Ex.No: 6 Roll.No: 210701158

### PYTHON PROGRAM TO BUILD AUTOENCODERS WITH KERAS

### Aim:

To build autoencoders with keras in python.

### **Procedure:**

- 1. Import NumPy and Keras modules for building and training the autoencoder.
- 2. Define the input dimension (e.g., 784 for flattened 28x28 MNIST images) and encoding dimension.
- 3. Create an input layer with the specified input dimension.
- 4. Build the encoder part of the autoencoder with three Dense layers, reducing dimensionality to the encoding dimension.
- 5. Construct the decoder part of the autoencoder with three Dense layers, reconstructing the input to the original dimension.
- 6. Define the autoencoder model with the input layer and the reconstructed output.
- 7. Create a separate encoder model to extract encoded features from the input.
- 8. Compile the autoencoder using Adam optimizer and binary crossentropy loss.
- 9. Generate dummy training and test data and train the autoencoder for 50 epochs.
- Use the encoder to obtain encoded representations and the autoencoder to reconstruct the input, printing the shapes of the results.

```
Code:
```

```
import numpy as np
from keras.layers import Input, Dense
from keras.models import Model
# Define input dimension
input_dim = 784 # For example, flattened 28x28 MNIST images
# Define encoding dimension
encoding_dim = 32
# Input layer
input_img = Input(shape=(input_dim,))
# Encoder layers
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(encoding_dim, activation='relu')(encoded)
# Decoder layers
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(input_dim, activation='sigmoid')(decoded)
# Autoencoder model
autoencoder = Model(input_img, decoded)
# Separate encoder model
encoder = Model(input_img, encoded)
# Compile the model
```

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# Generate dummy data for demonstration
x_train = np.random.random((1000, input_dim))
x_test = np.random.random((200, input_dim))
# Train the autoencoder
autoencoder.fit(x_train, x_train,
        epochs=50,
        batch_size=256,
        shuffle=True,
        validation_data=(x_test, x_test))
# Use the encoder to encode some input
encoded_imgs = encoder.predict(x_test)
# Use the autoencoder to reconstruct some input
decoded_imgs = autoencoder.predict(x_test)
print("Shape of encoded images:", encoded_imgs.shape)
print("Shape of decoded images:", decoded_imgs.shape)
```

# **Output:**

```
4/4 -
                       — 0s 9ms/step - loss: 0.6878 - val_loss: 0.6923
Epoch 38/50
4/4 —
                      — 0s 9ms/step - loss: 0.6878 - vαl_loss: 0.6922
Epoch 39/50
4/4 ----
                      — 0s 9ms/step - loss: 0.6876 - val_loss: 0.6922
Epoch 40/50
4/4 -
                      — 0s 9ms/step - loss: 0.6875 - vαl_loss: 0.6922
Epoch 41/50
4/4 —
                     — 0s 9ms/step - loss: 0.6874 - val_loss: 0.6922
Epoch 42/50
4/4 —
                      — 0s 9ms/step - loss: 0.6872 - val_loss: 0.6922
Epoch 43/50
4/4 ----
                      — 0s 9ms/step - loss: 0.6870 - val_loss: 0.6921
Epoch 44/50
4/4 -
                       — 0s 9ms/step - loss: 0.6869 - val_loss: 0.6921
Epoch 45/50
4/4 ---
                      — 0s 9ms/step - loss: 0.6868 - val_loss: 0.6921
Epoch 46/50
4/4 —
                      — 0s 10ms/step - loss: 0.6867 - val_loss: 0.6921
Epoch 47/50
4/4 -----
                     --- Os 9ms/step - loss: 0.6866 - vαl_loss: 0.6921
Epoch 48/50
4/4 —
                      — 0s 9ms/step - loss: 0.6864 - val_loss: 0.6921
Epoch 49/50
4/4 -----
                     — 0s 9ms/step - loss: 0.6862 - val_loss: 0.6921
Epoch 50/50
4/4 ----
                       - 0s 9ms/step - loss: 0.6860 - val_loss: 0.6922
7/7 —
                       - 0s 2ms/step
7/7 ____
               _____ 0s 3ms/step
Shape of encoded images: (200, 32)
Shape of decoded images: (200, 784)
```

# **Result:**

Thus, to build autoencoders with Keras has been done successfully.