Student Name: Yuan-Yao Lou

ECE 595 Machine Learning II

Project 4: Adversarial Machine Learning - Student Code

\

In [3]:

```
# Install Cleverhans (version Cleverhans 2.1.0 is most compatable with Python 2.x)
!pip install cleverhans==2.1.0
```

```
Requirement already satisfied: cleverhans==2.1.0 in /usr/local/lib/python3.7/dist-packages (2.1.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from cleverhans==2.1.
0) (3.2.2)
Requirement already satisfied: mnist~=0.2 in /usr/local/lib/python3.7/dist-packages (from cleverhans==2.1.
0) (0.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from cleverhans==2.1.0) (1
.19.5)
Requirement already satisfied: nose in /usr/local/lib/python3.7/dist-packages (from cleverhans==2.1.0) (1.
3.7)
Requirement already satisfied: pycodestyle in /usr/local/lib/python3.7/dist-packages (from cleverhans==2.1
.0) (2.8.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from cleverhans==2.1.0) (1
.4.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-p
ackages (from matplotlib->cleverhans==2.1.0) (3.0.6)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->cl
everhans==2.1.0) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplo
tlib->cleverhans==2.1.0) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotli
b->cleverhans==2.1.0) (1.3.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2
.1->matplotlib->cleverhans==2.1.0) (1.15.0)
```

In [4]:

```
%tensorflow_version 1.x

# Import necessary packages
from keras.datasets import mnist
from keras import Sequential
from keras.layers import Dense, BatchNormalization
from keras import backend
import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from cleverhans.utils_keras import KerasModelWrapper
from cleverhans.attacks import FastGradientMethod, MadryEtAl, DeepFool, CarliniWagnerL2
```

```
TensorFlow 1.x selected.
```

```
Using TensorFlow backend.
```

Part 1: Training a target classifier

In [6]:

```
# Load data MNIST data and normalize to [0, 1]
(data_train, labels_train), (data_test, labels_test) = mnist.load_data()
data_train = data_train / 255.0
data_test  = data_test  / 255.0

# Reshape training and testing data into 784-dimensional vectors
data_train = data_train.reshape(60000, 784)
```

```python
data_test  = data_test.reshape( 10000, 784)

# Convert integer labels for training and testing data into one-hot vectors
labels_train = keras.utils.np_utils.to_categorical(labels_train, num_classes=10)
labels_test  = keras.utils.np_utils.to_categorical(labels_test,  num_classes=10)

# Create classifier architecture, compile it, and train it
def Classifier():
    model = Sequential()

    model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
    model.add(BatchNormalization())

    model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
    model.add(BatchNormalization())

    model.add(Dense(10, activation='softmax'))

    return model

classifier = Classifier()
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics =['accuracy'])
history = classifier.fit(
                    data_train, labels_train,
                    validation_data=(data_test, labels_test),
                    epochs=50, batch_size=256, shuffle=True
                )
_, test_acc = classifier.evaluate(data_test, labels_test)
```

