Q-1 Explain the architecture and working of autoencoders, including their key variants such as sparse, denoising, and variational Autoencoders, highlighting their applications. implement a basic autoencoder using Tensorflow or Pytorch, and demonstrate its use for dimensionality reduction or data generation with a sample dataset.

1. Autoencoder

An **autoencoder** is a type of **unsupervised neural network** designed to learn efficient data representations (called *encodings*) by reconstructing the input data as accurately as possible.

It consists of two main parts:

- **Encoder**: Compresses the input data into a lower-dimensional *latent representation* (bottleneck).
- **Decoder**: Reconstructs the original data from this latent representation.

---

2. Architecture of Autoencoders

```
Input → Encoder → Latent Space → Decoder → Output
```

Component	Description
Encoder	Transforms input data xxx into a lower-dimensional latent vector zzz.
Latent Space (Bottleneck)	The compressed representation capturing key features of input data.
Decoder	Reconstructs the original input x^\hat{x}x^ from zzz.
Loss Function	Measures how well the reconstructed data matches the original. Common: Mean Squared Error (MSE).

---

3. Working Principle

Training minimizes **reconstruction loss**:

L(x,x^)=||x−x^||2L(x, \hat{x}) = ||x - \hat{x}||^2L(x,x^)=||x−x^||2

- The encoder learns to capture essential features.
- The decoder learns to reconstruct inputs from those features.

After training:

- The **encoder** can be used for **dimensionality reduction**.
- The **decoder** can be used for **data generation**.

---

4. Key Variants of Autoencoders

Variant	Key Idea	Benefit / Application
Sparse Autoencoder	Adds sparsity constraint (forces only a few neurons to activate).	Learns more meaningful, interpretable features.
Denoising Autoencoder (DAE)	Trains on noisy inputs but tries to reconstruct the clean version.	Robust feature learning, used in noise removal.
Variational Autoencoder (VAE)	Learns a *probabilistic* latent space (with mean & variance).	Can generate new, realistic samples (e.g., new images).

(a) Sparse Autoencoder

Adds a regularization term to the loss function that encourages sparsity in the hidden units:

L=||x−x^||2+β∑KL(ρ||ρj^)L = ||x - \hat{x}||^2 + \beta \sum KL(\rho || \hat{\rho_j})L=||x−x^||2+β∑KL(ρ||ρj^)

Used in **feature extraction** and **anomaly detection**.

(b) Denoising Autoencoder

Input xxx is corrupted with noise x~\tilde{x}x~, but output remains the original xxx.

L=||x−f(x~)||2L = ||x - f(\tilde{x})||^2L=||x−f(x~)||2

Used for **image denoising** and **robust representation learning**.

(c) Variational Autoencoder (VAE)

Encodes data into a *distribution* q(z|x)=N(μ,σ2)q(z|x) = N(\mu, \sigma^2)q(z|x)=N(μ,σ2), not a fixed vector.

Loss:

L=Reconstruction Loss+KL(q(z|x)||p(z))L = \text{Reconstruction Loss} + KL(q(z|x) || p(z))L=Reconstruction Loss+KL(q(z|x)||p(z))

Used in **data generation**, e.g. generating realistic images.

---

5. Applications of Autoencoders

 Dimensionality Reduction (alternative to PCA)
 Denoising Images
 Anomaly Detection
 Data Compression
 Image Generation (VAE, GANs)
 Feature Extraction for downstream ML tasks

---

6. Implementation: Basic Autoencoder (TensorFlow)

We'll demonstrate **dimensionality reduction** using the **MNIST dataset** (handwritten digits).

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Load MNIST data
(x_train, _), (x_test, _) = tf.keras.datasets.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), 28*28))
x_test = x_test.reshape((len(x_test), 28*28))

# Encoder
input_img = tf.keras.Input(shape=(784,))
encoded = layers.Dense(64, activation='relu')(input_img)

# Decoder
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# Autoencoder model
autoencoder = models.Model(input_img, decoded)

# Compile model
autoencoder.compile(optimizer='adam', loss='mse')

# Train
history = autoencoder.fit(x_train, x_train,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# Encode and reconstruct
encoded_imgs = models.Model(input_img, encoded).predict(x_test)
decoded_imgs = autoencoder.predict(x_test)
```

Visualizing Reconstruction

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
```

```
    plt.title("Original")
    plt.axis('off')

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis('off')
plt.show()
```

Dimensionality Reduction Visualization (using t-SNE)

```
from sklearn.manifold import TSNE
import seaborn as sns

# Reduce latent space to 2D for visualization
x_2d = TSNE(n_components=2).fit_transform(encoded_imgs)

plt.figure(figsize=(8,6))
sns.scatterplot(x=x_2d[:,0], y=x_2d[:,1], s=5)
plt.title("2D representation of encoded MNIST data")
plt.show()
```

---

7. Summary Table

Variant	Key Feature	Use Case
Basic AE	Reconstructs input	Compression, Feature Learning
Sparse AE	Adds sparsity penalty	Interpretable features
Denoising AE	Learns to remove noise	Image denoising
VAE	Learns latent distribution	Data generation