Multilayer Perceptron

Trong bài này, sinh viên được hướng dẫn sử dụng Tensorflow để hiện thực Multilayer Perceptron (neural network).

Code được tham khảo từ một số nguồn sau đây:

- Aymeric Damien (aymericdamien) (https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/multilayer_perceptron.py)

Dataset: the MNIST database of handwritten digits (http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/)).
Links: MNIST Dataset (http://yann.lecun.com/exdb/mnist/).

In [ ]:
```
from __future__ import print_function

import tensorflow as tf
```

In [ ]:
```
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data

# load dataset
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

In [ ]:
```
# Parameters
learning_rate = 0.001
training_epochs = 30
batch_size = 100
display_step = 1

# Network Parameters
n_hidden_1 = 256 # 1st layer number of neurons
n_hidden_2 = 256 # 2nd layer number of neurons
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)
```

In [ ]:
```
# tf Graph input
X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_classes])
```

Cách đầu tiên, sử dụng các phép toán đơn giản để định nghĩa MLP

In [ ]:
```
# Store layers weight & bias
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes]))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}


# Create model
def multilayer_perceptron(x):
    # Hidden fully connected layer with 256 neurons
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    # Hidden fully connected layer with 256 neurons
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    # Output fully connected layer with a neuron for each class
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
    return out_layer
```

Định nghĩa MLP gọn hơn với các phép toán đã được tensorflow hỗ trợ sẵn, ở đây ta dùng *tf.layers.dense*

Sinh viên chỉ dùng một trong 2 version của hàm multilayer_perceptron, ở đây hoặc ở phía trên

In [ ]:
```
def multilayer_perceptron(x):
    # Hidden fully connected layer with 256 neurons
    layer_1 = tf.layers.dense(x, units=n_hidden_1, kernel_initializer=tf.contrib.la
yers.xavier_initializer())
    # Hidden fully connected layer with 256 neurons
    layer_2 = tf.layers.dense(layer_1, units=n_hidden_2, kernel_initializer=tf.cont
rib.layers.xavier_initializer())
    # Output fully connected layer with a neuron for each class
    out_layer = tf.layers.dense(layer_2, units=n_classes, kernel_initializer=tf.con
trib.layers.xavier_initializer())
    return out_layer
```

Ở đây, thay vì khởi tạo các matrix weights và biases với random_normal, ta dùng xavier_initializer, nó có ảnh hưởng nhất định tới tốc độ hội tụ của model

Ở đây, chúng ta bắt đầu cấu trúc graph, định nghĩa loss (cost) và optimizer để đưa vào training

In [ ]:
```
# Construct model
logits = multilayer_perceptron(X)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, lab
els=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
# Initializing the variables
init = tf.global_variables_initializer()
```

Train model và test để xem kết quả

In [ ]:
```
with tf.Session() as sess:
    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_x, batch_y = mnist.train.next_batch(batch_size)
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, loss_op], feed_dict={X: batch_x,
                                                            Y: batch_y})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(avg_cost))
    print("Optimization Finished!")

    # Test model
    pred = tf.nn.softmax(logits)  # Apply softmax to logits
    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(Y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print("Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels}))
```

Bài tập

Bài nộp của sinh viên là chính là **file này** sau khi được đổi tên thành **MSSV.E10_Multilayer_Perceptron.ipynb** và đừng quên ghi thông tin sinh viên vào các ô ở dưới.

Địa chỉ nộp bài: https://www.dropbox.com/request/ny4gTnGl290Fs64ycR0b (https://www.dropbox.com/request/ny4gTnGl290Fs64ycR0b)

Deadline nộp bài: **10:00 thứ 3 tuần tiếp theo**

Điểm bài này sẽ được tổng hợp với điểm chấm trên lớp (nếu có) để ra điểm cuối cùng, do đó sinh viên vắng học trên lớp dù làm bài có toàn vẹn cũng có thể nhận được điểm thấp hơn điểm tối đa

Thông tin sinh viên:

In [ ]:
```
sid = 'MSSV'
name = 'Tên Bạn'

import tensorflow as tf
import numpy as np

# sinh viên import các thư viện cần thiết ở đây
```

Bài 1

Trong ví dụ ở trên, chúng ta đã sử dụng toàn bộ tập dữ liệu để train, và đồng thời cũng lấy chính dữ liệu đó để test, điều đó dẫn tới trường hợp "học khớp" (overfitting).

Sinh viên hãy chia tập dữ liệu ra làm 2 phần riêng biệt ~90% để train và ~10% còn lại dùng để test. (sửa trực tiếp ở code ví dụ ở trên)

Hãy cho biết kích thước của mỗi tập sau khi tách ra

Hãy cho biết các nhận xét của kết quả khi chia ra 2 tập rời như vậy

Bài 2

Hãy sử dụng tập dữ liệu cifar 10 (https://www.cs.toronto.edu/~kriz/cifar.html)

Hãy xử lý dữ liệu, thay đổi model (nếu cần) và điều chỉnh các thông số cần thiết phù hợp với dữ liệu này.

Sinh viên hãy train, test và cho biết kết quả đạt được (yêu cầu giữ kết quả output khi chạy train, test lúc nộp bài ở các cell tiếp theo ở dưới

In [ ]:
```
# data load, pre-processing
```

In [ ]:
```
# model (if need)
```

In [ ]:
```
# train, test (keep output when submit your exercise)
with tf.Session() as sess:
    # train, test code here
    pass
```

Bài 3

Các bạn đã train để tạo ra được model sử dụng trong classification đơn giản, tuy nhiên chúng ta chưa lưu lại các kết quả đạt được để tái sử dụng, do đó, mỗi lần test lại sẽ cần train lại mô hình, tốn kém thời gian.

Hãy tìm hiểu và hiện thực (vào code mẫu ở trên) để lưu lại model và tái sử dụng lại sau này.