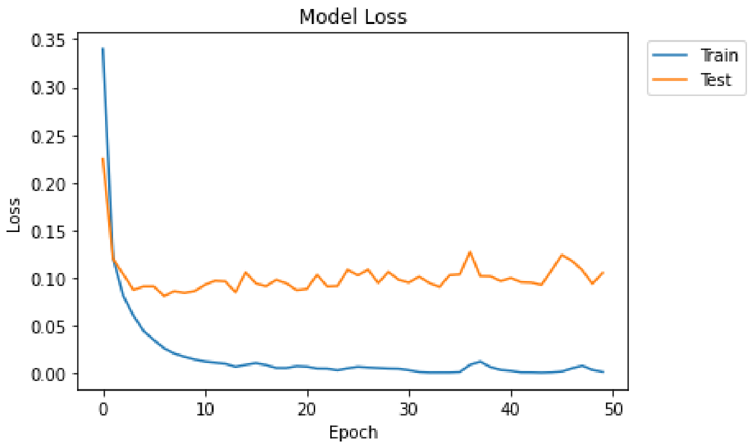
```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_backend.py:422: The name tf.
global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.3401 - accuracy: 0.8993 - val_loss:
0.2246 - val_accuracy: 0.9479
Epoch 2/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1212 - accuracy: 0.9653 - val_loss:
0.1201 - val_accuracy: 0.9617
Epoch 3/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0816 - accuracy: 0.9766 - val_loss:
0.1041 - val_accuracy: 0.9690
Epoch 4/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0606 - accuracy: 0.9824 - val_loss:
0.0876 - val_accuracy: 0.9720
Epoch 5/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0447 - accuracy: 0.9872 - val_loss:
0.0913 - val_accuracy: 0.9721
Epoch 6/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0350 - accuracy: 0.9896 - val_loss:
0.0913 - val_accuracy: 0.9733
Epoch 7/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0266 - accuracy: 0.9926 - val_loss:
0.0812 - val_accuracy: 0.9762
Epoch 8/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0211 - accuracy: 0.9940 - val_loss:
0.0862 - val_accuracy: 0.9750
Epoch 9/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0176 - accuracy: 0.9952 - val_loss:
0.0845 - val_accuracy: 0.9744
Epoch 10/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0147 - accuracy: 0.9961 - val_loss:
0.0863 - val_accuracy: 0.9754
Epoch 11/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0127 - accuracy: 0.9965 - val_loss:
0.0930 - val_accuracy: 0.9718
Epoch 12/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0115 - accuracy: 0.9969 - val_loss:
0.0973 - val_accuracy: 0.9740
Epoch 13/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0103 - accuracy: 0.9970 - val_loss:
0.0966 - val_accuracy: 0.9732
Epoch 14/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0072 - accuracy: 0.9982 - val_loss:
0.0852 - val_accuracy: 0.9763
Epoch 15/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0092 - accuracy: 0.9972 - val_loss:
0.1062 - val_accuracy: 0.9748
Epoch 16/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0111 - accuracy: 0.9968 - val_loss:
0.0944 - val_accuracy: 0.9770
Epoch 17/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0090 - accuracy: 0.9972 - val_loss:
```

```
0.0915 - val_accuracy: 0.9769
Epoch 18/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0059 - accuracy: 0.9985 - val_loss:
0.0982 - val_accuracy: 0.9753
Epoch 19/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0059 - accuracy: 0.9985 - val_loss:
0.0945 - val_accuracy: 0.9770
Epoch 20/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0078 - accuracy: 0.9976 - val_loss:
0.0874 - val_accuracy: 0.9779
Epoch 21/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0073 - accuracy: 0.9976 - val_loss:
0.0887 - val_accuracy: 0.9772
Epoch 22/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0053 - accuracy: 0.9984 - val_loss:
0.1035 - val_accuracy: 0.9762
Epoch 23/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0052 - accuracy: 0.9984 - val_loss:
0.0912 - val_accuracy: 0.9777
Epoch 24/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0035 - accuracy: 0.9991 - val_loss:
0.0917 - val_accuracy: 0.9760
Epoch 25/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0055 - accuracy: 0.9984 - val_loss:
0.1088 - val_accuracy: 0.9761
Epoch 26/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0071 - accuracy: 0.9977 - val_loss:
0.1031 - val_accuracy: 0.9760
Epoch 27/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0064 - accuracy: 0.9980 - val_loss:
0.1090 - val_accuracy: 0.9765
Epoch 28/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0058 - accuracy: 0.9982 - val_loss:
0.0949 - val_accuracy: 0.9782
Epoch 29/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0053 - accuracy: 0.9984 - val_loss:
0.1065 - val_accuracy: 0.9776
Epoch 30/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0051 - accuracy: 0.9984 - val_loss:
0.0983 - val_accuracy: 0.9777
Epoch 31/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0036 - accuracy: 0.9989 - val_loss:
0.0953 - val_accuracy: 0.9802
Epoch 32/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0017 - accuracy: 0.9997 - val_loss:
0.1016 - val_accuracy: 0.9788
Epoch 33/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0011 - accuracy: 0.9998 - val_loss:
0.0949 - val_accuracy: 0.9796
Epoch 34/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0011 - accuracy: 0.9998 - val_loss:
0.0908 - val_accuracy: 0.9810
Epoch 35/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0012 - accuracy: 0.9997 - val_loss:
0.1032 - val_accuracy: 0.9782
Epoch 36/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0016 - accuracy: 0.9997 - val_loss:
0.1040 - val_accuracy: 0.9788
Epoch 37/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0093 - accuracy: 0.9967 - val_loss:
0.1274 - val_accuracy: 0.9733
Epoch 38/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0125 - accuracy: 0.9955 - val_loss:
0.1022 - val_accuracy: 0.9788
Epoch 39/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0068 - accuracy: 0.9976 - val_loss:
0.1020 - val_accuracy: 0.9795
Epoch 40/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0039 - accuracy: 0.9988 - val_loss:
0.0969 - val_accuracy: 0.9800
Epoch 41/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0029 - accuracy: 0.9992 - val_loss:
0.1001 - val_accuracy: 0.9798
Epoch 42/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0013 - accuracy: 0.9997 - val_loss:
0.0958 - val_accuracy: 0.9798
Epoch 43/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0013 - accuracy: 0.9998 - val_loss:
0.0953 - val_accuracy: 0.9815
Epoch 44/50
60000/60000 [==============================] - 2s 36us/step - loss: 9.5756e-04 - accuracy: 0.9998 - val_lo
ss: 0.0929 - val_accuracy: 0.9818
Epoch 45/50
```

```
60000/60000 [==============================] - 2s 36us/step - loss: 0.0013 - accuracy: 0.9998 - val_loss:
0.1084 - val_accuracy: 0.9790
Epoch 46/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0022 - accuracy: 0.9994 - val_loss:
0.1241 - val_accuracy: 0.9764
Epoch 47/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0055 - accuracy: 0.9982 - val_loss:
0.1180 - val_accuracy: 0.9774
Epoch 48/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0083 - accuracy: 0.9970 - val_loss:
0.1083 - val_accuracy: 0.9768
Epoch 49/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.0039 - accuracy: 0.9990 - val_loss:
0.0941 - val_accuracy: 0.9806
Epoch 50/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0018 - accuracy: 0.9995 - val_loss:
0.1055 - val_accuracy: 0.9805
10000/10000 [==============================] - 1s 86us/step
```

