

## 7주차 실습과제

2016314786 김호진

### [14장 실습 – 기본적인 Autoencoder]

jupyter 14장 기본적인 Autoencoder Last Checkpoint: 4분 전 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | machinelearning C

```
In [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 알아준다.
        encoded = Dense(encoding_dim, activation='relu')(input_img)
        decoded = Dense(784, activation='sigmoid')(encoded)

        #최종 오토인코더 모델은 인코더, 인코더, 디코더의 아웃풋까지 연결해준다.
        autoencoder = Model([input_img], decoded)

In [2]: #다음과 같이 인코더와 디코더를 분리해서 경의해놓는 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))

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

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

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

In [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)

In [6]: #위에서 정의한 오토인코더 모델 학습
        autoencoder.fit(x_train, x_train,
                        epochs=60,
                        batch_size=256,
                        shuffle=True,
                        validation_data=(x_test, x_test))

Epoch 50/60
236/236 [=====] - 2s 9ms/step - loss: 0.0925 - val_loss: 0.0914
Epoch 53/60
236/236 [=====] - 2s 9ms/step - loss: 0.0925 - val_loss: 0.0915
Epoch 54/60
236/236 [=====] - 2s 9ms/step - loss: 0.0925 - val_loss: 0.0914
Epoch 55/60
236/236 [=====] - 2s 10ms/step - loss: 0.0925 - val_loss: 0.0914
Epoch 56/60
236/236 [=====] - 2s 10ms/step - loss: 0.0925 - val_loss: 0.0914
Epoch 57/60
236/236 [=====] - 2s 10ms/step - loss: 0.0925 - val_loss: 0.0915
Epoch 58/60
236/236 [=====] - 2s 10ms/step - loss: 0.0924 - val_loss: 0.0915
Epoch 59/60
236/236 [=====] - 2s 10ms/step - loss: 0.0924 - val_loss: 0.0913
Epoch 60/60
236/236 [=====] - 2s 9ms/step - loss: 0.0924 - val_loss: 0.0914


Out [6]: <tensorflow.python.keras.callbacks.History at 0x26497efe550>

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

In [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()
```



## [14장 실습 - Deep Autoencoder (stacked AE)]

jupyter

14장 Deep Autoencoder (stacked AE)

Last Checkpoint: 5분 전 (unsaved changes)

 Logout

FileEditViewInsertCellKernelWidgetsHelp

Trusted | machinelearning

Code

In [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()
```

In [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:])))
```

In [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)
```

In [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))
```

Out [4]:

```
<tensorflow.python.keras.callbacks.History at 0x20000199730>
```

In [5]:

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

In [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()
```



## [14장 실습 - Convolutional Autoencoder]

14장 Convolutional Autoencoder.ipynb

파일 수정 보기 삽입 런타임 도구 도움말

+ 코드 + 텍스트

RAM 디스크

수정 가능

```
[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
11453376/11450344 [=====] - 0s 0us/step
11501568/11450344 [=====] - 0s 0us/step

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', 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')
```

```
[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))

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=32,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 22/50
59/59 [=====] - 3s 48ms/step - loss: 0.1127 - val_loss: 0.1109
Epoch 23/50
59/59 [=====] - 3s 47ms/step - loss: 0.1120 - val_loss: 0.1103
Epoch 24/50
59/59 [=====] - 3s 48ms/step - loss: 0.1111 - val_loss: 0.1097
Epoch 25/50
59/59 [=====] - 3s 48ms/step - loss: 0.1107 - val_loss: 0.1092
Epoch 26/50
59/59 [=====] - 3s 48ms/step - loss: 0.1103 - val_loss: 0.1086
Epoch 27/50
59/59 [=====] - 3s 47ms/step - loss: 0.1096 - val_loss: 0.1080
Epoch 28/50
59/59 [=====] - 3s 47ms/step - loss: 0.1092 - val_loss: 0.1076
Epoch 29/50
59/59 [=====] - 3s 48ms/step - loss: 0.1085 - val_loss: 0.1070
Epoch 30/50
59/59 [=====] - 3s 47ms/step - loss: 0.1081 - val_loss: 0.1067
Epoch 31/50
59/59 [=====] - 3s 47ms/step - loss: 0.1079 - val_loss: 0.1062
Epoch 32/50
59/59 [=====] - 3s 47ms/step - loss: 0.1070 - val_loss: 0.1058
Epoch 33/50
59/59 [=====] - 3s 47ms/step - loss: 0.1069 - val_loss: 0.1055
Epoch 34/50
59/59 [=====] - 3s 50ms/step - loss: 0.1065 - val_loss: 0.1051
Epoch 35/50
59/59 [=====] - 3s 47ms/step - loss: 0.1064 - val_loss: 0.1048
Epoch 36/50
59/59 [=====] - 3s 48ms/step - loss: 0.1058 - val_loss: 0.1044
Epoch 37/50
59/59 [=====] - 3s 47ms/step - loss: 0.1052 - val_loss: 0.1041
Epoch 38/50
59/59 [=====] - 3s 48ms/step - loss: 0.1052 - val_loss: 0.1037
Epoch 39/50
59/59 [=====] - 3s 48ms/step - loss: 0.1047 - val_loss: 0.1034
Epoch 40/50
59/59 [=====] - 3s 47ms/step - loss: 0.1045 - val_loss: 0.1031
Epoch 41/50
59/59 [=====] - 3s 47ms/step - loss: 0.1042 - val_loss: 0.1028
Epoch 42/50
59/59 [=====] - 3s 47ms/step - loss: 0.1040 - val_loss: 0.1026
Epoch 43/50
59/59 [=====] - 3s 47ms/step - loss: 0.1037 - val_loss: 0.1022
Epoch 44/50
59/59 [=====] - 3s 48ms/step - loss: 0.1034 - val_loss: 0.1020
Epoch 45/50
59/59 [=====] - 3s 48ms/step - loss: 0.1031 - val_loss: 0.1017
Epoch 46/50
59/59 [=====] - 3s 48ms/step - loss: 0.1029 - val_loss: 0.1015
Epoch 47/50
59/59 [=====] - 3s 48ms/step - loss: 0.1031 - val_loss: 0.1014
Epoch 48/50
59/59 [=====] - 3s 48ms/step - loss: 0.1025 - val_loss: 0.1012
Epoch 49/50
59/59 [=====] - 3s 48ms/step - loss: 0.1022 - val_loss: 0.1008
Epoch 50/50
59/59 [=====] - 3s 48ms/step - loss: 0.1019 - val_loss: 0.1008
<keras.callbacks.History at 0x7f6290425fd0>
```

```
[5] decoded_imgs = autoencoder.predict(x_test)
```

```
[6] import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(30, 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()
```

## [14장 실습 - Denoising Autoencoder]

Untitled3.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 모든 notebook의 저장소

+ 코드 + 텍스트

RAM 사용량 100% CPU 사용량 100%

수정 가능

```
[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] #간혹무선 오로인코디를 활용하여 Denoising
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
x_train = np.reshape(x_train, (-1, x_train.shape[1]))
x_test = np.reshape(x_test, (-1, x_test.shape[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)
```

```
[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()
```



```
[5] #간혹무선 오로인코디를 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=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))
```

Epoch 22/50 - 2s 30ms/step - loss: 0.1056 - val\_loss: 0.1041  
Epoch 23/50 - 2s 30ms/step - loss: 0.1050 - val\_loss: 0.1042  
Epoch 24/50 - 2s 30ms/step - loss: 0.1046 - val\_loss: 0.1034  
Epoch 25/50 - 2s 30ms/step - loss: 0.1044 - val\_loss: 0.1034  
Epoch 26/50 - 2s 30ms/step - loss: 0.1041 - val\_loss: 0.1027  
Epoch 27/50 - 2s 30ms/step - loss: 0.1034 - val\_loss: 0.1029  
Epoch 28/50 - 2s 30ms/step - loss: 0.1035 - val\_loss: 0.1024  
Epoch 29/50 - 2s 30ms/step - loss: 0.1033 - val\_loss: 0.1019  
Epoch 30/50 - 2s 30ms/step - loss: 0.1027 - val\_loss: 0.1016  
Epoch 31/50 - 2s 30ms/step - loss: 0.1027 - val\_loss: 0.1015  
Epoch 32/50 - 2s 30ms/step - loss: 0.1024 - val\_loss: 0.1012  
Epoch 33/50 - 2s 30ms/step - loss: 0.1019 - val\_loss: 0.1010  
Epoch 34/50 - 2s 30ms/step - loss: 0.1020 - val\_loss: 0.1008  
Epoch 35/50 - 2s 30ms/step - loss: 0.1016 - val\_loss: 0.1007  
Epoch 36/50 - 2s 30ms/step - loss: 0.1017 - val\_loss: 0.1005  
Epoch 37/50 - 2s 30ms/step - loss: 0.1016 - val\_loss: 0.1003  
Epoch 38/50 - 2s 30ms/step - loss: 0.1011 - val\_loss: 0.1002  
Epoch 39/50 - 2s 30ms/step - loss: 0.1009 - val\_loss: 0.1000  
Epoch 40/50 - 2s 30ms/step - loss: 0.1007 - val\_loss: 0.1003  
Epoch 41/50 - 2s 30ms/step - loss: 0.1007 - val\_loss: 0.0997  
Epoch 42/50 - 2s 30ms/step - loss: 0.1004 - val\_loss: 0.0996  
Epoch 43/50 - 2s 30ms/step - loss: 0.1003 - val\_loss: 0.0996  
Epoch 44/50 - 2s 30ms/step - loss: 0.1003 - val\_loss: 0.0993  
Epoch 45/50 - 2s 30ms/step - loss: 0.1004 - val\_loss: 0.0991  
Epoch 46/50 - 2s 30ms/step - loss: 0.1000 - val\_loss: 0.0993  
Epoch 47/50 - 2s 30ms/step - loss: 0.0998 - val\_loss: 0.0989  
Epoch 48/50 - 2s 30ms/step - loss: 0.0997 - val\_loss: 0.0988  
Epoch 49/50 - 2s 30ms/step - loss: 0.0995 - val\_loss: 0.0987  
Epoch 50/50 - 2s 30ms/step - loss: 0.0995 - val\_loss: 0.0985  
<keras.callbacks.History at 0x7ef6008750>

```
[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()
```

