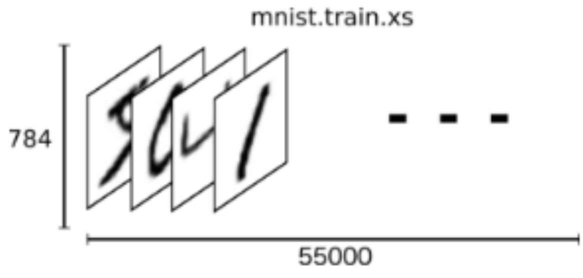
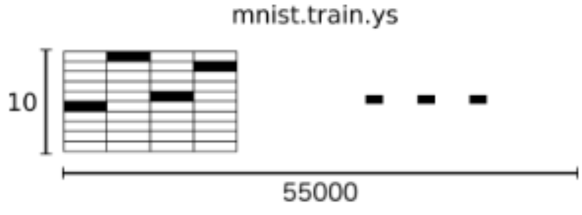
DNN - MNIST for handwritten character recognition

https://youtu.be/pnSBZ6TEVjY

- Automated handwritten digit recognition is widely used today - from recognizing zip codes on mail envelopes to recognizing amounts written on bank checks
- We want to use a *MNIST* dataset in TensorFlow
- MNIST is a database of 55000 training examples (mnist.train) of labeled handwritten digits '0' to '9' from 250 writers, formatted in 28x28 px matrix (each pixel is encoded as a unsigned byte). The dataset includes also 10000 labeled testing samples (mnist.test) from a different set of writers
- If 'x' is the image and 'y' is the label, then mnist.train.x and mnist.train.y are two tensors



rank is 2, shape is [55000, 784]



rank is 2, shape is [55000, 10]

- If our model does not take into consideration the 2D structure of the images, it is convenient to flatten a 28x28 square matrix as a 784 pixel vector. The pixel order is not important because all images are consistent
- Let's create the computation graph, or the DNN model

deep-net.py

```python
# Python
import os
os.environ ['TF_CPP_MIN_LOG_LEVEL']='2'

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

n_nodes_hl1 = 500
n_nodes_hl2 = 500
n_nodes_hl3 = 500

n_classes = 10  #the number of output nodes
batch_size = 100

x = tf.placeholder('float', [None, 784])    #these are the 28x28 px samples
y = tf.placeholder('float')         #these are the labels

# this function describes the feedforward process
def neural_network_model(data):
    # out = (wheights * in) + bias
    hidden_1_layer = {'weights': tf.Variable(tf.random_normal([784, n_nodes_hl1])),
                'biases': tf.Variable(tf.random_normal([n_nodes_hl1]))}

    hidden_2_layer = {'weights': tf.Variable(tf.random_normal([n_nodes_hl1, n_nodes_hl2])),
                'biases': tf.Variable(tf.random_normal([n_nodes_hl2]))}

    hidden_3_layer = {'weights': tf.Variable(tf.random_normal([n_nodes_hl2, n_nodes_hl3])),
                'biases': tf.Variable(tf.random_normal([n_nodes_hl3]))}

    output_layer = {'weights': tf.Variable(tf.random_normal([n_nodes_hl3, n_classes])),
                'biases': tf.Variable(tf.random_normal([n_classes]))}

    l1 = tf.add(tf.matmul(data, hidden_1_layer['weights']), hidden_1_layer['biases'])
    l1 = tf.nn.relu(l1)
    l2 = tf.add(tf.matmul(l1, hidden_2_layer['weights']), hidden_2_layer['biases'])
    l2 = tf.nn.relu(l2)
    l3 = tf.add(tf.matmul(l2, hidden_3_layer['weights']), hidden_3_layer['biases'])
    l3 = tf.nn.relu(l3)
    output = tf.add(tf.matmul(l3, output_layer['weights']), output_layer['biases'])
    return (output)
```

- Then add the training function

```python
def train_neural_network(x):
    prediction = neural_network_model(x)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y))
    optimizer = tf.train.AdamOptimizer().minimize(cost) #learning rate is 0.001

    hm_epochs = 10  # cycles feedforward + backpropagation

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        # network training:
        for epoch in range(hm_epochs):
            epoch_loss = 0
            for _ in range(int(mnist.train.num_examples/batch_size)):
                epoch_x, epoch_y = mnist.train.next_batch(batch_size)
                _, c = sess.run([optimizer, cost], feed_dict = {x: epoch_x, y: epoch_y})
                epoch_loss += c
            print('Epoch', epoch, 'completed oput of', hm_epochs, 'loss:', epoch_loss)

        # print results
        correct = tf.equal(tf.argmax(prediction,1), tf.argmax(y,1))
        accuracy =  tf.reduce_mean(tf.cast(correct, 'float'))
        print('Accuracy:', accuracy.eval({x: mnist.test.images, y:mnist.test.labels}))

train_neural_network(x)
```

- Run the program with Python3

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
Epoch 0 completed oput of 10 loss: 1821681.66306
Epoch 1 completed oput of 10 loss: 431706.899899
Epoch 2 completed oput of 10 loss: 238944.139815
Epoch 3 completed oput of 10 loss: 141129.630415
Epoch 4 completed oput of 10 loss: 87271.4430515
Epoch 5 completed oput of 10 loss: 57762.353711
Epoch 6 completed oput of 10 loss: 38104.9504944
Epoch 7 completed oput of 10 loss: 28384.9141481
Epoch 8 completed oput of 10 loss: 22437.0027955
Epoch 9 completed oput of 10 loss: 19794.682209
Accuracy: 0.9475
[Finished in 137.7s]
```

The accuracy achieved is 0.9475%.

Notes

- tf.placeholder represents a symbolic variable for the input
- For example, with x = tf.placeholder(tf.float32, [None, 784]) we can represent a 2D tensor of floating-point numbers with a [None, 784] shape
- tf.Variable is a modifiable tensor that stores the model parameters
- For example, with W=tf.Variables(tf.zeros([784, 10])) we set the initial values to the weights from any image and the output classes