In [17]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

In [18]:
```
sess = tf.InteractiveSession()
```

```
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py:1761: UserWarning: An interactive session is already active. This can cause out-of-memory errors in some cases. You must explicitly call `InteractiveSession.close()` to release resources held by the other session(s).
  warnings.warn('An interactive session is already active. This can '
```

In [19]:
```
import numpy as np
import math
```

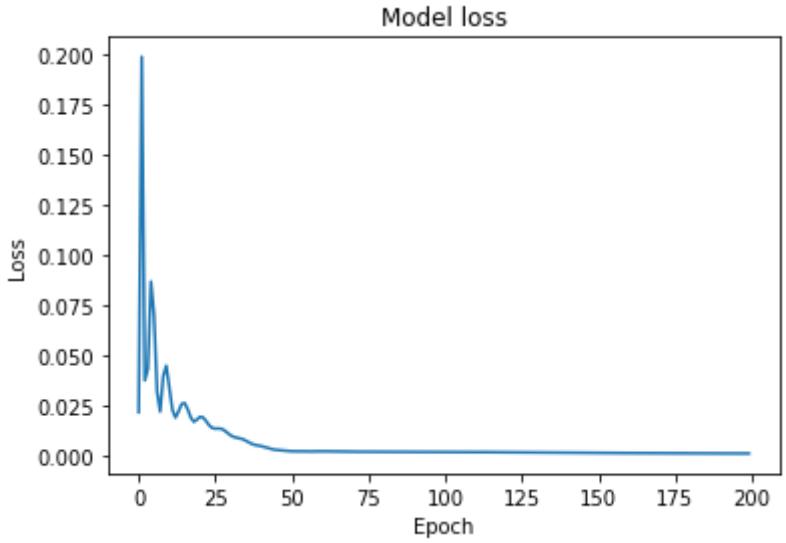
In [20]:
```
X =np.expand_dims(np.arange(0.0, 5.0, 0.01),1)
Y =np.sinc(5*X)
x = tf.placeholder(tf.float64, [500,1], name='x')
y = tf.placeholder(tf.float64, [500,1], name='y')
```

insert the data type for your tensor model 1 is 200+200*400+400*200+200+200*200+200=200400

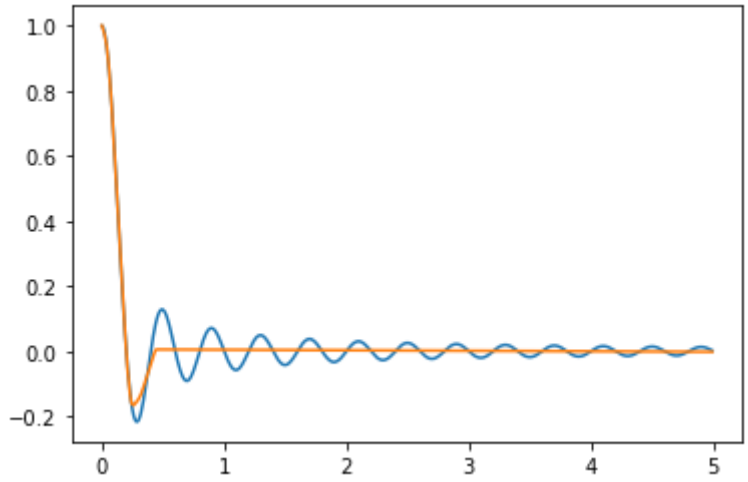
In [22]:
```
input_layer = tf.layers.dense(x, 200, activation= tf.nn.relu)
hidden_layer1 = tf.layers.dropout(input_layer,0.2)
hidden_layer2 = tf.layers.dense(hidden_layer1,400,activation=tf.nn.relu)
hidden_layer3 = tf.layers.dense(hidden_layer2,200,activation=tf.nn.relu)
output_layer = tf.layers.dense(hidden_layer3,1)
Loss =tf.losses.mean_squared_error(y , output_layer)
Optimizer = tf.train.AdamOptimizer(learning_rate= 0.001).minimize(Loss)
init = tf.global_variables_initializer()
```

In [23]:
```
loss_list=[]
sess.run(init)
for i in range(0,200):
  fd ={x:X, y:Y}
  _, loss_val = sess.run([Optimizer, Loss], feed_dict=fd)
  #print ('loss = %s' % loss_val)
  loss_list.append(loss_val)
YP = sess.run(output_layer,feed_dict={x:X})
```

In [24]:
```
# Plot training  loss values
import matplotlib.pyplot as plt
plt.plot(loss_list)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



In [25]:
```
plt.plot(X,Y)
plt.plot(X,YP)
plt.show()
```



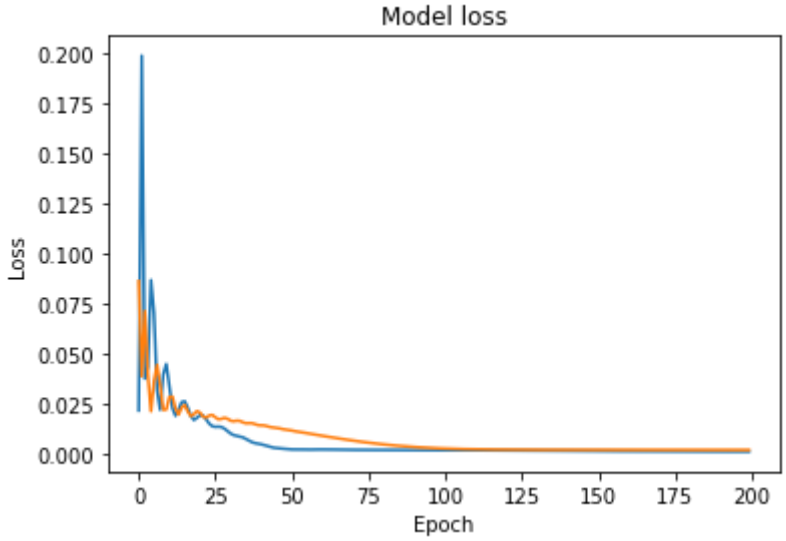
insert the data type for your tensor model 1 is 200+200*500+500*200+200 =200400variables

In [26]:
```
input_layer_m2 = tf.layers.dense(x, 200, activation= tf.nn.relu)
hidden_layer1_m2 = tf.layers.dropout(input_layer_m2,0.2)
hidden_layer2_m2 = tf.layers.dense(hidden_layer1_m2,500,activation=tf.nn.relu)

output_layer_m2 = tf.layers.dense(hidden_layer2_m2,1)
Loss_m2 =tf.losses.mean_squared_error(y , output_layer_m2)
Optimizer_m2 = tf.train.AdamOptimizer(learning_rate= 0.001).minimize(Loss_m2)
init_m2 = tf.global_variables_initializer()
```

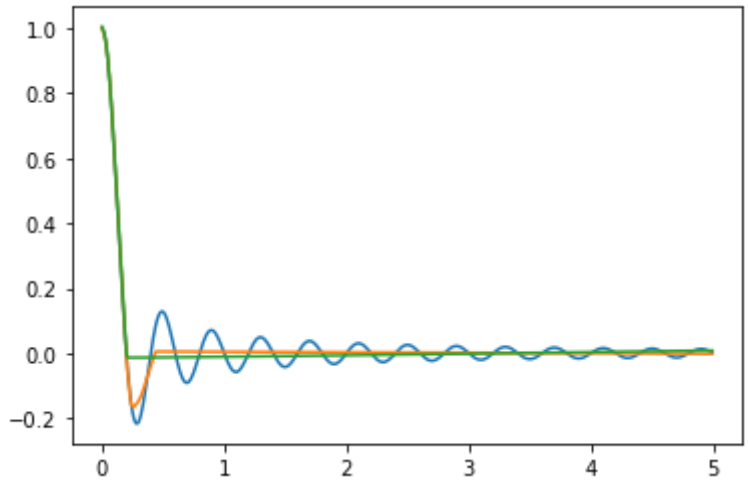
```
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:171: UserWarning: `tf.layers.dense` is deprecated and will
be removed in a future version. Please use `tf.keras.layers.Dense` instead.
  warnings.warn('`tf.layers.dense` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engi
ne/base_layer_v1.py:1719: UserWarning: `layer.apply` is deprecated and will be r
emoved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:268: UserWarning: `tf.layers.dropout` is deprecated and wil
l be removed in a future version. Please use `tf.keras.layers.Dropout` instead.
  warnings.warn('`tf.layers.dropout` is deprecated and '
```

In [27]:
```
loss_list_m2=[]
sess.run(init_m2)
for i in range(0,200):
  fd ={x:X, y:Y}
  _, loss_val_m2 = sess.run([Optimizer_m2, Loss_m2], feed_dict=fd)
  #print ('loss = %s' % loss_val)
  loss_list_m2.append(loss_val_m2)

YP_m2 = sess.run(output_layer_m2,feed_dict={x:X})
```

In [28]:
```
# Plot training  loss values
import matplotlib.pyplot as plt
plt.plot(loss_list)
plt.plot(loss_list_m2)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



In [29]:
```
plt.plot(X,Y)
plt.plot(X,YP)
plt.plot(X,YP_m2)
plt.show()
```



Describe the models you use, including the number of parameters (at least two models) and the

function you use.

In one graph, plot the predicted function curve of all models and the ground-truth function curve.