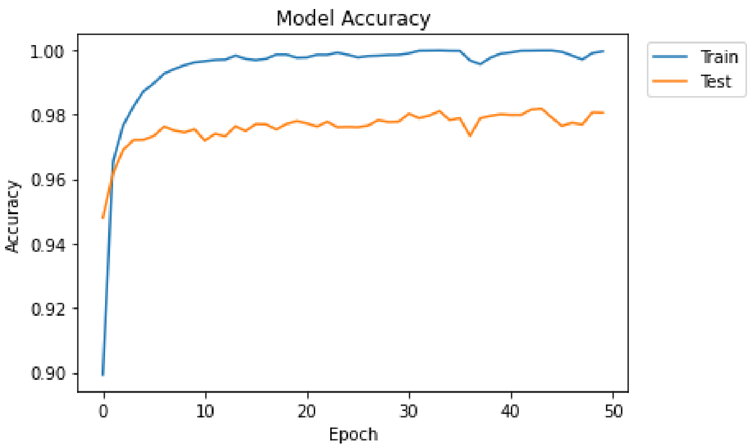
In [7]:

```
# Plot loss vs epoch
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], bbox_to_anchor=(1.23, 1), loc='upper right')
plt.show()
```



In [8]:

```
# Plot accuracy vs epoch
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], bbox_to_anchor=(1.23, 1), loc='upper right')
plt.show()
```



In [9]:

```
# Print accuracy of classifier on MNIST testing data
print('Accuracy (of the trained classifier on the MNIST testing data): {0}'.format(test_acc))
```

```
Accuracy (of the trained classifier on the MNIST testing data): 0.9804999828338623
```

In [10]:

```
# Edit the classifier name fed into KerasModel Wrapper with the name of the
# classifier from above and then run this block

# Get TensorFlow Session to pass into Cleverhans modules
sess = backend.get_session()

# Create wrapper for classifier model so that it can be passed into Cleverhans modules
wrap = KerasModelWrapper(classifier)
```

In [11]:

```
def print_samples(msg, data):
    row, col = 2, 5

    print(msg)
    fig, ax = plt.subplots(row, col)
    for i in range(row):
        for j in range(col):
            ax[i, j].imshow((data[i*col + j].reshape(28, 28)), cmap='gray')
            ax[i, j].axis('off')
    plt.show()
```

Part 2: The Fast Gradient Method (FGM)

In [19]:

```
# Implementing the FGSM attack

# FGM Instance on trained classifier from Part 1
FGM = FastGradientMethod(wrap, sess=sess)

# Attack parameters
attack_param = {
            'eps':      0.25,
            'clip_min': 0.0,
            'clip_max': 1.0
        }

# Generate adversarial data
FGM_data_test = FGM.generate_np(data_test, **attack_param)

# Evaluate accuracy on target classifier
FGM_score = classifier.evaluate(FGM_data_test, labels_test)
print('\nAccuracy (of FGSM Attack Data): {0}'.format(FGM_score[1]))
```

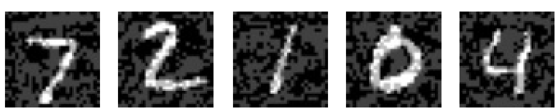
```
10000/10000 [==============================] - 1s 81us/step

Accuracy (of FGSM Attack Data): 0.0851999968290329
```

In [11]:

```
# Show ten original samples and their corresponding adversarial samples
print_samples('Original Samples:', data_test)
print()
print_samples('Corresponding Adversarial Samples (FGM):', FGM_data_test)
```

Original Samples:

Corresponding Adversarial Samples (FGM):



In [12]:

```python
# Implementing Detection via Autoencoders

def Autoencoder():
    model = Sequential()
    model.add(Dense(400, activation=None,       kernel_initializer="normal", input_dim=784))
    model.add(Dense(200, activation=None,       kernel_initializer='normal'))
    model.add(Dense(100, activation=None,       kernel_initializer='normal'))
    model.add(Dense(200, activation=None,       kernel_initializer='normal'))
    model.add(Dense(400, activation=None,       kernel_initializer='normal'))
    model.add(Dense(784, activation='sigmoid',  kernel_initializer='normal'))
    return model

# Create and train the autoencoder using the mean squared error loss and adam optimizer
autoencoder = Autoencoder()
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
history = autoencoder.fit(
                    data_train, data_train,
                    epochs=50, batch_size=256, shuffle=True
                )
```

```
Epoch 1/50
60000/60000 [==============================] - 2s 40us/step - loss: 0.0372
Epoch 2/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0118
Epoch 3/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0082
Epoch 4/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0066
Epoch 5/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0056
Epoch 6/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0050
Epoch 7/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0045
Epoch 8/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0041
Epoch 9/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0038
Epoch 10/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0035
Epoch 11/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0033
Epoch 12/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0031
Epoch 13/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0030
Epoch 14/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0028
Epoch 15/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0027
Epoch 16/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0026
Epoch 17/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0025
Epoch 18/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0025
Epoch 19/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0024
Epoch 20/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0023
Epoch 21/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0023
Epoch 22/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0022
Epoch 23/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0022
Epoch 24/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0022
Epoch 25/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0021
Epoch 26/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0021
Epoch 27/50
```

```
Epoch 27/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0021
Epoch 28/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0021
Epoch 29/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 30/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 31/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 32/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 33/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 34/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 35/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0020
Epoch 36/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0020
Epoch 37/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 38/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0019
Epoch 39/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 40/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 41/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 42/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0019
Epoch 43/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 44/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0019
Epoch 45/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 46/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 47/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 48/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.0019
Epoch 49/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
Epoch 50/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.0019
```

