NN&DeepLearning_Assignment_8: Autoencoders

Bhanu Chandrika Lakkimsetti (700747439)

Applications of Autoencoder

GitHub Link: https://github.com/bhanuchandrika99/NNDL_ICP_Assignment-8

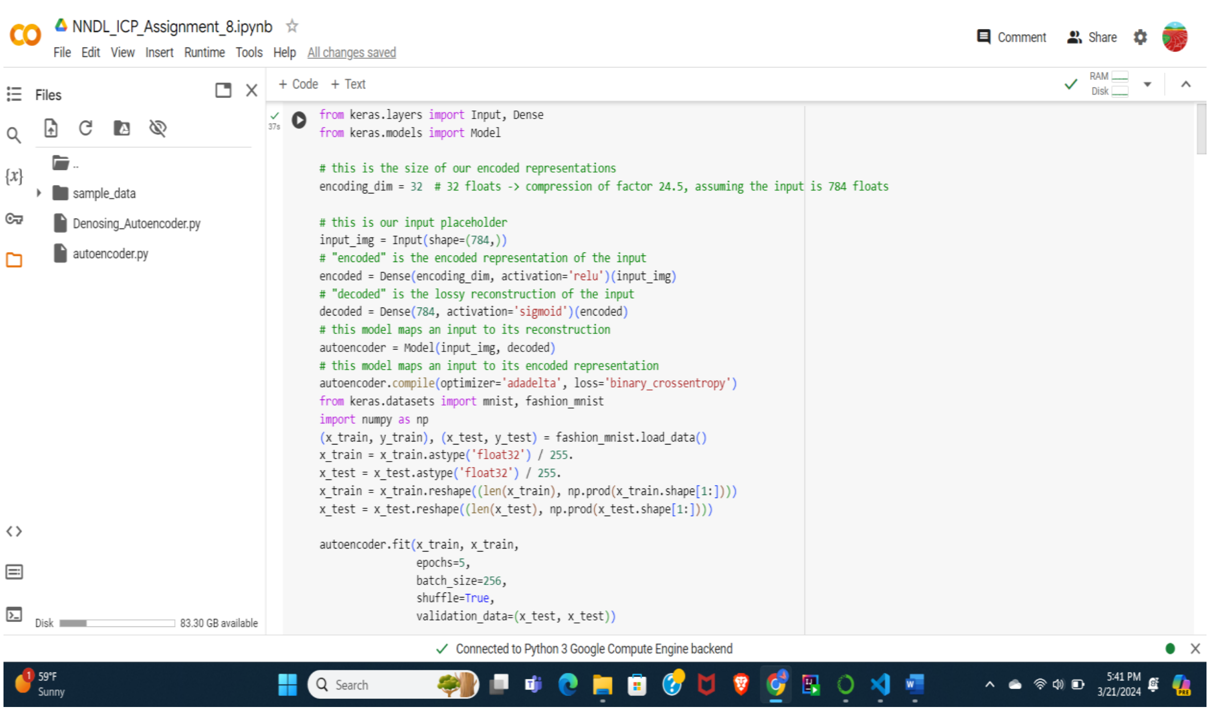
Lesson Overview: In this lesson, we are going to discuss types and applications of Autoencoder.

Programming elements:

1. Basics of Autoencoders
2. Role of Autoencoders in unsupervised learning
3. Types of Autoencoders
4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image
5. Use case: Stacked autoencoder.

In class programming:

1. Add one more hidden layer to autoencoder
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
3. Repeat the question 2 on the denoisening autoencoder
4. plot loss and accuracy using the history object.



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Epoch 1/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6963 - val_loss: 0.6961
Epoch 2/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6960 - val_loss: 0.6958
Epoch 3/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6957 - val_loss: 0.6955
Epoch 4/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6954 - val_loss: 0.6952
Epoch 5/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6951 - val_loss: 0.6949
<keras.src.callbacks.History at 0x7d250d343820>
```

Addition of one more hidden layer to autoencoder. Also, prediction on the test data. Visualization of one of the reconstructed version and Visualization using same test data before reconstruction using Matplotlib

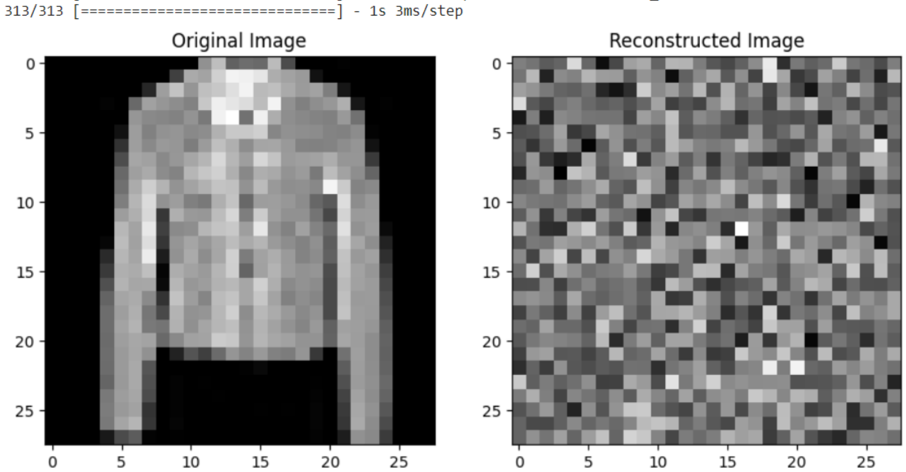
```
from keras.layers import Input, Dense
from keras.models import Model