In [ ]:

In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import math
import absl
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

```
WARNING:tensorflow:From /Users/gloria/anaconda3/lib/python3.8/site-packages/tens
orflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflo
w.python.ops.variable_scope) is deprecated and will be removed in a future versi
on.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [3]:
```
tf.__version__
```

Out[3]:
```
'2.4.1'
```

In [4]:
```
from tensorflow.examples.tutorials.mnist import input_data
#data = input_data.read_data_sets('data/MNIST/', one_hot=True)
mnist = input_data.read_data_sets('./mnist', one_hot=True)  # they has been norm
test_x = mnist.test.images[:10000]
test_y = mnist.test.labels[:10000]
```

```
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
```

In [5]:
```
print("Size of:")
print("- Training-set:\t\t{}".format(len(mnist.train.labels)))
print("- Test-set:\t\t{}".format(len(mnist.test.labels)))
print("- Validation-set:\t{}".format(len(mnist.validation.labels)))
```

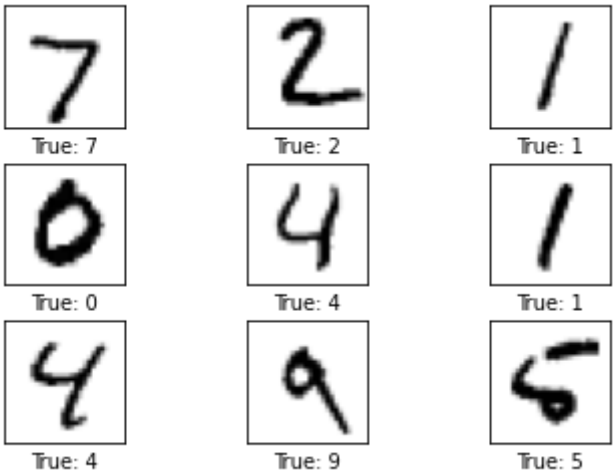
```
Size of:
- Training-set:         55000
- Test-set:             10000
- Validation-set:       5000
```

In [6]:
```
mnist.test.cls = np.argmax(mnist.test.labels, axis=1)
```

In [7]:
```
# We know that MNIST images are 28 pixels in each dimension.
img_size = 28

# Images are stored in one-dimensional arrays of this length.
img_size_flat = img_size * img_size

# Tuple with height and width of images used to reshape arrays.
img_shape = (img_size, img_size)

# Number of colour channels for the images: 1 channel for gray-scale.
num_channels = 1

# Number of classes, one class for each of 10 digits.
num_classes = 10
```

In [8]:
```
def plot_images(images, cls_true, cls_pred=None):
    assert len(images) == len(cls_true) == 9

    # Create figure with 3x3 sub-plots.
    fig, axes = plt.subplots(3, 3)
    fig.subplots_adjust(hspace=0.3, wspace=0.3)

    for i, ax in enumerate(axes.flat):
        # Plot image.
        ax.imshow(images[i].reshape(img_shape), cmap='binary')

        # Show true and predicted classes.
        if cls_pred is None:
            xlabel = "True: {0}".format(cls_true[i])
        else:
            xlabel = "True: {0}, Pred: {1}".format(cls_true[i], cls_pred[i])

        # Show the classes as the label on the x-axis.
        ax.set_xlabel(xlabel)

        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])

    # Ensure the plot is shown correctly with multiple plots
    # in a single Notebook cell.
    plt.show()
```

In [9]:
```
# Get the first images from the test-set.
images = mnist.test.images[0:9]

# Get the true classes for those images.
cls_true = mnist.test.cls[0:9]

# Plot the images and labels using our helper-function above.
plot_images(images=images, cls_true=cls_true)
```



In [10]:
```
x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x') / 255
```

In [11]:
```
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
```

In [12]:
```
y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
```

In [13]:
```
y_true_cls = tf.argmax(y_true, dimension=1)
```

In [14]:
```
conv1 = tf.layers.conv2d(   # shape (28, 28, 1)
    inputs=x_image,
    filters=16,
    kernel_size=5,
    strides=1,
    padding='same',
    activation=tf.nn.relu
)           # -> (28, 28, 16)
pool1 = tf.layers.max_pooling2d(
    conv1,
    pool_size=2,
    strides=2,
)           # -> (14, 14, 16)
conv2 = tf.layers.conv2d(pool1, 36, 5, 1, 'same', activation=tf.nn.relu)    # ->
pool2 = tf.layers.max_pooling2d(conv2, 2, 2)    # -> (7, 7, 36)
flat = tf.reshape(pool2, [-1, 7*7*36])          # -> (7*7*36, )
output = tf.layers.dense(flat, 10)              # output layer

loss = tf.losses.softmax_cross_entropy(onehot_labels=y_true, logits=output)
train_op = tf.train.AdamOptimizer(0.001).minimize(loss) #learning rate is 0.1

accuracy = tf.metrics.accuracy(          # return (acc, update_op), and create 2
    labels=tf.argmax(y_true, axis=1), predictions=tf.argmax(output, axis=1),)[1]
```

```
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/convolutional.py:414: UserWarning: `tf.layers.conv2d` is deprecated
and will be removed in a future version. Please Use `tf.keras.layers.Conv2D` ins
tead.
  warnings.warn('`tf.layers.conv2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engi
ne/base_layer_v1.py:1719: UserWarning: `layer.apply` is deprecated and will be r
emoved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/pooling.py:310: UserWarning: `tf.layers.max_pooling2d` is deprecate
d and will be removed in a future version. Please use `tf.keras.layers.MaxPoolin
g2D` instead.
  warnings.warn('`tf.layers.max_pooling2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:171: UserWarning: `tf.layers.dense` is deprecated and will
be removed in a future version. Please use `tf.keras.layers.Dense` instead.
  warnings.warn('`tf.layers.dense` is deprecated and '
```

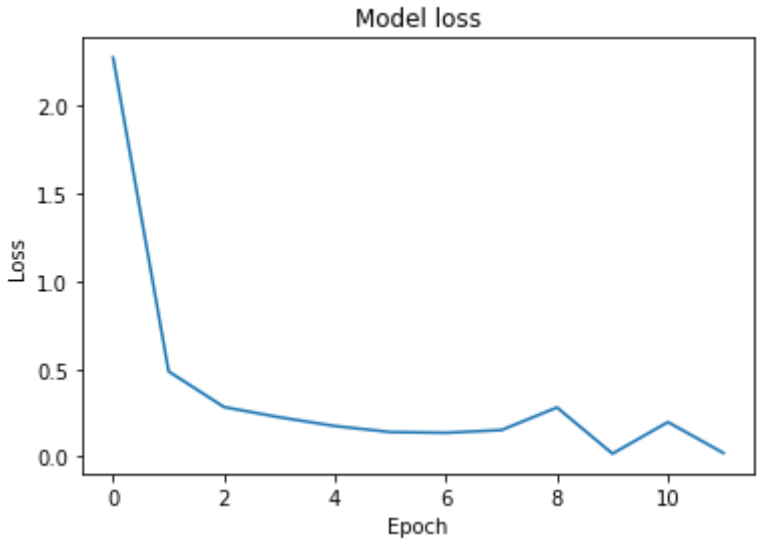
In [15]:
```
session=tf.Session()
```

In [16]:
```
init_op = tf.group(tf.global_variables_initializer(), tf.local_variables_initial
session.run(init_op)     # initialize var in graph
```

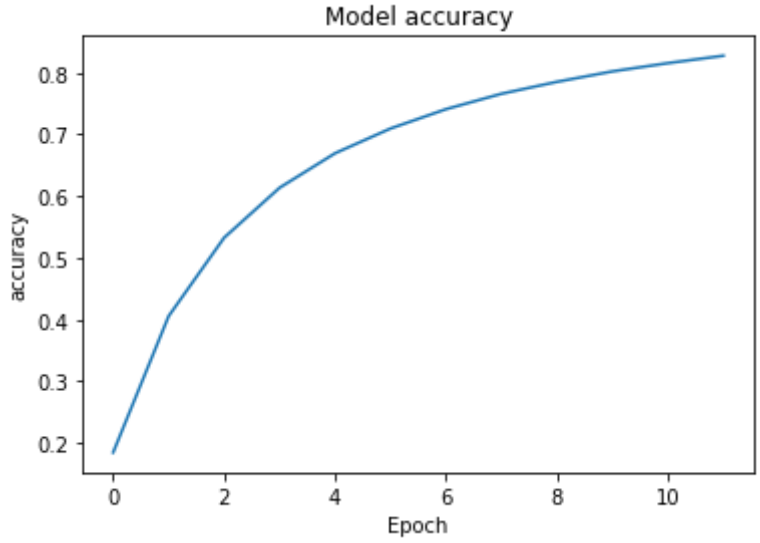
In [18]:
```
loss_list=[]
accuracy_list=[]
for step in range(600):
    b_x, b_y = mnist.train.next_batch(50) #Batch_size is 50
    _, loss_ = session.run([train_op, loss], {x: b_x, y_true: b_y})
    if step % 50 == 0:
        accuracy_, flat_representation = session.run([accuracy, flat], {x: test_
        print('Step:', step, '| train loss: %.4f' % loss_, '| test accuracy: %.2
        loss_list.append(loss_)
        accuracy_list.append(accuracy_)
```

```
Step: 0 | train loss: 2.2764 | test accuracy: 0.18
Step: 50 | train loss: 0.4868 | test accuracy: 0.41
Step: 100 | train loss: 0.2832 | test accuracy: 0.53
Step: 150 | train loss: 0.2249 | test accuracy: 0.61
Step: 200 | train loss: 0.1751 | test accuracy: 0.67
Step: 250 | train loss: 0.1406 | test accuracy: 0.71
Step: 300 | train loss: 0.1367 | test accuracy: 0.74
Step: 350 | train loss: 0.1521 | test accuracy: 0.77
Step: 400 | train loss: 0.2811 | test accuracy: 0.79
Step: 450 | train loss: 0.0176 | test accuracy: 0.80
Step: 500 | train loss: 0.1976 | test accuracy: 0.82
Step: 550 | train loss: 0.0212 | test accuracy: 0.83
```

In [19]:
```
plt.plot(loss_list)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



