COURSE CODE: DJ22DSL501 **DATE:**

COURSE NAME: Machine Learning - II **CLASS: AY 2024-25**

LAB EXPERIMENT NO. 8

AIM :

Compare the performance of PCA and Auto encoders on a given dataset

THEORY:

What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Principle Component Analysis

Principle Component Analysis is an unsupervised technique where the original data is projected to the direction of high variance. These directions of high variance are orthogonal to each other resulting in very low or almost close to 0 correlation in the projected data. These features transformation is linear and the methodology to do it is:

Step 1: Calculate the Correlation matrix data consisting of n dimensions. The Correlation matrix will be of shape n*n.

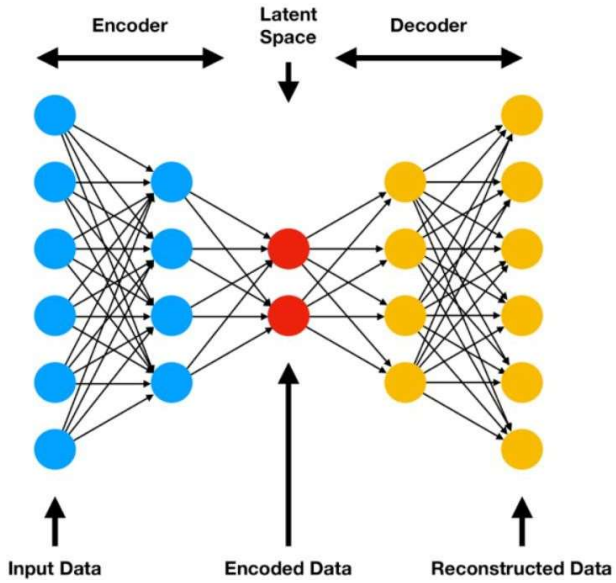
Step 2: Calculate the Eigenvectors and Eigenvalues of this matrix.

Step 3: Take the first k-eigenvectors with the highest eigenvalues.

Step 4: Project the original dataset into these k eigenvectors resulting in k dimensions where $k \leq n$.

Autoencoders

Autoencoder is an unsupervised artificial neural network that compresses the data to lower dimension and then reconstructs the input back. Autoencoder finds the representation of the data in a lower dimension by focusing more on the important features getting rid of noise and redundancy. It's based on Encoder-Decoder architecture, where encoder encodes the high-dimensional data to lower-dimension and decoder takes the lower-dimensional data and tries to reconstruct the original high-dimensional data.



Tasks to be performed:

1. **Use the Iris Dataset present in the scikit-learn library**
2. **Create an Auto Encoder and fit it with our data using 3 neurons in the dense layer**
3. **Use encoded layer to encode the training input**
4. **Plot loss for different encoders [PCA, Linear Autoencoder, Sigmoid based Non-Linear Autoencoder, ReLU based Non-Linear Auto encoder]**

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.datasets import load_iris
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     from keras.models import Model
     from keras.layers import Input, Dense
     from keras.callbacks import History
```

