7주차 실습과제

2017313107 이승태

1. 14장_기본적인 Autoencoder

(1) encoding부분과 decoding부분의 layer들을 만들어 준 뒤에 모델을 구성하였다.

```
[1]  from keras.layers import Input, Dense
     from keras.models import Model

     encoding_dim = 32

     #인풋이미지는 한 인스턴스 당 28x28=784개의 feature를 가지므로 shape을 다음과 같이 정해준다.
     input_img = Input(shape=(784,))

     #인코더와 디코더를 각각 한 층으로 쌓아준다.
     encoded = Dense(encoding_dim, activation='relu')(input_img)
     decoded = Dense(784, activation='sigmoid')(encoded)

     #최종 오토인코더 모형은 인풋부터 인코더, 디코더의 아웃풋까지 연결해준다.
     autoencoder = Model(input_img, decoded)
```

(2) 맨 위 부분은 인코더와 디코더를 분리해서 정의해놓은 것이며, (1)에서 만든 autoencoder를 complie한 후, MNISTset을 다운받는다.

```
[2]  #다음과 같이 인코더와 디코더를 분리해서 정의해놓는 것도 가능하다.
     encoder = Model(input_img, encoded)
     encoded_input = Input(shape=(encoding_dim,))
     decoder_layer = autoencoder.layers[-1]
     decoder = Model(encoded_input, decoder_layer(encoded_input))
```

```
[3]  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
[4]  from keras.datasets import mnist
     import numpy as np

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

     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
     11493376/11490434 [==============================] - 0s 0us/step
     11501568/11490434 [==============================] - 0s 0us/step
```

(3) 값들을 float으로 바꾼 뒤, train data와 test data를 분리시킨다.

```
[5]  #MNIST dataset의 x입력값들을 28x28=784의 크기로 벡터화한다.
     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:])))
     print(x_train.shape)
     print(x_test.shape)

     (60000, 784)
     (10000, 784)
```

(4) fit함수를 이용하여 오토인코더를 학습한다.

```
[6]  #위에서 정의한 오토인코더 모형 학습
     autoencoder.fit(x_train, x_train,
                     epochs=50,
                     batch_size=256,
                     shuffle=True,
                     validation_data=(x_test, x_test))
```

```
Epoch 1/50
235/235 [==============================] - 3s 11ms/step - loss: 0.2773 - val_loss: 0.1917
Epoch 2/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1722 - val_loss: 0.1551
Epoch 3/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1459 - val_loss: 0.1351
Epoch 4/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1282 - val_loss: 0.1200
Epoch 5/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1169 - val_loss: 0.1112
Epoch 6/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1095 - val_loss: 0.1053
Epoch 7/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1044 - val_loss: 0.1015
Epoch 8/50
235/235 [==============================] - 3s 11ms/step - loss: 0.1010 - val_loss: 0.0983
Epoch 9/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0986 - val_loss: 0.0964
Epoch 10/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0970 - val_loss: 0.0951
```

```
Epoch 39/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 40/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0928 - val_loss: 0.0918
Epoch 41/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 42/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 43/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 44/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 45/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 46/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 47/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 48/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 49/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 50/50
235/235 [==============================] - 3s 11ms/step - loss: 0.0926 - val_loss: 0.0915
<keras.callbacks.History at 0x7f7e08fa8dd0>
```

(5) encoded_imgs에 test data를 넣고, encoded_imgs에 encoding layer에서 나온 output을 넣어 decoded_imgs라는 data에 넣는다.

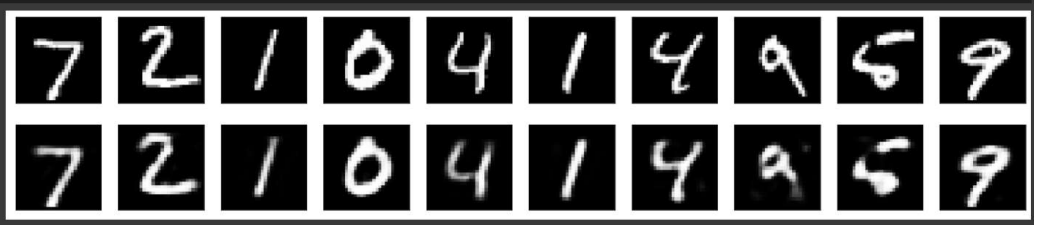
```
[7]  encoded_imgs = encoder.predict(x_test)
     decoded_imgs = decoder.predict(encoded_imgs)
```

(6) 위 그림은 실제 데이터 이고, 아래 데이터는 decoder에서 나온 값이다.

```
[8]  import matplotlib.pyplot as plt

     n = 10
     plt.figure(figsize=(20, 4))
     for i in range(n):
         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)

         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()
```



2. 14장_Deep Autoencoder (stacked AE)

