Assignment 6
AutoEncoder

Date : 10th April,2019

Nanavati Tilak D.
(201811001)

AutoEncode :

An AutoEncoder is a type of artificial neural network used to learn efficient data codings in unsupervised manner.The aim of an AutoEncoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to avoid signal noise.Along with the reduction side,a reconstructing side is learnt,where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input,hence its name is AutoEncoder.

1. Code

```
from IPython.display import Image, SVG
import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D,
Flatten, Reshape
from keras import regularizers
```

Output :

```
Using TensorFlow backend.
```

Code :

```
(x_train, _), (x_test, _) = mnist.load_data()

# Scales the training and test data to range between 0 and 1.
max_value = float(x_train.max())
x_train = x_train.astype('float32') / max_value
x_test = x_test.astype('float32') / max_value
```

```
# input dimension = 784
input_dim = x_train.shape[1]
encoding_dim = 32

compression_factor = float(input_dim) / encoding_dim
print("Compression factor: %s" % compression_factor)

autoencoder = Sequential()
autoencoder.add(
    Dense(encoding_dim, input_shape=(input_dim,),
activation='relu')
)
autoencoder.add(
    Dense(input_dim, activation='sigmoid')
)

autoencoder.summary()
```

Output:

```
Compression factor: 24.5
WARNING:tensorflow:From
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_libra
ry.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated
and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
```

```
Non-trainable params: 0
```

Observation :
The autoencoder basically reduces the dimensions of the input image in the bottleneck layer. And that we are showing inside the output.We get those reduced dimensions on the bottleneck layer.

Code :

```
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

Output :

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 5s 79us/step -
loss: 0.2736 - val_loss: 0.1882
Epoch 2/50
60000/60000 [==============================] - 4s 74us/step -
loss: 0.1701 - val_loss: 0.1531
Epoch 3/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.1435 - val_loss: 0.1329
Epoch 4/50
60000/60000 [==============================] - 4s 74us/step -
loss: 0.1276 - val_loss: 0.1204
Epoch 5/50
60000/60000 [==============================] - 5s 80us/step -
loss: 0.1175 - val_loss: 0.1123
```

```
Epoch 6/50
60000/60000 [==============================] - 4s 74us/step -
loss: 0.1105 - val_loss: 0.1062
Epoch 7/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.1054 - val_loss: 0.1021
Epoch 8/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.1018 - val_loss: 0.0990
Epoch 9/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0992 - val_loss: 0.0970
Epoch 10/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0974 - val_loss: 0.0954
Epoch 11/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0963 - val_loss: 0.0946
Epoch 12/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0955 - val_loss: 0.0939
Epoch 13/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0949 - val_loss: 0.0934
Epoch 14/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0946 - val_loss: 0.0931
Epoch 15/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0943 - val_loss: 0.0929
Epoch 16/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0941 - val_loss: 0.0927
Epoch 17/50
60000/60000 [==============================] - 4s 73us/step -
```

```
loss: 0.0940 - val_loss: 0.0926
Epoch 18/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0938 - val_loss: 0.0925
Epoch 19/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0937 - val_loss: 0.0924
Epoch 20/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0936 - val_loss: 0.0923
Epoch 21/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0935 - val_loss: 0.0922
Epoch 22/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0934 - val_loss: 0.0922
Epoch 23/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0934 - val_loss: 0.0921
Epoch 24/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0933 - val_loss: 0.0921
Epoch 25/50
60000/60000 [==============================] - 4s 74us/step -
loss: 0.0932 - val_loss: 0.0920
Epoch 26/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0932 - val_loss: 0.0919
Epoch 27/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0931 - val_loss: 0.0919
Epoch 28/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0931 - val_loss: 0.0918
Epoch 29/50
```

```
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0931 - val_loss: 0.0919
Epoch 30/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0930 - val_loss: 0.0918
Epoch 31/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0930 - val_loss: 0.0918
Epoch 32/50
60000/60000 [==============================] - 4s 75us/step -
loss: 0.0930 - val_loss: 0.0918
Epoch 33/50
60000/60000 [==============================] - 5s 78us/step -
loss: 0.0929 - val_loss: 0.0917
Epoch 34/50
60000/60000 [==============================] - 5s 77us/step -
loss: 0.0929 - val_loss: 0.0917
Epoch 35/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0929 - val_loss: 0.0917
Epoch 36/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0928 - val_loss: 0.0917
Epoch 37/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0928 - val_loss: 0.0917
Epoch 38/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0928 - val_loss: 0.0917
Epoch 39/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0928 - val_loss: 0.0917
Epoch 40/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0928 - val_loss: 0.0916
```

```
Epoch 41/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0928 - val_loss: 0.0916
Epoch 42/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0928 - val_loss: 0.0916
Epoch 43/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0927 - val_loss: 0.0917
Epoch 44/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0927 - val_loss: 0.0916
Epoch 45/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0927 - val_loss: 0.0915
Epoch 46/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0927 - val_loss: 0.0916
Epoch 47/50
60000/60000 [==============================] - 4s 71us/step -
loss: 0.0927 - val_loss: 0.0916
Epoch 48/50
60000/60000 [==============================] - 4s 71us/step -
loss: 0.0927 - val_loss: 0.0915
Epoch 49/50
60000/60000 [==============================] - 4s 72us/step -
loss: 0.0926 - val_loss: 0.0915
Epoch 50/50
60000/60000 [==============================] - 4s 73us/step -
loss: 0.0926 - val_loss: 0.0916
```

