Assignment 1. Handwritten Digit Recognition using TensorFlow and MNIST dataset

CSE 598. Assignment 1. Handwritten Digit Recognition using TensorFlow and MNIST dataset

Due date: Jan 24 2020

Files to submit:

- A report in PDF format. For each model that you have tested, please describe the model, the graph representation captured in TensorBoard, the accuracy you achieved and brief analysis, time you spent to complete this task and interesting problems that you've met;
- **a1a.py** Python source code for logistic regression. You need add your code to complete the functionality;
- a1b.py Additional python source code for other models like DNN (deep neural network), CNN (convolutional neural network) and so on. You need create the file.
- **utils.py** Python source code for utility functions that will be used by a1a.py. You do not need modify this file.

Submission Website:

GradeScope (We will NOT accept submissions via email and Canvas).

If you have any questions regarding GradeScope, please contact TA.

Learning Goal:

1. How to prepare image data for training?
   - batching
2. How to use a high-level framework (TensorFlow) to create a neural model?
   - simple logistic regression
   - fully-connected and convolution layers
   - stacking layers
3. How to train the model?
   - typical training loop
   - model evaluation

Platform: TensorFlow

Please follow the official instruction to install TensorFlow **here** **(https://www.tensorflow.org/install/)** . You're welcome to use either Python 2 or Python 3 for the assignments, but Python 3 will be recommended.

A reference list of dependencies:

tensorflow==1.4.1
numpy==1.18.1
scipy==1.0.0
scikit-learn==0.19.1
matplotlib==2.1.1
xlrd==1.1.0
ipdb==0.10.3
Pillow==5.0.0
lxml==4.1.1

We can encode above into a **requirements.txt** . In ubuntu, you can simply run following commands:

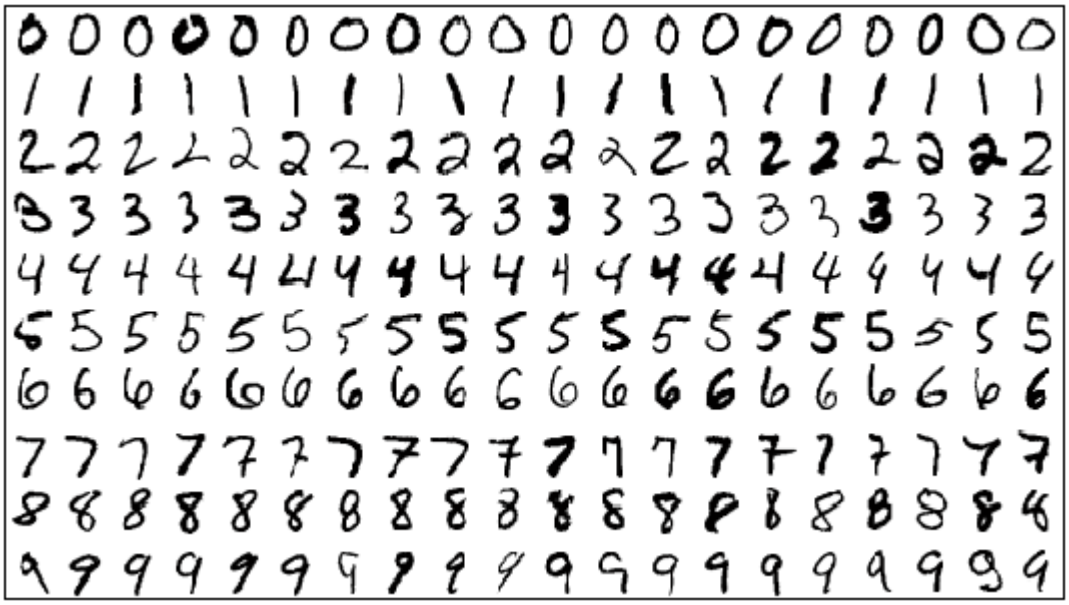
```
sudo apt-get install python3

sudo apt-get install python3-pip

sudo apt-get install python3-tk

pip3 install -r requirements.txt
```

Then you can try to run **logreg_example.py** using following command to verify whether tensorflow is correctly installled.

```
python3  logreg_example.py
```

Dataset: MNIST

The MNIST (Mixed National Institute of Standards and Technology database) is one of the most popular databases used for training various image processing systems. It is a database of handwritten digits. The images look like this:



Each image is 28 x 28 pixels. You can flatten each image to be a 1-d tensor of size 784. Each comes with a label from 0 to 9. For example, images on the first row is labelled as 0, the second as 1, and so on. The dataset is hosted on Yann Lecun's website **(http://yann.lecun.com/exdb/mnist/)** .

Data Loading Approach 1.

TF Learn (the simplified interface of TensorFlow) has a script that lets you load the MNIST dataset from Yann Lecun's website and divide it into train set, validation set, and test set.

