DL Lab #1

Spring 2019

Backends

- TensorFlow
- Theano

Frontend

Installation steps (1)

1. Install Python 3.6
2. Append paths to python.exe (aka C:\Python36) and path to directory Scripts (i.e. C:\Python36\Tools\Scripts) in the PATH environment variable:
   - open Control Panel » System » Advanced » Environment Variables
   - update Path variable
   - check by running ‘echo %path%’ in ‘cmd’ mode (Windows) or ‘echo $path’ in Linux

Installation steps (2) - suggested

Install Python virtual environment virtualenv and virtualenvwrapper

1. Use pip (pre-installed with python 3.6 & up)
   - py –m pip install virtualenv (or virtualenvwrapper)
2. Create a Python 3 virtual environment *exclusively* for Keras + TensorFlow-based projects:
   - virtualenv <my_env_name> -p <path to python>

   aka virtualenv env1 –p C:\Python36
3. cd to environment's dir

Installation steps (3)

Install Keras with tensorflow backend

1. Install Keras
   - pip install --upgrade tensorflow
2. Test the installation by opening a Python shell and trying to import the tensorflow package:
   - import tensorflow as tf

Example #1 ‘Hello world” (©Aymeric Damien)

```
import tensorflow as tf

# Simple hello world using TensorFlow

# Create a Constant op
# The op is added as a node to the default graph.
#
# The value returned by the constructor represents the output
# of the Constant op.
hello = tf.constant('Hello, TensorFlow!')

# Start tf session
sess = tf.Session()

# Run the op
print(sess.run(hello))
```

Example #2 Basic ops (1)

```
from __future__ import print_function
import tensorflow as tf
# Basic constant operations, The value returned by the constructor represents the output of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

# Launch the default graph.
with tf.Session() as sess:
    print("a=2, b=3")
    print("Addition with constants: %i" % sess.run(a+b))
    print("Multiplication with constants: %i" % sess.run(a*b))
```

Example #2 Basic ops (2)

Matrix Multiplication. 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.]])

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)

To run the matmul op we call the session 'run()' method, passing 'product‘ which represents the output of the matmul op. # The call 'run(product)' thus causes the execution of threes ops in the

The output of the op is returned in 'result' as a numpy `ndarray` object.

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

Should be ==> [[12.]]

Example #3 Eager API (1)

```
from __future__ import absolute_import, division, print_function
import numpy as np
import tensorflow as tf
import tensorflow.contrib.eager as tfe

# Set Eager API
print("Setting Eager mode...")
tfe.enable_eager_execution()

# Define constant tensors
print("Define constant tensors")
a = tf.constant(2)
print("a = %i" % a)
b = tf.constant(3)
print("b = %i" % b)
```

What is Eager API?

" Eager execution is an imperative, define-by-run interface where operations are executed immediately as they are called from Python. This makes it easier to get started with TensorFlow, and can make research and development more intuitive. A vast majority of the TensorFlow API remains the same whether eager execution is enabled or not. As a result, the exact same code that constructs TensorFlow graphs (e.g. using the layers API) can be executed imperatively by using eager execution. Conversely, most models written with Eager enabled can be converted to a graph that can be further optimized and/or extracted for deployment in production without changing code. " - Rajat Monga

Example #3 Eager API (2)

```
# Run the operation without the need for tf.Session
print("Running operations, without tf.Session")
c = a + b
print("a + b = %i" % c)
d = a * b
print("a * b = %i" % d)
# Full compatibility with Numpy
print("Mixing operations with Tensors and Numpy Arrays")
# Define constant tensors
a = tf.constant([[2., 1.], [1., 0.]], dtype=tf.float32)
print("Tensor:\n a = %s" % a)
b = np.array([[3., 0.],  [5., 1.]], dtype=np.float32)
print("NumpyArray:\n b = %s" % b)
```

Example #3 Eager API (3)

```
# Run the operation without the need for tf.Session
print("Running operations, without tf.Session")

c = a + b
print("a + b = %s" % c)

d = tf.matmul(a, b)
print("a * b = %s" % d)

print("Iterate through Tensor 'a':")
for i in range(a.shape[0]):
    for j in range(a.shape[1]):
        print(a[i][j])
```

Example #4: Linear Regression (1)

```
#[install matplotlib by 'pip install matplotlib']   # do not use eager mode!
from __future__ import print_function

import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

# Parameters
learning_rate = 0.01
training_epochs = 1000
display_step = 50
```

Example #4: Linear Regression (2)

```
# Training Data
train_X = numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                         7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]

# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")
```

Example #4: Linear Regression (3)

```
# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)

# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)

# Gradient descent
#  Note, minimize() knows how to modify W and b because Variable objects are trainable=True by default
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

Example #4: Linear Regression (4)

```
# Start training
with tf.Session() as sess:
    # Run the initializer
    sess.run(init)
    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c), \
                "W=", sess.run(W), "b=", sess.run(b))
    print("Optimization Finished!")
```

Example #4: Linear Regression (4)

```
training_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y})
print("Training cost=", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n')
# Graphic display
plt.plot(train_X, train_Y, 'ro', label='Original data')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
plt.legend()
plt.show()
# Testing example, as requested (Issue #2)
test_X = numpy.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
test_Y = numpy.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])
```

Example #4: Linear Regression (5)

