AAYUSH JHA
22
CSE(DS)
EXP 5

AUTOENCODER IN DL

```
import keras
from keras import layers

# 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 image
input_img = keras.Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)

encoder = keras.Model(input_img, encoded)

encoded_input = keras.Input(shape=(encoding_dim,))
# Retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

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

from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = 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:])))
print(x_train.shape)
print(x_test.shape)
```

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

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/50
235/235 [==============================] - 5s 18ms/step - loss: 0.2767 - val_loss: 0.1893
Epoch 2/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1700 - val_loss: 0.1536
Epoch 3/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1445 - val_loss: 0.1344
Epoch 4/50
235/235 [==============================] - 3s 13ms/step - loss: 0.1293 - val_loss: 0.1223
Epoch 5/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1192 - val_loss: 0.1140
Epoch 6/50
235/235 [==============================] - 3s 13ms/step - loss: 0.1120 - val_loss: 0.1082
Epoch 7/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1068 - val_loss: 0.1035
Epoch 8/50
235/235 [==============================] - 3s 12ms/step - loss: 0.1031 - val_loss: 0.1003
Epoch 9/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1003 - val_loss: 0.0979
Epoch 10/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0983 - val_loss: 0.0963
Epoch 11/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0969 - val_loss: 0.0951
Epoch 12/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0959 - val_loss: 0.0943
Epoch 13/50
235/235 [==============================] - 4s 17ms/step - loss: 0.0952 - val_loss: 0.0937
Epoch 14/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0948 - val_loss: 0.0933
Epoch 15/50
235/235 [==============================] - 3s 13ms/step - loss: 0.0945 - val_loss: 0.0931
Epoch 16/50
235/235 [==============================] - 3s 12ms/step - loss: 0.0942 - val_loss: 0.0929
Epoch 17/50
235/235 [==============================] - 4s 17ms/step - loss: 0.0940 - val_loss: 0.0927
```

```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
```

```
import matplotlib.pyplot as plt

n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
```

```
for i in range(n):
    # Display original
    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)

    # Display reconstruction
    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()
```