UNIT-4

Neural Network

An Artificial Neural Network (ANN) is a computational model inspired by the human brain. It consists of interconnected nodes (neurons) organized in layers that learn to recognize patterns in data.

These networks are particularly powerful for classification, regression, pattern recognition, and prediction tasks.

Structure of ANN

1. **Input Layer:**
    - Receives input data (features).
    - No computation happens here.
2. **Hidden Layers:**
    - Perform transformations using weights and activation functions.
    - The number of hidden layers depends on problem complexity.
3. **Output Layer:**
    - Produces the final result (e.g., class probabilities, numeric value).

Mathematical Model of a Neuron

Each neuron receives inputs and produces output as:

$$z = \sum_{i=1}^{n} w_i x_i + b$$
$$a = f(z)$$

Where:

- x_i: inputs
- w_i: weights
- b: bias
- f: activation function
- a: output of neuron

Activation Functions

Used to introduce non-linearity into the network.

Function	Formula	Range	Comments
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	(0,1)	Smooth, good for probabilities
Tanh	$f(x) = \tanh(x)$	(-1,1)	Centered at 0
ReLU	$f(x) = \max(0, x)$	(0,∞)	Fast, avoids vanishing gradients
Leaky ReLU	$f(x) = \max(0.01x, x)$		Prevents dead neurons
Softmax	$f_i(x) = \frac{e^{x_i}}{\sum e^{x_j}}$	(0,1)	Multi-clas

Objective of ANN Training

The aim is to find a set of weights that minimize the prediction error.
This is done through training, i.e., learning from examples.

Fitting Neural Networks (Training Process)

The fitting process involves adjusting weights using a learning algorithm.
The most popular method is Gradient Descent with Backpropagation.

Steps of Fitting a Neural Network

1. **Initialization:**
   Randomly assign weights and biases.
2. **Forward Propagation:**
   Inputs → Hidden layers → Output.
   Compute output using activation functions.
3. **Compute Loss:**
   Loss (Error) = Difference between actual output $y$and predicted output $\hat{y}$.
   Example:
   - MSE = $\frac{1}{n}\sum(y-\hat{y})^2$
   - Cross-Entropy = $-\sum y\log(\hat{y})$

4. **Backward Propagation:**
   Calculate gradients of loss w.r.t. each weight using chain rule.
5. **Update Weights:**
   Using Gradient Descent:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

where η= learning rate.

6. **Repeat:**
   Continue until convergence (loss becomes small or fixed).

Backpropagation Algorithm

The **Backpropagation** algorithm is the heart of neural network training.
It allows efficient computation of gradients for each weight.

Algorithm Steps

1. **Forward Pass:**
   Compute network output $\hat{y}$for given input.
2. **Compute Error:**

$$E = \frac{1}{2}(y - \hat{y})^2$$

3. **Backward Pass:**
   Compute partial derivatives $\frac{\partial E}{\partial w}$for each weight.
4. **Update Weights:**

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial E}{\partial w}$$

Advantages of Backpropagation

- Efficient and simple to implement.
- Works for any differentiable activation function.
- Adaptable to deep networks.

Limitations

- Can get stuck in local minima.
- Slow for large datasets.
- Sensitive to learning rate.
- Suffers from vanishing gradient in deep layers.

Issues in Training Neural Networks

Issue	**Description**	**Solution**
Overfitting	Learns noise from training data	Dropout, regularization
Underfitting	Too simple model	Increase layers/neurons
Vanishing Gradient	Gradient becomes too small	Use ReLU, normalization
Exploding Gradient	Gradients blow up	Gradient clipping
Local Minima	Gets stuck in suboptimal solution	Momentum, Adam optimizer
Training Time	Too many parameters	Mini-batch training, GPUs

Support Vector Machines (SVM)

Introduction

SVM is a supervised learning algorithm that classifies data by finding an optimal hyperplane separating different classes with maximum margin.

Basic Concept

For 2D data, the decision boundary is a line.
For 3D → plane,
For higher dimensions → hyperplane.

$$w \cdot x + b = 0$$

Maximizing the Margin

Margin = distance between the hyperplane and closest data points (support vectors).
SVM aims to maximize this margin.

$$\text{Margin} = \frac{2}{|| w ||}$$

Subject to:

