HW3 Problem 3

Name: Liam Jackson

Imports

In [1]:
```
import struct
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import sklearn.metrics as sk
import seaborn as sns
```

1. Loading the data

In [2]:
```
# training images
with open('train-images-idx3-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    nrows, ncols = struct.unpack(">II", f.read(8))
    train_data = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
    train_data = train_data.reshape((size, nrows, ncols))
# training labels
with open('train-labels-idx1-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    train_labels = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))

# test images
with open('t10k-images-idx3-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    nrows, ncols = struct.unpack(">II", f.read(8))
    test_data = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
    test_data = test_data.reshape((size, nrows, ncols))

# test labels
with open('t10k-labels-idx1-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    test_labels = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
```

In [3]:
```
print(f'The shape of the training image array is: {train_data.shape}')
print(f'The shape of the testing image array is: {test_data.shape}')
print(f'The shape of the training label array is: {train_labels.shape}')
print(f'The shape of the testing label array is: {test_labels.shape}')
```

```
The shape of the training image array is: (60000, 28, 28)
The shape of the testing image array is: (10000, 28, 28)
The shape of the training label array is: (60000,)
The shape of the testing label array is: (10000,)
```

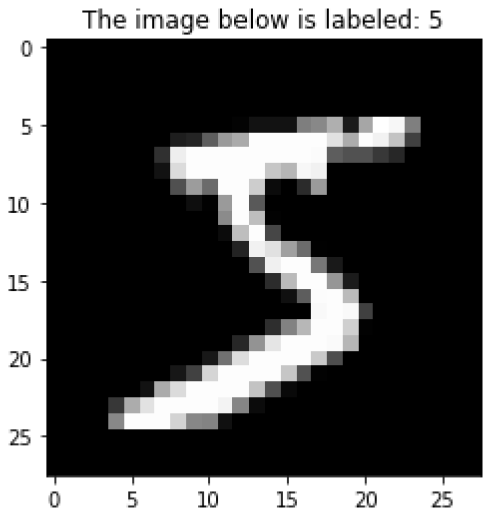
c. Image plots

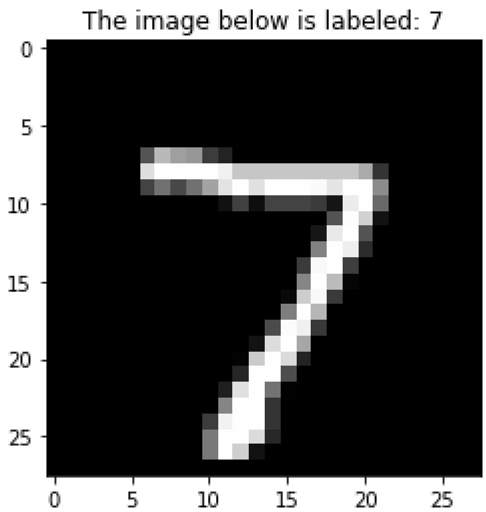
In [4]:
```
train_im1 = train_data[0,:,:]
```

```
plt.imshow(train_im1, cmap = 'gray')
plt.title(f'The image below is labeled: {train_labels[0]}')
plt.show()

test_im1 = test_data[0,:,:]
plt.imshow(test_im1, cmap = 'gray')
plt.title(f'The image below is labeled: {test_labels[0]}')
plt.show()

print('The images match their labels.')
```






```
The images match their labels.
```

d. Image plot (10x10 grid)

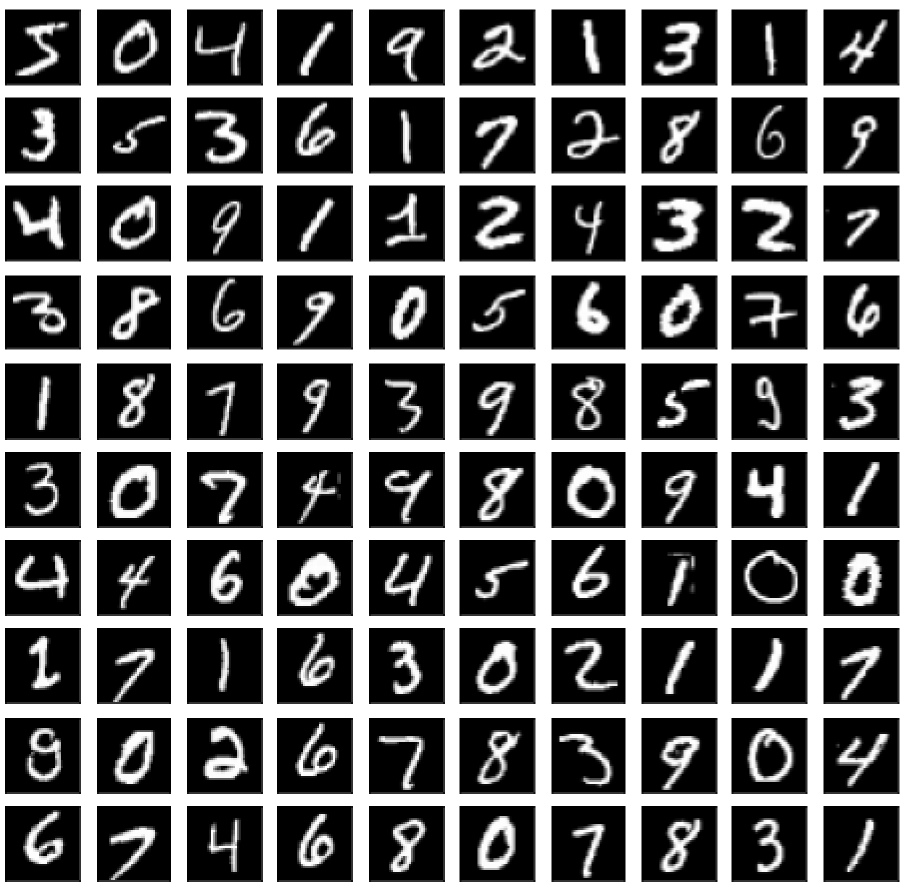
In [5]:

```
w= 28
h= 28
fig=plt.figure(figsize=(8, 8))
cols = 10
rows = 10
for i in range(1, cols*rows +1):
    train_im1010 = train_data[i-1,:,:]
    fig.add_subplot(rows, cols, i)
    plt.imshow(train_im1010,cmap='gray')
    plt.tick_params(
        axis='both',
        which='both',
```

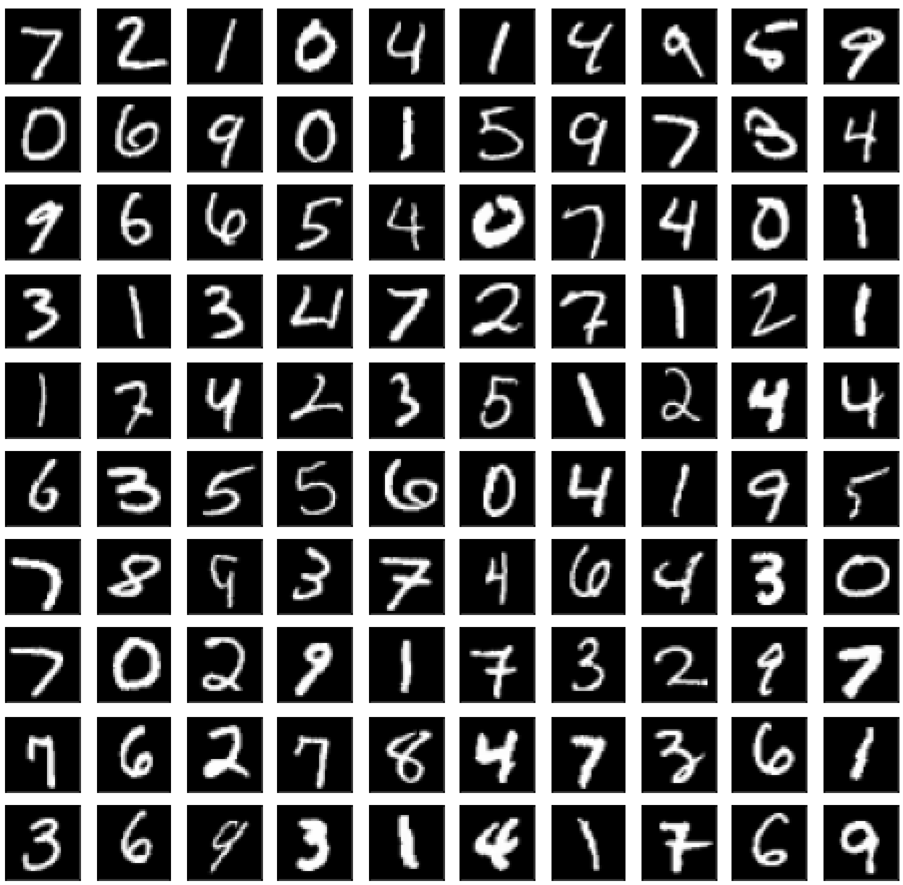
```
        bottom=False,
        left=False,
        labelbottom=False,
        labelleft=False)
plt.suptitle('First 100 Training Images')
plt.show()

fig=plt.figure(figsize=(8, 8))
for i in range(1, cols*rows +1):
    test_im1010 = test_data[i-1,:,:]
    fig.add_subplot(rows, cols, i)
    plt.imshow(test_im1010,cmap='gray')
    plt.tick_params(
        axis='both',
        which='both',
        bottom=False,
        left=False,
        labelbottom=False,
        labelleft=False)
plt.suptitle('First 100 Test Images')
plt.show()
```