In [20]:
```
plt.plot(accuracy_list)
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.show()
```



In [21]:
```
test_output = session.run(output, {x: test_x[:10]})
pred_y = np.argmax(test_output, 1)
print(pred_y, 'prediction number')
print(np.argmax(test_y[:10], 1), 'real number')
```

```
[7 2 1 0 4 1 4 9 5 9] prediction number
[7 2 1 0 4 1 4 9 5 9] real number
```

In [22]:
```
session.close()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import math
import absl
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

```
WARNING:tensorflow:From /Users/gloria/anaconda3/lib/python3.8/site-packages/tens
orflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflo
w.python.ops.variable_scope) is deprecated and will be removed in a future versi
on.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:
```
from tensorflow.examples.tutorials.mnist import input_data
#data = input_data.read_data_sets('data/MNIST/', one_hot=True)
mnist = input_data.read_data_sets('./mnist', one_hot=True)  # they has been norm
test_x = mnist.test.images[:10000]
test_y = mnist.test.labels[:10000]
```

```
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
```

In [3]:
```
mnist.test.cls = np.argmax(mnist.test.labels, axis=1)
```

In [4]:
```
# We know that MNIST images are 28 pixels in each dimension.
img_size = 28

# Images are stored in one-dimensional arrays of this length.
img_size_flat = img_size * img_size

# Tuple with height and width of images used to reshape arrays.
img_shape = (img_size, img_size)

# Number of colour channels for the images: 1 channel for gray-scale.
num_channels = 1

# Number of classes, one class for each of 10 digits.
num_classes = 10
```

In [5]:
```
x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x') / 255
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)
```

create the CNN

In [6]:
```
net = tf.layers.conv2d(inputs=x_image, name='layer_conv1', padding='same',
                       filters=16, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

# layer_conv2
net = tf.layers.conv2d(inputs=net, name='layer_conv2', padding='same',
```

```
                         filters=36, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

print(net)
net = tf.layers.flatten(net)

print(net)
net = tf.layers.dense(inputs=net, name='layer_fc1',
                      units=128, activation=tf.nn.relu)
logits = tf.layers.dense(inputs=net, name='layer_fc_out',
                      units=num_classes, activation=None)
print(logits)
```

```
Tensor("max_pooling2d_1/MaxPool:0", shape=(?, 7, 7, 36), dtype=float32)
Tensor("flatten/Reshape:0", shape=(?, 1764), dtype=float32)
Tensor("layer_fc_out/BiasAdd:0", shape=(?, 10), dtype=float32)
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/convolutional.py:414: UserWarning: `tf.layers.conv2d` is deprecated
and will be removed in a future version. Please Use `tf.keras.layers.Conv2D` ins
tead.
  warnings.warn('`tf.layers.conv2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engi
ne/base_layer_v1.py:1719: UserWarning: `layer.apply` is deprecated and will be r
emoved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/pooling.py:310: UserWarning: `tf.layers.max_pooling2d` is deprecate
d and will be removed in a future version. Please use `tf.keras.layers.MaxPoolin
g2D` instead.
  warnings.warn('`tf.layers.max_pooling2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:329: UserWarning: `tf.layers.flatten` is deprecated and wil
l be removed in a future version. Please use `tf.keras.layers.Flatten` instead.
  warnings.warn('`tf.layers.flatten` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:171: UserWarning: `tf.layers.dense` is deprecated and will
be removed in a future version. Please use `tf.keras.layers.Dense` instead.
  warnings.warn('`tf.layers.dense` is deprecated and '
```

In [7]:
```
y_pred = tf.nn.softmax(logits=logits)
y_pred_cls = tf.argmax(y_pred, dimension=1)
```

In [8]:
```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=lo
loss = tf.reduce_mean(cross_entropy)
```

Optimization

In [9]:
```
opt = tf.train.AdamOptimizer(learning_rate=1e-4)
optimizer = opt.minimize(loss)
```

Classification Accuracy

In [10]:
```
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Getting the weight

In [12]:
```
trainable_var_list = tf.trainable_variables()
def get_weights_variable(layer_name):
    with tf.variable_scope(layer_name, reuse=True):
        variable = tf.get_variable('kernel')
```

```
    return variable

weights_conv1 = get_weights_variable(layer_name='layer_conv1')
weights_conv2 = get_weights_variable(layer_name='layer_conv2')
print(weights_conv1)
print(weights_conv2)

weights_fc1 = get_weights_variable(layer_name='layer_fc1')
weights_fc_out = get_weights_variable(layer_name='layer_fc_out')
print(weights_fc1)
print(weights_fc_out)
```

```
<tf.Variable 'layer_conv1/kernel:0' shape=(5, 5, 1, 16) dtype=float32_ref>
<tf.Variable 'layer_conv2/kernel:0' shape=(5, 5, 16, 36) dtype=float32_ref>
<tf.Variable 'layer_fc1/kernel:0' shape=(1764, 128) dtype=float32_ref>
<tf.Variable 'layer_fc_out/kernel:0' shape=(128, 10) dtype=float32_ref>
```

start one session, and initialize the variables

In [13]:
```
session = tf.Session()
session.run(tf.global_variables_initializer())
```

get gradient during training; compute hessian

In [14]:
```
#grad = opt.compute_gradients(loss, weights_fc_out)[0]
grads = tf.gradients(loss, weights_fc_out)[0]
print(grads)
hessian = tf.reduce_sum(tf.hessians(loss, weights_fc_out)[0], axis = 2)
print(hessian)
train_batch_size = 64
```

```
Tensor("gradients_1/layer_fc_out/MatMul_grad/MatMul_1:0", shape=(128, 10), dtype
=float32)
Tensor("Sum:0", shape=(128, 10, 10), dtype=float32)
```

start the optimization

In [34]:
```
total_iterations = 0
def optimize(num_iterations):
    global total_iterations

    for i in range(total_iterations,
                   total_iterations + num_iterations):


        x_batch, y_true_batch = mnist.train.next_batch(train_batch_size)

        feed_dict_train = {x: x_batch,
                           y_true: y_true_batch}


        session.run(optimizer, feed_dict=feed_dict_train)
        if i % 10 == 0:
            # Calculate the accuracy on the training-set.
            los, acc = session.run([loss, accuracy], feed_dict=feed_dict_train)

            grads_vals, hess_vals = session.run([grads, hessian], feed_dict=feed
            print(grads_vals.shape)
            print(hess_vals.shape)

            # Message for printing.
            msg = "Iteration: {0:>6}, Training Loss: {1:>1.6}, Training Accuracy
```

```
            # Print it.
            print(msg.format(i + 1, los, acc))

    # Update the total number of iterations performed.
    total_iterations += num_iterations
```

In [18]:
```
optimize(num_iterations=99)
```

```
(128, 10)
(128, 10, 10)
Iteration:      1, Training Loss: 2.29611, Training Accuracy:  20.3%
(128, 10)
(128, 10, 10)
Iteration:     11, Training Loss: 2.22555, Training Accuracy:  29.7%
(128, 10)
(128, 10, 10)
Iteration:     21, Training Loss: 2.17334, Training Accuracy:  29.7%
(128, 10)
(128, 10, 10)
Iteration:     31, Training Loss: 2.16627, Training Accuracy:  43.8%
(128, 10)
(128, 10, 10)
Iteration:     41, Training Loss: 2.03829, Training Accuracy:  68.8%
(128, 10)
(128, 10, 10)
Iteration:     51, Training Loss: 1.99218, Training Accuracy:  64.1%
(128, 10)
(128, 10, 10)
Iteration:     61, Training Loss: 1.75646, Training Accuracy:  73.4%
(128, 10)
(128, 10, 10)
Iteration:     71, Training Loss: 1.53628, Training Accuracy:  84.4%
(128, 10)
(128, 10, 10)
Iteration:     81, Training Loss: 1.32736, Training Accuracy:  87.5%
(128, 10)
(128, 10, 10)
Iteration:     91, Training Loss: 1.35038, Training Accuracy:  67.2%
```

In [27]:
```
def plot_conv_weights(weights, input_channel=0):

    w = session.run(weights)

    w_min = np.min(w)
    w_max = np.max(w)

    num_filters = w.shape[3]

    num_grids = math.ceil(math.sqrt(num_filters))

    fig, axes = plt.subplots(num_grids, num_grids)

    for i, ax in enumerate(axes.flat):
        if i<num_filters:

            img = w[:, :, input_channel, i]

            ax.imshow(img, vmin=w_min, vmax=w_max,
                      interpolation='nearest', cmap='seismic')

        ax.set_xticks([])
        ax.set_yticks([])
```

```
    plt.show()
```

In [28]:
```
from sklearn.decomposition import PCA
def plot_fc_weights(weights_list):

    w_list = session.run(weights_list)

    pca = PCA(n_components=2)

    fig = plt.figure(figsize = (8,8))
    ax = fig.add_subplot(1,1,1)
    ax.set_xlabel('Principal Component 1', fontsize = 15)
    ax.set_ylabel('Principal Component 2', fontsize = 15)
    ax.set_title('2 component PCA', fontsize = 20)

    for w in w_list:

        print(w.shape)

        principalComponents = pca.fit_transform(w)

        ax.scatter(principalComponents[:,0], principalComponents[:,1], label=w.s

    ax.legend()
    plt.show()
```

In [39]:
```
weights_list=[]
for step in range(100):
    if step % 3 == 0:
        weights=session.run(trainable_var_list)
        weights=PCA(weights)
        weights_list.append(weights)
```

In [40]:
```
print(weights_list)
```

```
[PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],
```

```
         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
```

```
                 array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                 array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                 array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                 array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
```