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('data/mnist', one_hot=True)
```

One-hot encoding

In digital circuits, one-hot refers to a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0).

In this case, one-hot encoding means that if the output of the image is the digit 7, then the output will be encoded as a vector of 10 elements with all elements being 0, except for the element at index 7 which is 1.

input_data.read_data_sets('data/mnist', one_hot=True)

Above statement returns an instance of learn.datasets.base.Datasets, which contains three generators to 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation). You get the samples of these datasets by calling next_batch(batch_size), for example, mnist.train.next_batch(batch_size) with a batch_size of your choice.

Data Loading Approach 2.

You can use the provided utils.py, which implemented functions

downloading and parsing MNIST data into numpy arrays in the file utils.py. All you need to do in your program is:

```
mnist_folder = 'data/mnist'
utils.download_mnist(mnist_folder)
train, val, test = utils.read_mnist(mnist_folder, flatten=True)
```

We need choose flatten = True for logistics regression and DNN, because we want each image to be flattened into a 1-d tensor. Each of train, val, and test in this case is a tuple of NumPy arrays, the first is a NumPy array of images, the second of labels. We need to create two Dataset objects, one for train set and one for test set (in this example, we won't be using val set).

```
train_data = tf.data.Dataset.from_tensor_slices(train)
# train_data = train_data.shuffle(10000) # if you want to shuffle your data
test_data = tf.data.Dataset.from_tensor_slices(test)
```

However, now we have A LOT more data. If we calculate gradient after every single data point it'd be painfully slow. Fortunately, we can process the data in batches.

```
train_data = train_data.batch(batch_size)
test_data = test_data.batch(batch_size)
```

The next step is to create an iterator to get samples from the two datasets. In the linear regression example, we used only the train set, so it was okay to create an iterator for that dataset and just draw samples from that dataset. When we have more than one dataset, if we have one iterator for each dataset, we would need to build one graph for each iterator! A better way to do it is to create one single iterator and initialize it with a dataset when we need to draw data from that dataset.

```
iterator = tf.data.Iterator.from_structure(train_data.output_types,
                                           train_data.output_shapes)
img, label = iterator.get_next()
train_init = iterator.make_initializer(train_data) # initializer for train_data
test_init  = iterator.make_initializer(test_data) # initializer for test_data
 ...
with tf.Session() as sess:
```

```
...
for i in range(n_epochs):       # train the model n_epochs times
    sess.run(train_init)        # drawing samples from train_data
    try:
        while True:
            _, l = sess.run([optimizer, loss])
    except tf.errors.OutOfRangeError:
        pass
# test the model
sess.run(test_init) # drawing samples from test_data
try:
    while True:
        sess.run(accuracy)
except tf.errors.OutOfRangeError:
    pass
```

Task 1. Using logistic regression to classify image data

You need fill in your code to **a1a.py**, which is a skeleton of logistic regression using Data Loading Approach 2 as described above.

Please first read and understand a1a.py. Try to complete the code by yourself. Note that a1a.py will use functions defined in **utils.py**.

There is some **example code** that uses Data Loading Approach 1.

If you meet problems and really cannot solve the problem, you can take a look at this **hint list**.

Task 2. Improve the model of Task 1.

We got the accuracy of ~91% on our MNIST dataset with our vanilla model, which is unacceptable. The state of the art is above 99% **(http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)**. You have full

freedom to do whatever you want here, e.g. to use a different model like DNN or CNN, as long as your model is built in TensorFlow.

You can reuse the code from part 1, but please save your code for part 2 in a separate file and name it a1b.py. Anything above 97% accuracy will get a bonus point that will be directly added to your final score.

Directly copying code from internet or other students will get 0 points for this assignment.

Task 3. Write a report

In the report, FOR EACH MODEL that you have tested in TASK1 and TASK 2, please:

- describe the model;
- paste a picture of the graph representation captured in TensorBoard; report the accuracy the model has achieved on MNIST dataset;
- the time you spent to complete task 1 and task 2;
- and interesting problems that you've met during this assignment.

Please export the report as a PDF file.

To use TensorBoard in Linux:

In the directory where you run the python script, type following command:

tensorboard --logdir .

Then you can view the graph in following website:

localhost:6006

Grading Criteria:

total points: 20

- a1a.py can run correctly: 4 points
- a1b.py can run correctly: 4 points
- a1b.py have better accuracy than a1a.py: 4 points
- report correctly describes models in a1a.py and a1b.py: 2 points
- report contains the graphs captured from tensorboard for a1a.py and a1b.py: 3 points
- report contains accuracy and analysis: 1 point
- report contains total estimated time spent in completing this assignment: 1 point
- report contains descriptions of problems met: 1 point

If a1b.py accuracy > 97%, we will give you 1 bonus point that adds directly to your final grade for this course.

Autograder Results

Results | Code

This assignment does not have an autograder configured.

STUDENT
Parth Rajendra Doshi

AUTOGRADER SCORE
0.0 / 0.0

QUESTION 2
a1a.py can run correctly — **4.0** / 4.0 pts

QUESTION 3
a1b.py can run correctly — **4.0** / 4.0 pts

QUESTION 4
report correctly describes models in a1a.py and a1b.py — **2.0** / 2.0 pts