First 100 Training Images



First 100 Test Images



e. Digit frequency

In [6]:

```
train_labels_100_uniq, train_labels_100_cts = np.unique(train_labels[0:100], ret
print('Digit frequencies in the first 100 training images:')
print(np.asarray((train_labels_100_uniq, train_labels_100_cts)))

test_labels_100_uniq, test_labels_100_cts  = np.unique(test_labels[0:100], retur
print('Digit frequencies in the first 100 test images:')
print(np.asarray((test_labels_100_uniq, test_labels_100_cts)))
```

```
Digit frequencies in the first 100 training images:
[[ 0  1  2  3  4  5  6  7  8  9]
 [13 14  6 11 11  5 11 10  8 11]]
Digit frequencies in the first 100 test images:
[[ 0  1  2  3  4  5  6  7  8  9]
 [ 8 14  8 11 14  7 10 15  2 11]]
```

2. Data prepartion

Normalization and reshaping

In [7]:

```
# Reduce data to first 6000 to save computation time
train_data = train_data[0:6000,:,:]
train_labels = train_labels[0:6000]

train_labels_uniq, train_labels_cts = np.unique(train_labels, return_counts=True
```

```
print('Digit frequencies in the first 6000 training images:')
print(np.asarray((train_labels_uniq, train_labels_cts)))

test_labels_uniq, test_labels_cts  = np.unique(test_labels, return_counts=True)
print('\nDigit frequencies in the first 6000 test images:')
print(np.asarray((test_labels_uniq, test_labels_cts)))

train_data_norm = np.reshape(train_data / 255.0, (-1, 28*28)).T
print(f'\nThe shape of the new training data array is: {train_data_norm.shape}')

test_data_norm = np.reshape(test_data / 255.0, (-1, 28*28)).T
print(f'The shape of the new test data array is: {test_data_norm.shape}')
```

```
Digit frequencies in the first 6000 training images:
[[  0   1   2   3   4   5   6   7   8   9]
 [592 671 581 608 623 514 608 651 551 601]]

Digit frequencies in the first 6000 test images:
[[   0    1    2    3    4    5    6    7    8    9]
 [ 980 1135 1032 1010  982  892  958 1028  974 1009]]

The shape of the new training data array is: (784, 6000)
The shape of the new test data array is: (784, 10000)
```

One-hot encoding of labels

In [8]:

```
train_labels_ohenc = np.zeros([10, len(train_labels)])
for inst in range(len(train_labels)):
    train_labels_ohenc_idx = train_labels[inst]
    train_labels_ohenc[train_labels_ohenc_idx, inst] = 1

test_labels_ohenc = np.zeros([10, len(test_labels)])
for inst in range(len(test_labels)):
    test_labels_ohenc_idx = test_labels[inst]
    test_labels_ohenc[test_labels_ohenc_idx, inst] = 1

print(f'The shape of the one-hot encoded training labels array is: {train_labels
print(f'The shape of the one-hot encoded testing labels array is: {test_labels_c

print(f'\nThe first training label is: {train_labels[0]}')
print(f'Its associated vector is: {np.asarray(train_labels_ohenc[:,0])}')

print(f'\nThe first test label is: {test_labels[0]}')
print(f'Its associated vector is: {np.asarray(test_labels_ohenc[:,0])}')
```

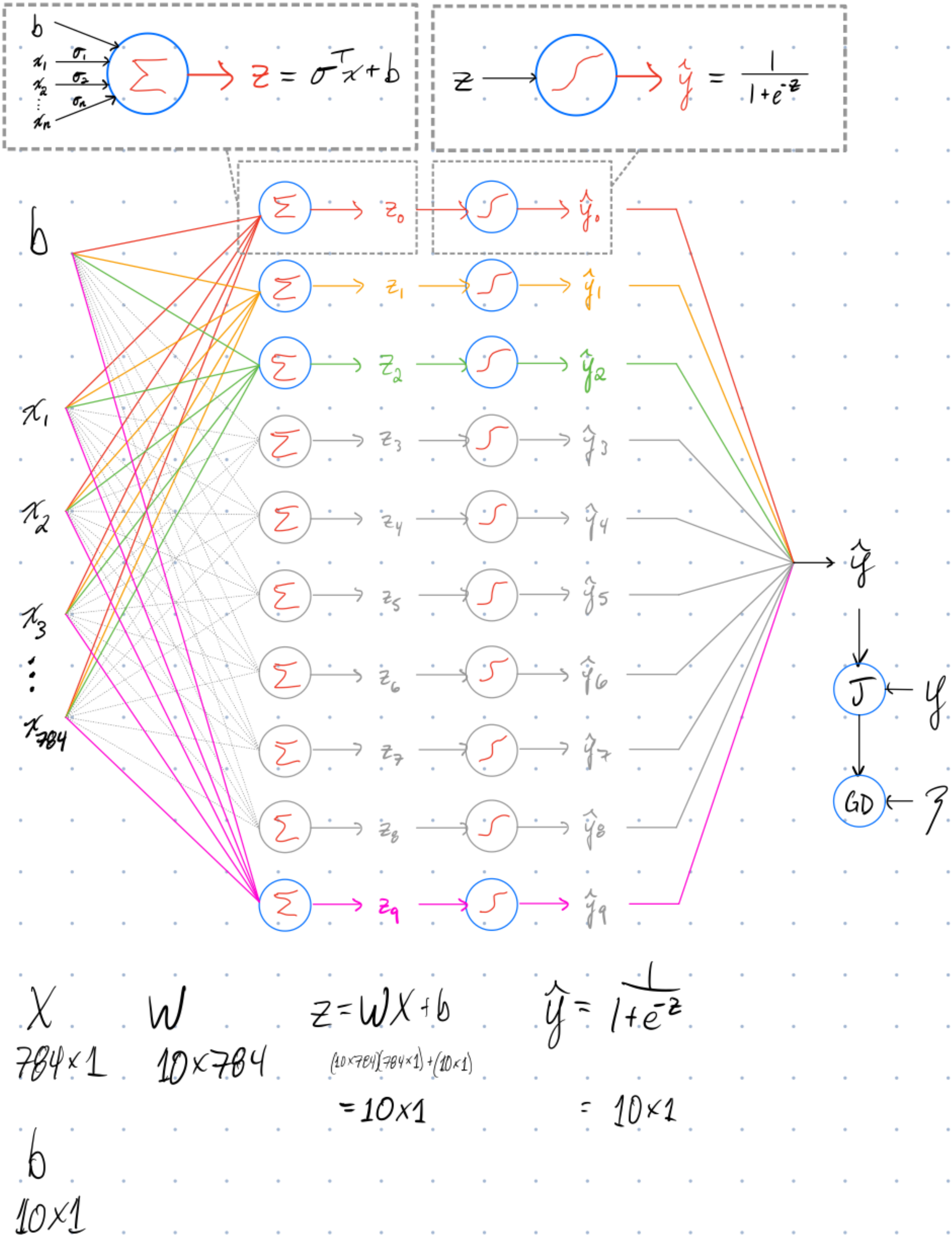
```
The shape of the one-hot encoded training labels array is: (10, 6000)
The shape of the one-hot encoded testing labels array is: (10, 10000)

The first training label is: 5
Its associated vector is: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

The first test label is: 7
Its associated vector is: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

3. Neural Network

Computational graph



In [9]:

```
tf.reset_default_graph()
# Number of features/dimensions
n_dim = 784
# Number of neurons in layers 1 and 2
n1 = 10
n2 = 1
# --- Input / Output Placeholders
X = tf.placeholder(tf.float64, [n_dim, None])
```

```
Y = tf.placeholder(tf.float64, [10, None])
# --- First Layer Weights and Bias
W1 = tf.Variable(tf.random.normal([n1, n_dim], stddev=.1, seed=12345, dtype=tf.f
                 dtype=tf.float64)
b1 = tf.Variable(tf.random.normal([n1,1], stddev=.1, seed=12345, dtype=tf.float6
                 dtype=tf.float64)
# --- Learning Rate
learning_rate = tf.placeholder(tf.float64, shape = ())
# --- First Layer z vals, Output
z = tf.matmul(W1, X) + b1
Y_ = tf.sigmoid(z)
# --- Training Statistics
digit_predictions = tf.argmax(Y_,0)
digit_true = tf.argmax(Y,0)
correct_predictions = tf.equal(digit_predictions, digit_true)
accuracy = tf.reduce_mean(tf.cast(correct_predictions,
                                  dtype=tf.float64))
# --- Variable Initialization
init = tf.global_variables_initializer()
# --- Cost Function
cost_fxn = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=z, labe
# --- Gradient Descent
training_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_f
# --- Saving Node
saver = tf.train.Saver()
```

```
WARNING:tensorflow:From /home/liam/anaconda3/envs/tf/lib/python3.7/site-package
s/tensorflow_core/python/ops/nn_impl.py:183: where (from tensorflow.python.ops.a
rray_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Training function

In [10]:

```
def nn_train(training_data, training_labels, num_epochs, learn_rate):
    sess = tf.Session()
    sess.run(init)
    cost_history = np.empty(shape=[0], dtype = float)

    for epoch in range(num_epochs):
        training_step_, cost_ = sess.run([training_step, cost_fxn],
                                         feed_dict ={X: training_data,
                                                     Y: training_labels,
                                                     learning_rate: learn_rate})
        cost_history = np.append(cost_history, np.mean(cost_))

        accuracy_ = accuracy.eval({X: training_data,
                                   Y: training_labels,
                                   learning_rate: learn_rate},
                                  session=sess)

        if epoch % 500 == 0:
            print(f'Epoch {epoch} reached, with mean cost of {np.mean(cost_)}')
            print("Accuracy:", accuracy_)
    print ("Final Accuracy:", accuracy_)

    ep_str = str(num_epochs)
    lr_str = '_'.join(str(learn_rate).split('.'))
    file_name_ = '_'.join((ep_str, 'ep', lr_str, 'lr'))
    file_name = "%s/trained_model.cpkt" % file_name_
```

```
    save_path = saver.save(sess, file_name)
    return sess, cost_history, save_path
```

4. Training and testing

a. Cost history

In [11]:

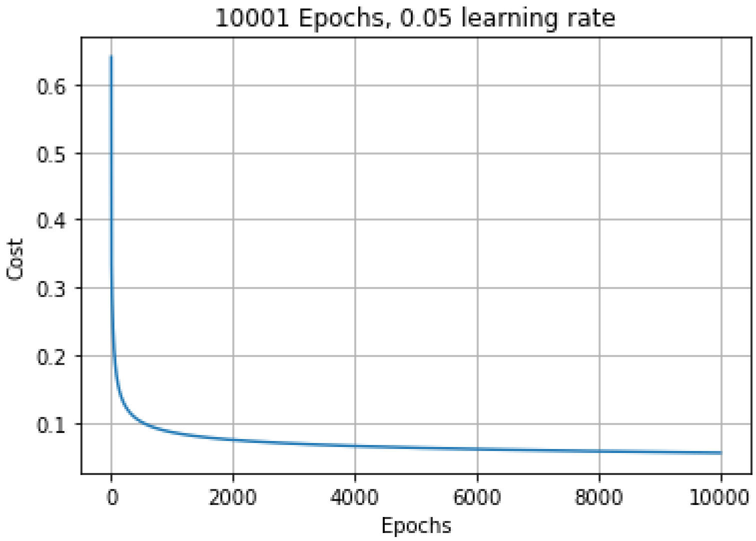
```
epochs1 = 10001
learning_rate1 = .05
sess1, cost_history1, spath1 = nn_train(train_data_norm,
                                        train_labels_ohenc,
                                        epochs1,
                                        learning_rate1)
sess1.close()
```

```
Epoch 0 reached, with mean cost of 0.6404056534110572
Accuracy: 0.20333333333333334
Epoch 500 reached, with mean cost of 0.10094980011976343
Accuracy: 0.88
Epoch 1000 reached, with mean cost of 0.08585776065521449
Accuracy: 0.8965
Epoch 1500 reached, with mean cost of 0.07891588946091989
Accuracy: 0.9043333333333333
Epoch 2000 reached, with mean cost of 0.07458169668748314
Accuracy: 0.9093333333333333
Epoch 2500 reached, with mean cost of 0.07148200457608998
Accuracy: 0.9133333333333333
Epoch 3000 reached, with mean cost of 0.06909044570536527
Accuracy: 0.916
Epoch 3500 reached, with mean cost of 0.06715414883272393
Accuracy: 0.919
Epoch 4000 reached, with mean cost of 0.06553334983331613
Accuracy: 0.9225
Epoch 4500 reached, with mean cost of 0.0641431023235331
Accuracy: 0.9255
Epoch 5000 reached, with mean cost of 0.06292812968998782
Accuracy: 0.928
Epoch 5500 reached, with mean cost of 0.061850564862116465
Accuracy: 0.9288333333333333
Epoch 6000 reached, with mean cost of 0.060883403345195694
Accuracy: 0.9303333333333333
Epoch 6500 reached, with mean cost of 0.06000675349088088
Accuracy: 0.9323333333333333
Epoch 7000 reached, with mean cost of 0.05920556652368445
Accuracy: 0.9328333333333333
Epoch 7500 reached, with mean cost of 0.05846819920194292
Accuracy: 0.9336666666666666
Epoch 8000 reached, with mean cost of 0.05778546914989804
Accuracy: 0.9353333333333333
Epoch 8500 reached, with mean cost of 0.057150014144827954
Accuracy: 0.9353333333333333
Epoch 9000 reached, with mean cost of 0.05655584567646318
Accuracy: 0.9361666666666667
Epoch 9500 reached, with mean cost of 0.05599803049997034
Accuracy: 0.9366666666666666
Epoch 10000 reached, with mean cost of 0.05547245876916988
Accuracy: 0.9376666666666666
Final Accuracy: 0.9376666666666666
```

In [12]:

```
plt.plot(np.arange(epochs1), cost_history1)
```

```
plt.title(f'{epochs1} Epochs, {learning_rate1} learning rate')
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.grid('on')
plt.show()
```



In [13]:

```
epochs2 = 50001
learning_rate2 = .05
sess2, cost_history2, spath2 = nn_train(train_data_norm,
                                        train_labels_ohenc,
                                        epochs2,
                                        learning_rate2)
sess2.close()
```

```
Epoch 0 reached, with mean cost of 0.6404056534110572
Accuracy: 0.20333333333333334
Epoch 500 reached, with mean cost of 0.10094980011976343
Accuracy: 0.88
Epoch 1000 reached, with mean cost of 0.08585776065521449
Accuracy: 0.8965
Epoch 1500 reached, with mean cost of 0.07891588946091989
Accuracy: 0.9043333333333333
Epoch 2000 reached, with mean cost of 0.07458169668748314
Accuracy: 0.9093333333333333
Epoch 2500 reached, with mean cost of 0.07148200457608998
Accuracy: 0.9133333333333333
Epoch 3000 reached, with mean cost of 0.06909044570536527
Accuracy: 0.916
Epoch 3500 reached, with mean cost of 0.06715414883272393
Accuracy: 0.919
Epoch 4000 reached, with mean cost of 0.06553334983331613
Accuracy: 0.9225
Epoch 4500 reached, with mean cost of 0.0641431023235331
Accuracy: 0.9255
Epoch 5000 reached, with mean cost of 0.06292812968998782
Accuracy: 0.928
Epoch 5500 reached, with mean cost of 0.061850564862116465
Accuracy: 0.9288333333333333
Epoch 6000 reached, with mean cost of 0.060883403345195694
Accuracy: 0.9303333333333333
Epoch 6500 reached, with mean cost of 0.06000675349088088
Accuracy: 0.9323333333333333
Epoch 7000 reached, with mean cost of 0.05920556652368445
```