```
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
        -0.097877  ,  0.01333893],
       [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
        -0.097877  ,  0.01333893],
       [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
        -0.097877  ,  0.01333893],
       [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
```

```
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                   array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                   array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                   array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                   array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
```

4.86401096e021.26112536e013.08717322e02
4.23055924e025.49924783e028.90975073e02
6.02630228e029.03390869e021.13589466e01
1.15379550e011.19940154e011.08178295e01
9.53982845e02

1.21127687e019.15692896e022.99458336e02
7.25981370e028.97659659...
0.178886530.137434140.0086686...0.16559619
0.163273410.07722243
0.169259220.131860210.05882641...0.06325659
0.0978770.01333893
0.105820940.092717860.00425618...0.17365101
0.133021860.01772023dt eloat32
arra0.000257080.005007240.00162470.002117080.
00095305
0.000592950.002238830.001162890.002907410.00169671
dt eloat32 nom onentsarra7.02866837e021.08631827
e014.99007441e02
4.86401096e021.26112536e013.08717322e02
4.23055924e025.49924783e028.90975073e02
6.02630228e029.03390869e021.13589466e01
1.15379550e011.19940154e011.08178295e01
9.53982845e02

1.21127687e019.15692896e022.99458336e02
7.25981370e028.97659659...
0.178886530.137434140.0086686...0.16559619
0.163273410.07722243
0.169259220.131860210.05882641...0.06325659
0.0978770.01333893
0.105820940.092717860.00425618...0.17365101
0.133021860.01772023dt eloat32
arra0.000257080.005007240.00162470.002117080.
00095305
0.000592950.002238830.001162890.002907410.00169671
dt eloat32 nom onentsarra7.02866837e021.08631827
e014.99007441e02
4.86401096e021.26112536e013.08717322e02
4.23055924e025.49924783e028.90975073e02
6.02630228e029.03390869e021.13589466e01
1.15379550e011.19940154e011.08178295e01
9.53982845e02

1.21127687e019.15692896e022.99458336e02
7.25981370e028.97659659...
0.178886530.137434140.0086686...0.16559619
0.163273410.07722243
0.169259220.131860210.05882641...0.06325659
0.0978770.01333893
0.105820940.092717860.00425618...0.17365101
0.133021860.01772023dt eloat32
arra0.000257080.005007240.00162470.002117080.
00095305
0.000592950.002238830.001162890.002907410.00169671
dt eloat32 nom onentsarra7.02866837e021.08631827
e014.99007441e02
4.86401096e021.26112536e013.08717322e02
4.23055924e025.49924783e028.90975073e02
6.02630228e029.03390869e021.13589466e01
1.15379550e011.19940154e011.08178295e01
9.53982845e02

1.21127687e019.15692896e022.99458336e02
7.25981370e028.97659659...

```
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
           -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
            4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
```

```
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

         [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
           -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
         0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
       dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],
```

```
        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
         -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
        [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
         -0.16327341, -0.07722243],
        [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
         -0.097877  ,  0.01333893],
        [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
```

```
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
        -0.097877  ,  0.01333893],
       [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
        -0.097877  ,  0.01333893],
       [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
           6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
           1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
           9.53982845e-02]],

        [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
          -7.25981370e-02, -8.97659659...
       [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
        -0.16327341, -0.07722243],
       [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
        -0.097877  ,  0.01333893],
       [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
        -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
        0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
      dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
          -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
           4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
```

```
            6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
            1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
            9.53982845e-02]],

          [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
            -7.25981370e-02, -8.97659659...
         [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
          -0.16327341, -0.07722243],
         [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
          -0.097877  ,  0.01333893],
         [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
          -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
          0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
        dtype=float32)]), PCA(n_components=[array([[[[-7.02866837e-02,  1.08631827
e-01,  4.99007441e-02,
            -4.86401096e-02,  1.26112536e-01, -3.08717322e-02,
             4.23055924e-02, -5.49924783e-02,  8.90975073e-02,
             6.02630228e-02, -9.03390869e-02, -1.13589466e-01,
             1.15379550e-01,  1.19940154e-01,  1.08178295e-01,
             9.53982845e-02]],

          [[ 1.21127687e-01,  9.15692896e-02,  2.99458336e-02,
            -7.25981370e-02, -8.97659659...
         [-0.17888653,  0.13743414,  0.0086686 , ...,  0.16559619,
          -0.16327341, -0.07722243],
         [ 0.16925922,  0.13186021,  0.05882641, ..., -0.06325659,
          -0.097877  ,  0.01333893],
         [-0.10582094,  0.09271786, -0.00425618, ..., -0.17365101,
          -0.13302186,  0.01772023]], dtype=float32),
                  array([-0.00025708,  0.00500724, -0.0016247 , -0.00211708, -0.
00095305,
          0.00059295, -0.00223883,  0.00116289, -0.00290741,  0.00169671],
        dtype=float32)])]
```

In [ ]:

Github see: https://github.com/Gloriabhsfer/DeeplearningCPS843

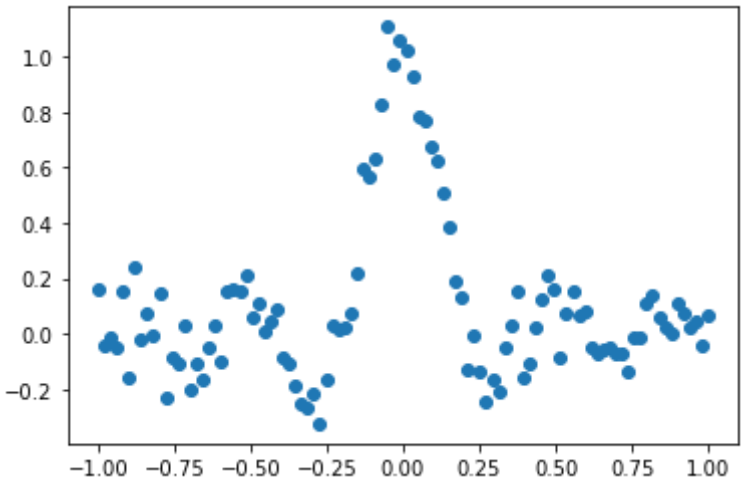
In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import math
import absl
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

```
WARNING:tensorflow:From /Users/gloria/anaconda3/lib/python3.8/site-packages/tens
orflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflo
w.python.ops.variable_scope) is deprecated and will be removed in a future versi
on.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:
```
tf.set_random_seed(1)
np.random.seed(1)
```

In [3]:
```
# fake data
x = np.linspace(-1, 1, 100)[:, np.newaxis]          # shape (100, 1)
noise = np.random.normal(0, 0.1, size=x.shape)
y = np.sinc(5*x) + noise                            # shape (100, 1) + some noise
```

In [4]:
```
plt.scatter(x, y)
plt.show()
```



In [5]:
```
input_x = tf.placeholder(tf.float32, [None, 1])      # input x
#input_x = tf.placeholder(tf.float32, [100, 1])
output_y = tf.placeholder(tf.float32, [None, 1])     # input y
#output_y = tf.placeholder(tf.float32, [100, 1])
```

In [6]:
```
# neural network layers
h1 = tf.layers.dense(inputs=input_x, units=10, activation=tf.nn.relu, name='h1')
h2 = tf.layers.dense(inputs=h1, units=10, activation=tf.nn.relu, name='h2')
output = tf.layers.dense(inputs=h2, units=1, name='output')
```

```
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:171: UserWarning: `tf.layers.dense` is deprecated and will
be removed in a future version. Please use `tf.keras.layers.Dense` instead.
```

```
  warnings.warn('`tf.layers.dense` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engi
ne/base_layer_v1.py:1719: UserWarning: `layer.apply` is deprecated and will be r
emoved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
```

In [7]:
```
loss = tf.losses.mean_squared_error(output_y, output)   # compute cost
```

In [8]:
```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5)
train_op = optimizer.minimize(loss)
```

In [9]:
```
def get_weights_variable(layer_name):
    with tf.variable_scope(layer_name, reuse=True):
        variable = tf.get_variable('kernel')
    return variable
```

In [10]:
```
weights_h1 = get_weights_variable(layer_name='h1')
weights_h2 = get_weights_variable(layer_name='h2')
print(weights_h1)
print(weights_h2)

weights_out = get_weights_variable(layer_name='output')
print(weights_out)
```

```
<tf.Variable 'h1/kernel:0' shape=(1, 10) dtype=float32_ref>
<tf.Variable 'h2/kernel:0' shape=(10, 10) dtype=float32_ref>
<tf.Variable 'output/kernel:0' shape=(10, 1) dtype=float32_ref>
```

In [11]:
```
session = tf.Session()
session.run(tf.global_variables_initializer())
```

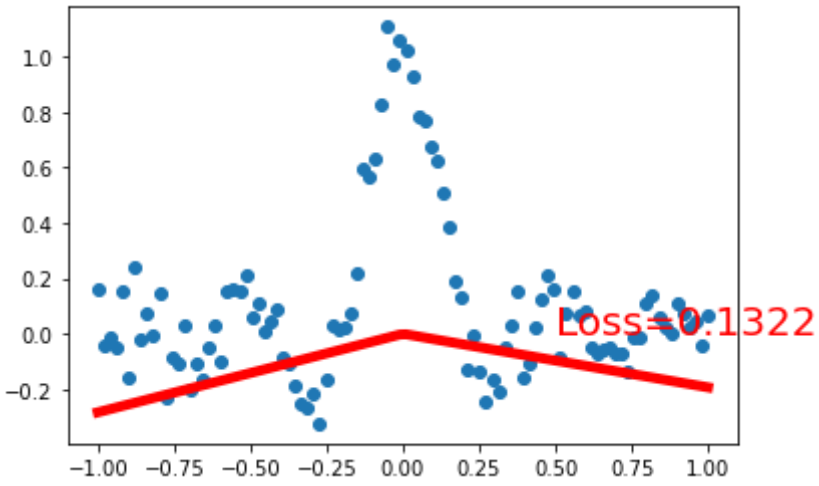
In [12]:
```
grads = tf.gradients(loss, weights_out)[0]
print(grads)
```