$$y_i(w \cdot x_i + b) \geq 1$$

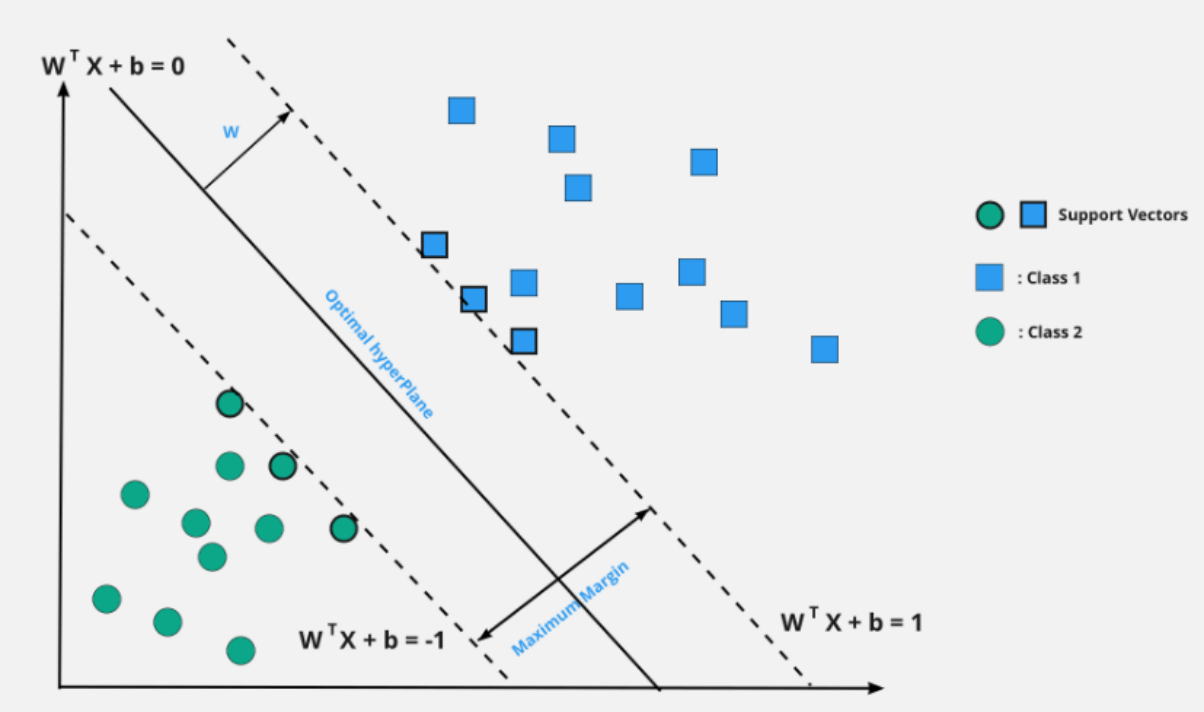
Soft Margin SVM

Real-world data may have overlapping classes.
We introduce **slack variables** ξ_i and **penalty parameter** C:

$$\text{Minimize } \frac{1}{2} || w ||^2 + C\sum\xi_i$$

Subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0$$



Reproducing Kernels (Kernel Trick)

Some data is **not linearly separable**.
Instead of drawing a complex boundary in 2D, SVM uses a **kernel function** to map data into a **higher-dimensional space** where it becomes linearly separable.

Common Kernels

Kernel	**Application**
Linear	Linearly separable data
Polynomial	Curved boundaries
RBF (Gaussian)	
Sigmoid	Neural-net type problems

SVM for Regression (SVR)

Support Vector Regression applies SVM principles to predict **continuous values**.

Key Idea:

Instead of classifying, SVR tries to fit the best line that lies within an **ε-tube** around the data points.

Only points outside the tube affect the model (support vectors).

$$| y_i - (w \cdot x_i + b) | \leq \epsilon$$

If the deviation is > ε, a penalty is applied.

Advantages of SVM

- Works well for both linear and non-linear data.
- High accuracy.
- Effective in high-dimensional spaces.
- Robust against overfitting.

Disadvantages

- Computationally expensive for large datasets.
- Requires careful choice of kernel and parameters.
- Not suitable for noisy data.

K-Nearest Neighbour (KNN)

KNN is a lazy learning, non-parametric, supervised algorithm.
It does not build a model — instead, it memorizes training examples.

When a new data point comes, it classifies it by majority vote among its K nearest neighbors.

Algorithm Steps

1. Choose K (number of neighbors).
2. Compute distance between new point and all training points.
    - Euclidean: $d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$
3. Select K smallest distances.
4. Assign the most frequent class among neighbors.

Choosing K

- Small K → Sensitive to noise, overfitting
- Large K → Smoother, may underfit
- Usually odd K used to avoid ties.

KNN for Image Scene Classification

In image classification:

1. Each image is converted into a feature vector (color, texture, edges, etc.)
2. Given a test image:
    - Compute distance from all training images.
    - Select K nearest ones.
    - Assign label based on the majority class.

Example:
If 3 nearest neighbors = [Beach, Beach, Forest] → predicted class = "Beach".

Distance metrics

Euclidean Distance

Formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Explanation:

- Measures the straight-line distance between two points in n-dimensional space.
- Most commonly used distance metric.
- Sensitive to scale of data, so normalization may be required.

Use case:

- Continuous numerical data, e.g., image pixel values, height-weight datasets.

Manhattan Distance (L1 Distance / City Block Distance)

Formula:

$$d(x, y) = \sum_{i=1}^{n} \quad | x_i - y_i |$$

Explanation:

- Measures distance if movement is restricted to grid-like paths (like city blocks).
- Sum of absolute differences of coordinates.

Use case:

- Grid-based problems, e.g., robotics, warehouse navigation, categorical data after encoding.

Advantages

- Simple, intuitive, and non-parametric.
- No training time.
- Effective for small datasets.

Limitations

- High computational cost for large data.
- Sensitive to irrelevant features.
- Requires normalization.
- Poor performance in high dimensions (curse of dimensionality).