INSTRUCTOR
PROF. VOLKAN CEVHER

HEAD TA
YA-PING HSIEH

TENSORFLOW TUTORIAL (FOR EXERCISE 2)

This tutorial introduces the basics of the deep learning framework **TensorFlow 1.10** required for **Exercise 2**.

1 Installing TensorFlow

The installation of TensorFlow is straightforward via a package manager like *conda* (`https://conda.io`) or *pip*. The exact installation command depends on your OS, package manager, python version, and whether you have a GPU with CUDA support. In TensorFlow's installation page (`https://www.tensorflow.org/install/`) you can find detailed instructions in your particular case. For example, if you use the conda package manager, running the following command in a Linux terminal

```
conda install -c conda-forge tensorflow
```

will install TensorFlow without GPU support. On the other hand, using the python's pip you would run

```
pip install tensorflow
```

2 Defining a neural network architecture

In TensorFlow it is easy to define a network architecture. The idea is to create a *computational graph* by defining how the input data is transformed and finally how the loss function is computed. First we define *placeholders*, objects that define where the input data will flow into the computational graph.

```python
import tensorflow as tf

input_dim = 784
classes = 10
x = tf.placeholder(tf.float32, shape=[None, input_dim])
y = tf.placeholder(tf.float32, shape=[None, classes])
```

now we can define the weights and biases of our network as *variables*, and we describe how the output of the network and the loss function are computed, using TensorFlow's available operations

```python
hidden = 100

W1 = tf.Variable(tf.zeros([input_dim, hidden]))
b1 = tf.Variable(tf.zeros([hidden]))

W2 = tf.Variable(tf.zeros([hidden, classes]))
b2 = tf.Variable(tf.zeros([classes]))

output = tf.add(tf.matmul(x, W1), b1)
output = tf.nn.relu(output)
output = tf.add(tf.matmul(output, W2), b2)

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
    logits=output, labels=y))
```

In this case we chose the cross entropy loss as our loss function.

3 Loading the training data

TensorFlow includes the MNIST dataset and we can easily loop over it in batches in the following way

```
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
batch_size = 100
n_batches = int(mnist.train.num_examples / batch_size)

for i in range(n_batches):
    batch_x, batch_y = mnist.train.next_batch(batch_size)
    print('batch size: ' + str(x.shape[0]))
    print('input dimension: ' + str(x[0].shape))
```

4 Evaluating the computational graph in some data

Now that we have a neural network architecture, a loss function and some labeled data, we can run computations in a TensorFlow *session*:

```
epochs = 5
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init) # Initialize weights and biases

    for epoch in range(epochs):
        total_loss = 0
        for i in range(n_batches):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            batch_loss = sess.run([loss], feed_dict={x: batch_x, y:batch_y})[0]
            total_loss += batch_loss
        print('total loss: ' + str(total_loss))
```

notice that via the `feed_dict` parameter of the session's `run` method, we specify the value that the placeholders we defined earlier will take. In this case the placeholders `x,y` that represent the input data and their labels, are replaced by the mini-batch data and labels `batch_x, batch_y`.

5 Computing the gradients of the loss function

We can compute the gradient of the loss functions using TensorFlow's `tf.train.GradientDescentOptimizer`.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
grads_and_vars = optimizer.compute_gradients(loss)
train = optimizer.apply_gradients(grads_and_vars)

with tf.Session() as sess:
    sess.run(init) # Initialize weights and biases

    for epoch in range(epochs):
        for i in range(n_batches):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            _, batch_loss = sess.run([train, loss], feed_dict={x: batch_x, y:batch_y})
            print('mini-batch loss: ' + str(batch_loss))
```

6 Some useful low-level functions

- `tf.zeros(shape)`: create a tensor of zeros of a particular shape.
- `tf.zeros_like(input_tensor)`: create a tensor of zeros of the same shape as an input tensor.

- `tf.ones(shape), tf.ones_like(input_tensor)`: similar to `tf.zeros, tf.zeros_like`.
- `tf.add, tf.multiply, tf.divide`: tensor equivalents of the binary operations `+, *, /`.
- `tf.sqrt`: element-wise square root of a tensor.
- see `https://www.tensorflow.org/api_docs/python/tf` for a complete review of TensorFlow low-level functions.