(1) 1번과 마찬가지로 data를 구성하는 것은 같지만, [3]과 같이 깊은 신경망을 쌓아 오토인코더를 구성하였다.

```
[1]  from keras.layers import Input, Dense
     from keras.models import Model
     from keras.datasets import mnist
     import numpy as np
     (x_train, _), (x_test, _) = mnist.load_data()

     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
     11493376/11490434 [==============================] - 0s 0us/step
     11501568/11490434 [==============================] - 0s 0us/step

[2]  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:])))

[3]  #기본적인 오토인코더에서 인코더 및 디코더를 더 깊은 층으로 구성한다.
     input_img = Input(shape=(784,))
     encoded = Dense(128, activation='relu')(input_img)
     encoded = Dense(64, activation='relu')(encoded)
     encoded = Dense(32, activation='relu')(encoded)

     decoded = Dense(64, activation='relu')(encoded)
     decoded = Dense(128, activation='relu')(decoded)
     decoded = Dense(784, activation='sigmoid')(decoded)
```

(2) fit을 이용하여 모델을 학습했다.

```
[4]  autoencoder = Model(input_img, decoded)
     autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

     autoencoder.fit(x_train, x_train,
                     epochs=100,
                     batch_size=256,
                     shuffle=True,
                     validation_data=(x_test, x_test))
```

```
Epoch 84/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0822 - val_loss: 0.0822
Epoch 85/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0822 - val_loss: 0.0820
Epoch 86/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0821 - val_loss: 0.0821
Epoch 87/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0821 - val_loss: 0.0820
Epoch 88/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0820 - val_loss: 0.0817
Epoch 89/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0819 - val_loss: 0.0817
Epoch 90/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0819 - val_loss: 0.0821
Epoch 91/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0819 - val_loss: 0.0817
Epoch 92/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0818 - val_loss: 0.0817
Epoch 93/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0818 - val_loss: 0.0817
Epoch 94/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0818 - val_loss: 0.0817
Epoch 95/100
235/235 [==============================] - 4s 18ms/step - loss: 0.0817 - val_loss: 0.0814
Epoch 96/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0817 - val_loss: 0.0816
Epoch 97/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0816 - val_loss: 0.0816
Epoch 98/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0816 - val_loss: 0.0812
Epoch 99/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0816 - val_loss: 0.0816
Epoch 100/100
235/235 [==============================] - 4s 17ms/step - loss: 0.0815 - val_loss: 0.0811
<keras.callbacks.History at 0x7f5fcabdcbd0>
```

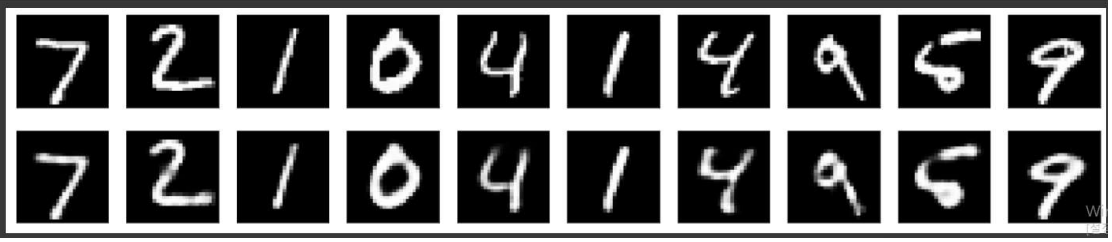
(3) output을 저장한 후에 기존 데이터와 비교해 보았다.

```
[5]  #오토인코더의 아웃풋을 변수에 저장
     decoded_imgs = autoencoder.predict(x_test)
```

```
[6]  import matplotlib.pyplot as plt

     n = 10
     plt.figure(figsize=(20, 4))
     for i in range(n):
         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)

         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()
```



3. 14장_Convolutional Autoencoder

(1) convolution network로 모델을 구성한 후 마찬가지로 mnist data를 이용한다.

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
[2] from keras.layers import Conv2D, MaxPooling2D, UpSampling2D

    #컨볼루션 오토인코더에서는 인풋이 이미지 그대로 들어간다. (28x28)
    #기본적인 오토인코더가 MLP 모형으로 이루어졌다면, 컨볼루션 오토인코더는 CNN(Convolutional Neural Networks)으로 이루어졌다고 볼 수 있다.

    input_img = Input(shape=(28, 28, 1))

    x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    encoded = MaxPooling2D((2, 2), padding='same')(x)

    x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(16, (3, 3), activation='relu')(x)
    x = UpSampling2D((2, 2))(x)
    decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
[3] x_train = x_train.astype('float32') / 255
    x_test = x_test.astype('float32') / 255
    x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
    x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

(2) fit함수를 이용하여 fit한다.

```
[4] autoencoder.fit(x_train, x_train,
                    epochs=50,
                    batch_size=1024,
                    shuffle=True,
                    validation_data=(x_test, x_test))
```

```
Epoch 35/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1068 - val_loss: 0.1055
Epoch 36/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1063 - val_loss: 0.1051
Epoch 37/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1061 - val_loss: 0.1048
Epoch 38/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1058 - val_loss: 0.1046
Epoch 39/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1053 - val_loss: 0.1041
Epoch 40/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1050 - val_loss: 0.1038
Epoch 41/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1048 - val_loss: 0.1037
Epoch 42/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1043 - val_loss: 0.1032
Epoch 43/50
59/59 [==============================] - 3s 48ms/step - loss: 0.1042 - val_loss: 0.1030
Epoch 44/50
59/59 [==============================] - 3s 48ms/step - loss: 0.1039 - val_loss: 0.1026
Epoch 45/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1035 - val_loss: 0.1024
Epoch 46/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1033 - val_loss: 0.1020
Epoch 47/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1031 - val_loss: 0.1022
Epoch 48/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1027 - val_loss: 0.1021
Epoch 49/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1027 - val_loss: 0.1015
Epoch 50/50
59/59 [==============================] - 3s 47ms/step - loss: 0.1023 - val_loss: 0.1010
<keras.callbacks.History at 0x7f8470660650>
```

