**DL UNIT 4**

1. **Autoencoders**

Autoencoders are a type of artificial neural network that helps compress data into a smaller size and then reconstructs it back to its original form. Think of them like a zip file: they take something big, shrink it into a small file, and later unzip it to get it back.

They are used in **unsupervised learning**, which means they don't need labels or specific answers in the data. Instead, they work on raw data to find patterns.

**How Do Autoencoders Work?**

Autoencoders have two main parts:

1. **Encoder**: This is like a "compressor." It reduces the input into a smaller-sized summary (called **code** or **latent representation**).
2. **Decoder**: This is like an "unzipper." It takes the compressed summary and tries to recreate the original input.

The network trains itself to make the output as close as possible to the input.

**Why Use Autoencoders?**

They are good for:

1. **Dimensionality Reduction**: Shrinking big data into a smaller, more manageable size without losing important information.
2. **Denoising**: Cleaning up noisy data like blurry or corrupted images.
3. **Finding Anomalies**: Spotting unusual things in data, like detecting faulty sensors or fraud.

**Simple Example**

Imagine you give an autoencoder an image of a cat.

* The **encoder** compresses this image into a small, simplified version, capturing only the key features (like the shape of ears and eyes).
* The **decoder** takes this compressed version and recreates the image of the cat.

Even though the recreated image may not be 100% perfect, it will look similar because the network learns the most important parts of the data.

For instance, if you give a noisy, blurry cat image to a **denoising autoencoder**, it can clean up the noise and recreate a clear cat image.

1. **Architecture of Autoencoder**

An autoencoder consists of three main parts:

1. **Encoder**: Compresses the input data.
2. **Code (Latent Space)**: The compact representation of the input.
3. **Decoder**: Reconstructs the original data from the compressed representation.

**How It Works**

1. **Input**: The original data is passed into the network.
2. **Hidden Layers**: The input data is processed through multiple layers.
   * In the **encoder**, the layers reduce the data into a smaller size (latent space / Code).
   * In the **decoder**, the layers expand the data back to its original size.
3. **Output**: The network tries to make the output as similar to the input as possible.

**Detailed Structure**

1. **Encoder**:
   * A function h=f(x)h = f(x) that maps input xx to the code hh.
   * It compresses the data into a smaller dimension (features like edges, shapes, etc.).
2. **Code (Latent Space)**:
   * A middle layer representing the compressed data.
   * The size of this layer is a **hyperparameter** (decided before training).
3. **Decoder**:
   * A function r=g(h)r = g(h) that maps the code hh back to the reconstructed data rr.
   * It expands the compressed data to make it resemble the input.
4. **Symmetry**:
   * The encoder and decoder often have a mirror-like structure, meaning the layers in the encoder shrink data, and the decoder layers expand it.

**Key Hyperparameters**

1. **Code Size**: Number of neurons in the code layer. Smaller size means more compression.
2. **Number of Layers**: How deep the network is (number of encoder and decoder layers).
3. **Number of Nodes Per Layer**: Determines how much compression happens at each layer.
4. **Loss Function**: Used to measure reconstruction error (e.g., Mean Squared Error or Binary Cross-Entropy).

**Diagram**

Input → [Encoder] → Latent Space (Code) → [Decoder] → Output

Example with layers:

Input Layer (784 nodes, e.g., pixels of an image)

↓

Hidden Layer (300 nodes, part of Encoder)

↓

Latent Space (50 nodes, compressed representation)

(Code)

↓

Hidden Layer (300 nodes, part of Decoder)

↓

Output Layer (784 nodes, reconstructed image)

**Example**

Suppose you input a grayscale image of size 28×2828 \times 28 pixels (784 values).

1. **Encoder**: Reduces the 784 values to 50 (latent space).
2. **Latent Space / Code**: The 50 values represent key features (edges, shapes).
3. **Decoder**: Reconstructs the 50 values back to 784, recreating the original image.

Though the reconstructed image might not be identical, it will capture the essence of the original.

1. ***Types of Autoencoders***

Here’s a **detailed explanation** of each of the nine types of autoencoders:

**1. Undercomplete Autoencoder**

* **Definition**: The latent space (**code**) has fewer dimensions than the input data. This forces the autoencoder to learn compressed representations by focusing on key patterns.
* **Objective**: Minimize reconstruction error while ensuring the Code (latent space) captures meaningful features.

Forces the network to focus on the most important features of the data.

* **Features**:
  + Reduces overfitting by constraining capacity.
  + Latent space is smaller than the input, so the model learns the essence of the data.
  + Encodes only critical information.
  + Works well for dimensionality reduction.
* **Applications**:
  + Dimensionality reduction (alternative to PCA).
  + Feature extraction.
* **Example**: Compressing a 1000-pixel image into a 10-number representation, then reconstructing the original image.

**2. Regularized Autoencoder**

* **Definition**: Uses additional constraints during training to avoid trivial solutions (like copying input to output without learning meaningful patterns).

Adds constraints to the learning process to prevent the network from simply copying the input to the output.

It’s like adding rules to ensure the autoencoder doesn’t cheat by just memorizing the data.