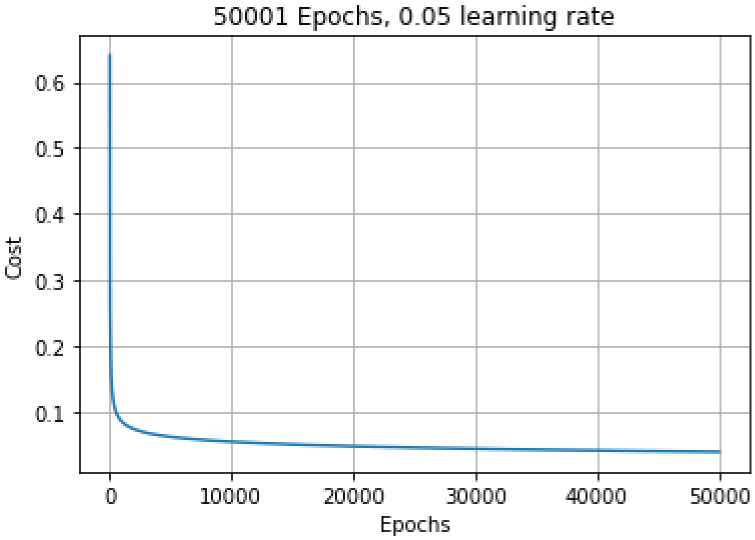
```
Accuracy: 0.9328333333333333
Epoch 7500 reached, with mean cost of 0.05846819920194292
Accuracy: 0.9336666666666666
Epoch 8000 reached, with mean cost of 0.05778546914989804
Accuracy: 0.9353333333333333
Epoch 8500 reached, with mean cost of 0.057150014144827954
Accuracy: 0.9353333333333333
Epoch 9000 reached, with mean cost of 0.05655584567646318
Accuracy: 0.9361666666666667
Epoch 9500 reached, with mean cost of 0.05599803049997034
Accuracy: 0.9366666666666666
Epoch 10000 reached, with mean cost of 0.05547245876916988
Accuracy: 0.9376666666666666
Epoch 10500 reached, with mean cost of 0.054975672109041265
Accuracy: 0.9378333333333333
Epoch 11000 reached, with mean cost of 0.054504734045153845
Accuracy: 0.9378333333333333
Epoch 11500 reached, with mean cost of 0.054057130919498944
Accuracy: 0.9378333333333333
Epoch 12000 reached, with mean cost of 0.053630695114041636
Accuracy: 0.9381666666666667
Epoch 12500 reached, with mean cost of 0.05322354484300411
Accuracy: 0.9385
Epoch 13000 reached, with mean cost of 0.05283403641963484
Accuracy: 0.9386666666666666
Epoch 13500 reached, with mean cost of 0.05246072603227907
Accuracy: 0.9393333333333334
Epoch 14000 reached, with mean cost of 0.05210233885249786
Accuracy: 0.9396666666666667
Epoch 14500 reached, with mean cost of 0.051757743856195026
Accuracy: 0.9398333333333333
Epoch 15000 reached, with mean cost of 0.05142593313968036
Accuracy: 0.9401666666666667
Epoch 15500 reached, with mean cost of 0.05110600480431212
Accuracy: 0.9408333333333333
Epoch 16000 reached, with mean cost of 0.0507971486981061
Accuracy: 0.9415
Epoch 16500 reached, with mean cost of 0.050498634462523874
Accuracy: 0.9418333333333333
Epoch 17000 reached, with mean cost of 0.050209801452825685
Accuracy: 0.9423333333333334
Epoch 17500 reached, with mean cost of 0.04993005019159765
Accuracy: 0.9428333333333333
Epoch 18000 reached, with mean cost of 0.04965883508493861
Accuracy: 0.9431666666666667
Epoch 18500 reached, with mean cost of 0.04939565818476651
Accuracy: 0.9436666666666667
Epoch 19000 reached, with mean cost of 0.04914006382272984
Accuracy: 0.9438333333333333
Epoch 19500 reached, with mean cost of 0.04889163397417891
Accuracy: 0.944
Epoch 20000 reached, with mean cost of 0.04864998423669859
Accuracy: 0.9445
Epoch 20500 reached, with mean cost of 0.0484147603284215
Accuracy: 0.9451666666666667
Epoch 21000 reached, with mean cost of 0.04818563502792385
Accuracy: 0.9456666666666667
Epoch 21500 reached, with mean cost of 0.047962305490858725
Accuracy: 0.9461666666666667
Epoch 22000 reached, with mean cost of 0.04774449088929585
Accuracy: 0.9463333333333334
Epoch 22500 reached, with mean cost of 0.04753193032854043
Accuracy: 0.9468333333333333
Epoch 23000 reached, with mean cost of 0.0473243810034093
Accuracy: 0.9468333333333333
```

```
Epoch 23500 reached, with mean cost of 0.04712161656186682
Accuracy: 0.9471666666666667
Epoch 24000 reached, with mean cost of 0.046923425648817636
Accuracy: 0.9471666666666667
Epoch 24500 reached, with mean cost of 0.0467296106069139
Accuracy: 0.9476666666666667
Epoch 25000 reached, with mean cost of 0.04653998631461899
Accuracy: 0.9481666666666667
Epoch 25500 reached, with mean cost of 0.04635437914459907
Accuracy: 0.9481666666666667
Epoch 26000 reached, with mean cost of 0.04617262602789171
Accuracy: 0.9481666666666667
Epoch 26500 reached, with mean cost of 0.04599457361130327
Accuracy: 0.9483333333333334
Epoch 27000 reached, with mean cost of 0.04582007749718174
Accuracy: 0.9488333333333333
Epoch 27500 reached, with mean cost of 0.04564900155614924
Accuracy: 0.949
Epoch 28000 reached, with mean cost of 0.045481217304603416
Accuracy: 0.9491666666666667
Epoch 28500 reached, with mean cost of 0.045316603339842845
Accuracy: 0.9493333333333334
Epoch 29000 reached, with mean cost of 0.045155044826568
Accuracy: 0.9495
Epoch 29500 reached, with mean cost of 0.044996433029279796
Accuracy: 0.9493333333333334
Epoch 30000 reached, with mean cost of 0.04484066488576186
Accuracy: 0.9495
Epoch 30500 reached, with mean cost of 0.04468764261740673
Accuracy: 0.9498333333333333
Epoch 31000 reached, with mean cost of 0.04453727337264361
Accuracy: 0.95
Epoch 31500 reached, with mean cost of 0.04438946890015799
Accuracy: 0.9501666666666667
Epoch 32000 reached, with mean cost of 0.04424414524896918
Accuracy: 0.9505
Epoch 32500 reached, with mean cost of 0.044101222492761674
Accuracy: 0.951
Epoch 33000 reached, with mean cost of 0.043960624476152954
Accuracy: 0.9511666666666667
Epoch 33500 reached, with mean cost of 0.0438222785808332
Accuracy: 0.9513333333333334
Epoch 34000 reached, with mean cost of 0.04368611550973368
Accuracy: 0.9516666666666667
Epoch 34500 reached, with mean cost of 0.04355206908757591
Accuracy: 0.9516666666666667
Epoch 35000 reached, with mean cost of 0.04342007607632561
Accuracy: 0.9518333333333333
Epoch 35500 reached, with mean cost of 0.043290076004227414
Accuracy: 0.9518333333333333
Epoch 36000 reached, with mean cost of 0.043162011007231035
Accuracy: 0.9518333333333333
Epoch 36500 reached, with mean cost of 0.04303582568173887
Accuracy: 0.9518333333333333
Epoch 37000 reached, with mean cost of 0.04291146694771032
Accuracy: 0.9523333333333334
Epoch 37500 reached, with mean cost of 0.04278888392125319
Accuracy: 0.9523333333333334
Epoch 38000 reached, with mean cost of 0.04266802779591584
Accuracy: 0.9525
Epoch 38500 reached, with mean cost of 0.04254885173196834
Accuracy: 0.9531666666666667
Epoch 39000 reached, with mean cost of 0.04243131075302874
Accuracy: 0.9531666666666667
Epoch 39500 reached, with mean cost of 0.04231536164944895
```

```
Accuracy: 0.9535
Epoch 40000 reached, with mean cost of 0.04220096288792977
Accuracy: 0.9536666666666667
Epoch 40500 reached, with mean cost of 0.04208807452688187
Accuracy: 0.954
Epoch 41000 reached, with mean cost of 0.041976658137092636
Accuracy: 0.954
Epoch 41500 reached, with mean cost of 0.04186667672729865
Accuracy: 0.9541666666666667
Epoch 42000 reached, with mean cost of 0.04175809467429744
Accuracy: 0.9545
Epoch 42500 reached, with mean cost of 0.04165087765726465
Accuracy: 0.9545
Epoch 43000 reached, with mean cost of 0.041544992595970956
Accuracy: 0.9546666666666667
Epoch 43500 reached, with mean cost of 0.04144040759261902
Accuracy: 0.9548333333333333
Epoch 44000 reached, with mean cost of 0.04133709187704411
Accuracy: 0.9548333333333333
Epoch 44500 reached, with mean cost of 0.041235015755043174
Accuracy: 0.9551666666666667
Epoch 45000 reached, with mean cost of 0.0411341505596165
Accuracy: 0.9551666666666667
Epoch 45500 reached, with mean cost of 0.0410344686049236
Accuracy: 0.9551666666666667
Epoch 46000 reached, with mean cost of 0.04093594314277047
Accuracy: 0.9555
Epoch 46500 reached, with mean cost of 0.04083854832146043
Accuracy: 0.9555
Epoch 47000 reached, with mean cost of 0.040742259146853144
Accuracy: 0.9556666666666667
Epoch 47500 reached, with mean cost of 0.04064705144548927
Accuracy: 0.9561666666666667
Epoch 48000 reached, with mean cost of 0.04055290182964809
Accuracy: 0.9561666666666667
Epoch 48500 reached, with mean cost of 0.04045978766421668
Accuracy: 0.9565
Epoch 49000 reached, with mean cost of 0.04036768703525682
Accuracy: 0.9566666666666667
Epoch 49500 reached, with mean cost of 0.04027657872016586
Accuracy: 0.9568333333333333
Epoch 50000 reached, with mean cost of 0.04018644215933365
Accuracy: 0.9568333333333333
Final Accuracy: 0.9568333333333333
```

