DEEP LEARNING

TENSORFLOW

OVERVIEW

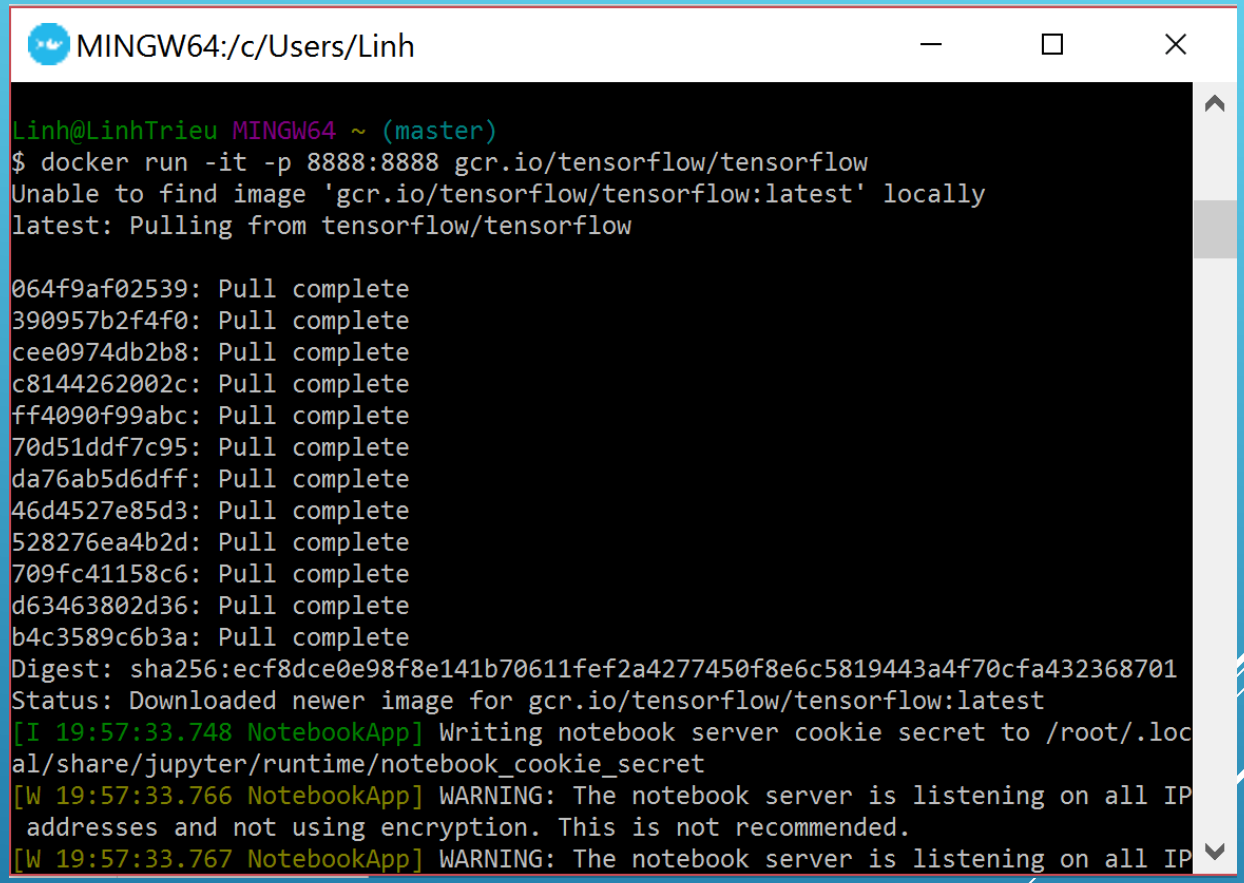
- MNIST Dataset
  - MNIST is a labeled set of images of handwritten digits.
  - Google's AI tool TensorFlow is used in this project to train the traditional dataset MNIST where the model looks at images and predict what digits they are.

