ML THEORY AND ALGORITHM

Experiment 1: Linear Regression

Theory:
Linear Regression is a foundational algorithm in supervised learning, used to predict a **continuous dependent variable** based on one or more independent variables. It assumes a linear relationship, where the change in the dependent variable is directly proportional to the change in the independent variables. The primary goal is to find the **best-fitting straight line** (regression line) that minimizes the difference between actual values and predicted values. This difference is quantified using the **Mean Squared Error (MSE)**. Linear regression is widely used in domains like economics (predicting income), healthcare (predicting disease progression), and business (forecasting sales).

Algorithm:

1. Initialize weights (w) and bias (b).
2. Predict: $\hat{y} = wX + b$
3. Compute loss: $MSE = \frac{1}{n}\sum(y - \hat{y})^2$
4. Update weights using Gradient Descent:
   - $w = w - \alpha \cdot \frac{\partial MSE}{\partial w}$
   - $b = b - \alpha \cdot \frac{\partial MSE}{\partial b}$
5. Repeat until convergence.

Experiment 2: Logistic Regression

Theory:
Logistic Regression is a classification algorithm used to predict **binary outcomes** (e.g., yes/no, 0/1). Unlike linear regression, it uses the **logistic (sigmoid) function** to map predicted values to probabilities between 0 and 1. It estimates the probability that a given input point belongs to a certain class. The decision boundary is determined using a threshold (commonly 0.5). Logistic regression is interpretable and efficient, making it ideal for applications such as spam detection, medical diagnosis, and customer churn prediction.

Algorithm:

1. Initialize weights.
2. Predict: $\hat{y} = \sigma(wX + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$
3. Compute binary cross-entropy loss.
4. Update weights using Gradient Descent.
5. Repeat until convergence.

Experiment 3: Support Vector Machine (SVM)

Theory:
SVM is a powerful supervised learning algorithm used for both **classification and regression**. It works by finding the **optimal hyperplane** that separates data points of different classes with the **maximum margin**. The data points closest to the hyperplane are called **support vectors**, and they define the position and orientation of the hyperplane. For non-linearly separable data, SVM uses kernel functions (like RBF, polynomial) to transform the input space into a higher-dimensional space where a linear separator can be found. SVMs are effective in high-dimensional spaces and used in applications like image classification, text categorization, and bioinformatics.

Algorithm (Linear SVM):

1. Initialize weights and bias.
2. For each data point:
    - If correctly classified with margin: do nothing.
    - Else update:
        - $w = w + \alpha \cdot (y_i \cdot x_i)$
        - $b = b + \alpha \cdot y_i$
3. Repeat until convergence.

Experiment 4: Hebbian Learning Rule

Theory:
Hebbian learning is an unsupervised learning rule inspired by the way biological neurons learn. It is based on the principle: *"neurons that fire together, wire together."* When an input and its corresponding output neuron are activated simultaneously, the connection between them is strengthened. This rule is one of the earliest models of learning and is primarily used in **associative learning**. Although simple, Hebbian learning provides the theoretical foundation for several neural learning techniques and is often used in **pattern recognition and cognitive models**.

$$w_{ij} = w_{ij} + \eta \cdot x_i \cdot y_j$$

Where:

- w_{ij}: weight from input neuron i to output neuron j
- η: learning rate (a small positive constant)
- x_i: input from neuron i
- y_j: output of neuron j

Steps:

1. Initialize the weights w_{ij} to small random values or zeros.
2. For each input pattern x:
    - Compute the output y_j
    - Update each weight using:

$$w_{ij} = w_{ij} + \eta \cdot x_i \cdot y_j$$

3. Repeat for all training samples or until convergence.

Experiment 5: Expectation Maximization (EM)

Theory:
The Expectation-Maximization (EM) algorithm is used for finding **maximum likelihood estimates** in models with **latent (hidden) variables**. It is particularly useful when the data is incomplete or has missing labels. The algorithm alternates between the **E-step**, where it estimates the hidden variables based on current parameters, and the **M-step**, where it updates the model parameters to maximize the expected likelihood. EM is widely applied in clustering problems (e.g., **Gaussian Mixture Models**), computer vision, and natural language processing.