In [14]:

```python
# Using the autoencoder for detection and to determine a threshold

# Create adversarial examples using FGSM on training data
FGM_data_train = FGM.generate_np(data_train, **attack_param)

# Obtain reconstruction errors on training set and determine a threshold
FGM_data_reconstruction = autoencoder.predict(FGM_data_train)
error = keras.losses.mean_squared_error(FGM_data_train, FGM_data_reconstruction)

# Convert error tensor into NumPy array
err = error.eval(session=sess)

# Determine threshold (based on min in this case) and print it
threshold = np.amin(err)
print('Threshold (Reconstruction of Adversarial Samples): {0}'.format(threshold))

# =====================================================================================

# Calculate error of adversarial testing set
FGM_data_reconstruction = autoencoder.predict(FGM_data_test)
error = keras.losses.mean_squared_error(FGM_data_test, FGM_data_reconstruction)
err = error.eval(session=sess)

# Determine how many examples are above threshold and consider them adversarial
# (true positive count)
# Hint: Use a 'for' loop to compare each error value to the threshold
pos = np.zeros_like(err)
pos[np.where(err > threshold)] = 1

# Print number of true positive samples
true_pos = int(np.sum(pos))
print('\nNumber of True Positive Samples: {0}'.format(true_pos))
```

```
# ==========================================================================================

# Determine false positives on benign testing set
BENIGH_data_reconstruction = autoencoder.predict(data_test)
error = keras.losses.mean_squared_error(data_test, BENIGH_data_reconstruction)
err = error.eval(session=sess)

# Determine how many examples are above threshold and consider them adversarial
# (false positive count)
# Hint: Use a 'for' loop to compare each error value to the threshold
pos = np.zeros_like(err)
pos[np.where(err > threshold)] = 1

# Print number of false positive samples
false_pos = int(np.sum(pos))
print('Number of False Positive Samples: {0}'.format(false_pos))
```

```
Threshold (Reconstruction of Adversarial Samples): 0.02156517468392849

Number of True Positive Samples: 10000
Number of False Positive Samples: 0
```

Part 3: Projected Gradient Descent (PGD)

In [16]:

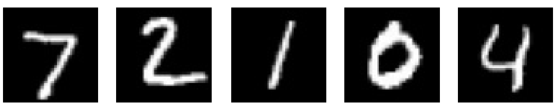
```
# Implementing the PGD attack

# PGD Instance on trained classifier from Part 1
PGD = MadryEtAl(wrap, sess=sess)

# Attack parameters
attack_param = {
            'eps':      0.25,
            'eps_iter': 0.01,
            'nb_iter':  20,
            'clip_min': 0.0,
            'clip_max': 1.0
        }

# Generate adversarial data
PGD_data_test = PGD.generate_np(data_test, **attack_param)

# Evaluate accuracy of perturbed data on target classifier
PGD_score = classifier.evaluate(PGD_data_test, labels_test)
print('Accuracy (of PGD Attack Data): {0}'.format(PGD_score[1]))
```

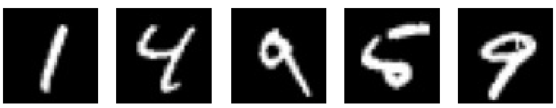
```
10000/10000 [==============================] - 1s 79us/step
Accuracy (of PGD Attack Data): 0.009800000116229057
```

In [17]:

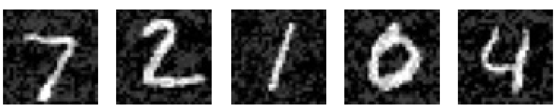
```
# Show ten original samples and their corresponding adversarial samples
print_samples('Original Samples:', data_test)
print()
print_samples('Corresponding Adversarial Samples (PGD):', PGD_data_test)
```

Original Samples:





Corresponding Adversarial Samples (PGD):



