Double-click (or enter) to edit

NN&DeepLearning_Lesson: Autoencoders

Assignment-4

NAME: Saharsha Muddagauni Student Id: 700742679

Github Link: https://github.com/sxm26790/Assignment4_neural/tree/main

1. Add one more hidden layer to autoencoder

```
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt


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

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# Adding of one more layer to the encoder
encoded_1 = Dense(4, activation='relu')(encoded)
# Adding of one more layer to the decoder
decoded_1 = Dense(encoding_dim, activation='sigmoid')(encoded_1)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy' ,metrics=['accuracy'])
from keras.datasets import mnist, fashion_mnist
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
#Visualize the data before reconstructed.
plt.imshow(x_test[15].reshape(28,28))
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:])))

history=autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

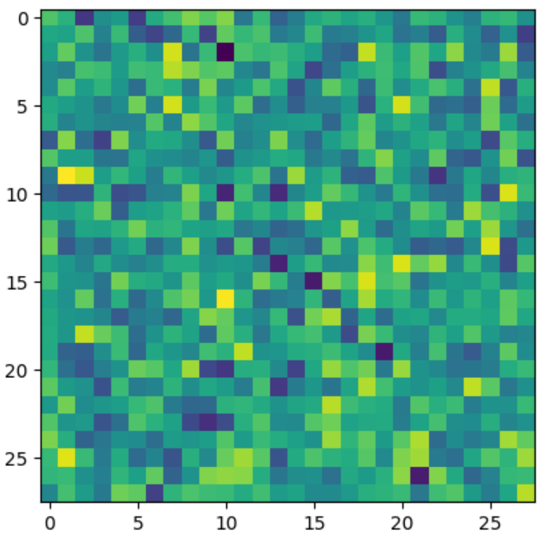
```
Epoch 1/5
235/235 [==============================] - 9s 32ms/step - loss: 0.6943 - accuracy: 0.0010 - val_loss: 0.6942 - val_accuracy:
Epoch 2/5
235/235 [==============================] - 4s 18ms/step - loss: 0.6941 - accuracy: 0.0011 - val_loss: 0.6940 - val_accuracy:
Epoch 3/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6939 - accuracy: 0.0011 - val_loss: 0.6939 - val_accuracy:
Epoch 4/5
235/235 [==============================] - 4s 18ms/step - loss: 0.6938 - accuracy: 0.0011 - val_loss: 0.6937 - val_accuracy:
Epoch 5/5
235/235 [==============================] - 3s 13ms/step - loss: 0.6936 - accuracy: 0.0012 - val_loss: 0.6936 - val_accuracy:
```

Double-click (or enter) to edit

2. Do the prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib

```
#After reconstructed version of that test data.
X_test = autoencoder.predict(x_test)
plt.imshow(X_test[15].reshape(28,28))
```

```
313/313 [==============================] - 1s 2ms/step
<matplotlib.image.AxesImage at 0x7d328e26e8f0>
```

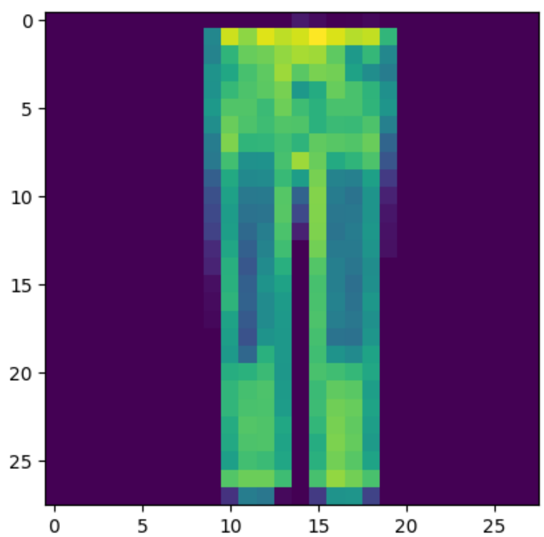


3. Repeat the question 2 on the denoisening autoencoder

```
from keras.layers import Input, Dense
from keras.models import Model

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

# this is our 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)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
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:])))
#Visualize the data before reconstruction
plt.imshow(x_test[15].reshape(28,28))
```

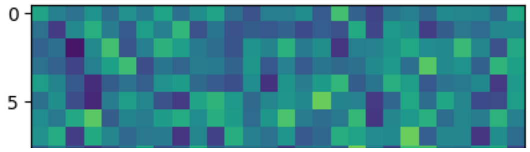
```
#introducing noise
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

```
Epoch 1/10
235/235 [==============================] - 4s 13ms/step - loss: 0.6994 - val_loss: 0.6992
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6991 - val_loss: 0.6988
Epoch 3/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6987 - val_loss: 0.6985
Epoch 4/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6984 - val_loss: 0.6982
Epoch 5/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6981 - val_loss: 0.6979
Epoch 6/10
235/235 [==============================] - 4s 15ms/step - loss: 0.6978 - val_loss: 0.6976
Epoch 7/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6975 - val_loss: 0.6973
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6972 - val_loss: 0.6970
Epoch 9/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6969 - val_loss: 0.6967
Epoch 10/10
235/235 [==============================] - 4s 15ms/step - loss: 0.6966 - val_loss: 0.6964
<keras.callbacks.History at 0x7d328e19fee0>
```




```
#After reconstructed version of that test data.
X_test = autoencoder.predict(x_test)
plt.imshow(X_test[15].reshape(28,28))
```

```
313/313 [==============================] - 2s 5ms/step
<matplotlib.image.AxesImage at 0x7d328de17b20>
```



4. plot loss and accuracy using the history object

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```