SNS COLLEGE OF TECHNOLOGY

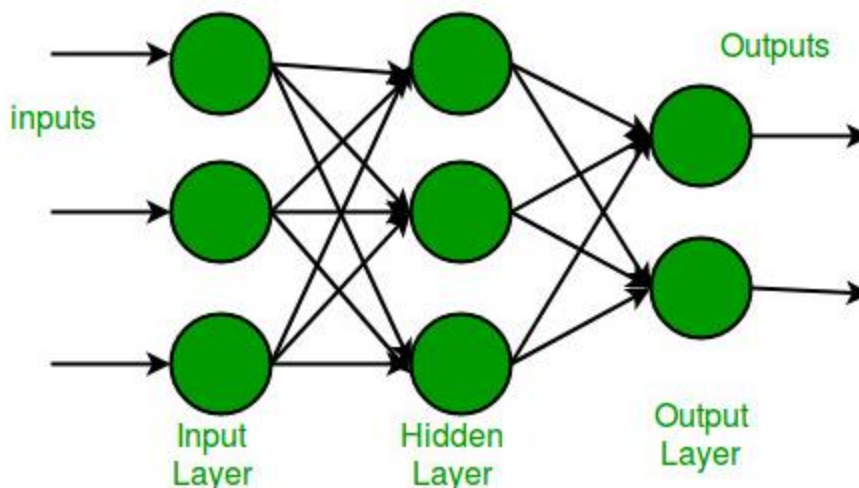
19ECT303 – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

UNIT- 4 –NEURON & NEURAL NETWORKS

MULTILAYER PERCEPTRON

Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perception is a neural network that has multiple layers. To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.

A multi-layer perceptron has one input layer and for each input, there is one neuron(or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes. A schematic diagram of a Multi-Layer Perceptron (MLP) is depicted below.



In the multi-layer perceptron diagram above, we can see that there are three inputs and thus three input nodes and the hidden layer has three nodes. The output layer gives two outputs, therefore there are two output nodes. The nodes in the input layer take input and forward it for further process, in the diagram above the nodes in the input layer forwards their output to each of the three nodes in the hidden layer, and in the same way, the hidden layer processes the information and passes it to the output layer.

Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.

Input layer

The input layer consists of nodes or neurons that receive the initial input data. Each neuron represents a feature or dimension of the input data. The number of neurons in the input layer is determined by the dimensionality of the input data.

Hidden layer

Between the input and output layers, there can be one or more layers of neurons. Each neuron in a hidden layer receives inputs from all neurons in the previous layer (either the input layer or another hidden layer) and produces an output that is passed to the next layer. The number of hidden layers and the number of neurons in each hidden layer are hyperparameters that need to be determined during the model design phase.

Output layer

This layer consists of neurons that produce the final output of the network. The number of neurons in the output layer depends on the nature of the task. In binary classification, there may be either one or two neurons depending on the activation function and representing the probability of belonging to one class; while in multi-class classification tasks, there can be multiple neurons in the output layer.

Weights

Neurons in adjacent layers are fully connected to each other. Each connection has an associated weight, which determines the strength of the connection. These weights are learned during the training process.

Bias neurons

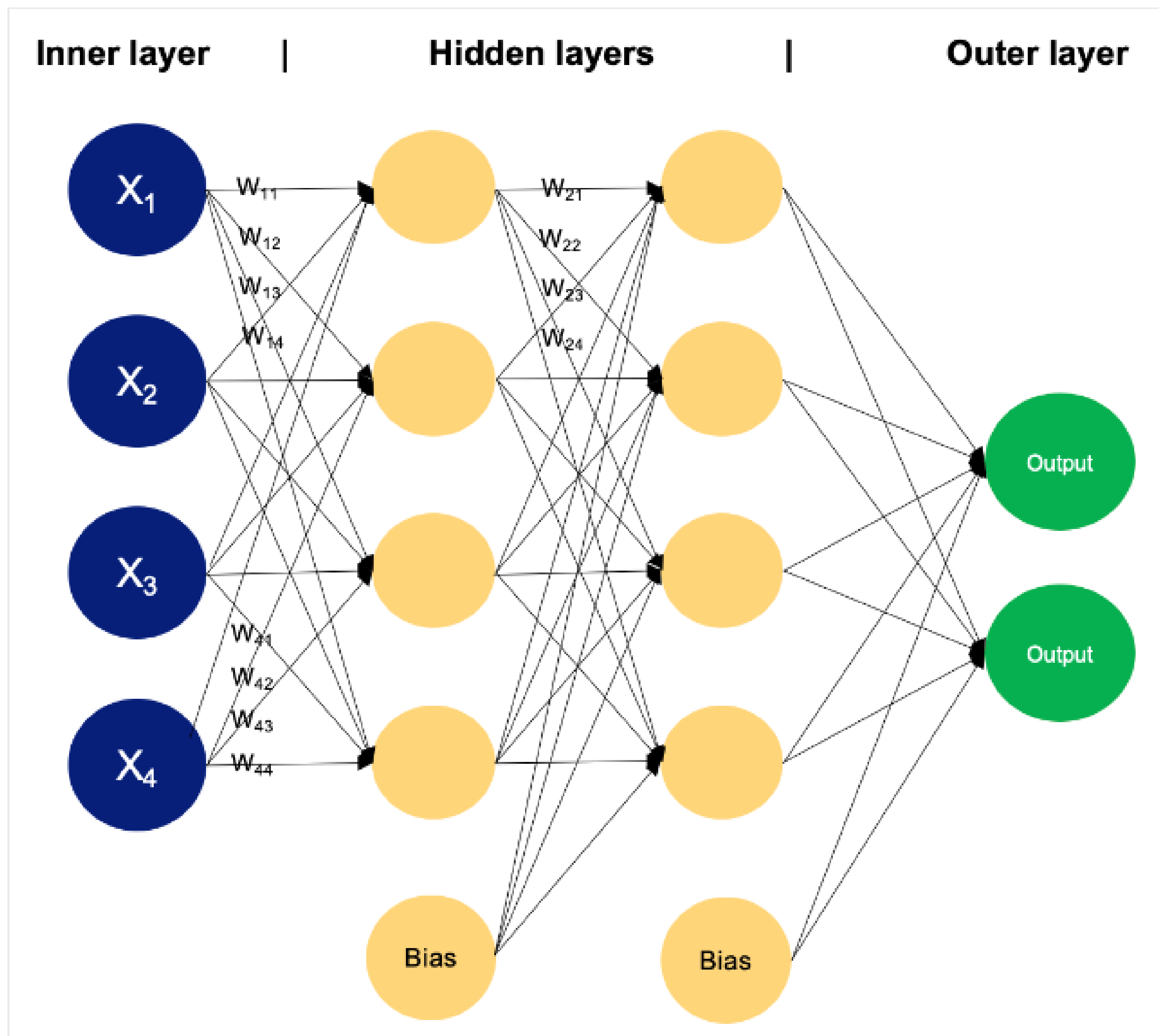
In addition to the input and hidden neurons, each layer (except the input layer) usually includes a bias neuron that provides a constant input to the neurons in the next layer. Bias neurons have their own weight associated with each connection, which is also learned during training.