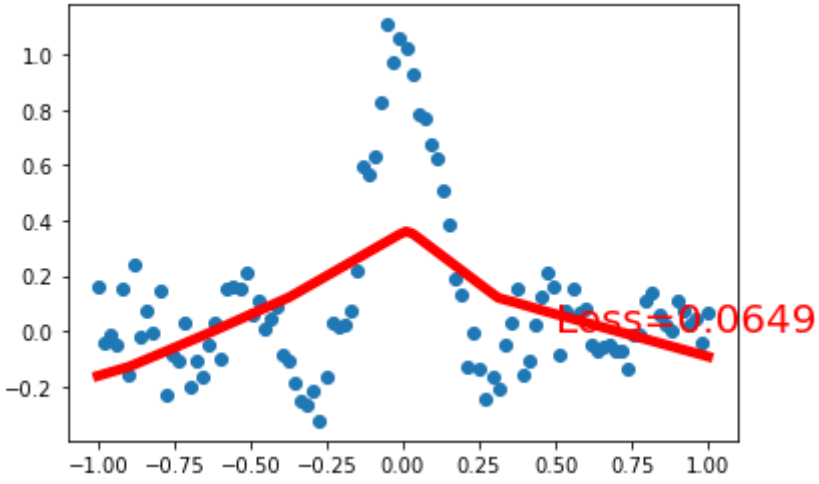
```
Tensor("gradients_1/output/MatMul_grad/MatMul_1:0", shape=(10, 1), dtype=float3
2)
```

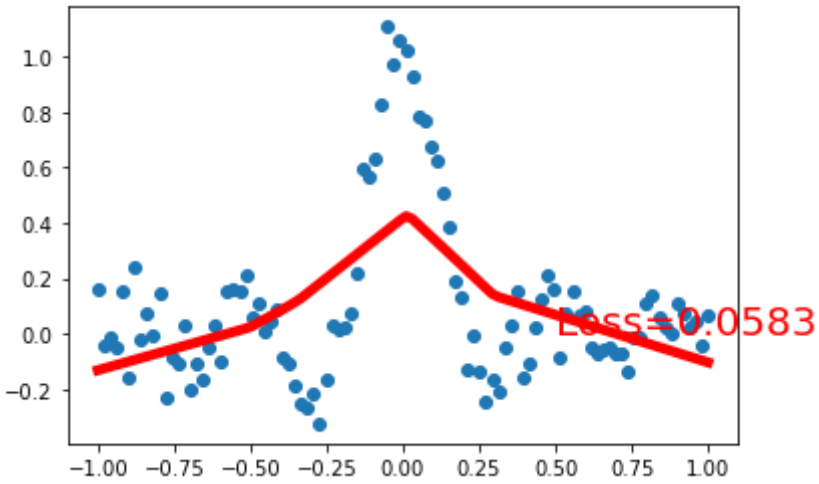
In [13]:
```
hessian = tf.reduce_sum(tf.hessians(loss, weights_out)[0], axis = 2)
print(hessian)
```

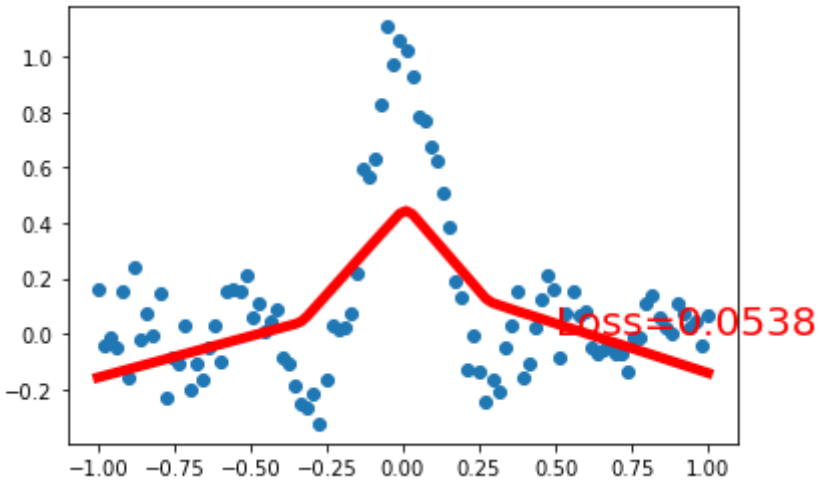
```
Tensor("Sum:0", shape=(10, 1, 1), dtype=float32)
```

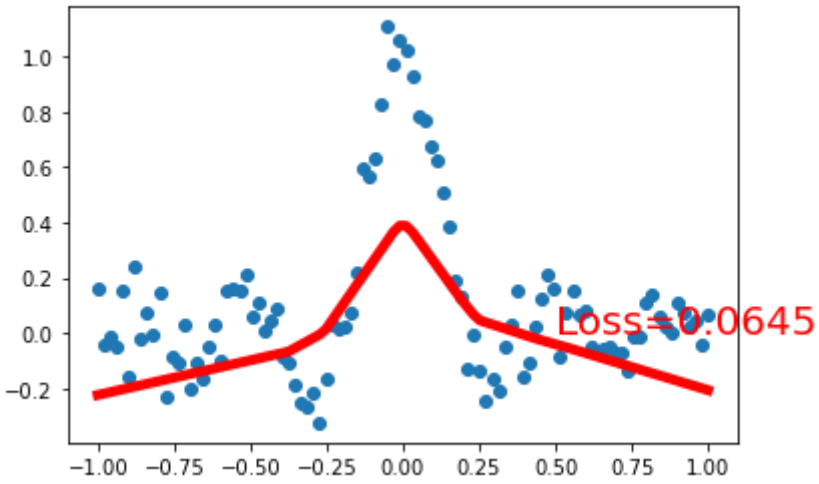
In [14]:
```
train_op = optimizer.minimize(loss)
loss = tf.losses.mean_squared_error(output_y, output)
grad_list=[]
for iteration in range(100):
    # train and net output
    _, l, pred = session.run([train_op, loss, output], feed_dict={input_x: x, ou
    grads_vals, hess_vals = session.run([grads, hessian], feed_dict={input_x: x,
    grad_list.append(grads_vals)
    if iteration % 20 == 0:
        # plot and show learning process
        plt.cla()
        plt.scatter(x, y)
        plt.plot(x, pred, 'r-', lw=5)
        plt.text(0.5, 0, 'Loss=%.4f' % l, fontdict={'size': 20, 'color': 'red'})
        plt.pause(0.1)

plt.ioff()
plt.show()
```


Loss=0.1322


Loss=0.0649


Loss=0.0583





In [15]:
```
print(np.array(grad_list)[:,1])
```

```
[[ 0.0000000e+00]
 [-7.8980060e-04]
 [-1.0695765e-03]
 [-4.8301509e-03]
 [-4.7677797e-03]
 [-9.3810521e-03]
 [-7.3561757e-03]
 [-1.2528453e-02]
 [-7.7160024e-03]
 [-1.4058841e-02]
 [-7.0987376e-03]
 [-1.4022825e-02]
 [-6.0905674e-03]
 [-1.4796937e-02]
 [-5.7100896e-03]
 [-1.4008171e-02]
 [-4.9671270e-03]
 [-1.3931324e-02]
 [-5.4148789e-03]
 [-1.3930539e-02]
 [-5.8548488e-03]
 [-1.3511472e-02]
 [-5.9597297e-03]
 [-1.4357337e-02]
 [-7.2026704e-03]
 [-1.4265257e-02]
 [-8.1384927e-03]
 [-1.3989154e-02]
```

```
[-8.6637046e-03]
[-1.3730988e-02]
[-8.7621585e-03]
[-1.3933657e-02]
[-9.1672121e-03]
[-1.3594082e-02]
[-8.9389831e-03]
[-1.4210852e-02]
[-9.4519332e-03]
[-1.3906778e-02]
[-9.7884713e-03]
[-1.4076617e-02]
[-1.0410661e-02]
[-1.4455598e-02]
[-1.0436121e-02]
[-1.5210116e-02]
[-1.0600964e-02]
[-1.5431061e-02]
[-1.0328175e-02]
[-1.6042177e-02]
[-1.0618213e-02]
[-1.6053556e-02]
[-1.0719976e-02]
[-1.6368756e-02]
[-1.0473526e-02]
[-1.6791299e-02]
[-1.0394309e-02]
[-1.6872868e-02]
[-9.8758899e-03]
[-1.7407937e-02]
[-9.6798949e-03]
[-1.7772447e-02]
[-8.8173887e-03]
[-1.8435292e-02]
[-8.3800722e-03]
[-1.8802870e-02]
[-6.5830722e-03]
[-1.9527199e-02]
[-5.4314313e-03]
[-2.0260584e-02]
[-3.2129863e-03]
[-2.0847376e-02]
[-2.2784125e-03]
[-2.1440562e-02]
[ 3.4779657e-04]
[-2.1615600e-02]
[ 9.1947493e-04]
[-2.1912297e-02]
[ 1.3826289e-03]
[-2.2122620e-02]
[ 1.5806267e-03]
[-2.2686904e-02]
[ 1.2445450e-03]
[-2.2885662e-02]
[ 9.6987031e-04]
[-2.3141067e-02]
[ 6.2804917e-05]
[-2.3551786e-02]
[-5.3269105e-05]
[-2.3642881e-02]
[ 1.2487029e-04]
[-2.3946270e-02]
[-5.6667515e-04]
[-2.4217928e-02]
[ 4.6457927e-04]
```

```
[-2.4391733e-02]
[ 1.3372998e-03]
[-2.4842920e-02]
[ 7.8088709e-04]
[-2.4964593e-02]
[ 1.9344395e-03]
[-2.5296777e-02]]
```