In [14]:

```
plt.plot(np.arange(epochs2), cost_history2)
plt.title(f'{epochs2} Epochs, {learning_rate2} learning rate')
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.grid('on')
plt.show()
```



In [15]:

```
epochs3 = 50001
learning_rate3 = .01
sess3, cost_history3, spath3 = nn_train(train_data_norm,
                                        train_labels_ohenc,
                                        epochs3,
                                        learning_rate3)
sess3.close()
```

```
Epoch 0 reached, with mean cost of 0.6404056534110572
Accuracy: 0.18783333333333332
Epoch 500 reached, with mean cost of 0.16238585527267363
Accuracy: 0.8128333333333333
Epoch 1000 reached, with mean cost of 0.13030331971019782
Accuracy: 0.8466666666666667
Epoch 1500 reached, with mean cost of 0.11574564134735313
Accuracy: 0.8628333333333333
Epoch 2000 reached, with mean cost of 0.10698931307260624
Accuracy: 0.8721666666666666
Epoch 2500 reached, with mean cost of 0.10097976325049056
Accuracy: 0.8798333333333334
Epoch 3000 reached, with mean cost of 0.09652288908498582
Accuracy: 0.8838333333333334
Epoch 3500 reached, with mean cost of 0.0930423613486847
Accuracy: 0.8876666666666667
Epoch 4000 reached, with mean cost of 0.09022162459618319
Accuracy: 0.8915
Epoch 4500 reached, with mean cost of 0.08787078818251433
Accuracy: 0.894
Epoch 5000 reached, with mean cost of 0.08586834612679052
Accuracy: 0.8963333333333333
Epoch 5500 reached, with mean cost of 0.0841325877053026
Accuracy: 0.8976666666666666
Epoch 6000 reached, with mean cost of 0.08260634217030498
Accuracy: 0.899
Epoch 6500 reached, with mean cost of 0.0812482897959905
Accuracy: 0.9006666666666666
Epoch 7000 reached, with mean cost of 0.08002774674550159
Accuracy: 0.902
Epoch 7500 reached, with mean cost of 0.0789213969030522
Accuracy: 0.9043333333333333
Epoch 8000 reached, with mean cost of 0.07791116822886227
Accuracy: 0.9055
Epoch 8500 reached, with mean cost of 0.0769828095155952
Accuracy: 0.9066666666666666
```