The bias neuron effectively shifts the activation function of the neurons in the subsequent layer, allowing the network to learn an offset or bias in the decision boundary. By adjusting the weights connected to the bias neuron, the MLP can learn to control the threshold for activation and better fit the training data.

Note: It is important to note that in the context of MLPs, bias can refer to two related but distinct concepts: bias as a general term in machine learning and the bias neuron (defined above). In general machine learning, bias refers to the error introduced by approximating a real-world problem with a simplified model. Bias measures how well the model can capture the underlying patterns in the data. A high bias indicates that the model is too simplistic and may underfit the data, while a low bias suggests that the model is capturing the underlying patterns well.

Activation function

Typically, each neuron in the hidden layers and the output layer applies an activation function to its weighted sum of inputs. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax. These functions introduce nonlinearity into the network, allowing it to learn complex patterns in the data.



Example of an MLP with two hidden layers. Image by Author

General Guidelines for Implementing Multilayer Perceptron

Implementing a MLP involves several steps, from data preprocessing to model training and evaluation. Selecting the number of layers and neurons for a MLP involves balancing model complexity, training time, and generalization performance. There is no one-size-fits-all answer, as the optimal architecture depends on factors such as the complexity of the task,

the amount of available data, and computational resources. However, here are some general guidelines to consider when implementing MLP:

1. Model architecture

- Begin with a simple architecture and gradually increase complexity as needed. Start with a single hidden layer and a small number of neurons, and then experiment with adding more layers and neurons if necessary.

2. Task complexity

- For simple tasks with relatively low complexity, such as binary classification or regression on small datasets, a shallow architecture with fewer layers and neurons may suffice.
- For more complex tasks, such as multi-class classification or regression on high-dimensional data, deeper architectures with more layers and neurons may be necessary to capture intricate patterns in the data.

3. Data preprocessing

- Clean and preprocess your data, including handling missing values, encoding categorical variables, and scaling numerical features.
- Split your data into training, validation, and test sets to evaluate the model's performance.

4. Initialization

- Initialize the weights and biases of your MLP appropriately. Common initialization techniques include random initialization with small weights or using techniques like Xavier or He initialization.

5. Experimentation

- Ultimately, the best approach is to experiment with different architectures, varying the number of layers and neurons, and evaluate their performance empirically.
- Use techniques such as cross-validation and hyperparameter tuning to systematically explore different architectures and find the one that performs best on the task at hand.

6. Training

- Train your MLP using the training data and monitor its performance on the validation set.

- Experiment with different batch sizes, number of epochs, and other hyperparameters to find the optimal training settings.
- Visualize training progress using metrics such as loss and accuracy to diagnose issues and track convergence.

7. Optimization algorithm

- Experiment with different learning rates and consider using techniques like learning rate schedules or adaptive learning rates.

8. Avoid overfitting

- Be cautious not to overfit the model to the training data by introducing unnecessary complexity.
- Use techniques such as regularization (e.g., L1, L2 regularization), dropout, and early stopping to prevent overfitting and improve generalization performance.
- Tune the regularization strength based on the model's performance on the validation set.

9. Model evaluation

- Monitor the model's performance on a separate validation set during training to assess how changes in architecture affect performance.
- Evaluate the trained model on the test set to assess its generalization performance.
- Use metrics such as accuracy, loss, and validation error to evaluate the model's performance and guide architectural decisions.

10. Iterate and experiment

- Experiment with different architectures, hyperparameters, and optimization strategies to improve the model's performance.
- Iterate on your implementation based on insights gained from training and evaluation results.