(3) autoencoder를 이용해 데이터를 생성한 후 기존의 데이터와 비교했다.

```
[6]  import matplotlib.pyplot as plt
     n = 10
     plt.figure(figsize=(20, 4))
     for i in range(1,n+1):
         ax = plt.subplot(2, n, i)
         plt.imshow(x_test[i].reshape(28, 28))
         plt.gray()
         ax.get_xaxis().set_visible(False)
         ax.get_yaxis().set_visible(False)

         ax = plt.subplot(2, n, i+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()
```

4. 14장_Denoising Autoencoder

(1) 데이터에 가우시안 noise를 집어넣는다.

```
[1]  from keras.layers import Input, Dense
     from keras.models import Model
     from keras.datasets import mnist
     import numpy as np
     (x_train, _), (x_test, _) = mnist.load_data()

     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
     11493376/11490434 [==============================] - 0s 0us/step
     11501568/11490434 [==============================] - 0s 0us/step

[2]  #컨볼루션 오토인코더를 활용하여 Denoising
     x_train = x_train.astype('float32') / 255
     x_test = x_test.astype('float32') / 255
     x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
     x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

[3]  #가우스 노이즈 데이터 생성
     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)

     x_train_noisy = np.clip(x_train_noisy, 0, 1)
     x_test_noisy = np.clip(x_test_noisy, 0, 1)
```

(2) 노이즈가 있는 데이터를 시각화한 것이다.

```
[4]  #노이즈 데이터 시각화
     import matplotlib.pyplot as plt

     n = 10
     plt.figure(figsize=(20, 2))
     for i in range(1,n+1):
         ax = plt.subplot(1, n, i)
         plt.imshow(x_test_noisy[i].reshape(28, 28))
         plt.gray()
         ax.get_xaxis().set_visible(False)
         ax.get_yaxis().set_visible(False)
     plt.show()
```

(3) 컨볼루션 오토인코더를 구성한 후에 fit으로 학습시킨다.

```
[5]  #컨볼루션 오토인코더를 활용
     #Deep autoencoder를 활용해도 된다.(이 경우 인풋을 784로 맞춰야 한다.)

     from keras.layers import Conv2D, MaxPooling2D, UpSampling2D
     input_img = Input(shape=(28, 28, 1))

     x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
     x = MaxPooling2D((2, 2), padding='same')(x)
     x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
     encoded = MaxPooling2D((2, 2), padding='same')(x)

     x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
     x = UpSampling2D((2, 2))(x)
     x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
     x = UpSampling2D((2, 2))(x)
     decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

     autoencoder = Model(input_img, decoded)
     autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
[6]  autoencoder.fit(x_train_noisy, x_train,
                     epochs=50,
                     batch_size=1024,
                     shuffle=True,
                     validation_data=(x_test_noisy, x_test))
```

```
Epoch 33/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1026 - val_loss: 0.1016
Epoch 34/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1025 - val_loss: 0.1015
Epoch 35/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1022 - val_loss: 0.1015
Epoch 36/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1020 - val_loss: 0.1012
Epoch 37/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1019 - val_loss: 0.1010
Epoch 38/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1018 - val_loss: 0.1013
Epoch 39/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1017 - val_loss: 0.1007
Epoch 40/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1015 - val_loss: 0.1008
Epoch 41/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1013 - val_loss: 0.1007
Epoch 42/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1012 - val_loss: 0.1002
Epoch 43/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1011 - val_loss: 0.1001
Epoch 44/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1009 - val_loss: 0.1005
Epoch 45/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1008 - val_loss: 0.1000
Epoch 46/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1006 - val_loss: 0.0998
Epoch 47/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1005 - val_loss: 0.0997
Epoch 48/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1003 - val_loss: 0.0996
Epoch 49/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1003 - val_loss: 0.0994
Epoch 50/50
59/59 [==============================] - 2s 38ms/step - loss: 0.1001 - val_loss: 0.0994
<keras.callbacks.History at 0x7fb47025ead0>
```

(4) 데이터를 오토인코더에 넣은 결과 데이터가 노이즈가 없이 잘 출력되었다.

```
[7]  decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
[8]  n = 10
     plt.figure(figsize=(20, 2))
     for i in range(1,n+1):
         ax = plt.subplot(1, n, i)
         plt.imshow(decoded_imgs[i].reshape(28, 28))
         plt.gray()
         ax.get_xaxis().set_visible(False)
         ax.get_yaxis().set_visible(False)

     plt.show()
```