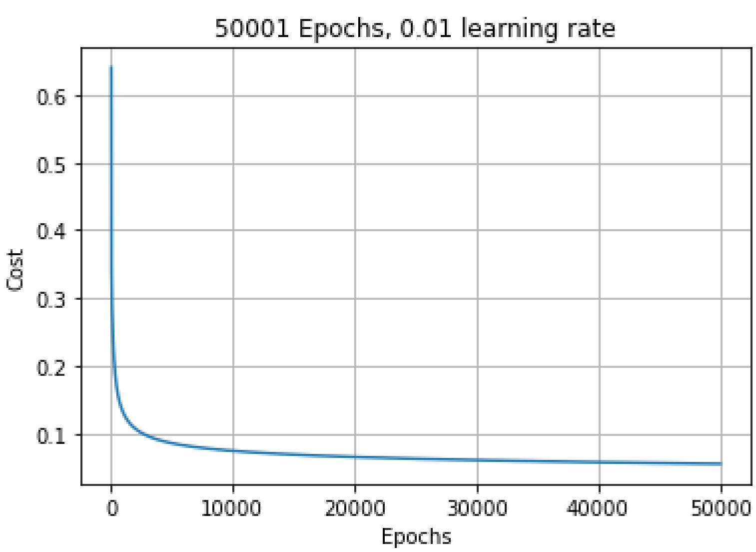
```
Epoch 9000 reached, with mean cost of 0.07612491075456872
Accuracy: 0.908
Epoch 9500 reached, with mean cost of 0.07532821295696161
Accuracy: 0.9088333333333334
Epoch 10000 reached, with mean cost of 0.07458511182423908
Accuracy: 0.9093333333333333
Epoch 10500 reached, with mean cost of 0.07388929425525097
Accuracy: 0.9105
Epoch 11000 reached, with mean cost of 0.07323546775778322
Accuracy: 0.9111666666666667
Epoch 11500 reached, with mean cost of 0.07261915603475314
Accuracy: 0.9123333333333333
Epoch 12000 reached, with mean cost of 0.07203654248829594
Accuracy: 0.9131666666666667
Epoch 12500 reached, with mean cost of 0.07148434894345455
Accuracy: 0.9133333333333333
Epoch 13000 reached, with mean cost of 0.07095974061257901
Accuracy: 0.914
Epoch 13500 reached, with mean cost of 0.07046025085548693
Accuracy: 0.9148333333333334
Epoch 14000 reached, with mean cost of 0.06998372104528666
Accuracy: 0.9151666666666667
Epoch 14500 reached, with mean cost of 0.06952825208344052
Accuracy: 0.9156666666666666
Epoch 15000 reached, with mean cost of 0.06909216498697819
Accuracy: 0.916
Epoch 15500 reached, with mean cost of 0.06867396860556217
Accuracy: 0.9166666666666666
Epoch 16000 reached, with mean cost of 0.06827233298979649
Accuracy: 0.917
Epoch 16500 reached, with mean cost of 0.06788606727459802
Accuracy: 0.9175
Epoch 17000 reached, with mean cost of 0.0675141011969174
Accuracy: 0.9188333333333333
Epoch 17500 reached, with mean cost of 0.06715546955950978
Accuracy: 0.919
Epoch 18000 reached, with mean cost of 0.0668092990986705
Accuracy: 0.9198333333333333
Epoch 18500 reached, with mean cost of 0.06647479732589719
Accuracy: 0.9205
Epoch 19000 reached, with mean cost of 0.0661512429999759
Accuracy: 0.921
Epoch 19500 reached, with mean cost of 0.0658379779533255
Accuracy: 0.9216666666666666
Epoch 20000 reached, with mean cost of 0.06553440004919739
Accuracy: 0.9225
Epoch 20500 reached, with mean cost of 0.06523995708795058
Accuracy: 0.9233333333333333
Epoch 21000 reached, with mean cost of 0.06495414151366219
Accuracy: 0.9241666666666667
Epoch 21500 reached, with mean cost of 0.06467648579872284
Accuracy: 0.925
Epoch 22000 reached, with mean cost of 0.0644065584052613
Accuracy: 0.9251666666666667
Epoch 22500 reached, with mean cost of 0.06414396023936333
Accuracy: 0.9255
Epoch 23000 reached, with mean cost of 0.06388832152794806
Accuracy: 0.9261666666666667
Epoch 23500 reached, with mean cost of 0.06363929905950752
Accuracy: 0.927
Epoch 24000 reached, with mean cost of 0.06339657373921466
Accuracy: 0.9273333333333333
Epoch 24500 reached, with mean cost of 0.06315984841656525
Accuracy: 0.9278333333333333
Epoch 25000 reached, with mean cost of 0.0629288459500571
```

```
Accuracy: 0.928
Epoch 25500 reached, with mean cost of 0.06270330747867586
Accuracy: 0.9281666666666667
Epoch 26000 reached, with mean cost of 0.06248299087435111
Accuracy: 0.9281666666666667
Epoch 26500 reached, with mean cost of 0.062267669353224905
Accuracy: 0.9283333333333333
Epoch 27000 reached, with mean cost of 0.06205713022667013
Accuracy: 0.9285
Epoch 27500 reached, with mean cost of 0.06185117377560565
Accuracy: 0.9288333333333333
Epoch 28000 reached, with mean cost of 0.0616496122338656
Accuracy: 0.929
Epoch 28500 reached, with mean cost of 0.06145226886825861
Accuracy: 0.9291666666666667
Epoch 29000 reached, with mean cost of 0.06125897714455235
Accuracy: 0.9298333333333333
Epoch 29500 reached, with mean cost of 0.06106957996998789
Accuracy: 0.93
Epoch 30000 reached, with mean cost of 0.06088392900410132
Accuracy: 0.9303333333333333
Epoch 30500 reached, with mean cost of 0.060701884030639565
Accuracy: 0.9306666666666666
Epoch 31000 reached, with mean cost of 0.06052331238422648
Accuracy: 0.9308333333333333
Epoch 31500 reached, with mean cost of 0.06034808842618966
Accuracy: 0.9315
Epoch 32000 reached, with mean cost of 0.06017609306460886
Accuracy: 0.9318333333333333
Epoch 32500 reached, with mean cost of 0.06000721331421473
Accuracy: 0.9323333333333333
Epoch 33000 reached, with mean cost of 0.059841341892260855
Accuracy: 0.9325
Epoch 33500 reached, with mean cost of 0.059678376846922024
Accuracy: 0.9323333333333333
Epoch 34000 reached, with mean cost of 0.05951822121515078
Accuracy: 0.9323333333333333
Epoch 34500 reached, with mean cost of 0.05936078270725388
Accuracy: 0.9328333333333333
Epoch 35000 reached, with mean cost of 0.059205973415742075
Accuracy: 0.9328333333333333
Epoch 35500 reached, with mean cost of 0.05905370954626251
Accuracy: 0.933
Epoch 36000 reached, with mean cost of 0.05890391116864927
Accuracy: 0.933
Epoch 36500 reached, with mean cost of 0.058756501986327446
Accuracy: 0.9333333333333333
Epoch 37000 reached, with mean cost of 0.05861140912248308
Accuracy: 0.9333333333333333
Epoch 37500 reached, with mean cost of 0.05846856292156853
Accuracy: 0.9336666666666666
Epoch 38000 reached, with mean cost of 0.058327896764852574
Accuracy: 0.9341666666666667
Epoch 38500 reached, with mean cost of 0.05818934689884834
Accuracy: 0.9353333333333333
Epoch 39000 reached, with mean cost of 0.058052852275563935
Accuracy: 0.9353333333333333
Epoch 39500 reached, with mean cost of 0.057918354403618755
Accuracy: 0.9351666666666667
Epoch 40000 reached, with mean cost of 0.05778579720935749
Accuracy: 0.9353333333333333
Epoch 40500 reached, with mean cost of 0.0576551269071736
Accuracy: 0.9351666666666667
Epoch 41000 reached, with mean cost of 0.05752629187832427
Accuracy: 0.9351666666666667
```

```
Epoch 41500 reached, with mean cost of 0.05739924255758364
Accuracy: 0.9353333333333333
Epoch 42000 reached, with mean cost of 0.0572739313271388
Accuracy: 0.9353333333333333
Epoch 42500 reached, with mean cost of 0.057150312417183935
Accuracy: 0.9353333333333333
Epoch 43000 reached, with mean cost of 0.05702834181271588
Accuracy: 0.9356666666666666
Epoch 43500 reached, with mean cost of 0.05690797716607594
Accuracy: 0.9358333333333333
Epoch 44000 reached, with mean cost of 0.056789177714821726
Accuracy: 0.936
Epoch 44500 reached, with mean cost of 0.05667190420454612
Accuracy: 0.9363333333333334
Epoch 45000 reached, with mean cost of 0.0565561188162936
Accuracy: 0.9361666666666667
Epoch 45500 reached, with mean cost of 0.05644178509825103
Accuracy: 0.9363333333333334
Epoch 46000 reached, with mean cost of 0.05632886790141704
Accuracy: 0.9366666666666666
Epoch 46500 reached, with mean cost of 0.05621733331897706
Accuracy: 0.9363333333333334
Epoch 47000 reached, with mean cost of 0.056107148629132596
Accuracy: 0.9365
Epoch 47500 reached, with mean cost of 0.05599828224115304
Accuracy: 0.9366666666666666
Epoch 48000 reached, with mean cost of 0.05589070364443636
Accuracy: 0.9368333333333333
Epoch 48500 reached, with mean cost of 0.055784383360380274
Accuracy: 0.937
Epoch 49000 reached, with mean cost of 0.05567929289688204
Accuracy: 0.9371666666666667
Epoch 49500 reached, with mean cost of 0.05557540470529691
Accuracy: 0.9375
Epoch 50000 reached, with mean cost of 0.055472692139699374
Accuracy: 0.9376666666666666
Final Accuracy: 0.9376666666666666
```

In [16]:

```
plt.plot(np.arange(epochs3), cost_history3)
plt.title(f'{epochs3} Epochs, {learning_rate3} learning rate')
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.grid('on')
plt.show()
```



b. Confusion matrix

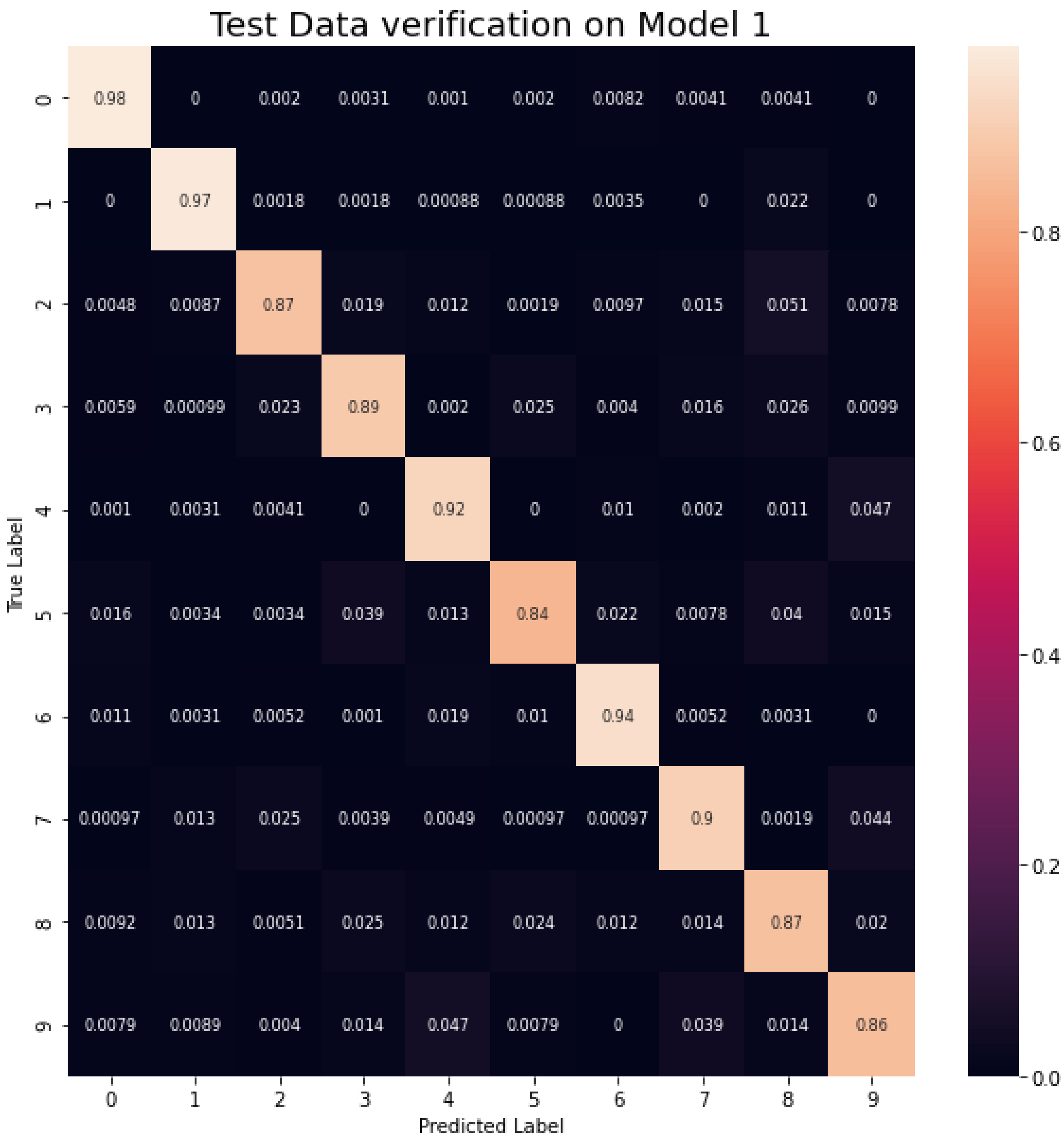
In [17]:
```
sess1_res = tf.Session()
saver.restore(sess1_res, spath1)
y_pred1 = sess1_res.run(Y_, {X: test_data_norm})
y_pred1 = np.argmax(y_pred1,0)
y_true = test_labels

w1 = sess1_res.run(W1, {X: test_data_norm})

conf_mtx1 = sk.confusion_matrix(y_true, y_pred1, normalize='true')

fig_cm1, ax_cm1 = plt.subplots(figsize=(10,10))
ax_cm1 = sns.heatmap(conf_mtx1, annot=True, annot_kws={'size': 8})
ax_cm1.set_title('Test Data verification on Model 1', fontdict={'fontsize':18})
ax_cm1.set_xlabel('Predicted Label')
ax_cm1.set_ylabel('True Label')
plt.show()
```

```
INFO:tensorflow:Restoring parameters from 10001_ep_0_05_lr/trained_model.cpkt
```