In [18]:

```
# Implementing the adversarial training defense
PGD_data_train   = PGD.generate_np(data_train, **attack_param)
all_data_train   = np.concatenate((data_train, PGD_data_train), axis=0)
all_data_test    = np.concatenate((data_test,  PGD_data_test),  axis=0)
all_labels_train = np.concatenate((labels_train, labels_train), axis=0)
all_labels_test  = np.concatenate((labels_test,  labels_test),  axis=0)

adv_trained_clf = Classifier()
adv_trained_clf.compile(optimizer='adam', loss='categorical_crossentropy', metrics =['accuracy'])

history = adv_trained_clf.fit(
                        all_data_train, all_labels_train,
                        validation_data = (all_data_test, all_labels_test),
                        epochs=50, batch_size=256, shuffle=True
                    )
```

```
Train on 120000 samples, validate on 20000 samples
Epoch 1/50
120000/120000 [==============================] - 5s 39us/step - loss: 0.2085 - accuracy: 0.9392 - val_loss
: 0.1269 - val_accuracy: 0.9677
Epoch 2/50
120000/120000 [==============================] - 4s 32us/step - loss: 0.0659 - accuracy: 0.9806 - val_loss
: 0.1194 - val_accuracy: 0.9706
Epoch 3/50
120000/120000 [==============================] - 4s 33us/step - loss: 0.0443 - accuracy: 0.9869 - val_loss
: 0.1086 - val_accuracy: 0.9742
Epoch 4/50
120000/120000 [==============================] - 4s 32us/step - loss: 0.0318 - accuracy: 0.9906 - val_loss
: 0.1132 - val_accuracy: 0.9747
Epoch 5/50
120000/120000 [==============================] - 4s 33us/step - loss: 0.0251 - accuracy: 0.9926 - val_loss
: 0.1163 - val_accuracy: 0.9748
Epoch 6/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0197 - accuracy: 0.9938 - val_loss
: 0.1169 - val_accuracy: 0.9755
Epoch 7/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0162 - accuracy: 0.9951 - val_loss
: 0.1129 - val_accuracy: 0.9763
Epoch 8/50
120000/120000 [==============================] - 4s 34us/step - loss: 0.0134 - accuracy: 0.9959 - val_loss
: 0.1233 - val_accuracy: 0.9754
Epoch 9/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0123 - accuracy: 0.9961 - val_loss
: 0.1224 - val_accuracy: 0.9768
Epoch 10/50
120000/120000 [==============================] - 4s 34us/step - loss: 0.0103 - accuracy: 0.9969 - val_loss
: 0.1294 - val_accuracy: 0.9748
Epoch 11/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0098 - accuracy: 0.9969 - val_loss
: 0.1295 - val_accuracy: 0.9781
Epoch 12/50
120000/120000 [==============================] - 4s 34us/step - loss: 0.0074 - accuracy: 0.9979 - val_loss
: 0.1325 - val_accuracy: 0.9764
Epoch 13/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0065 - accuracy: 0.9980 - val_loss
: 0.1347 - val_accuracy: 0.9760
Epoch 14/50
120000/120000 [==============================] - 4s 34us/step - loss: 0.0072 - accuracy: 0.9977 - val_loss
: 0.1414 - val_accuracy: 0.9765
Epoch 15/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0079 - accuracy: 0.9975 - val_loss
: 0.1369 - val_accuracy: 0.9776
Epoch 16/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0065 - accuracy: 0.9979 - val_loss
: 0.1402 - val_accuracy: 0.9765
Epoch 17/50
120000/120000 [==============================] - 4s 34us/step - loss: 0.0057 - accuracy: 0.9981 - val_loss
: 0.1396 - val_accuracy: 0.9772
Epoch 18/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0061 - accuracy: 0.9979 - val_loss
: 0.1442 - val_accuracy: 0.9779
Epoch 19/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0049 - accuracy: 0.9984 - val_loss
: 0.1432 - val_accuracy: 0.9769
```

```
Epoch 20/50
120000/120000 [==============================] - 4s 34us/step - loss: 0.0035 - accuracy: 0.9990 - val_loss
: 0.1390 - val_accuracy: 0.9783
Epoch 21/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0029 - accuracy: 0.9992 - val_loss
: 0.1578 - val_accuracy: 0.9779
Epoch 22/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0070 - accuracy: 0.9976 - val_loss
: 0.1612 - val_accuracy: 0.9761
Epoch 23/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0047 - accuracy: 0.9985 - val_loss
: 0.1436 - val_accuracy: 0.9785
Epoch 24/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0034 - accuracy: 0.9990 - val_loss
: 0.1397 - val_accuracy: 0.9779
Epoch 25/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0025 - accuracy: 0.9993 - val_loss
: 0.1387 - val_accuracy: 0.9790
Epoch 26/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0037 - accuracy: 0.9987 - val_loss
: 0.1562 - val_accuracy: 0.9768
Epoch 27/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0046 - accuracy: 0.9984 - val_loss
: 0.1576 - val_accuracy: 0.9775
Epoch 28/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0038 - accuracy: 0.9987 - val_loss
: 0.1518 - val_accuracy: 0.9778
Epoch 29/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0035 - accuracy: 0.9989 - val_loss
: 0.1627 - val_accuracy: 0.9773
Epoch 30/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0029 - accuracy: 0.9990 - val_loss
: 0.1464 - val_accuracy: 0.9789
Epoch 31/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0027 - accuracy: 0.9990 - val_loss
: 0.1413 - val_accuracy: 0.9789
Epoch 32/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0034 - accuracy: 0.9988 - val_loss
: 0.1499 - val_accuracy: 0.9788
Epoch 33/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0033 - accuracy: 0.9989 - val_loss
: 0.1443 - val_accuracy: 0.9796
Epoch 34/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0026 - accuracy: 0.9992 - val_loss
: 0.1478 - val_accuracy: 0.9789
Epoch 35/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0029 - accuracy: 0.9991 - val_loss
: 0.1524 - val_accuracy: 0.9790
Epoch 36/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0034 - accuracy: 0.9989 - val_loss
: 0.1578 - val_accuracy: 0.9780
Epoch 37/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0031 - accuracy: 0.9990 - val_loss
: 0.1450 - val_accuracy: 0.9801
Epoch 38/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0024 - accuracy: 0.9992 - val_loss
: 0.1588 - val_accuracy: 0.9791
Epoch 39/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0020 - accuracy: 0.9994 - val_loss
: 0.1518 - val_accuracy: 0.9807
Epoch 40/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0021 - accuracy: 0.9993 - val_loss
: 0.1512 - val_accuracy: 0.9797
Epoch 41/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0019 - accuracy: 0.9994 - val_loss
: 0.1619 - val_accuracy: 0.9789
Epoch 42/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0031 - accuracy: 0.9990 - val_loss
: 0.1610 - val_accuracy: 0.9787
Epoch 43/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0022 - accuracy: 0.9993 - val_loss
: 0.1592 - val_accuracy: 0.9794
Epoch 44/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0019 - accuracy: 0.9994 - val_loss
: 0.1603 - val_accuracy: 0.9791
Epoch 45/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0010 - accuracy: 0.9998 - val_loss
: 0.1630 - val_accuracy: 0.9797
Epoch 46/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0023 - accuracy: 0.9992 - val_loss
: 0.1657 - val_accuracy: 0.9773
Epoch 47/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0029 - accuracy: 0.9990 - val_loss
```

```
: 0.1630 - val_accuracy: 0.9791
Epoch 48/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0016 - accuracy: 0.9995 - val_loss
: 0.1570 - val_accuracy: 0.9804
Epoch 49/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0016 - accuracy: 0.9995 - val_loss
: 0.1534 - val_accuracy: 0.9796
Epoch 50/50
120000/120000 [==============================] - 4s 35us/step - loss: 0.0019 - accuracy: 0.9995 - val_loss
: 0.1529 - val_accuracy: 0.9791
```

