CS5590/490 - Python and Deep Learning Programming
In Class Programming Report – DL Lesson 6
Class ID 24 - Anurag Thantharate
Date Submitted: 05/03/2020

---

1. Add one more hidden layer to autoencoder

```python
# Import libraries
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model
from matplotlib import pyplot as plt

# Size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# Input placeholder
input_img = Input(shape=(784,))
# Additional hidden layers added to the model
hidden_1 = Dense(encoding_dim, activation='relu')(input_img)
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(hidden_1)
hidden_2 = Dense(encoding_dim, activation='relu')(encoded)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(hidden_2)

# Mapping input to its reconstruction
autoencoder = Model(input_img, decoded)
# Model mapping an input to its encoded representation
encoder = Model(input_img, encoded)
# Creating a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# Retrieving the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Creating the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
# Compiling the model defined
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['acc'])
```

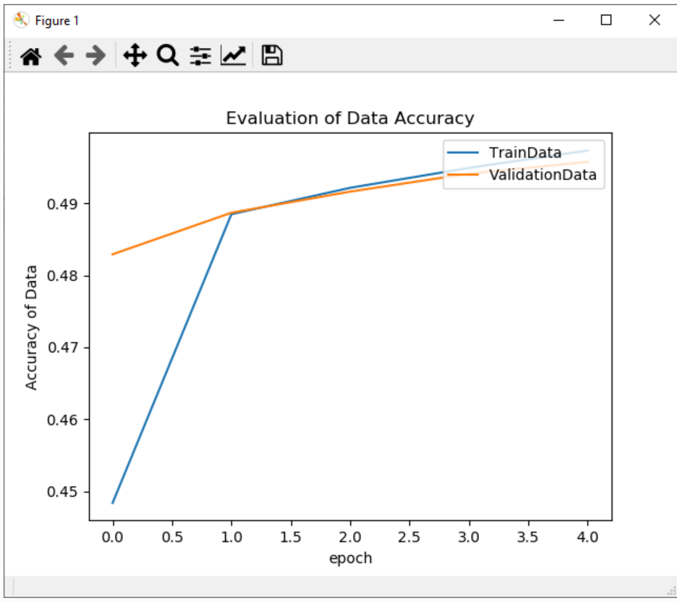
2. visualize the input and reconstructed representation of the autoencoder using Matplotlib

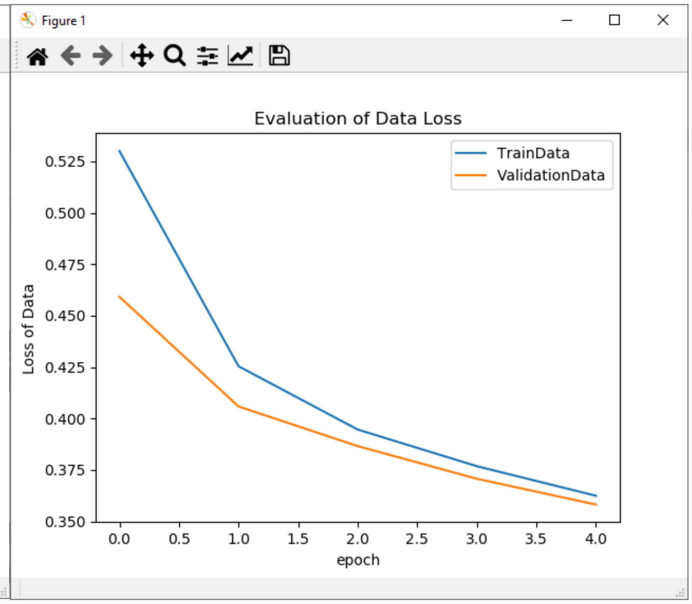
```python
# Import libraries
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model

# Size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# Input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# Mapping input to its reconstruction
autoencoder = Model(input_img, decoded)

# Model mapping an input to its encoded representation
encoder = Model(input_img, encoded)

# Decoder model representation
# Creating a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# Retrieving the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Creating the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

# Compiling the model defined
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['acc'])
```

```python
# Loading input data set
from keras.datasets import fashion_mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Fitting the model defined on training data set
history = autoencoder.fit(x_train, x_train, epochs=5, batch_size=256, shuffle=True, validation_data=(x_test, x_test))

# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

# Visualization of reconstructed inputs and decoded representations
# Library imported to be used
from matplotlib import pyplot as plt
# Identify the number of digits to be displayed
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original input
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```python
    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