In [16]:
```
min(np.array(grad_list)[:,1])
```

Out[16]:
```
array([-0.02529678], dtype=float32)
```

In [ ]:
```

```

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import math
import absl
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

```
WARNING:tensorflow:From /Users/gloria/anaconda3/lib/python3.8/site-packages/tens
orflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflo
w.python.ops.variable_scope) is deprecated and will be removed in a future versi
on.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:
```
from tensorflow.examples.tutorials.mnist import input_data
#data = input_data.read_data_sets('data/MNIST/', one_hot=True)
mnist = input_data.read_data_sets('./mnist', one_hot=True)  # they has been norm
test_x = mnist.test.images[:10000]
test_y = mnist.test.labels[:10000]
```

```
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
```

In [3]:
```
mnist.test.cls = np.argmax(mnist.test.labels, axis=1)
```

In [4]:
```
# We know that MNIST images are 28 pixels in each dimension.
img_size = 28

# Images are stored in one-dimensional arrays of this length.
img_size_flat = img_size * img_size

# Tuple with height and width of images used to reshape arrays.
img_shape = (img_size, img_size)

# Number of colour channels for the images: 1 channel for gray-scale.
num_channels = 1

# Number of classes, one class for each of 10 digits.
num_classes = 10
x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x') / 255
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)
```

In [5]:
```
np.random.shuffle(mnist.train.labels)
```

In [6]:
```
#network1
net = tf.layers.conv2d(inputs=x_image, name='layer_conv1_1', padding='same',
                       filters=16, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

# layer_conv2
```

```
net = tf.layers.conv2d(inputs=net, name='layer_conv2_1', padding='same',
                       filters=36, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

print(net)
net = tf.layers.flatten(net)

print(net)
net = tf.layers.dense(inputs=net, name='layer_fc1_1',
                      units=128, activation=tf.nn.relu)
logits = tf.layers.dense(inputs=net, name='layer_fc_out_1',
                      units=num_classes, activation=None)
print(logits)
y_pred = tf.nn.softmax(logits=logits)
y_pred_cls = tf.argmax(y_pred, dimension=1)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=lo
loss = tf.reduce_mean(cross_entropy)
```

```
Tensor("max_pooling2d_1/MaxPool:0", shape=(?, 7, 7, 36), dtype=float32)
Tensor("flatten/Reshape:0", shape=(?, 1764), dtype=float32)
Tensor("layer_fc_out_1/BiasAdd:0", shape=(?, 10), dtype=float32)
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/convolutional.py:414: UserWarning: `tf.layers.conv2d` is deprecated
and will be removed in a future version. Please Use `tf.keras.layers.Conv2D` ins
tead.
  warnings.warn('`tf.layers.conv2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engi
ne/base_layer_v1.py:1719: UserWarning: `layer.apply` is deprecated and will be r
emoved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/pooling.py:310: UserWarning: `tf.layers.max_pooling2d` is deprecate
d and will be removed in a future version. Please use `tf.keras.layers.MaxPoolin
g2D` instead.
  warnings.warn('`tf.layers.max_pooling2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:329: UserWarning: `tf.layers.flatten` is deprecated and wil
l be removed in a future version. Please use `tf.keras.layers.Flatten` instead.
  warnings.warn('`tf.layers.flatten` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:171: UserWarning: `tf.layers.dense` is deprecated and will
be removed in a future version. Please use `tf.keras.layers.Dense` instead.
  warnings.warn('`tf.layers.dense` is deprecated and '
```

In [7]:
```
def get_weights_variable(layer_name):

    with tf.variable_scope(layer_name, reuse=True):
        variable = tf.get_variable('kernel')

    return variable

total_weights = []
weights_conv1 = get_weights_variable(layer_name='layer_fc_out_1')

print(weights_conv1)
```

```
<tf.Variable 'layer_fc_out_1/kernel:0' shape=(128, 10) dtype=float32_ref>
```

In [11]:
```
opt = tf.train.AdamOptimizer(learning_rate=1e-4)
optimizer = opt.minimize(loss)
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

In [13]:
```
train_batch_size = 64
loss_list=[]
loss_list_test=[]
accuracy_list=[]
accuracy_list_test=[]
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for iteration in range(200000):
        x_batch, y_true_batch = mnist.train.next_batch(train_batch_size)
        feed_dict_train = {x: x_batch, y_true: y_true_batch}
        sess.run(optimizer, feed_dict = feed_dict_train)
        if iteration % 100 == 0:
            feed_dict_test = {x: mnist.test.images, y_true: mnist.test.labels}
            y_true_cls_batch = np.argmax(y_true_batch, axis=1)
            loss_,accuracy_ = sess.run([loss,  accuracy], feed_dict = feed_dict_
            loss_test_, accuracy_test_ = sess.run([loss, accuracy], feed_dict =
            loss_list.append(loss_)
            loss_list_test.append(loss_test_)
            accuracy_list.append(accuracy_)
            accuracy_list_test.append(accuracy_test_)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-13-c29c6df58936> in <module>
      9             x_batch, y_true_batch = mnist.train.next_batch(train_batch_size)
     10             feed_dict_train = {x: x_batch, y_true: y_true_batch}
---> 11             sess.run(optimizer, feed_dict = feed_dict_train)
     12             if iteration % 100 == 0:
     13                 feed_dict_test = {x: mnist.test.images, y_true: mnist.test.l
abels}

~/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py in r
un(self, fetches, feed_dict, options, run_metadata)
    965
    966     try:
--> 967       result = self._run(None, fetches, feed_dict, options_ptr,
    968                          run_metadata_ptr)
    969       if run_metadata:

~/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py in _
run(self, handle, fetches, feed_dict, options, run_metadata)
   1188     # or if the call is a partial run that specifies feeds.
   1189     if final_fetches or final_targets or (handle and feed_dict_tensor):
-> 1190       results = self._do_run(handle, final_targets, final_fetches,
   1191                              feed_dict_tensor, options, run_metadata)
   1192     else:

~/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py in _
do_run(self, handle, target_list, fetch_list, feed_dict, options, run_metadata)
   1366
   1367     if handle is None:
-> 1368       return self._do_call(_run_fn, feeds, fetches, targets, options,
   1369                            run_metadata)
   1370     else:

~/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py in _
do_call(self, fn, *args)
   1373   def _do_call(self, fn, *args):
   1374     try:
-> 1375       return fn(*args)
   1376     except errors.OpError as e:
   1377       message = compat.as_text(e.message)
```

```
~/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py in _
run_fn(feed_dict, fetch_list, target_list, options, run_metadata)
   1357       # Ensure any changes to the graph are reflected in the runtime.
   1358       self._extend_graph()
-> 1359       return self._call_tf_sessionrun(options, feed_dict, fetch_list,
   1360                                       target_list, run_metadata)
   1361

~/anaconda3/lib/python3.8/site-packages/tensorflow/python/client/session.py in _
call_tf_sessionrun(self, options, feed_dict, fetch_list, target_list, run_metada
ta)
   1449   def _call_tf_sessionrun(self, options, feed_dict, fetch_list, target_l
ist,
   1450                           run_metadata):
-> 1451     return tf_session.TF_SessionRun_wrapper(self._session, options, feed
_dict,
   1452                                             fetch_list, target_list,
   1453                                             run_metadata)

KeyboardInterrupt:
```

In [ ]:

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import math
import absl
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

```
WARNING:tensorflow:From /Users/gloria/anaconda3/lib/python3.8/site-packages/tens
orflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflo
w.python.ops.variable_scope) is deprecated and will be removed in a future versi
on.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:
```
from tensorflow.examples.tutorials.mnist import input_data
#data = input_data.read_data_sets('data/MNIST/', one_hot=True)
mnist = input_data.read_data_sets('./mnist', one_hot=True)  # they has been norm
test_x = mnist.test.images[:10000]
test_y = mnist.test.labels[:10000]
```

```
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
```

In [3]:
```
mnist.test.cls = np.argmax(mnist.test.labels, axis=1)
```

In [5]:
```
# We know that MNIST images are 28 pixels in each dimension.
img_size = 28

# Images are stored in one-dimensional arrays of this length.
img_size_flat = img_size * img_size

# Tuple with height and width of images used to reshape arrays.
img_shape = (img_size, img_size)

# Number of colour channels for the images: 1 channel for gray-scale.
num_channels = 1

# Number of classes, one class for each of 10 digits.
num_classes = 10
```

In [6]:
```
x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x') / 255
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)
```

create the CNN

In [10]:
```
conv1 = tf.layers.conv2d(   # shape (28, 28, 1)
    inputs=x_image,
    filters=16,
    kernel_size=5,
    strides=1,
    padding='same',
```

```
    activation=tf.nn.relu
)           # -> (28, 28, 16)
pool1 = tf.layers.max_pooling2d(
    conv1,
    pool_size=2,
    strides=2,
)           # -> (14, 14, 16)
conv2 = tf.layers.conv2d(pool1, 36, 5, 1, 'same', activation=tf.nn.relu)    # ->
pool2 = tf.layers.max_pooling2d(conv2, 2, 2)    # -> (7, 7, 36)
flat = tf.reshape(pool2, [-1, 7*7*36])          # -> (7*7*36, )
output = tf.layers.dense(flat, 10)              # output layer

loss = tf.losses.softmax_cross_entropy(onehot_labels=y_true, logits=output)
train_op = tf.train.AdamOptimizer(0.001).minimize(loss) #learning rate is 0.1

accuracy = tf.metrics.accuracy(          # return (acc, update_op), and create 2
    labels=tf.argmax(y_true, axis=1), predictions=tf.argmax(output, axis=1),)[1]
```

