Example 2: classify digits 0-9 using fully-connected ANN using one-hot encoding of classes and softmax activation

with help from: https://liufuyang.github.io/2017/03/17/just-another-tensorflow-beginner-guide-2.html

```
import numpy as np
import tensorflow as tf
import math

import matplotlib.pyplot as plt
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
!ls '/content/gdrive/My Drive/Colab Notebooks/'

path_to_data = '/content/gdrive/My Drive/Colab Notebooks/'
```

mount your google drive folder

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien

Enter your authorization code:
··········
Mounted at /content/gdrive
ASL_X.npy                                       mnist_train_images.npy
ASL_Y.npy                                       mnist_train_labels.npy
digits_0to9_ANN_NgNotation.ipynb                tmp
digits_0to9_logisticRegression_NgNotation.ipynb
```

here how to get your code

```
from datetime import datetime
import time


# use skikits learn helper function to split data
from sklearn.model_selection import train_test_split
```

```
# progress bar and loop timing
# from tqdm import tnrange, tqdm_notebook
from tqdm import trange
```

fix for progress bar

```
# reset everything
tf.reset_default_graph()
np.random.seed(1)
seed = 3
```

```
# configuration
batch_size = 64
learning_rate = 0.0001
training_epochs = 25
```

functions

```
def random_mini_batches(X, Y, mini_batch_size = 64, seed = 0):
    """
    Creates a list of random minibatches from (X, Y)

    (Check: will only work with flattened images because of indexing?)

    Arguments:
    X -- input data, of shape (input size, number of examples)
    Y -- true "label" vector (containing 0 if cat, 1 if non-cat), of shape (1, numb
    mini_batch_size - size of the mini-batches, integer
    seed -- this is only for the purpose of grading, so that you're "random minibat

    Returns:
    mini_batches -- list of synchronous (mini_batch_X, mini_batch_Y)

    from Andrew Ng
    """

    m = X.shape[1]                  # number of training examples
    mini_batches = []
    np.random.seed(seed)

    # Step 1: Shuffle (X, Y)
    permutation = list(np.random.permutation(m))
    shuffled_X = X[:, permutation]
    shuffled_Y = Y[:, permutation].reshape((Y.shape[0],m))

    # Step 2: Partition (shuffled_X, shuffled_Y). Minus the end case.
    num_complete_minibatches = math.floor(m/mini_batch_size) # number of mini batch
    for k in range(0, num_complete_minibatches):
        mini_batch_X = shuffled_X[:, k * mini_batch_size : k * mini_batch_size + mi
        mini_batch_Y = shuffled_Y[:, k * mini_batch_size : k * mini_batch_size + mi
        mini_batch = (mini_batch_X, mini_batch_Y)
        mini_batches.append(mini_batch)

    # Handling the end case (last mini-batch < mini_batch_size)
    if m % mini_batch_size != 0:
        mini_batch_X = shuffled_X[:, num_complete_minibatches * mini_batch_size : m
        mini_batch_Y = shuffled_Y[:, num_complete_minibatches * mini_batch_size : m
        mini_batch = (mini_batch_X, mini_batch_Y)
        mini_batches.append(mini_batch)

    return mini_batches
```

```
def one_hot_matrix(labels, C):
    """
    Creates a matrix where the i-th row corresponds to the ith class number and the
                     corresponds to the jth training example. So if example j had a
                     will be 1.

    Arguments:
    labels -- vector containing the labels
    C -- number of classes, the depth of the one hot dimension

    Returns:
    one_hot -- one hot matrix
```

```
    from Andrew Ng
    """

    # Create a tf.constant equal to C (depth), name it 'C'.
    C = tf.constant(C, name = "C")

    # Use tf.one_hot, be careful with the axis
    one_hot_matrix = tf.one_hot(labels, C, axis=0)

    # Create and run the session
    with tf.Session() as sess:
        one_hot = sess.run(one_hot_matrix)

    return one_hot


def shuffle_in_unison(X, y):
    shuffled_X = np.empty(X.shape, dtype=X.dtype)
    shuffled_y = np.empty(y.shape, dtype=y.dtype)
    permutation = np.random.permutation(X.shape[0])
    for old_index, new_index in enumerate(permutation):
        shuffled_X[new_index,:,:] = X[old_index,:,:]
        shuffled_y[new_index,:] = y[old_index,:]
    return shuffled_X, shuffled_y
```

load dataset