- TensorFlow

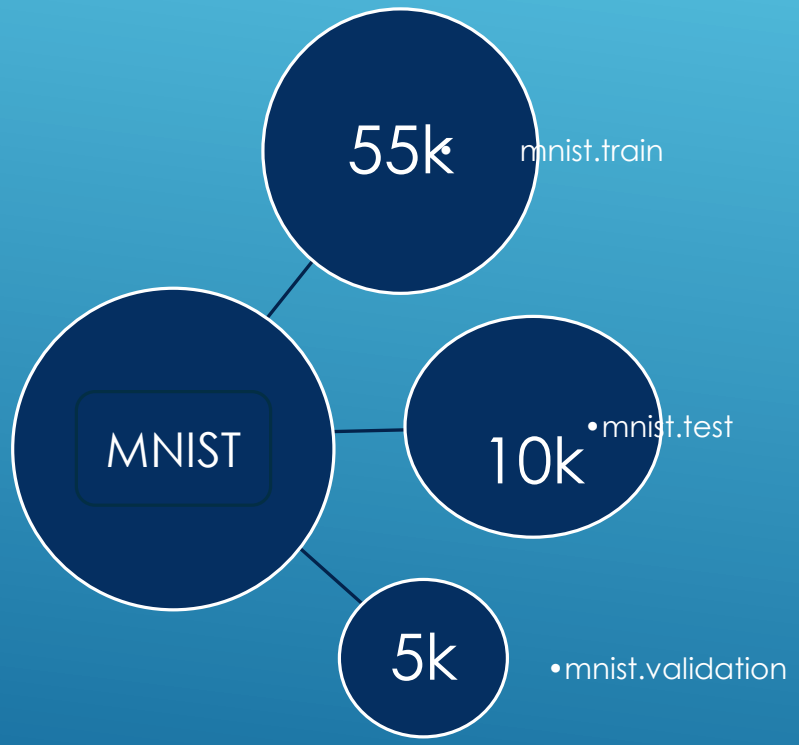
  This presentation is not necessarily building the best model, but rather showing my first training model leveraging powerful tool TensorFlow directly on my computer.

DOCKER

On Windows environment, I installed Docker, launched docker container, and ran Ipython Notebook



LOAD DATASET MNIST



- Imported tensorflow into ipynb
- Loaded dataset as mnist

```
In [1]: from __future__ import absolute_import
        from __future__ import division
        from __future__ import print_function
        import argparse

In [54]: from tensorflow.examples.tutorials.mnist import input_data
         mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

         import tensorflow as tf

         Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
         Extracting MNIST_data/train-images-idx3-ubyte.gz
         Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
         Extracting MNIST_data/train-labels-idx1-ubyte.gz
         Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
         Extracting MNIST_data/t10k-images-idx3-ubyte.gz
         Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
         Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

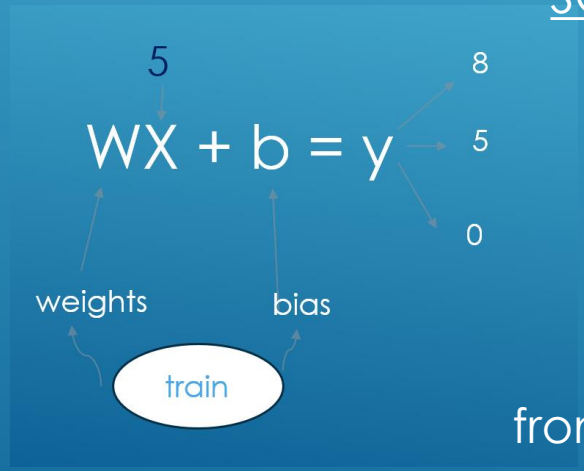
SOFTMAX

- Flattened array into a vector of 28*28 = 748 numbers
- Each image has 28pixels by 28 pixels.

- Declared variables to the model for recognizing digits by looking at every pixel in the image