Algorithm:

1. Initialize parameters randomly.
2. **E-Step:** Estimate hidden variables (posterior probabilities).
3. **M-Step:** Maximize the likelihood to update parameters.
4. Repeat E and M steps until convergence.

Experiment 6: McCulloch-Pitts Model

Theory:
The McCulloch-Pitts model is one of the earliest models of an **artificial neuron**, proposed in 1943. It is a binary threshold logic unit, which outputs 1 if the weighted sum of its inputs exceeds a predefined threshold and 0 otherwise. The model is simplistic—it only works with binary inputs and does not involve learning or weight updates. Despite its simplicity, the model laid the groundwork for the development of neural networks. It is a useful tool to understand basic computation in neural architectures and boolean logic functions.

Algorithm:

1. Inputs are binary.
2. Compute: $net = \sum w_i x_i$
3. Output:
   - If $net \geq \theta$: output = 1
   - Else output = 0

Experiment 7: Single Layer Perceptron

Theory:
A Single Layer Perceptron is a type of **feedforward neural network** that can only learn **linearly separable functions**. It consists of a single layer of output nodes connected to input features with weights. The perceptron uses an activation function (typically a step function) to produce binary outputs. During training, weights are adjusted based on the **error between predicted and actual outputs** using a simple update rule. While perceptrons are limited in capacity, they serve as the foundation for more complex models like multilayer perceptrons (MLPs).

1. Initialize weights and threshold.
2. Predict output using activation:
   - $y = 1$ if $w \cdot x \geq \theta$, else 0
3. Update weights:
   - $w_i = w_i + \eta \cdot (y - \hat{y}) \cdot x_i$
4. Repeat for all samples.

Experiment 8: Backpropagation

Theory:
Backpropagation is the core algorithm behind training **multilayer neural networks**. It uses the **chain rule of calculus** to compute gradients of the loss function with respect to weights in the network. During training, the algorithm performs a **forward pass** to compute the output and loss, then a **backward pass** to propagate the error and update weights using **gradient descent**. Backpropagation allows deep networks to learn complex patterns in data, and it's the basis for modern deep learning applications in fields like speech recognition, computer vision, and NLP.

Algorithm:

1. Forward pass: Compute activations.
2. Compute error at output.
3. Backward pass: Compute gradients using chain rule.
4. Update weights:
   - $w = w - \eta \cdot \frac{\partial E}{\partial w}$
5. Repeat until convergence.

Experiment 9: Principal Component Analysis (PCA)

Theory:
PCA is a **dimensionality reduction** technique used to compress data while preserving its most important structures. It identifies the directions (**principal components**) along which the data varies the most and projects the data onto these new axes. PCA helps in reducing computational complexity, removing multicollinearity, and

visualizing high-dimensional data. It is commonly used in exploratory data analysis, face recognition, and preprocessing before machine learning modeling.

Algorithm:

1. Standardize the data.
2. Compute covariance matrix.
3. Compute eigenvalues and eigenvectors.
4. Sort and select top-k eigenvectors.
5. Project data onto these eigenvectors.

Experiment 10: MNIST Digit Classification

Theory:
The MNIST dataset is a classic benchmark in machine learning, consisting of **70,000 images of handwritten digits (0–9)**. Each image is 28x28 pixels. Classification involves building a model—commonly a neural network—that learns to map pixel values to digit labels. The task requires good **image processing and pattern recognition** ability. Using models like CNNs or dense neural networks, MNIST serves as a great starting point to understand image classification and evaluate model performance using metrics like accuracy and confusion matrix.

Typical Steps (using a neural network):

1. Normalize input images.
2. Define model architecture (e.g., input → hidden → output layer).
3. Train using backpropagation.
4. Evaluate accuracy on test set.