In [20]:

```
# Using the defense to evaluate the accuracy of the perturbed data
PGD_score = adv_trained_clf.evaluate(PGD_data_test, labels_test)
print('\nAccuracy (of PGD Attack Data): {0}'.format(PGD_score[1]))
```

```
10000/10000 [==============================] - 1s 84us/step

Accuracy (of PGD Attack Data): 0.9797999858856201
```

Part 4: Carlini and Wagner Attack (CW)

In [19]:

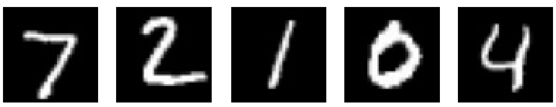
```
# Implementing the CW attack

# CW Instance on trained classifier from Part 1
CW = CarliniWagnerL2(wrap, sess=sess)

# Attack parameters
attack_param = {
            'binary_search_steps':  1,
            'y':                    None,
            'learning_rate':        1.25,
            'batch_size':           16,
            'initial_const':        10,
            'clip_min':             0.0,
            'clip_max':             1.0
        }

# Generate adversarial data
CW_data_test = CW.generate_np(data_test, **attack_param)

# Evaluate accuracy of perturbed data on target classifier
CW_score = classifier.evaluate(CW_data_test, labels_test)
print('\nAccuracy (of CW Attack Data): {0}'.format(CW_score[1]))
```

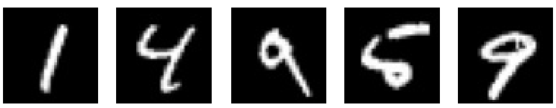
```
10000/10000 [==============================] - 1s 82us/step

Accuracy (of CW Attack Data): 0.013500000350177288
```

In [20]:

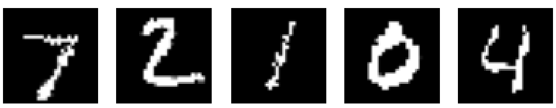
```
# Show ten original samples and their corresponding adversarial samples
print_samples('Original Samples:', data_test)
print()
print_samples('Corresponding Adversarial Samples (CW):', CW_data_test)
```

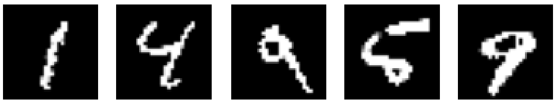
Original Samples:





Corresponding Adversarial Samples (CW):