Code :

```
num_images = 10
np.random.seed(42)
random_test_images = np.random.randint(x_test.shape[0],
size=num_images)

encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

plt.figure(figsize=(18, 4))

for i, image_idx in enumerate(random_test_images):
    # plot original image
    ax = plt.subplot(3, num_images, i + 1)
    plt.imshow(x_test[image_idx].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # plot encoded image
    ax = plt.subplot(3, num_images, num_images + i + 1)
    plt.imshow(encoded_imgs[image_idx].reshape(8, 4))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # plot reconstructed image
    ax = plt.subplot(3, num_images, 2*num_images + i + 1)
    plt.imshow(decoded_imgs[image_idx].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Output :

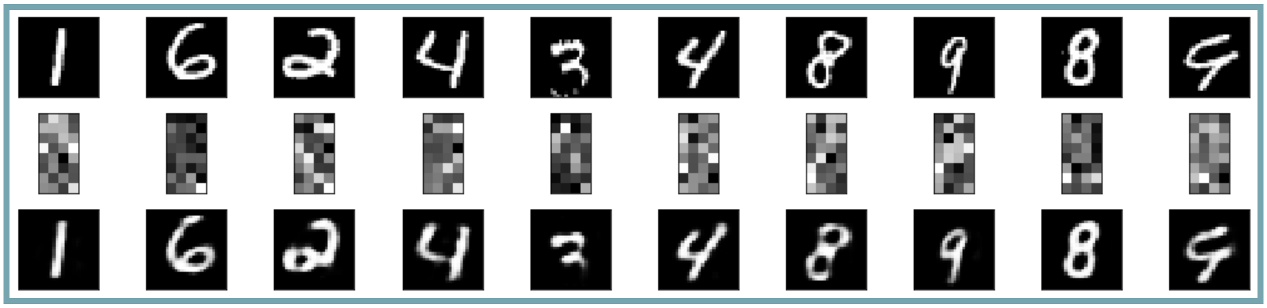

Fig. (a) First row represents the random input images that we are giving to the encoder. (b) Second row shows the intermediate (bottleneck layer) representation of the input images. (c). Third row shows the final output that we are obtaining at the output layer of the AutoEncoder.

Code: Z values of First Test 5 Images

```
for i, image_idx in enumerate(random_test_images):
  print(encoded_imgs[image_idx])
```

Output :

```
[ 4.8686795   9.724134    7.829077    3.5875263   8.016051
7.9123898
  8.047392    4.0871396   4.7480836   5.1603847   8.472448
7.1361957
 10.788681    3.7810907   5.847143   11.122015    4.0106854
5.976287
  0.49805975  7.549961    3.5597792   6.369248    2.203638
3.958158
  6.5987616   3.2125907   5.8932877   3.861183    6.3623695
6.0588965
  3.0127392   9.873814  ]
[ 5.550332   4.58055    3.7144377  7.052125  15.066132
9.778974
 11.272017   2.5515172 11.154323   9.498934   5.039446
9.942481
  9.808603  10.311965   6.6647577  3.7137     7.8571525
```

```
13.349087
 12.876808  13.163999   5.9904566 11.954491   4.977845
9.633353
 15.32916    9.455136  19.448708   3.3635848  8.894091
14.333416
 15.005651  30.22148  ]
[11.757883   9.031895  10.019552   3.2484937  4.415995
3.4182343
 17.097261  18.848288  12.024426   7.5011873  7.102546   9.32479
 16.818256   6.984645   8.987232   2.6030836 12.093388
18.346716
  5.66778   11.992239  10.261052  14.568375   6.448857   6.27266
  6.175054  10.852571  16.431114  11.286544  10.225961
11.768716
  6.8281975 19.453487 ]
[11.685593   4.422655   3.7250922  5.078603   3.3191905
12.071592
 11.838171  18.52755    7.222043   5.913079   6.5832734
7.9630923
  5.053716   5.1109343  6.7120266  0.          6.3887167
5.7052264
  6.418667  14.005888   7.211956   9.332304  15.944404
8.773998
  8.032375   8.930192   6.7909584  5.748738   8.586849
10.065658
  5.8243365 16.634575 ]
[ 2.8536131  5.728632   4.374846   6.5096664  4.74376
16.935616
  3.1226685  5.2766967 13.327088   8.997061   7.7389364
9.543337
 12.008819   6.1560454  8.087344   6.609338   6.497387
7.1451373
 12.790136   6.202892   5.9142118  4.7408032 11.315119
8.494477
  3.400912   5.731931   6.381925   5.172523   4.560462
```

```
4.205854
  2.7905931 12.477829 ]
```