# Define input shape
input_shape = (784,)

# Define encoding dimensions
encoding_dim1 = 64
encoding_dim2 = 32

# Define input layer
input_img = Input(shape=input_shape)

encoded1 = Dense(encoding_dim1, activation='relu')(input_img)
encoded2 = Dense(encoding_dim2, activation='relu')(encoded1)
decoded1 = Dense(encoding_dim1, activation='relu')(encoded2)
decoded2 = Dense(input_shape[0], activation='sigmoid')(decoded1)
autoencoder = Model(input_img, decoded2)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
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.
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:])))
```

```
# Train model
history = autoencoder.fit(x_train, x_train,
                          epochs=20,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test, x_test))

# Predict on test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize reconstructed image and original image
import matplotlib.pyplot as plt

# Choose an index of a test image to visualize
idx = 10

# Reshape the test image
test_img = x_test[idx].reshape(28, 28)

# Reshape the reconstructed image
reconstructed_img = decoded_imgs[idx].reshape(28, 28)
```

```python
# Plot the original and reconstructed images side by side
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(test_img, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(reconstructed_img, cmap='gray')
plt.title('Reconstructed Image')
plt.show()
```

```
Epoch 1/20
235/235 [==============================] - 4s 13ms/step - loss: 0.6930 - val_loss: 0.6930
Epoch 2/20
235/235 [==============================] - 3s 14ms/step - loss: 0.6930 - val_loss: 0.6930
Epoch 3/20
235/235 [==============================] - 3s 13ms/step - loss: 0.6929 - val_loss: 0.6929
Epoch 4/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6929 - val_loss: 0.6928
Epoch 5/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6928 - val_loss: 0.6928
Epoch 6/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6928 - val_loss: 0.6927
Epoch 7/20
235/235 [==============================] - 6s 24ms/step - loss: 0.6927 - val_loss: 0.6927
Epoch 8/20
235/235 [==============================] - 4s 19ms/step - loss: 0.6927 - val_loss: 0.6926
```

```
235/235 [==============================] - 3s 14ms/step - loss: 0.6921 - val_loss: 0.6921
Epoch 18/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6921 - val_loss: 0.6920
Epoch 19/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6920 - val_loss: 0.6920
Epoch 20/20
235/235 [==============================] - 4s 17ms/step - loss: 0.6920 - val_loss: 0.6919
313/313 [==============================] - 1s 3ms/step
```



Repeating the question 2 on the denoisening autoencoder.
Prediction on the test data.Visualization of one of the reconstructed version

```
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', metrics=['accuracy'])

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:])))

#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)
```

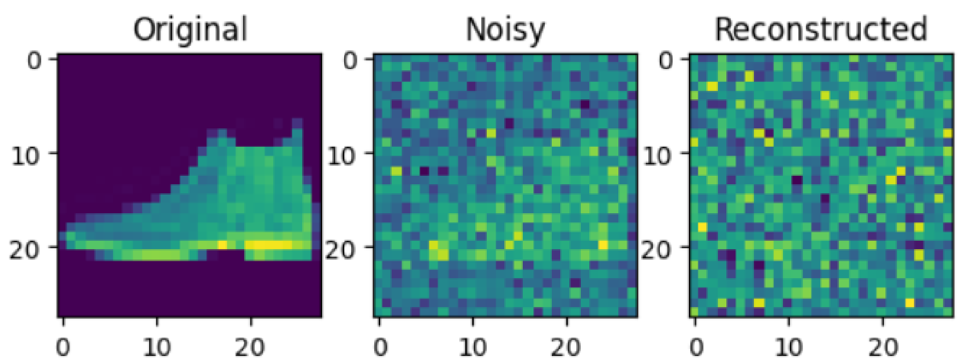
```
history=autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

```
Epoch 1/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6980 - accuracy: 8.1667e-04 - val_loss: 0.6979 - val_accuracy: 0.0012
Epoch 2/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6976 - accuracy: 8.1667e-04 - val_loss: 0.6976 - val_accuracy: 0.0012
Epoch 3/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6973 - accuracy: 8.6667e-04 - val_loss: 0.6972 - val_accuracy: 0.0012
Epoch 4/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6969 - accuracy: 8.3333e-04 - val_loss: 0.6969 - val_accuracy: 0.0012
Epoch 5/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6966 - accuracy: 8.3333e-04 - val_loss: 0.6965 - val_accuracy: 0.0012
Epoch 6/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6963 - accuracy: 8.3333e-04 - val_loss: 0.6962 - val_accuracy: 0.0012
Epoch 7/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6960 - accuracy: 8.6667e-04 - val_loss: 0.6959 - val_accuracy: 0.0012
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6957 - accuracy: 9.1667e-04 - val_loss: 0.6956 - val_accuracy: 0.0012
Epoch 9/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6954 - accuracy: 9.3333e-04 - val_loss: 0.6953 - val_accuracy: 0.0012
Epoch 10/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6951 - accuracy: 9.6667e-04 - val_loss: 0.6950 - val_accuracy: 0.0013
```

Visualization using same test data before reconstruction using Matplotlib

```python
import matplotlib.pyplot as plt

# Get the reconstructed images
reconstructed_imgs = autoencoder.predict(x_test_noisy)

# Select one image to display
img_to_display = 0

# Display the original, noisy, and reconstructed images side by side
plt.subplot(1, 3, 1)
plt.imshow(x_test[img_to_display].reshape(28, 28))
plt.title('Original')

plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[img_to_display].reshape(28, 28))
plt.title('Noisy')

plt.subplot(1, 3, 3)
plt.imshow(reconstructed_imgs[img_to_display].reshape(28, 28))
plt.title('Reconstructed')

plt.show()
```

```
313/313 [==============================] - 1s 1ms/step
```



Plotting loss and accuracy using the history object.

```python
# Plot the loss and accuracy over epochs
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()

plt.show()
```