```
# these examples are gray scale images
# nc = 1
# and are for digits 0-9 --> 10 classes with one-hot encoding
n_classes = 10

#################################################################
# try this dataset
# ASL digits 0-9

# X_data = np.load('ASL_X.npy').astype(np.float32)
# y_data = np.load('ASL_Y.npy').astype(np.float32)
# y_data = one_hot_matrix(y_data,C=n_classes).T

# X_data, y_data = shuffle_in_unison(X_data,y_data) # this dataset is ordered, so m
# m,nh,nw = X_data.shape

#################################################################
# or this dataset
# MNIST digits 0-9
X_data = np.load(path_to_data+'mnist_train_images.npy').astype(np.float32) / 255.0
y_data = np.load(path_to_data+'mnist_train_labels.npy').astype(np.float32).squeeze(
y_data = one_hot_matrix(y_data,C=n_classes).T

m,hxw = X_data.shape
nh = np.int(np.sqrt(hxw))
nw = np.int(np.sqrt(hxw))
X_data = np.reshape(X_data,(m,nh,nw))


print (X_data.shape)
print (y_data.shape)
# print (hxw)
```

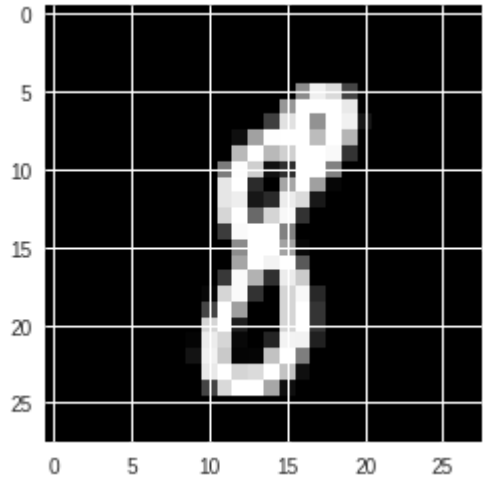
```
(60000, 28, 28)
(60000, 10)
```

path to data files

take a look at one example

```
plt.imshow(X_data[300],cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f8ff5b37908>
```




```
print (y_data[300])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

split data into test / train sets

```
X_train, X_test, y_train, y_test = train_test_split(X_data,y_data, test_size=0.33,
                                                    shuffle=True, random_state = 42
num_examples = X_train.shape[0]

print (X_train.shape, y_train.shape)
print (y_test.shape, y_test.shape)
```

```
(40200, 28, 28) (40200, 10)
(19800, 10) (19800, 10)
```

reshape arrays to match Andrew Ng's notation and flatten 2-D images to 1-D features

```
# many data sets use X[m,nh,nw,nc] and tesnsorflow and scikits learn use m as first
# Andrew Ng, uses X[nh,nw,nc,m] to make matrix algebra make text-book sense
# I propose to follow Andrew Ng until I get comfortable with notation

# X_data = X_data.reshape(m,nh,nw,1)
X_train = X_train.reshape(X_train.shape[0],-1).T
X_test = X_test.reshape(X_test.shape[0],-1).T
y_train = y_train.T
y_test = y_test.T

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(784, 40200) (10, 40200)
(784, 19800) (10, 19800)
```

data input and formatted properly!

3 layer ANN

start up tensorboard with $ tensorboard --logdir=./tmp/example --port=8002 --reload_interval=5

open browser at http://localhost:8002/ then you should be able to see the computation graph

```
# tensorboard log file path
logs_path = path_to_data+'/tmp/example2/'+datetime.now().isoformat()


# reset everything
tf.reset_default_graph()
np.random.seed(1)
seed = 3
```