* **Types of Regularization**:
  + **Sparse Regularization**: Enforces sparsity in the latent space (explained below in Sparse Autoencoder).
  + **Robustness Regularization**: Encourages robustness to noise or input variations.
  + **Contractive Regularization**: Penalizes large derivatives of activations with respect to inputs (explained in Contractive Autoencoder).
* **Purpose**: Helps learn useful patterns even when the latent space has high capacity.
* **Advantages**:
  + Can have latent space dimensions larger than the input.
  + Prevents overfitting even for complex data.
* **Applications**:
  + Learning robust features in noisy datasets.
* **Example**: Adding penalties like sparsity or contractive terms.

**3. Convolutional Autoencoder (CAE)**

* **Definition**: Replaces fully connected layers with convolutional layers, making the model highly effective for image data.

A special type of autoencoder for image data, using techniques that work well with pictures.

* **Purpose**: ***Captures spatial hierarchies in images*** (e.g., edges, textures).
* **Features**:
  + Preserves spatial relationships in data.
  + Efficient for high-dimensional inputs like images.
* **Architecture**:
  + **Encoder**: Convolutional layers extract spatial features.
  + **Latent Space**: A compressed representation retaining image-specific features.
  + **Decoder**: Uses transposed convolutions (or upsampling) to reconstruct the image.
* **Applications**:
  + Image compression.
  + Image denoising.
  + Super-resolution tasks.
  + Image reconstruction.
* **Example**: Compressing high-definition images while retaining their core structures.

Compressing and reconstructing a picture of a cat while keeping its key features like eyes and whiskers.

**4. Sparse Autoencoder**

* **Definition**: Introduces sparsity by **adding a constraint that forces only a few neurons to activate during encoding.**

Only a few neurons in the middle layer are allowed to work at the same time (makes the network focus on key parts of the data).

* **How It Works**:
  + A penalty term is added to the loss function to ensure sparsity.
  + The network learns to represent data using a small subset of all neurons.
* **Applications**:
  + Feature extraction for downstream tasks (e.g., clustering, classification).
  + Finding sparse representations in signals or images.
* **Example**: Identifying key patterns in text or image data.

**5. Stacked Autoencoder**

* **Definition**: Builds a deep architecture by stacking multiple layers of encoders and decoders.

Multiple autoencoders are stacked on top of each other to build a deep network.

* **How It Works**:
  + Train one layer at a time.
  + The output of one autoencoder becomes the input to the next.
  + To learn complex patterns step by step, layer by layer.
* **Advantages**:
  + Learns hierarchical features (e.g., edges → textures → shapes).
* **Applications**:
  + Pretraining deep neural networks.
  + High-level feature extraction.
* **Example**: Extracting multi-level patterns in image recognition tasks.
* Recognizing a face in a photo by first learning edges, then shapes, then the overall face structure.

**6. Denoising Autoencoder**

* **Definition**: Trains the network to reconstruct the original input from corrupted or noisy input data.
* Reconstructs clean data from a noisy or corrupted input.
* **How It Works**:
  + Input data is deliberately corrupted (e.g., adding noise).
  + The model learns to reconstruct the clean, original data.
* **Applications**:
  + Image denoising.
  + Noise reduction in audio signals.
  + Data recovery.
* **Example**: Removing random pixel noise from a blurry photo.

**7. Variational Autoencoder (VAE)**

* **Definition**: Learns the probability distribution of data and generates new, similar data samples.

A generative autoencoder that learns the underlying probability distribution of the data and generates new samples.

To generate similar but new samples based on the training data.

* **How It Works**:
  + Instead of encoding inputs into fixed values, encodes them as a probability distribution (mean and variance).
  + Samples latent vectors from this distribution during training.
* **Applications**:
  + Generating synthetic(कृत्रिम) images or music.
  + Creating realistic new music or text.
  + Anomaly detection by identifying inputs that don’t fit the learned distribution.
* **Example**: Creating realistic human faces or new fashion designs.

**8. Stochastic Autoencoder**

* **Definition**: Introduces randomness into encoding and decoding, modeling probabilistic relationships between inputs and latent variables.

Adds randomness during the process, like rolling a dice, so it learns to deal with uncertain data.

* **How It Works**:
  + Encoder and decoder are treated as probabilistic models.
  + Often used to approximate distributions for tasks involving uncertainty(अनिश्चितता).
* **Applications**:
  + Uncertainty estimation in predictions.
  + Data generalization.
* **Example**: Predicting multiple possible future outcomes for time-series data.

**9. Contractive Autoencoder (CAE)**

* **Definition**: Adds a penalty to the loss function to ensure the learned representation is robust to small variations in the input.
* **How It Works**:
  + Trains the network to ignore small changes in input and focus on the bigger picture.
  + Minimizes the Frobenius norm of the Jacobian matrix of the encoder’s output with respect to its input.
  + Penalizes large changes in the latent representation for small input variations.
* **Applications**:
  + Learning stable and meaningful representations.
  + Applications in noisy environments.
* **Example**: Ensuring the model outputs consistent representations even with slight lighting changes in an image.