In [21]:

```
# Implementing the dimensionality reduction (PCA) defense

# Calculate PCA projection
pca = PCA(100)
pca.fit(data_train)
PCA_train = pca.transform(data_train)
PCA_test  = pca.transform(data_test)

# Transform perturbed CW data using the subspace from the original training data
PCA_CW_test = pca.transform(CW_data_test)

# Create model for PCA
def pca_model():
    model = Sequential()

    model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
    model.add(BatchNormalization())

    model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
    model.add(BatchNormalization())

    model.add(Dense(10, activation='softmax'))

    return model

# Create model graph, compile it, and train it using pca_train lables_train
PCA_model = pca_model()
PCA_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics =['accuracy'])
history = PCA_model.fit(
                    PCA_train, labels_train,
                    validation_data=(PCA_test, labels_test),
                    epochs=50, batch_size=256, shuffle=True
                )
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_backend.py:431: The name tf.
is_variable_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 2s 41us/step - loss: 0.5302 - accuracy: 0.8390 - val_loss:
0.2245 - val_accuracy: 0.9361
Epoch 2/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.1773 - accuracy: 0.9483 - val_loss:
0.1425 - val_accuracy: 0.9577
Epoch 3/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.1199 - accuracy: 0.9658 - val_loss:
0.1123 - val_accuracy: 0.9655
Epoch 4/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0902 - accuracy: 0.9738 - val_loss:
0.0972 - val_accuracy: 0.9700
Epoch 5/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0712 - accuracy: 0.9794 - val_loss:
0.0896 - val_accuracy: 0.9734
Epoch 6/50
60000/60000 [==============================] - 2s 30us/step - loss: 0.0581 - accuracy: 0.9835 - val_loss:
0.0818 - val_accuracy: 0.9752
Epoch 7/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0472 - accuracy: 0.9868 - val_loss:
0.0806 - val_accuracy: 0.9751
Epoch 8/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0394 - accuracy: 0.9892 - val_loss:
0.0799 - val_accuracy: 0.9752
Epoch 9/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0326 - accuracy: 0.9910 - val_loss:
0.0772 - val_accuracy: 0.9769
Epoch 10/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0273 - accuracy: 0.9926 - val_loss:
0.0802 - val_accuracy: 0.9762
Epoch 11/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0230 - accuracy: 0.9942 - val_loss:
0.0795 - val_accuracy: 0.9770
Epoch 12/50
```

```
60000/60000 [==============================] - 2s 31us/step - loss: 0.0194 - accuracy: 0.9952 - val_loss:
0.0796 - val_accuracy: 0.9772
Epoch 13/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0163 - accuracy: 0.9964 - val_loss:
0.0825 - val_accuracy: 0.9769
Epoch 14/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0135 - accuracy: 0.9969 - val_loss:
0.0810 - val_accuracy: 0.9781
Epoch 15/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0120 - accuracy: 0.9974 - val_loss:
0.0851 - val_accuracy: 0.9780
Epoch 16/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0099 - accuracy: 0.9979 - val_loss:
0.0875 - val_accuracy: 0.9765
Epoch 17/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0088 - accuracy: 0.9983 - val_loss:
0.0901 - val_accuracy: 0.9781
Epoch 18/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0075 - accuracy: 0.9984 - val_loss:
0.0947 - val_accuracy: 0.9760
Epoch 19/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0072 - accuracy: 0.9985 - val_loss:
0.0901 - val_accuracy: 0.9782
Epoch 20/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0063 - accuracy: 0.9987 - val_loss:
0.0942 - val_accuracy: 0.9763
Epoch 21/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0059 - accuracy: 0.9987 - val_loss:
0.0946 - val_accuracy: 0.9775
Epoch 22/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0054 - accuracy: 0.9990 - val_loss:
0.0926 - val_accuracy: 0.9790
Epoch 23/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0052 - accuracy: 0.9989 - val_loss:
0.0991 - val_accuracy: 0.9776
Epoch 24/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0060 - accuracy: 0.9985 - val_loss:
0.1013 - val_accuracy: 0.9770
Epoch 25/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0053 - accuracy: 0.9988 - val_loss:
0.0979 - val_accuracy: 0.9773
Epoch 26/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0043 - accuracy: 0.9990 - val_loss:
0.1051 - val_accuracy: 0.9758
Epoch 27/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0038 - accuracy: 0.9993 - val_loss:
0.1070 - val_accuracy: 0.9763
Epoch 28/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0053 - accuracy: 0.9985 - val_loss:
0.1039 - val_accuracy: 0.9778
Epoch 29/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0047 - accuracy: 0.9987 - val_loss:
0.1043 - val_accuracy: 0.9774
Epoch 30/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0045 - accuracy: 0.9987 - val_loss:
0.1126 - val_accuracy: 0.9753
Epoch 31/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0038 - accuracy: 0.9990 - val_loss:
0.1014 - val_accuracy: 0.9778
Epoch 32/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0030 - accuracy: 0.9994 - val_loss:
0.1067 - val_accuracy: 0.9769
Epoch 33/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0026 - accuracy: 0.9995 - val_loss:
0.1142 - val_accuracy: 0.9764
Epoch 34/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0025 - accuracy: 0.9995 - val_loss:
0.1156 - val_accuracy: 0.9761
Epoch 35/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0034 - accuracy: 0.9991 - val_loss:
0.1159 - val_accuracy: 0.9766
Epoch 36/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0041 - accuracy: 0.9988 - val_loss:
0.1131 - val_accuracy: 0.9777
Epoch 37/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0041 - accuracy: 0.9988 - val_loss:
0.1133 - val_accuracy: 0.9782
Epoch 38/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0025 - accuracy: 0.9994 - val_loss:
0.1129 - val_accuracy: 0.9774
Epoch 39/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0027 - accuracy: 0.9994 - val_loss:
0.1147 - val_accuracy: 0.9781
Epoch 40/50
```

```
Epoch 40/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0023 - accuracy: 0.9994 - val_loss:
0.1195 - val_accuracy: 0.9762
Epoch 41/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0021 - accuracy: 0.9996 - val_loss:
0.1177 - val_accuracy: 0.9766
Epoch 42/50
60000/60000 [==============================] - 2s 31us/step - loss: 0.0028 - accuracy: 0.9992 - val_loss:
0.1198 - val_accuracy: 0.9759
Epoch 43/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0048 - accuracy: 0.9984 - val_loss:
0.1277 - val_accuracy: 0.9751
Epoch 44/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0057 - accuracy: 0.9980 - val_loss:
0.1313 - val_accuracy: 0.9762
Epoch 45/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0039 - accuracy: 0.9987 - val_loss:
0.1266 - val_accuracy: 0.9760
Epoch 46/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0020 - accuracy: 0.9995 - val_loss:
0.1189 - val_accuracy: 0.9782
Epoch 47/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0016 - accuracy: 0.9997 - val_loss:
0.1230 - val_accuracy: 0.9781
Epoch 48/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0026 - accuracy: 0.9991 - val_loss:
0.1389 - val_accuracy: 0.9763
Epoch 49/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0020 - accuracy: 0.9995 - val_loss:
0.1327 - val_accuracy: 0.9777
Epoch 50/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0013 - accuracy: 0.9997 - val_loss:
0.1288 - val_accuracy: 0.9775
```

