(tf.float32, shape=[None,1])  # Define a placeholder for the lab
w = tf.Variable(tf.random_normal([2,1]))  # Random initialize the weight and bias
y_hat = tf.matmul(X_placeholder,w)
loss = tf.reduce_sum(tf.square(y_placeholder-y_hat))  # Define the l2 difference loss

opt = tf.train.GradientDescentOptimizer(0.001)  # Create a gradient descent optimizer
update_op = opt.minimize(loss)  # Tensorflow computes the gradients for you!

sess = tf.Session()  # Defines a session
sess.run(tf.initialize_all_variables())
# Performs gradient descent
for step in range(0,100):
    feed_dict = {X_placeholder:x, y_placeholder:y}
    _, loss_np = sess.run([update_op, loss], feed_dict=feed_dict)
    if step % 5 == 0:
        print("Loss at step %d: %f" % (step, loss_np))
print sess.run(w)
w_np = sess.run(w)
```

```
Loss at step 0: 372.072510
Loss at step 5: 44.258831
Loss at step 10: 11.404384
Loss at step 15: 3.394811
Loss at step 20: 1.372020
Loss at step 25: 0.860742
Loss at step 30: 0.731509
Loss at step 35: 0.698844
```

```
Loss at step 40: 0.690587
Loss at step 45: 0.688500
Loss at step 50: 0.687973
Loss at step 55: 0.687839
Loss at step 60: 0.687805
Loss at step 65: 0.687797
Loss at step 70: 0.687795
Loss at step 75: 0.687794
Loss at step 80: 0.687794
Loss at step 85: 0.687794
Loss at step 90: 0.687794
Loss at step 95: 0.687794
[[ 0.49833012]
 [ 1.09262776]]
```

```
In [20]: plt.scatter(x[:,0],y)
         x_axis = np.arange(-1.5, 1.5,0.001)
         y_predict = w_np[0]*x_axis + w_np[1]
         plt.plot(x_axis,y_predict,'r',linewidth=3)
```

```
Out[20]: [<matplotlib.lines.Line2D at 0x118e42cd0>]
```