Encryption Algorithm using Autoencoders

#encryption.py

#importing python libraries

import keras

from keras.layers import Input, Dense

from keras.models import Model

from keras.callbacks import TensorBoard

import numpy as np

from tensorflow import set\_random\_seed

import os

import pyAesCrypt

#to set random seed for np random generator and tensorflow backend

def seedy(s):

np.random.seed(s)

set\_random\_seed(s)

#defines the overall template

class AutoEncoder:

#we set the encoding dimension and we use randomly generated arrays of random numbers to train our autoencoder

def \_\_init\_\_(self, encoding\_dim=3):

self.encoding\_dim = encoding\_dim

r = lambda: np.random.randint(1, 3)

self.x = np.array([[r(),r(),r(),r(),r()] for \_ in range(2000)])

print(self.x)

#the encoder part of the network ,the encoded layers are dense layers ,dimensions reduce from 5 to 3 to 3

def \_encoder(self):

inputs = Input(shape=(self.x[0].shape))

encoded = Dense(self.encoding\_dim, activation='relu')(inputs)

encoded = Dense(self.encoding\_dim, activation='relu')(encoded)

model = Model(inputs, encoded)

self.encoder = model

return model

#the decoder part of the network , the encoded layers are dense layers ,dimensions reduce from 3 to 5 to 5

def \_decoder(self):

inputs = Input(shape=(self.encoding\_dim,))

decoded = Dense(5)(inputs)

decoded = Dense(5)(decoded)

model = Model(inputs, decoded)

self.decoder = model

return model

#here we concatenate our encoder and decoder models , the output of encoder will be the input of the decoder

def encoder\_decoder(self):

ec = self.\_encoder()

dc = self.\_decoder()

inputs = Input(shape=self.x[0].shape)

ec\_out = ec(inputs)

dc\_out = dc(ec\_out)

model = Model(inputs, dc\_out)

self.model = model

return model

#here we compile our model using stochastic gradient descent , we are comparing the reconstructed output with the input and try to minimize the difference .

def fit(self, batch\_size=10, epochs=300):

self.model.compile(optimizer='sgd', loss='mse')

log\_dir = './log/'

tbCallBack = keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=0, write\_graph=True, write\_images=True)

self.model.fit(self.x, self.x,

epochs=epochs,

batch\_size=batch\_size,

callbacks=[tbCallBack])

#we save our weights of the encoder and decoder for reconstruction , these weights files will be encrypted using aes and the decoder weights will be sent to the receiever while the encoder weights will be used by the sender

def save(self):

if not os.path.exists(r'./weights'):

os.mkdir(r'./weights')

else:

self.encoder.save(r'./weights/encoder\_weights.h5')

self.decoder.save(r'./weights/decoder\_weights.h5')

self.model.save(r'./weights/ae\_weights.h5')

bufferSize = 64 \* 1024

password = "hello" pyAesCrypt.encryptFile("./weights/encoder\_weights.h5","./weights/encoder\_weights.txt.aes", password, bufferSize)

pyAesCrypt.encryptFile("./weights/decoder\_weights.h5", "./weights/decoder\_weights.txt.aes", password, bufferSize)

#main function

if \_\_name\_\_ == '\_\_main\_\_':

seedy(2)

ae = AutoEncoder(encoding\_dim=3)

ae.encoder\_decoder()

ae.fit(batch\_size=50, epochs=300)

ae.save()

#sender.py

#run this file on sender end

#import python libraries

from keras.models import load\_model

import numpy as np

import pyAesCrypt

bufferSize = 64 \* 1024

password = "hello"

# we first decrypt our encrypted weights of the encoder , load the model from the weights.

pyAesCrypt.decryptFile("./weights/encoder\_weights.txt.aes","./weights/encoder\_weights.h5", password, bufferSize)

encoder = load\_model(r'./weights/encoder\_weights.h5')

inputs = np.array([[1,2,2,3,1]])

#we now reduce our 5 numbers to 3 and these numbers are saved in .npy file to be sent to the receiver

x = encoder.predict(inputs)

np.save('encodedValues',x)

print('Input: {}'.format(inputs))

print('Encoded: {}'.format(x))