In [22]:

```
# Using the defense (and comparing to baseline accuracy)
CW_score = PCA_model.evaluate(PCA_CW_test, labels_test)
print('\nAccuracy (of CW Attack Data): {0}'.format(CW_score[1]))
```

```
10000/10000 [==============================] - 1s 86us/step

Accuracy (of CW Attack Data): 0.8572999835014343
```

Part 5: DeepFool (DF)

In [13]:

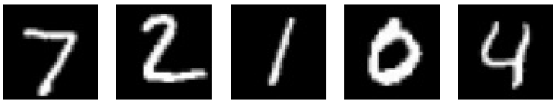
```
# Implementing the DeepFool attack

# DeepFool Instance on trained classifier from Part 1
DF = DeepFool(wrap, sess=sess)

# Attack parameters
attack_param = {
            'nb_candidate': 10,
            'max_iter':     50,
            'clip_min':     0.0,
            'clip_max':     1.0
        }

# Generate adversarial data
DF_data_test = DF.generate_np(data_test, **attack_param)

# Evaluate accuracy of perturbed data on target classifier
DF_score = classifier.evaluate(DF_data_test, labels_test)
print('\nAccuracy (of DeepFool Attack Data): {0}'.format(DF_score[1]))
```

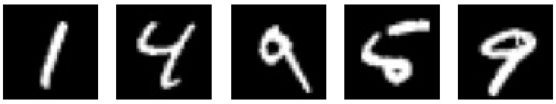
```
10000/10000 [==============================] - 1s 81us/step

Accuracy (of DeepFool Attack Data): 0.014100000262260437
```

In [14]:

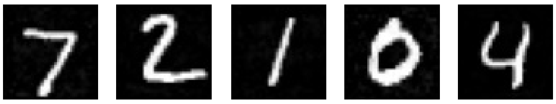
```
# Show ten original samples and their corresponding adversarial samples
print_samples('Original Samples:', data_test)
print()
print_samples('Corresponding Adversarial Samples (DeepFool):', DF_data_test)
```

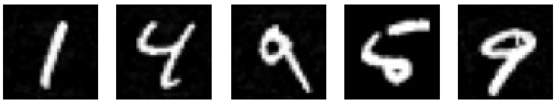
```
Original Samples:
```





```
Corresponding Adversarial Samples (DeepFool):
```





In [15]:

```
# Implementing the Denoising Autoencoder Defense

def Autoencoder():
    model = Sequential()
    model.add(Dense(400, activation=None,       kernel_initializer="normal", input_dim=784))
    model.add(Dense(200, activation=None,       kernel_initializer='normal'))
    model.add(Dense(100, activation=None,       kernel_initializer='normal'))
    model.add(Dense(200, activation=None,       kernel_initializer='normal'))
    model.add(Dense(400, activation=None,       kernel_initializer='normal'))
    model.add(Dense(784, activation='sigmoid',  kernel_initializer='normal'))
    return model

# Create training data for DAE
DF_data_train     = DF.generate_np(data_train, **attack_param)
data_total_train  = np.concatenate([DF_data_train, data_train])
labal_total_train = np.concatenate([data_train, data_train])

# Create and train DAE graph
DAE = Autoencoder()
DAE.compile(optimizer='adam', loss='mean_squared_error')
history = DAE.fit(
            data_total_train, labal_total_train,
            epochs=50, batch_size=256, shuffle=True
        )
```

```
Epoch 1/50
120000/120000 [==============================] - 5s 39us/step - loss: 0.0257
Epoch 2/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0077
Epoch 3/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0055
Epoch 4/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0044
Epoch 5/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0037
Epoch 6/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0033
Epoch 7/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0030
Epoch 8/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0028
Epoch 9/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0026
Epoch 10/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0025
Epoch 11/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0023
Epoch 12/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0023
Epoch 13/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0022
Epoch 14/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0021
```

```
Epoch 15/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0021
Epoch 16/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0021
Epoch 17/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 18/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 19/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 20/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 21/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 22/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 23/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0020
Epoch 24/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 25/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 26/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 27/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 28/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 29/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 30/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 31/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 32/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 33/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 34/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 35/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 36/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 37/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 38/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 39/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 40/50
120000/120000 [==============================] - 5s 39us/step - loss: 0.0019
Epoch 41/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 42/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 43/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 44/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 45/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 46/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 47/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 48/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 49/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
Epoch 50/50
120000/120000 [==============================] - 5s 38us/step - loss: 0.0019
```

In [16]:

```
# Using the defense

# Use DAE to to remove adversarial perturbation
DAE_data_reconstruction = DAE.predict(DF_data_test)

# Evaluate accuracy of FGM samples after denoising
DF_DAE_score = classifier.evaluate(DAE_data_reconstruction, labels_test)
```

```
print('\nAccuracy (of DeepFool Attack Data): {0}'.format(DF_DAE_score[1]))
```

```
10000/10000 [==============================] - 1s 87us/step

Accuracy (of DeepFool Attack Data): 0.9782999753952026
```

In [17]:

```
# Show ten samples of adversarial samples after denoising
print_samples('Corresponding Adversarial Samples (DeepFool):', DAE_data_reconstruction)
```

```
Corresponding Adversarial Samples (DeepFool):
```