```
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/convolutional.py:414: UserWarning: `tf.layers.conv2d` is deprecated
and will be removed in a future version. Please Use `tf.keras.layers.Conv2D` ins
tead.
  warnings.warn('`tf.layers.conv2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/engi
ne/base_layer_v1.py:1719: UserWarning: `layer.apply` is deprecated and will be r
emoved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/pooling.py:310: UserWarning: `tf.layers.max_pooling2d` is deprecate
d and will be removed in a future version. Please use `tf.keras.layers.MaxPoolin
g2D` instead.
  warnings.warn('`tf.layers.max_pooling2d` is deprecated and '
/Users/gloria/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/lega
cy_tf_layers/core.py:171: UserWarning: `tf.layers.dense` is deprecated and will
be removed in a future version. Please use `tf.keras.layers.Dense` instead.
  warnings.warn('`tf.layers.dense` is deprecated and '
```

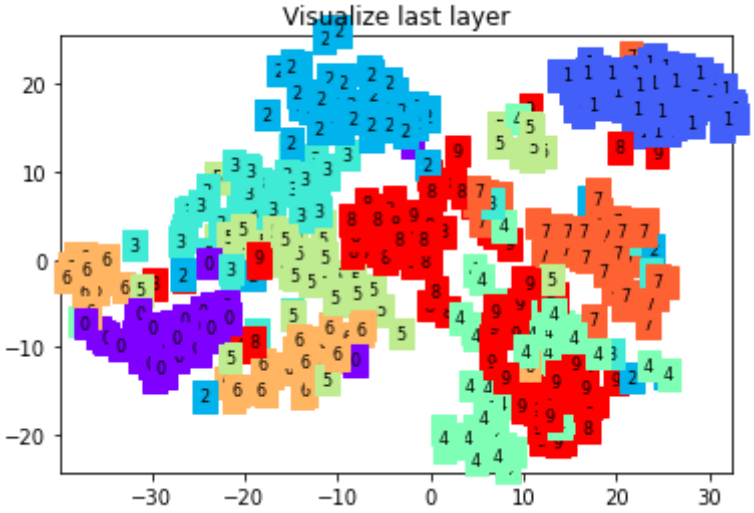
In [11]:
```
from matplotlib import cm
try: from sklearn.manifold import TSNE; HAS_SK = True
except: HAS_SK = False; print('\nPlease install sklearn for layer visualization\
def plot_with_labels(lowDWeights, labels):
    plt.cla(); X, Y = lowDWeights[:, 0], lowDWeights[:, 1]
    for x, y, s in zip(X, Y, labels):
        c = cm.rainbow(int(255 * s / 8)); plt.text(x, y, s, backgroundcolor=c, f
    plt.xlim(X.min(), X.max()); plt.ylim(Y.min(), Y.max()); plt.title('Visualize

plt.ion()
```

In [12]:
```
session=tf.Session()
init_op = tf.group(tf.global_variables_initializer(), tf.local_variables_initial
session.run(init_op)     # initialize var in graph
```

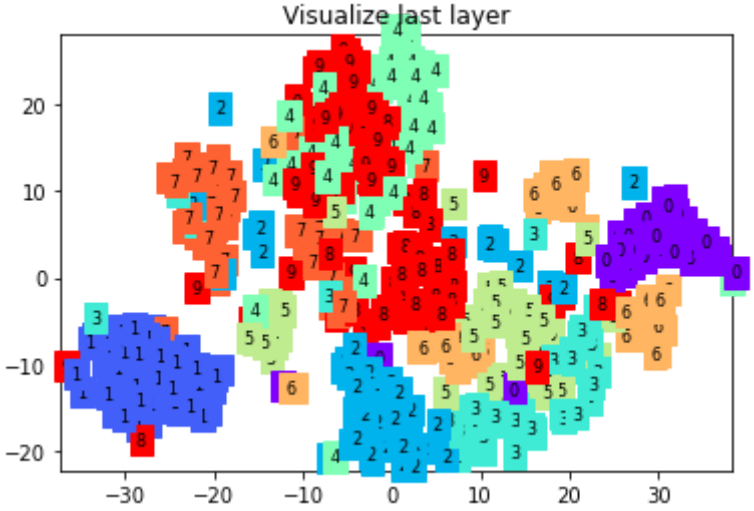
In [13]:
```
loss_list=[]
accuracy_list=[]
for step in range(600):
    b_x, b_y = mnist.train.next_batch(50) #Batch_size is 50
    _, loss_ = session.run([train_op, loss], {x: b_x, y_true: b_y})
    if step % 50 == 0:
        accuracy_, flat_representation = session.run([accuracy, flat], {x: test_
        print('Step:', step, '| train loss: %.4f' % loss_, '| test accuracy: %.2
        loss_list.append(loss_)
```

```
        accuracy_list.append(accuracy_)
        if HAS_SK:
            # Visualization of trained flatten layer (T-SNE)
            tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000);
            low_dim_embs = tsne.fit_transform(flat_representation[:plot_only, :]
            labels = np.argmax(test_y, axis=1)[:plot_only]; plot_with_labels(low
plt.ioff()
```

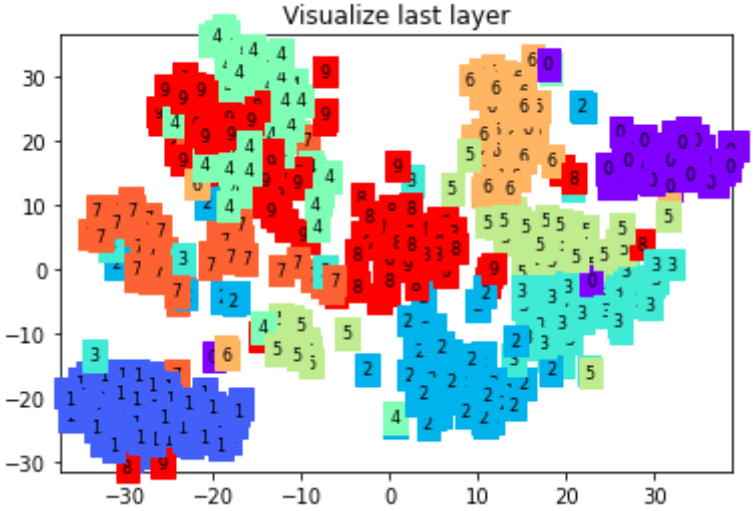
```
Step: 0 | train loss: 2.2945 | test accuracy: 0.18
```



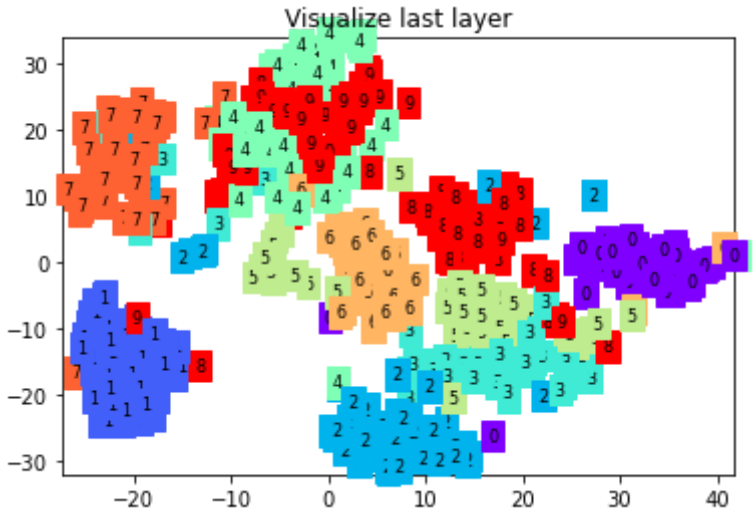

```
Step: 50 | train loss: 0.4907 | test accuracy: 0.53
```



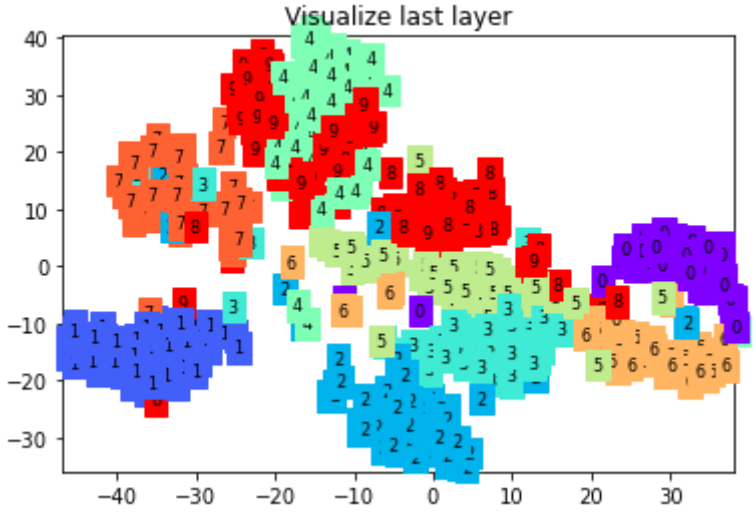