```
53760/60000 [========================>....] - ETA: 0s - loss: 0.3631 - acc: 0.4971
54784/60000 [=========================>...] - ETA: 0s - loss: 0.3630 - acc: 0.4973
55808/60000 [=========================>...] - ETA: 0s - loss: 0.3629 - acc: 0.4972
56832/60000 [==========================>..] - ETA: 0s - loss: 0.3627 - acc: 0.4973
57856/60000 [==========================>..] - ETA: 0s - loss: 0.3627 - acc: 0.4973
58880/60000 [===========================>.] - ETA: 0s - loss: 0.3626 - acc: 0.4973
59904/60000 [===========================>.] - ETA: 0s - loss: 0.3624 - acc: 0.4973
60000/60000 [==============================] - 4s 62us/step - loss: 0.3624 - acc: 0.4973 - val_loss: 0.3581 - val_acc: 0.4957
```



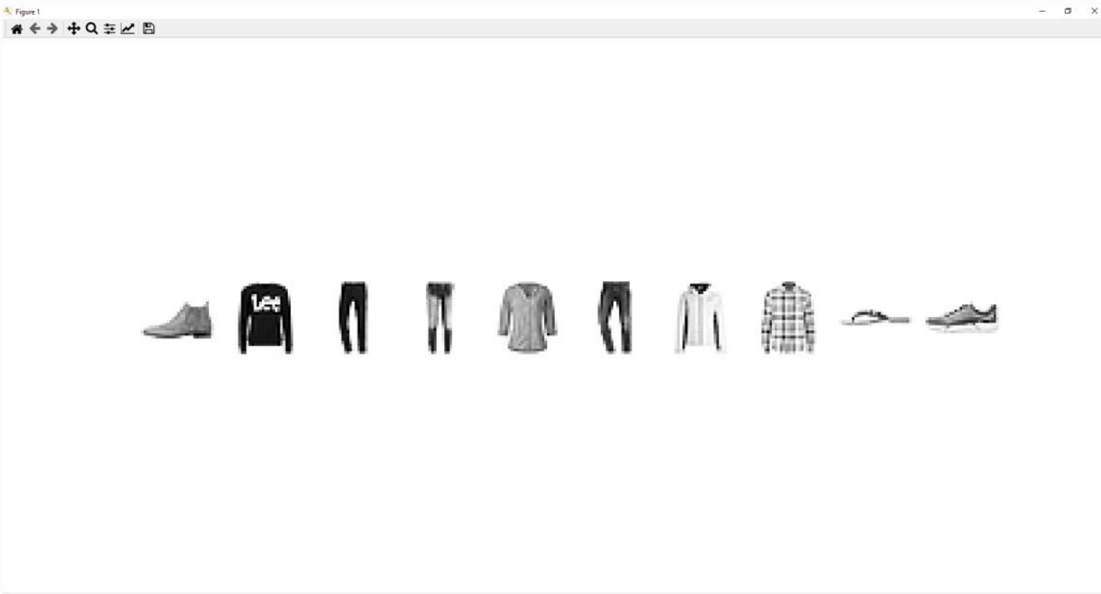


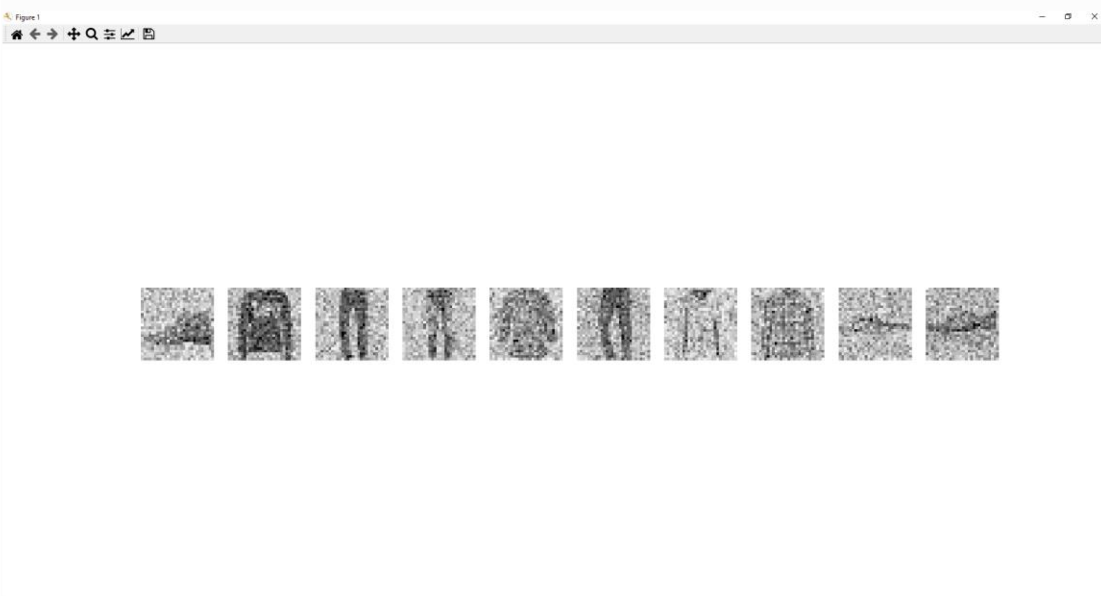
3. visualize the input, noisy input and reconstructed representation (denoised output) of the Denosing_Autoencoder using Matplotlib

```
# Import Libraries
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model
from matplotlib import pyplot as plt, rcParams

# Size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# Input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# Mapping input to its reconstruction
autoencoder = Model(input_img, decoded)
# Model mapping an input to its encoded representation
encoder = Model(input_img, encoded)
# Creating a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# Retrieving the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Creating the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
# Compiling the model defined
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['acc'])
```

```
# Loading input data set
from keras.datasets import fashion_mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Sparsity constraint for activity regularization
# Adding noise to test data set
n_rows = x_test.shape[0]
n_cols = x_test.shape[1]
mean = 0.5
stddev = 0.3
noise = np.random.normal(mean, stddev, (n_rows, n_cols))
# creating the noisy test data by adding X_test with noise
x_test_noisy = x_test + noise

# Fitting the model defined on training data set
history = autoencoder.fit(x_train, x_train, epochs=5, batch_size=256, shuffle=True, validation_data=(x_test_noisy, x_test_noisy))

# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test_noisy)
decoded_imgs = decoder.predict(encoded_imgs)
```

```python
# Evaluation of the results of the model obtained using the test data set
[test_loss, test_acc] = autoencoder.evaluate(x_test_noisy, x_test_noisy)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```





4. plot loss and accuracy using the history object