tensorflow constants, variables, placeholders

```
with tf.name_scope('input'):
    X = tf.placeholder(tf.float32, shape=[nh*nw, None], name="X")
    y_true = tf.placeholder(tf.float32, shape=[n_classes, None], name="y_true")


with tf.name_scope('weights'):
    W1 = tf.get_variable("W1", [25,nh*nw], initializer = tf.contrib.layers.xavier_i
    W2 = tf.get_variable("W2", [12,25], initializer = tf.contrib.layers.xavier_init
    W3 = tf.get_variable("W3", [n_classes,12], initializer = tf.contrib.layers.xavi


with tf.name_scope('biases'):
    b1 = tf.get_variable("b1", [25,1], initializer = tf.zeros_initializer())
    b2 = tf.get_variable("b2", [12,1], initializer = tf.zeros_initializer())
    b3 = tf.get_variable("b3", [n_classes,1], initializer = tf.zeros_initializer())
```

define computation graph: simple logistic regression

```
with tf.name_scope('layer1'):
    z1 = tf.add(tf.matmul(W1,X),b1)
    a1 = tf.nn.relu(z1)
with tf.name_scope('layer2'):
    z2 = tf.add(tf.matmul(W2,a1),b2)
```

```
    a2 = tf.nn.relu(z2)
with tf.name_scope('softmax'):
    z3 = tf.add(tf.matmul(W3,a2),b3)
    y  = tf.transpose(tf.nn.softmax(tf.transpose(z3)))

# instead do this if we use "softmax_cross_entropy" function for cost below
# with tf.name_scope('softmax'):
#     y = tf.matmul(X,W) + b
print(y)
```

```
Tensor("softmax/transpose_1:0", shape=(10, ?), dtype=float32)
```

cost function

```
with tf.name_scope('cross_entropy'):
    cost = tf.reduce_mean(-tf.reduce_sum(y_true * tf.log(y)))

# with tf.name_scope('cross_entropy'):
#     cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=tf.tr
#                                                                      logits=tf.tr
```

optimizer

```
with tf.name_scope('train'):
#     optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    train = optimizer.minimize(cost)
```

metrics

```
with tf.name_scope('accuracy'):
    correct_predictions = tf.equal(tf.argmax(y,0),tf.argmax(y_true,0))
    accuracy = tf.reduce_mean(tf.cast(correct_predictions,tf.float32))
```

log results

```
# summary for tensorboard
train_cost_summary = tf.summary.scalar("train_cost", cost)
train_accuracy_summary = tf.summary.scalar("train_accuracy", accuracy)

test_cost_summary = tf.summary.scalar("test_cost", cost)
test_accuracy_summary = tf.summary.scalar("test_accuracy", accuracy)
```

main program

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

```
with tf.Session() as sess:
    sess.run(init)
```

```
    # tensorboard log file writer
    writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())

    batch_count = np.int(num_examples/batch_size)
    print ('number of mini_batches: ',batch_count)

#     for epoch in tnrange(training_epochs, desc='epoch'):
    for epoch in trange(training_epochs, desc='epoch'):
        time.sleep(0.1)

        seed = seed+1
        mini_batches = random_mini_batches(X_train, y_train, batch_size, seed=seed)

        iter = 0
        for mini_batch in mini_batches:

            iter = iter+1

            (batch_x, batch_y) = mini_batch

            _, train_cost, train_accuracy, _train_cost_summary, _train_accuracy_sum
               sess.run([train, cost, accuracy, train_cost_summary, train_accuracy_
                   feed_dict={X: batch_x, y_true: batch_y})

            writer.add_summary(_train_cost_summary, epoch * batch_count + iter)
            writer.add_summary(_train_accuracy_summary, epoch * batch_count + iter)

            if iter % 100 == 0:
                # for log on test data:
                test_cost, test_accuracy, _test_cost_summary, _test_accuracy_summar
                    sess.run([cost, accuracy, test_cost_summary, test_accuracy_summ
                        feed_dict={X:X_test, y_true:y_test})
                # write log
                writer.add_summary(_test_cost_summary, epoch * batch_count + iter)
                writer.add_summary(_test_accuracy_summary, epoch * batch_count + it

#                 print('Epoch {0:3d}, Batch {1:3d} | Train Cost: {2:.2f} | Test Co

    print('Final Test Set Accuracy: {}'.format(accuracy.eval(feed_dict={X:X_test, y
```

make progress bar work ok

```
epoch:   0%|          | 0/25 [00:00<?, ?it/s]number of mini_batches:  628
epoch: 100%|██████████| 25/25 [00:55<00:00,  2.21s/it]Final Test Set Accuracy
```