```
In [56]: # Create the model
         x = tf.placeholder(tf.float32, [None, 784])
         W = tf.Variable(tf.zeros([784, 10]))
         b = tf.Variable(tf.zeros([10]))
         y = tf.nn.softmax(tf.matmul(x, W) + b)
```

- Model function:



SCORE

2.0

3.0

1.0

SOFTMAX

from SCORE TO PROBABILITY

PROBABILITY

y

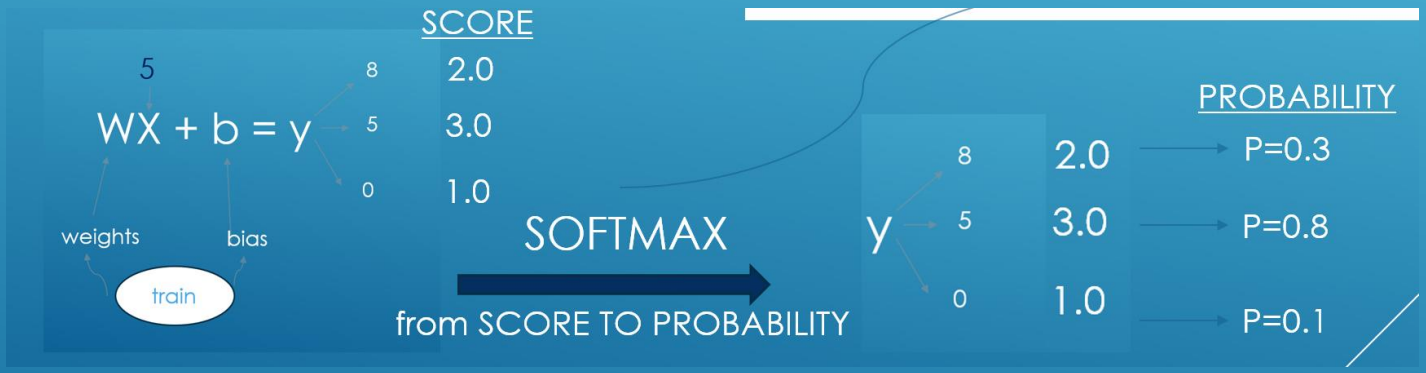
8 → 2.0 → P=0.3

5 → 3.0 → P=0.8

0 → 1.0 → P=0.1

ONE-HOT ENCODING

- Set the probability of the correct class (P=0.8) be close to 1
- For all others, be close to 0



one_hot=True

0.0

1.0 = 5

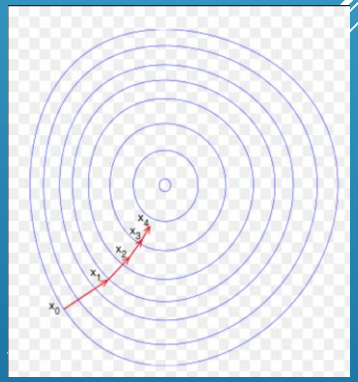
0.0

CROSS-ENTROPY

```
In [57]: # Define loss and optimizer
         y_ = tf.placeholder(tf.float32, [None, 10])
         cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
         train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

So far, model looks like this

- The step above is called Multinomial Logistic Classification
- Cross-entropy function: distance D(S(WX+b), L)
- Loss is the average of cross-entropy, which is lots of sums and matrices
- Gradient descent algorithm is chose to find the least loss with learning rate at 0.5



SCHOLASTIC GRADIENT DESCENT

```
In [59]: init = tf.initialize_all_variables()
         sess = tf.Session()
         sess.run(init)
         for i in range(1000):
           batch_xs, batch_ys = mnist.train.next_batch(100)
           sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

- As loss function is very huge and dependent on all single element in the training set, it uses lots of compute to train the data.
- Instead, batch included 100 random data points was used, many times, to increase efficiency.
- TensorFlow session was initialized

MODEL EVALUATION

```
In [60]: correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

         0.9092
```

First time: 90.9% accuracy

```
In [63]: correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

         0.9189
```

Second time: 91.9% accuracy

```
In [79]: correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

         0.9224
```

Third time: 92.2% accuracy

REFERENCE

This project follows instructions on www.tensorflow.com

Linh Trieu
Advanced Business Intelligence
The University of Texas at Dallas