Implementation of Classifier using Error Backpropagation Algorithm

Aim: To implement and analyze a feed-forward neural network classifier using the Error Back Propagation Algorithm (EBPA) for pattern classification tasks.

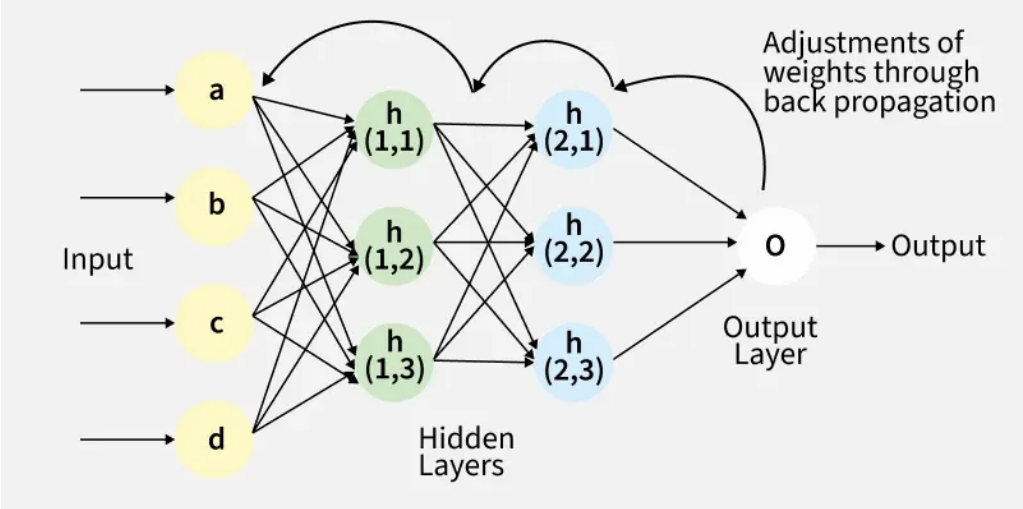
Problem Definition: Design and implement a **multi-layer feed-forward neural network (MLFFNN)** that can classify input data into multiple classes using supervised learning. The network should:

Take a set of labeled input patterns (training data).
Adjust its weights using the **Error Back Propagation Algorithm (EBPA)**.
Minimize the error between the predicted output and the actual target output.
Successfully classify unseen test data after training.
Classifying datasets such as **Iris**, **Wine**, or **MNIST** (for advanced implementation).

Theory: A **classifier** is a model that assigns an input vector x to one of several classes C1,C2,...,Ck. In neural networks, classification is achieved by adjusting the connection weights based on training examples so that the output neurons produce high activation for the correct class and low for others.

A **feed-forward neural network (FFNN)** consists of:

Input Layer – receives input features.
Hidden Layer(s) – performs weighted computation and applies nonlinear activation.
Output Layer – produces the classification result.



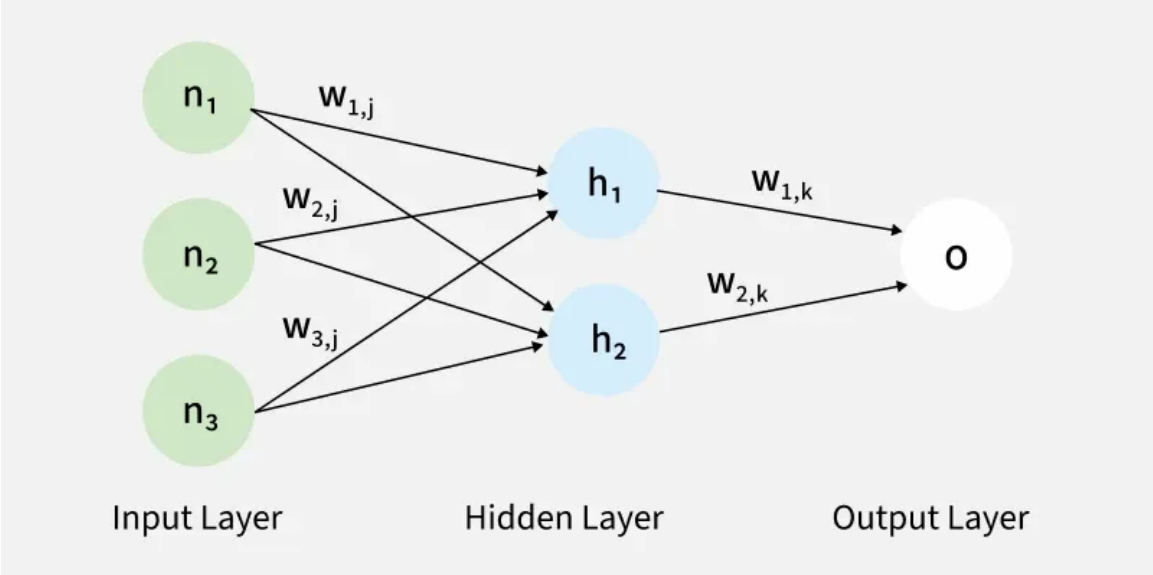
EBPA is a supervised learning algorithm used for training **multi-layer perceptrons (MLPs)**. It minimizes the mean squared error between the network's predicted output and the desired output.

The Back Propagation algorithm involves two main steps: the Forward Pass and the Backward Pass.

1. Forward Pass Work

In forward pass the input data is fed into the input layer. These inputs combined with their respective weights are passed to hidden layers. For example in a network with two hidden layers (h1 and h2) the output from h1 serves as the input to h2. Before applying an activation function, a bias is added to the weighted inputs.

Each hidden layer computes the weighted sum (`a`) of the inputs then applies an activation function like ReLU (Rectified Linear Unit) to obtain the output (`o`). The output is passed to the next layer where an activation function such as softmax converts the weighted outputs into probabilities for classification.



2. Backward Pass

In the backward pass the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases. One common method for error calculation is the Mean Squared Error (MSE) given by:

$$MSE=(\text{Predicted Output}-\text{Actual Output})^2$$

Once the error is calculated the network adjusts weights using gradients which are computed with the chain rule. These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration. The backward pass continues layer by layer ensuring that the network learns and improves its performance. The activation function through its derivative plays a crucial role in computing these gradients during Back Propagation.

Advantages

The key benefits of using the Back Propagation algorithm are:

1. Ease of Implementation: Back Propagation is beginner-friendly requiring no prior neural network knowledge and simplifies programming by adjusting weights with error derivatives.
2. Simplicity and Flexibility: Its straightforward design suits a range of tasks from basic feedforward to complex convolutional or recurrent networks.
3. Efficiency: Back Propagation accelerates learning by directly updating weights based on error especially in deep networks.
4. Generalization: It helps models generalize well to new data improving prediction accuracy on unseen examples.

5. Scalability: The algorithm scales efficiently with larger datasets and more complex networks making it ideal for large-scale tasks.

Challenges

While Back Propagation is useful it does face some challenges:

1. Vanishing Gradient Problem: In deep networks the gradients can become very small during Back Propagation making it difficult for the network to learn. This is common when using activation functions like sigmoid or tanh.
2. Exploding Gradients: The gradients can also become excessively large causing the network to diverge during training.
3. Overfitting: If the network is too complex it might memorize the training data instead of learning general patterns.