AutoEncoder(AE)

- 입력데이터의 차원축소를 통해 의미있는 잠재공간(Latent Space/Manifold)을 학습
  - Encoder 학습을 위해 Decoder를 사용

```
import warnings
warnings.filterwarnings('ignore')
```

+ 코드 + 텍스트

I. Load MNIST Dataset

1) X_train and X_test

```
from keras.datasets import mnist

(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
```

```
X_train.shape, X_test.shape
```

```
((60000, 28, 28), (10000, 28, 28))
```

2) Normalization and Reshape

```
X_train = X_train / 255.
X_test = X_test / 255.

X_train = X_train.reshape(60000, 28 * 28)
X_test = X_test.reshape(10000, 28 * 28)
```

```
X_train.shape, X_test.shape
```

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

II. Keras Modeling with Functional API

1) 'Latent Space' Size

```
encoding_dim = 32
```

2) Encoder

- Input

```
from keras import layers

input_img = layers.Input(shape = (784, ))
```

- Encoding Layers and Encoding Model

```
encoded = layers.Dense(256, activation = 'elu')(input_img)
encoded = layers.Dense(128, activation = 'elu')(encoded)
encoded = layers.Dense(encoding_dim, activation = 'elu')(encoded)
```

3) Decoder

- Decoding Lyaers and Decoding Model

```
decoded = layers.Dense(128, activation = 'elu')(encoded)
decoded = layers.Dense(256, activation = 'elu')(decoded)
decoded = layers.Dense(784, activation = 'sigmoid')(decoded)
```

III. AutoEncoder Model

1) 'autoencoder' Model

```
from keras import models

autoencoder = models.Model(input_img, decoded)
```

```
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
dense (Dense)                (None, 256)               200960
```

```
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_3 (Dense)              (None, 128)               4224
_________________________________________________________________
dense_4 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_5 (Dense)              (None, 784)               201488
=================================================================
Total params: 476,720
Trainable params: 476,720
Non-trainable params: 0
_________________________________________________________________
```

2) 'encoder' Model

```
encoder = models.Model(input_img, encoded)
```

```
encoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
dense (Dense)                (None, 256)               200960
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 32)                4128
=================================================================
Total params: 237,984
Trainable params: 237,984
Non-trainable params: 0
_________________________________________________________________
```

3) 'decoder' Model

```
encoded_input = layers.Input(shape = (encoding_dim,))

decoder_layer = autoencoder.layers[-3](encoded_input)
decoder_layer = autoencoder.layers[-2](decoder_layer)
decoder_layer = autoencoder.layers[-1](decoder_layer)

decoder = models.Model(encoded_input, decoder_layer)
```

```
decoder.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 32)]              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               4224
_________________________________________________________________
dense_4 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_5 (Dense)              (None, 784)               201488
=================================================================
Total params: 238,736
Trainable params: 238,736
Non-trainable params: 0
_________________________________________________________________
```

IV. Model Fit

1) Model Compile

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

2) Model Training

```
autoencoder.fit(X_train, X_train,
                epochs = 50,
                batch_size = 256,
                validation_data = (X_test, X_test))
```

```
Epoch 22/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0880 - val_loss: 0.0876
Epoch 23/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0874 - val_loss: 0.0875
Epoch 24/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0870 - val_loss: 0.0869
Epoch 25/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0868 - val_loss: 0.0864
Epoch 26/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0864 - val_loss: 0.0866
Epoch 27/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0862 - val_loss: 0.0858
Epoch 28/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0854 - val_loss: 0.0854
Epoch 29/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0851 - val_loss: 0.0851
Epoch 30/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0848 - val_loss: 0.0849
Epoch 31/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0845 - val_loss: 0.0847
Epoch 32/50
```

```
Epoch 32/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0843 - val_loss: 0.0845
Epoch 33/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0841 - val_loss: 0.0844
Epoch 34/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0838 - val_loss: 0.0840
Epoch 35/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0836 - val_loss: 0.0838
Epoch 36/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0834 - val_loss: 0.0836
Epoch 37/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0833 - val_loss: 0.0835
Epoch 38/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0830 - val_loss: 0.0832
Epoch 39/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0825 - val_loss: 0.0834
Epoch 40/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0826 - val_loss: 0.0827
Epoch 41/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0822 - val_loss: 0.0826
Epoch 42/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0820 - val_loss: 0.0824
Epoch 43/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0819 - val_loss: 0.0824
Epoch 44/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0819 - val_loss: 0.0823
Epoch 45/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0818 - val_loss: 0.0822
Epoch 46/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0815 - val_loss: 0.0823
Epoch 47/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0815 - val_loss: 0.0823
Epoch 48/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0816 - val_loss: 0.0818
Epoch 49/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0812 - val_loss: 0.0817
Epoch 50/50
235/235 [==============================] - 1s 4ms/step - loss: 0.0809 - val_loss: 0.0816
<tensorflow.python.keras.callbacks.History at 0x7f637c96cfd0>
```

V. Model Predict

1) Imgae Encoding

- Create 'Latent Space'

```
encoded_imgs = encoder.predict(X_test)

encoded_imgs.shape
```

```
(10000, 32)
```

2) 'Latent Space' Decoding

```
decoded_imgs = decoder.predict(encoded_imgs)

decoded_imgs.shape
```

```
(10000, 784)
```

VI. Visualization

- 10개의 이미지 데이터를 모델에 적용

```
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()
```

#

#

#

The End

#

#

#

✓ 0초 오전 8:44에 완료됨