```
[ ]  # Load and preprocess the Iris dataset
     iris = load_iris()
     data = iris.data
     scaler = StandardScaler()
     data_scaled = scaler.fit_transform(data)
```

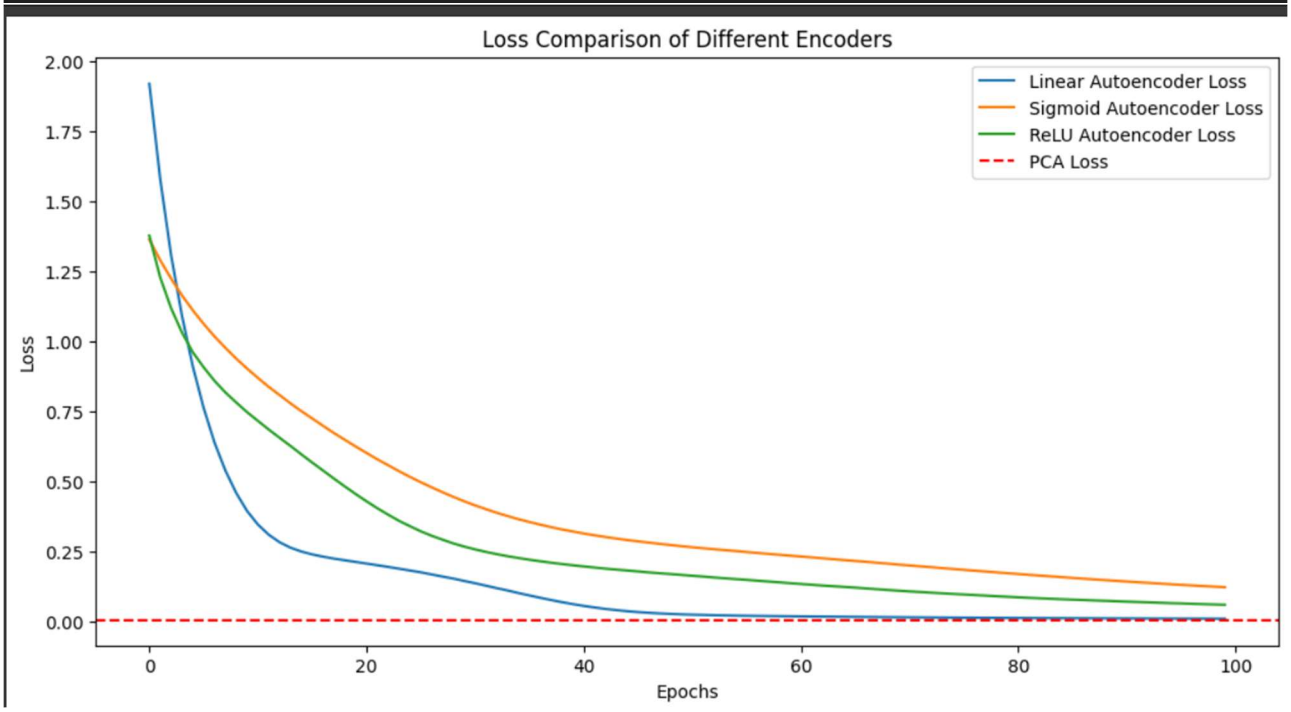
```
     # Autoencoder functions
     def create_autoencoder(activation='linear'):
         input_layer = Input(shape=(data_scaled.shape[1],))
         encoded = Dense(3, activation=activation)(input_layer)
         decoded = Dense(data_scaled.shape[1], activation='linear')(encoded)

         autoencoder = Model(input_layer, decoded)
         autoencoder.compile(optimizer='adam', loss='mse')
         return autoencoder
```

```
[ ]  # Train and collect losses
     def train_autoencoder(activation):
         autoencoder = create_autoencoder(activation)
         history = History()
         autoencoder.fit(data_scaled, data_scaled, epochs=100, batch_size=5, shuffle=True, verbose=0, callbacks=[history])
         return history.history['loss']
```

```
[ ]  # PCA
     pca = PCA(n_components=3)
     pca.fit(data_scaled)
     pca_transformed = pca.transform(data_scaled)
     pca_reconstructed = pca.inverse_transform(pca_transformed)
     pca_loss = np.mean((data_scaled - pca_reconstructed) ** 2)
```

```
[ ]  # Train autoencoders
     linear_loss = train_autoencoder(activation='linear')
     sigmoid_loss = train_autoencoder(activation='sigmoid')
     relu_loss = train_autoencoder(activation='relu')
```

```
[ ]  # Plotting
     plt.figure(figsize=(12, 6))
     plt.plot(linear_loss, label='Linear Autoencoder Loss')
     plt.plot(sigmoid_loss, label='Sigmoid Autoencoder Loss')
     plt.plot(relu_loss, label='ReLU Autoencoder Loss')
     plt.axhline(y=pca_loss, color='r', linestyle='--', label='PCA Loss')
     plt.title('Loss Comparison of Different Encoders')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```



OBSERVATION:

The plot shows that the linear autoencoder outperforms the others, achieving the lowest loss, closely matching PCA's baseline. The ReLU autoencoder also performs well but has a slightly higher final loss. In contrast, the sigmoid autoencoder struggles the most, showing the slowest decrease in loss and stabilizing at the highest value. PCA, represented by the red dashed line, sets a reference loss, and the linear autoencoder performs comparably to it, while the non-linear activations (ReLU and sigmoid) perform less effectively in this case.

CONCLUSION:

This code compares the performance of autoencoders with different activation functions (linear, sigmoid, and ReLU) against PCA by evaluating their reconstruction loss on the Iris dataset. The PCA model serves as a baseline, while the autoencoders are trained using mean squared error loss. From the plotted results, the performance of each activation function can be observed over 100 epochs, showing how well the models reconstruct the data. The comparison helps illustrate the effectiveness of non-linear activation functions like ReLU and sigmoid in capturing complex patterns in the data, as opposed to the linear activation used in PCA.