#reciever.py

#run this file on reciever end

#import python libraries

from keras.models import load\_model

import numpy as np

import pyAesCrypt

# we first decrypt our encrypted weights of the decoder , load the model from the weights.

bufferSize = 64 \* 1024

password = "hello"

pyAesCrypt.decryptFile("./weights/decoder\_weights.txt.aes","./weights/decoder\_weights.h5", password, bufferSize)

decoder = load\_model(r'./weights/decoder\_weights.h5')

#here we generate our original values again from the 3 values that were sent to us in .npy format

inputs = np.array(np.load('encodedValues.npy'))

y = decoder.predict(inputs)

print('Decoded: {}'.format(y))

print(np.round(y))

```

OUTPUT

Ruthss -MacBook-Pro:Desktop ravismbp$ **python3 encryption.py**

Using TensorFlow backend.

[[1 2 2 1 1]

[2 1 2 1 2]

[1 2 2 2 2]

...

[2 2 1 2 1]

[2 2 1 2 2]

[2 2 2 2 2]]

Epoch 1/300

2000/2000 [==============================] - 0s 34us/step - loss: 1.5250

Epoch 2/300

2000/2000 [==============================] - 0s 28us/step - loss: 0.2087

Epoch 3/300

2000/2000 [==============================] - 0s 28us/step - loss: 0.1989

Epoch 4/300

2000/2000 [==============================] - 0s 27us/step - loss: 0.1946

Epoch 5/300

2000/2000 [==============================] - 0s 28us/step - loss: 0.1904

Epoch 6/300

2000/2000 [==============================] - 0s 28us/step - loss: 0.1852

Epoch 7/300

2000/2000 [==============================] - 0s 27us/step - loss: 0.1809

Epoch 8/300

2000/2000 [==============================] - 0s 29us/step - loss: 0.1770

Epoch 9/300

2000/2000 [==============================] - 0s 28us/step - loss: 0.1729

Epoch 10/300

2000/2000 [==============================] - 0s 28us/step - loss: 0.1696

Epoch 11/300

2000/2000 [==============================] - 0s 29us/step - loss: 0.1668

Epoch 12/300

.

.

.

.

.

Epoch 297/300

2000/2000 [==============================] - 0s 27us/step - loss: 0.1516

Epoch 298/300

2000/2000 [==============================] - 0s 33us/step - loss: 0.1517

Epoch 299/300

2000/2000 [==============================] - 0s 32us/step - loss: 0.1517

Epoch 300/300

2000/2000 [==============================] - 0s 30us/step - loss: 0.1516

In encryption.py we create our autoencoder model , here 300 epochs of batch size 10 are run to improve the accuracy. Also the files to store the weights of the encoder and decoder are created at this point and are encrypted using aes in .aes files.

Ruthss-MacBook-Pro:Desktop ravismbp$ python3 **sender.py**

Using TensorFlow backend.

/usr/local/lib/python3.7/site-packages/keras/engine/saving.py:292: UserWarning: No training configuration found in save file: the model was \*not\* compiled. Compile it manually.

warnings.warn('No training configuration found in save file: '

**Input: [[1 2 2 3 1]]**

**Encoded: [[1.284911 2.435414 5.4132247]]**

In Sender.py the weights from the encoder model are used to compress the input .Here we can see how the 5 dimensional input is reduced to 3 and they are not similar to the inputs hence making it impossible for the hacker to decrypt the actual values without the weights of the decoder.

Ruthss -MacBook-Pro:Desktop ravismbp$ python3 **reciever.py**

Using TensorFlow backend.

/usr/local/lib/python3.7/site-packages/keras/engine/saving.py:292: UserWarning: No training configuration found in save file: the model was \*not\* compiled. Compile it manually.

warnings.warn('No training configuration found in save file: '

**Decoded: [[1.0325072 1.8651003 2.3601866 2.7759185 0.92045224]]**

**[[1. 2. 2. 3. 1.]]**

In reciever.py the weights from the decoder model are used to generate the original values .Here we cann see how the 3 dimensional compressed values are constructed back. The values are very close to the original input . On rounding the values we do get the original values back .