```
Step: 100 | train loss: 0.3523 | test accuracy: 0.66
```



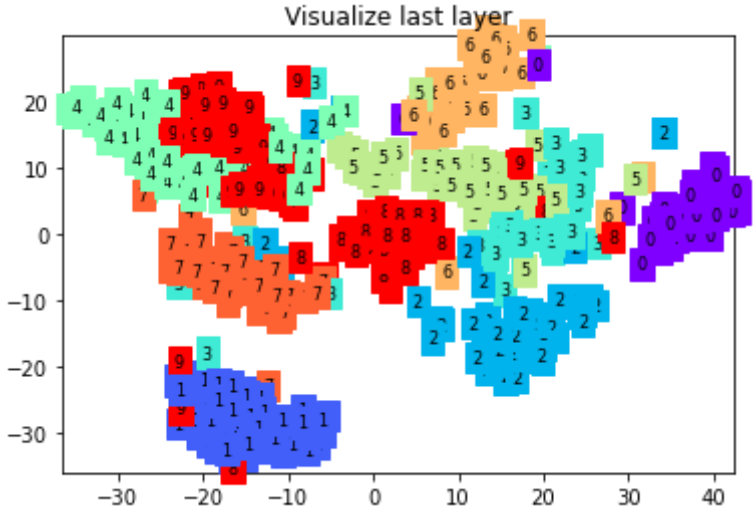
```
Step: 150 | train loss: 0.2431 | test accuracy: 0.73
```



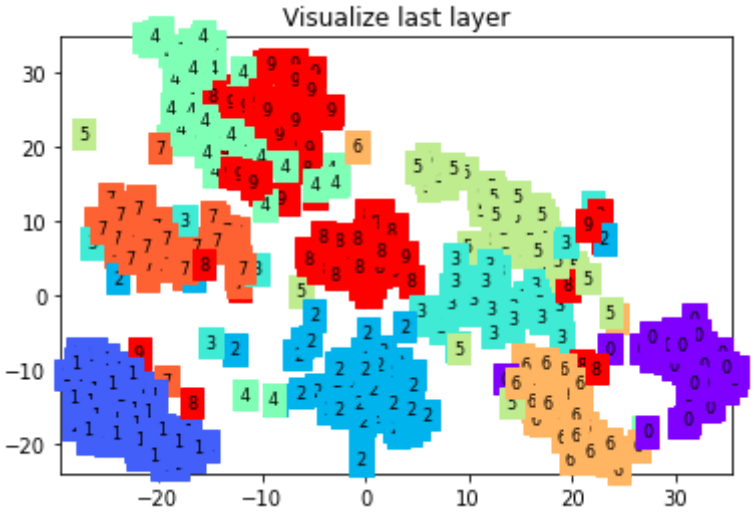

```
Step: 200 | train loss: 0.1333 | test accuracy: 0.77
```



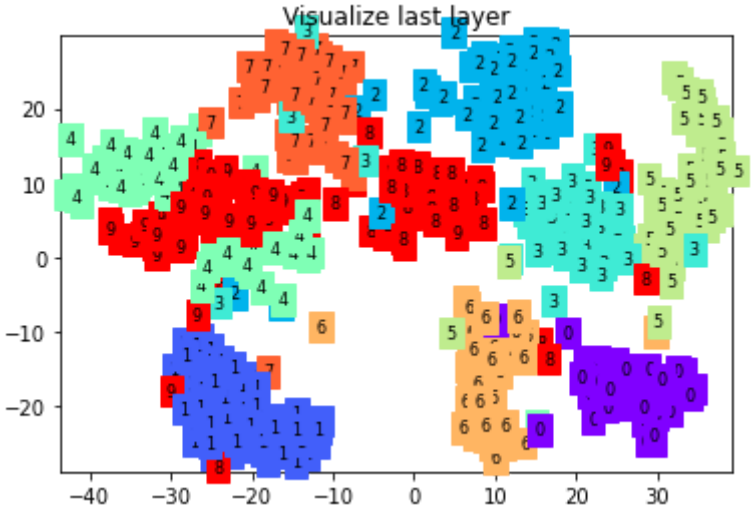

```
Step: 250 | train loss: 0.1475 | test accuracy: 0.80
```



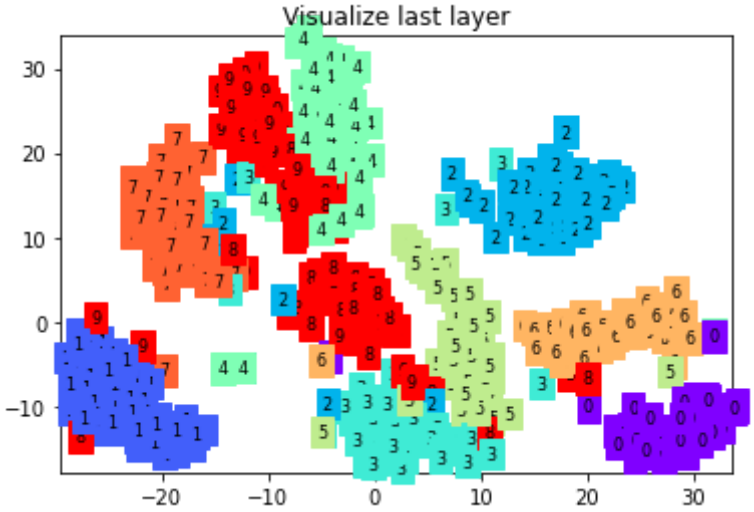

```
Step: 300 | train loss: 0.1584 | test accuracy: 0.83
```



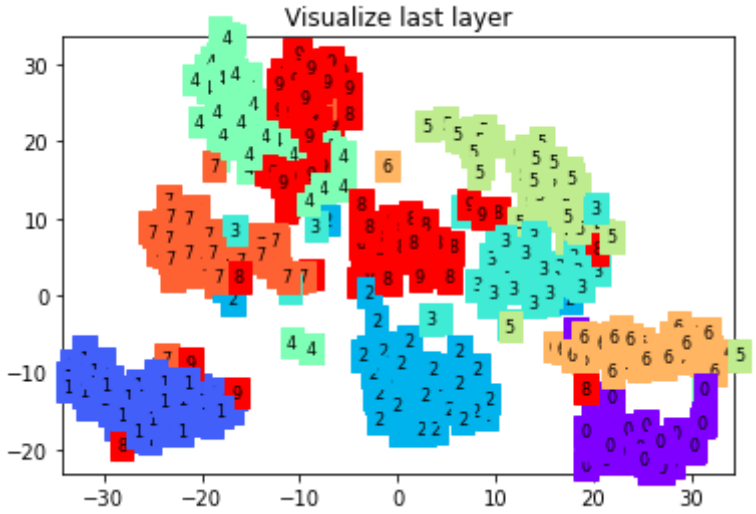

```
Step: 350 | train loss: 0.0986 | test accuracy: 0.84
```



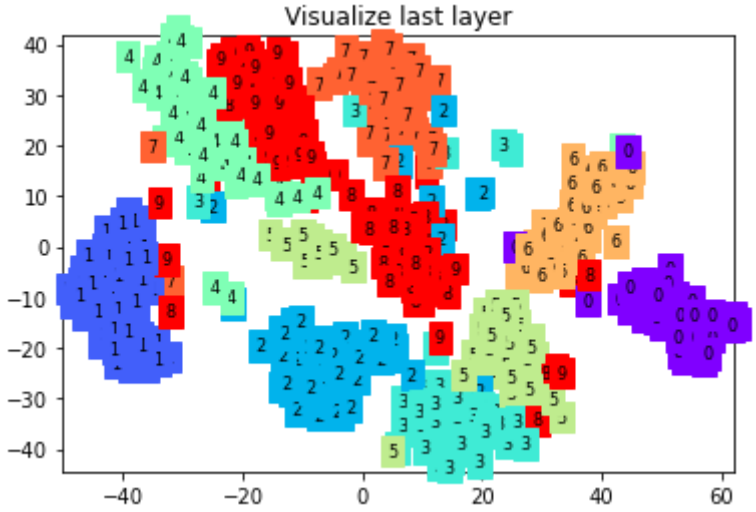

```
Step: 400 | train loss: 0.0751 | test accuracy: 0.86
```



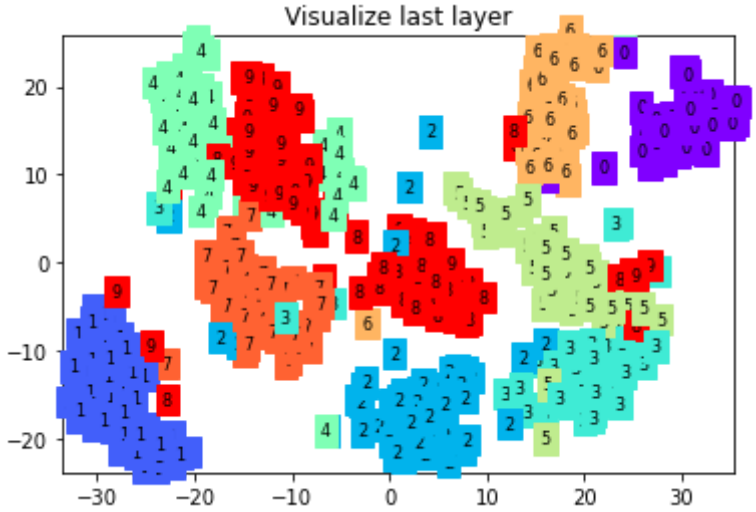

```
Step: 450 | train loss: 0.1182 | test accuracy: 0.87
```




```
Step: 500 | train loss: 0.1666 | test accuracy: 0.88
```




```
Step: 550 | train loss: 0.1771 | test accuracy: 0.88
```



In [14]:
```
test_output = session.run(output, {x: test_x[:10]})
pred_y = np.argmax(test_output, 1)
print(pred_y, 'prediction number')
print(np.argmax(test_y[:10], 1), 'real number')
```

```
[7 2 1 0 4 1 4 9 5 9] prediction number
[7 2 1 0 4 1 4 9 5 9] real number
```

Github see: https://github.com/Gloriabhsfer/DeeplearningCPS843

In [ ]: