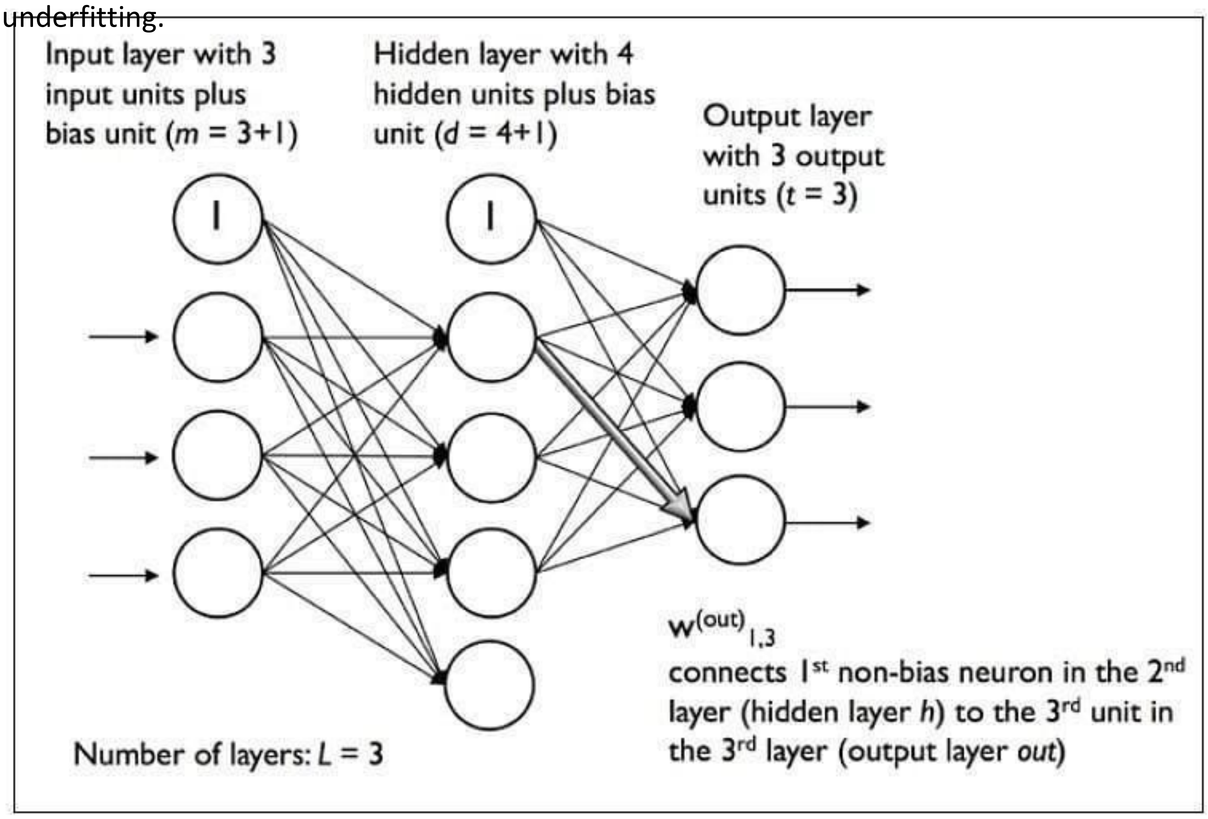
Experiment No. 2
Implement Multilayer Perceptron algorithm to simulate XOR gate
Date of Performance:
Date of Submission:

Aim: Implement Multilayer Perceptron algorithm to simulate XOR gate.

Objective: Ability to perform experiments on different architectures of multilayer perceptrons.

Theory:

Multilayer artificial neuron networks are an integral part of deep learning. And this lesson will help you with an overview of multilayer ANN along with overfitting and underfitting.


Input layer with 3 input units plus bias unit (m = 3+1)
Hidden layer with 4 hidden units plus bias unit (d = 4+1)
Output layer with 3 output units (t = 3)
1
1
$w^{(out)}_{1,3}$
connects 1st non-bias neuron in the 2nd layer (hidden layer h) to the 3rd unit in the 3rd layer (output layer out)
Number of layers: L = 3

A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP).

At has 3 layers including one hidden layer. If it has more than 1 hidden layer, it is called a deep ANN. An MLP is a typical example of a feedforward artificial neural network. In this figure, the ith activation unit in the lth layer is denoted as ai(l).

The number of layers and the number of neurons are referred to as hyperparameters of a neural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these.

The weight adjustment training is done via backpropagation. Deeper neural networks are better at processing data. However, deeper layers can lead to vanishing gradient problems. Special algorithms are required to solve this issue.

A multilayer perceptron (MLP) is a feed forward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers. MLP uses backpropagation for training the network. MLP is a deep learning method.

Conclusion:

1. Neural Networks (NNs)

Overview:
A neural network is a computational model inspired by the structure and functioning of the human brain. It consists of layers of interconnected nodes (neurons):

Input Layer: Accepts features from the dataset.

Hidden Layers: One or more layers that perform transformations on inputs using weights, biases, and activation functions.

Output Layer: Produces the final prediction or classification.

Key Concepts:

Weights & Biases: Parameters that control the strength of the connection between neurons.

Activation Functions: Introduce non-linearity to the model (e.g., ReLU, Sigmoid, Tanh).

Forward Pass: Input is propagated through the network to generate output.

Merits:

Can approximate complex non-linear functions.

Flexible for various tasks (classification, regression, image processing, NLP, etc.).

Demerits:

Requires large datasets to perform well.

Prone to overfitting if network is too complex.

Computationally expensive, especially deep networks.

2. Backpropagation Algorithm

Overview:
Backpropagation (short for backward propagation of errors) is the key algorithm used to train neural networks by minimizing the loss function.

Steps in Backpropagation:

Forward Pass: Compute the network output for a given input using current weights.

Compute Loss: Calculate the error (difference between predicted output and actual output) using a loss function (e.g., Mean Squared Error, Cross-Entropy).

Backward Pass (Gradient Calculation):

Use chain rule of calculus to compute gradients of the loss with respect to each weight in the network.

This tells us how much each weight contributed to the error.

Weight Update:

Repeat for multiple epochs until the network converges to minimal loss.

Merits:

Efficient method to train multi-layer neural networks.

Works well with differentiable activation functions.

Foundation for most deep learning frameworks (TensorFlow, PyTorch).

Demerits / Challenges:

Vanishing/Exploding Gradients: In very deep networks, gradients can become extremely small or large, making training difficult.

Local Minima / Saddle Points: Gradient descent can sometimes get stuck in suboptimal solutions.

Computational Cost: Backpropagation can be expensive for large networks.

Overfitting: Needs regularization (dropout, L2 norm) and sufficient data to generalize.

Improvements / Solutions:

Use of ReLU activation to mitigate vanishing gradient.

Batch normalization to stabilize training.

Adaptive optimizers (Adam, RMSprop) to improve convergence.

Residual networks (ResNets) to allow training of very deep networks.

CSL701: Deep Learning Lab