```
print("Testing... (Mean square loss Comparison)")
testing_cost = sess.run(
    tf.reduce_sum(tf.pow(pred - Y, 2)) / (2 * test_X.shape[0]),
    feed_dict={X: test_X, Y: test_Y})  # same function as cost above
print("Testing cost=", testing_cost)
print("Absolute mean square loss difference:", abs(
    training_cost - testing_cost))

plt.plot(test_X, test_Y, 'bo', label='Testing data')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
plt.legend()
plt.show()
```

LR in Keras

- You can look at https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/ and use the code

Example #4: Linear Regression (6)

Now, let's play with it and get different results:

Tasks:

1. Change test data as follows: test_X = [9 digits of your ID], text_Y = [9 digits of your ID, inversed]. Explain what happens
2. Change training data as follows: all Y's are the same. Can you fit training data now? Does it overfit or underfit?
3. Suggest a solution to (3)
4. Now, load the **coffee_data** data set and use LR to predict coffee temperature. Use 66%-33% training-test random data split to do so.
5. In (4), do you overfit or underfit? Explain why.

Example #5: Logistic regression (1) on MNIST

```
from __future__ import print_function
import tensorflow as tf
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

# Parameters
learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 1

# tf Graph Input
x = tf.placeholder(tf.float32, [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder(tf.float32, [None, 10]) # 0-9 digits recognition => 10 classes
```

Example #5: Logistic regression (2)

```
# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct model
pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

Example #5: Logistic regression (3)

```
# Start training
with tf.Session() as sess:
    # Run the initializer
    sess.run(init)
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([optimizer, cost], feed_dict={x: batch_xs,
                                                          y: batch_ys})
```

Example #5: Logistic regression (4)

```
    # Compute average loss
     avg_cost += c / total_batch
    # Display logs per epoch step
    if (epoch+1) % display_step == 0:
        print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

print("Optimization Finished!")

# Test model
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
```

LogR in Keras

- You can use code https://medium.com/@the1ju/simple-logistic-regression-using-keras-249e0cc9a970 as a reference

Example #5: Logistic regression (5)

Now, let’s play with it and get different results (save your results):

Tasks:

1. Change batch size to 10. Does it overfit or underfit? Why?
2. Change batch size to 400. Does it overfit or underfit? Why?
3. Change learning rate to 0.5. Does it overfit or underfit? Why?
4. Now, load the **coffee_data** data set yet again and use LogR to predict coffee temperature, using same split as in LR. Report your accuracy.

Example #6: Random Forest (1) with MNIST

```
from __future__ import print_function
import tensorflow as tf
from tensorflow.contrib.tensor_forest.python import tensor_forest
from tensorflow.python.ops import resources
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
# Parameters
num_steps = 500 # Total steps to train
batch_size = 1024 # The number of samples per batch
num_classes = 10 # The 10 digits
num_features = 784 # Each image is 28x28 pixels
num_trees = 10
max_nodes = 1000
```

Example #6: Random Forest (2)

```
# Input and Target data
X = tf.placeholder(tf.float32, shape=[None, num_features])
# For random forest, labels must be integers (the class id)
Y = tf.placeholder(tf.int32, shape=[None])
# Random Forest Parameters
hparams = tensor_forest.ForestHParams(num_classes=num_classes,
                                      num_features=num_features,
                                      num_trees=num_trees,
                                      max_nodes=max_nodes).fill()
# Build the Random Forest
forest_graph = tensor_forest.RandomForestGraphs(hparams)
# Get training graph and loss
train_op = forest_graph.training_graph(X, Y)
loss_op = forest_graph.training_loss(X, Y)
```

Example #6: Random Forest (3)

```
# Measure the accuracy
infer_op, _, _ = forest_graph.inference_graph(X)
correct_prediction = tf.equal(tf.argmax(infer_op, 1), tf.cast(Y, tf.int64))
accuracy_op = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# Initialize the variables (i.e. assign their default value) and forest resources
init_vars = tf.group(tf.global_variables_initializer(),
    resources.initialize_resources(resources.shared_resources()))

# Start TensorFlow session
sess = tf.Session()

# Run the initializer
sess.run(init_vars)
```

Example #6: Random Forest (4)

```
# Training
for i in range(1, num_steps + 1):
    # Prepare Data
    # Get the next batch of MNIST data (only images are needed, not labels)
    batch_x, batch_y = mnist.train.next_batch(batch_size)
    _, l = sess.run([train_op, loss_op], feed_dict={X: batch_x, Y: batch_y})
    if i % 50 == 0 or i == 1:
        acc = sess.run(accuracy_op, feed_dict={X: batch_x, Y: batch_y})
        print('Step %i, Loss: %f, Acc: %f' % (i, l, acc))

# Test Model
test_x, test_y = mnist.test.images, mnist.test.labels
print("Test Accuracy:", sess.run(accuracy_op, feed_dict={X: test_x, Y: test_y}))
```

RF in Keras

- You can use https://medium.com/analytics-vidhya/build-your-first-neural-network-model-on-a-structured-dataset-using-keras-d9e7de5c6724 as a reference

Example #6: Random Forest (5)

Now, let’s play with it and get different results (save your results):

Tasks:

1. Change number of trees to 5. Does it overfit or underfit? Why?
2. Change max_nodes to 100. Does it overfit or underfit? Why?
3. Change number of steps to 300. Does it overfit or underfit? Why?
4. Load and run RF on *coffee_survey* dataset with # of cups as predicted parameter, using 66%-33% random split . Save your chart and report accuracy.

Result submission

Fill the following table for all algorithms and tasks and submit in moodle (.pdf format):

Algorithm	Task #	Results/answer	Your explanation
Linear regression	1-5	…	…
Logistic regression	1-4	…	…
Random forest	1-4		