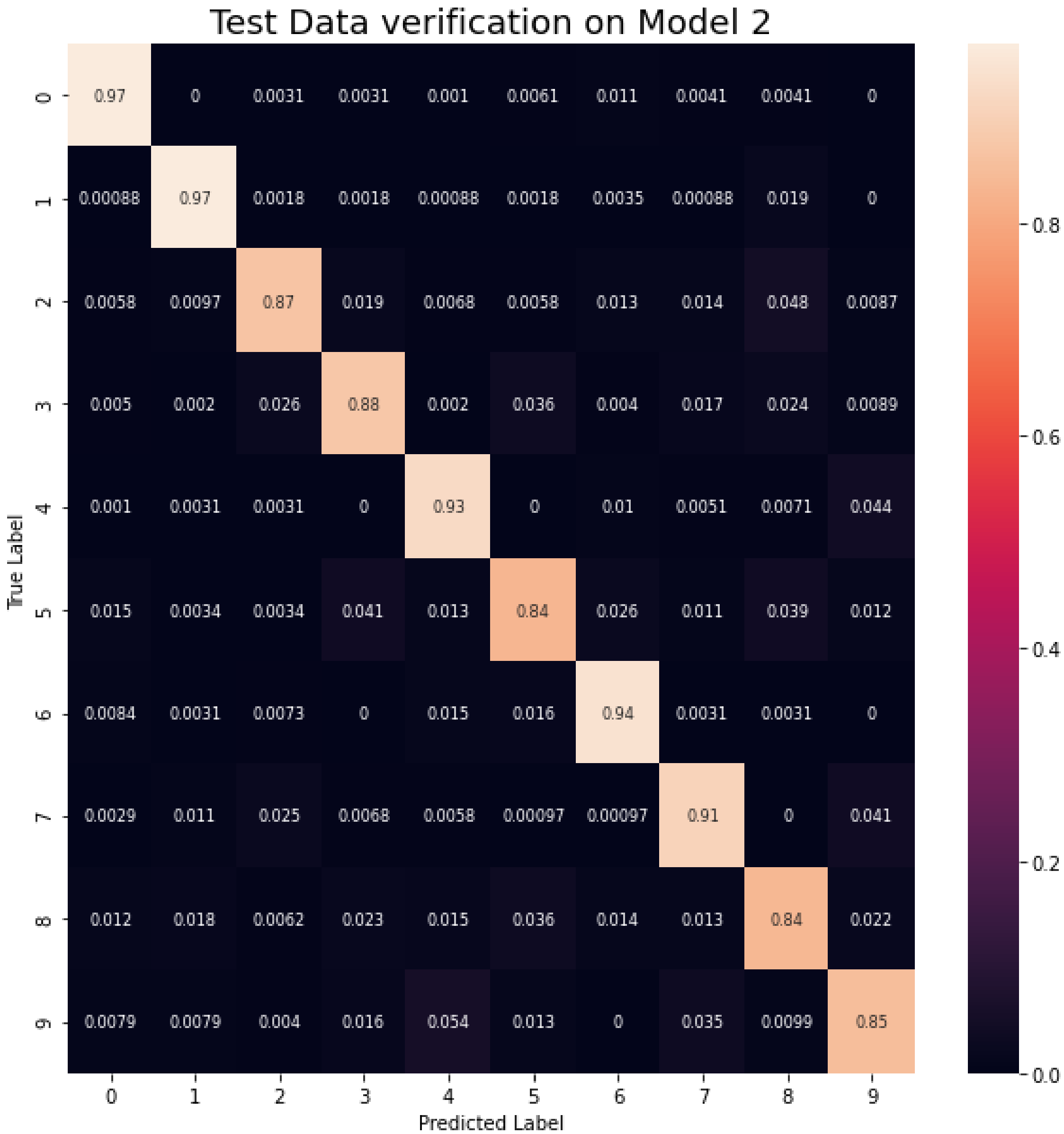


In [18]:
```
sess2_res = tf.Session()
saver.restore(sess2_res, spath2)
y_pred2 = sess2_res.run(Y_, {X: test_data_norm})
y_pred2 = np.argmax(y_pred2, 0)
y_true = test_labels

conf_mtx2 = sk.confusion_matrix(y_true, y_pred2, normalize='true')

fig_cm2, ax_cm2 = plt.subplots(figsize=(10,10))
ax_cm2 = sns.heatmap(conf_mtx2, annot=True, annot_kws={'size': 8})
ax_cm2.set_title('Test Data verification on Model 2', fontdict={'fontsize':18})
ax_cm2.set_xlabel('Predicted Label')
ax_cm2.set_ylabel('True Label')
plt.show()
```

```
INFO:tensorflow:Restoring parameters from 50001_ep_0_05_lr/trained_model.cpkt
```



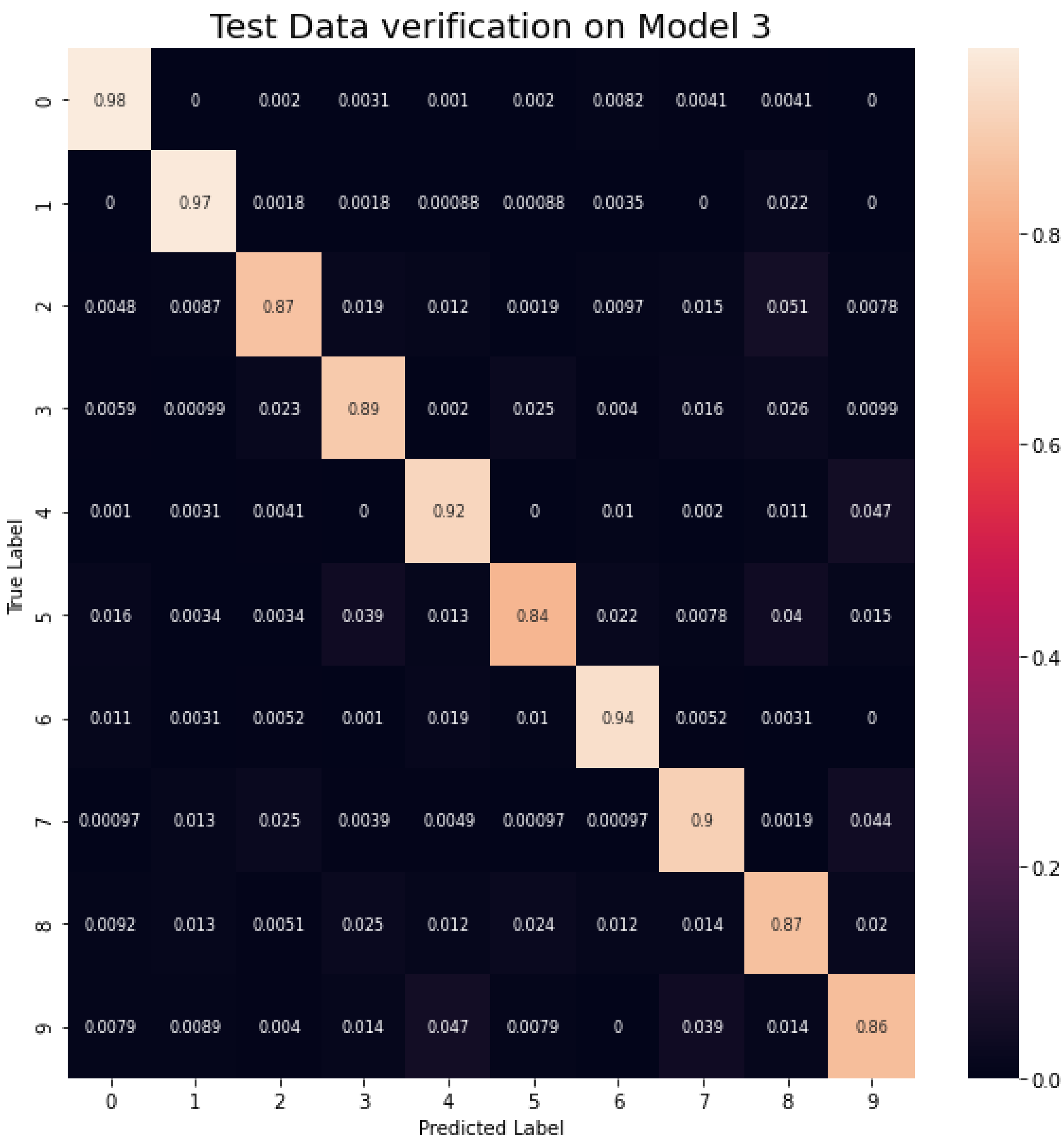
In [19]:
```
sess3_res = tf.Session()
saver.restore(sess3_res, spath3)
y_pred3 = sess3_res.run(Y_, {X: test_data_norm})
y_pred3 = np.argmax(y_pred3, 0)
y_true = test_labels
```

```
w3 = sess3_res.run(W1, {X: test_data_norm})

conf_mtx3 = sk.confusion_matrix(y_true, y_pred3, normalize='true')

fig_cm3, ax_cm3 = plt.subplots(figsize=(10,10))
ax_cm3 = sns.heatmap(conf_mtx3, annot=True, annot_kws={'size': 8})
ax_cm3.set_title('Test Data verification on Model 3', fontdict={'fontsize':18})
ax_cm3.set_xlabel('Predicted Label')
ax_cm3.set_ylabel('True Label')
plt.show()
```

```
INFO:tensorflow:Restoring parameters from 50001_ep_0_01_lr/trained_model.cpkt
```



c. Common misclassifications

In [20]:

```
N = 5
miss_class_conf_mtx3 = conf_mtx3
np.fill_diagonal(miss_class_conf_mtx3, 0)

miss_class_conf_mtx3_flat = miss_class_conf_mtx3.flatten()
```

```
idx_most_common = miss_class_conf_mtx3_flat.argsort()[-2*N:]
i_idx, j_idx = np.unravel_index(idx_most_common, miss_class_conf_mtx3.shape)
top_2N_misses = np.fliplr(np.flipud(np.asarray([i_idx, j_idx]).T))
print(f'The most common misclassifications from session 3:\n {top_2N_misses}')
print('where the first column is the predicted digit and the second is the true
```

```
The most common misclassifications from session 3:
 [[8 2]
 [9 4]
 [4 9]
 [9 7]
 [8 5]
 [3 5]
 [7 9]
 [8 3]
 [2 7]
 [5 3]]
where the first column is the predicted digit and the second is the true digit.
```

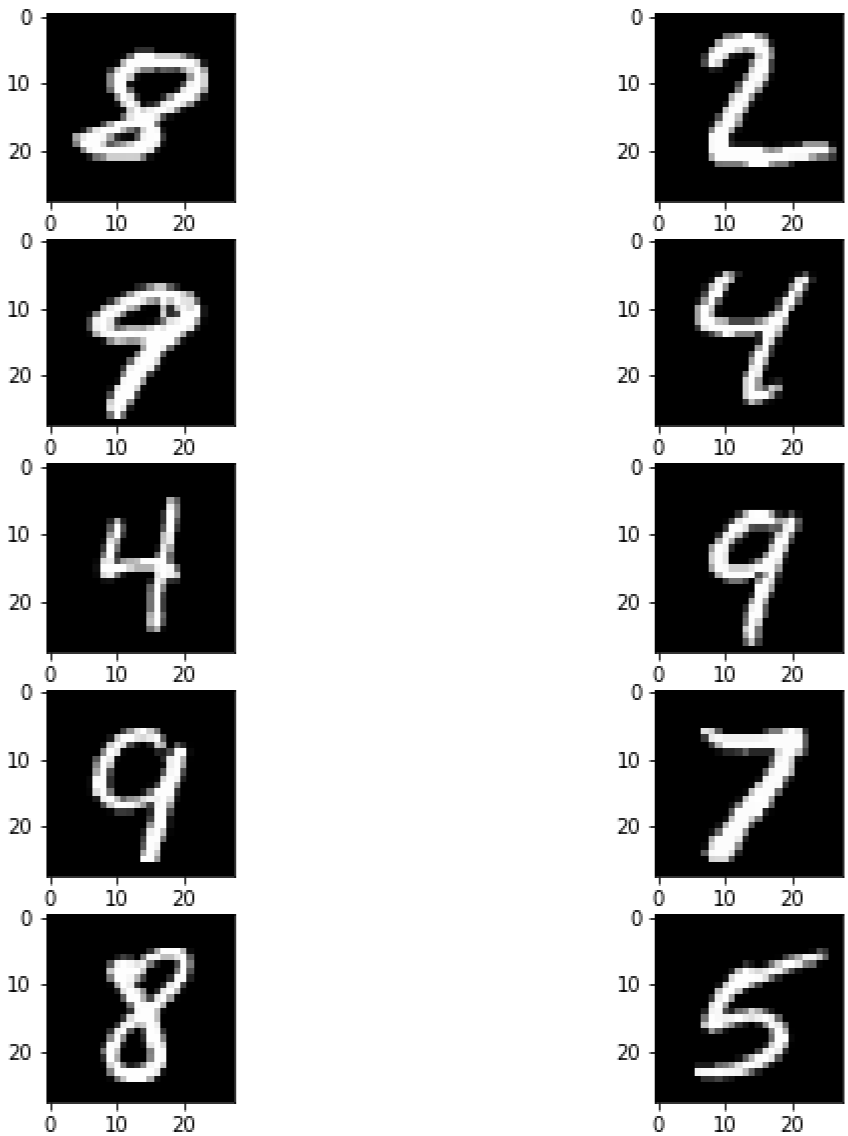
In [21]:

```
fig_miss, ax_miss  = plt.subplots(N, 2, figsize=(10,10))
fig_miss.suptitle('Left (Predicted Digit) vs Right (True Digit)', fontsize=20)
for miss_img in range(N):
    pred_dig = top_2N_misses[miss_img,0]
    true_dig = top_2N_misses[miss_img,1]

    pred_dig_idx = np.where(test_labels == pred_dig)[0][miss_img]
    true_dig_idx = np.where(test_labels == true_dig)[0][miss_img]

    ax_miss[miss_img,0].imshow(test_data[pred_dig_idx,:,:], cmap = 'gray')
    ax_miss[miss_img,1].imshow(test_data[true_dig_idx,:,:], cmap = 'gray')
```

Left (Predicted Digit) vs Right (True Digit)



The predicted digits are based on the positions and intensities of their pixels, and the similarity of that data to training image data. Between the predicted and true digits above, it's easy to recognize that white pixels fall within similar regions between left/right pairs. This indicates to the model that the input data is statistically similar to the data it is familiar with, and so a missclassification results.

It's interesting to note that the values for permuted "predicted" and "true" labels (ie: 8,2 vs 2,8) are not the same. I looked into this a bit, and found a paper describing that behavior as an indicator of a bad classifier. I'm pretty interested in how this type of thing might be optimized.