Lecture 4
Forward Propogation

Contents

1 Forward Propagation

In simple terms, **forward propagation** refers to the *flow of data from input to output* through a neural network. It is the process by which the network processes information and produces an output before any learning (i.e., weight adjustment) takes place.

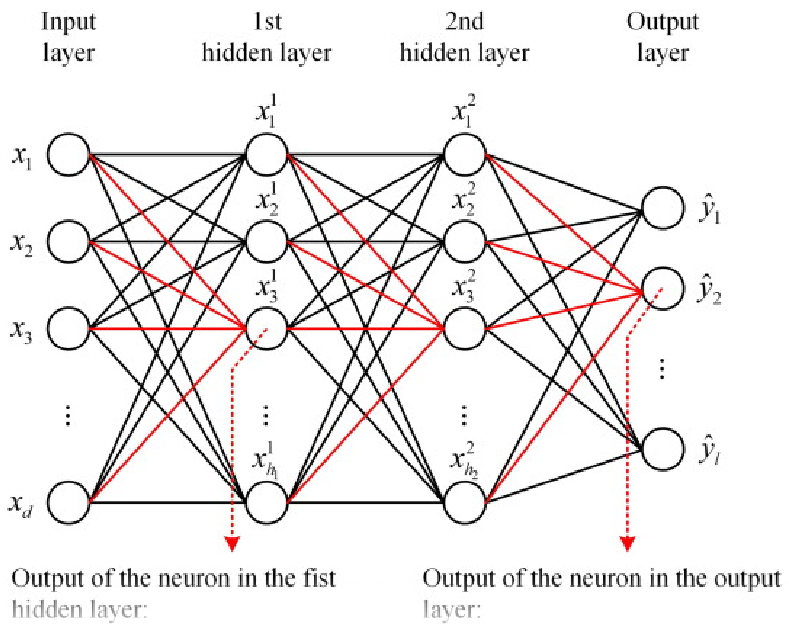

Figure 1: Data flow in forward propagation — from input layer to output layer.

- **Input Reception:** Each neuron (node) in a layer receives inputs from the neurons in the previous layer.
- **Weighted Sum + Bias:** Each input is multiplied by its corresponding *weight*—a value that represents how important that input is—and then a *bias* term is added.
- **Activation Function:** The neuron then applies an *activation function* (such as ReLU, sigmoid, or tanh) to this weighted sum. This introduces non-linearity, enabling the network to learn complex relationships.
- **Propagation to Next Layer:** The output from each neuron becomes the input for the next layer.
- **Output Layer:** This process continues layer by layer until the data reaches the output layer, where the network produces its final prediction (for example, a class label or probability).

2 Weights and Biases

Each connection between neurons has a **weight** (w), which determines the strength or importance of that connection. Similarly, every neuron has an additional parameter called a **bias** (b), which helps the neuron shift the activation function and fit the data better — making the model more flexible and capable of learning complex patterns.

To understand this intuitively, think about a simple linear equation:

$$y = mx$$

This line always passes through the origin (0, 0). But if we add a bias term c:

$$y = mx + c$$

the line shifts up or down depending on the value of c. In a neural network, the same idea applies — the **bias** allows the activation function to move away from the origin, which helps the model learn patterns even when inputs are zero.

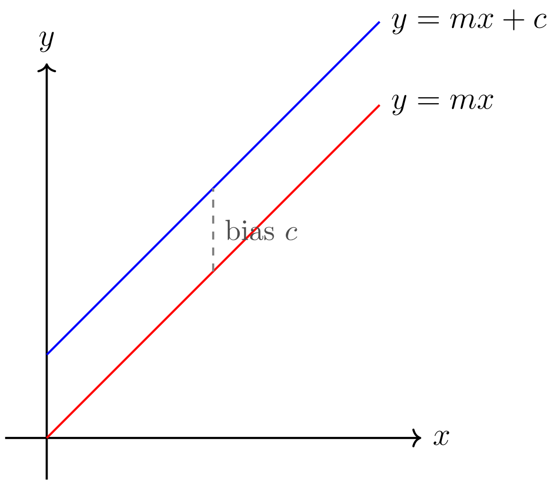

Figure 2: Illustration of how bias shifts a line upward or downward, similar to how bias in neurons shifts the activation function.

In the context of a neuron:

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

where:

- x_i — inputs from previous layer,
- w_i — connection weights,
- b — bias term that shifts the activation,
- $f(z)$ — activation function that introduces non-linearity.

The bias ensures that even if all input values are zero, the neuron can still output a non-zero activation — making it essential for deep networks to model real-world data accurately.

The figure 1 illustrates a fully connected feedforward neural network.

- The **input layer** (left) takes input features $x_1, x_2, \ldots, x_d$.

- The signal passes through two **hidden layers**, where each neuron computes a weighted sum of all inputs from the previous layer, adds a bias, and applies an activation function.
- The final **output layer** produces predictions $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_l$.
- Red arrows highlight the path of forward propagation — how data moves from input to output.

Trainable Parameters

Consider a network with:

- 4 input neurons,
- 1st hidden layer: 4 neurons,
- 2nd hidden layer: 4 neurons,
- Output layer: 3 neurons.

We can calculate the total number of **trainable parameters** as follows:

- **Input → 1st Hidden Layer:** $4 \times 4 = 16$ weights + 4 biases = **20 parameters.**
- **1st Hidden → 2nd Hidden Layer:** $4 \times 4 = 16$ weights + 4 biases = **20 parameters.**
- **2nd Hidden → Output Layer:** $4 \times 3 = 12$ weights + 3 biases = **15 parameters.**

Total trainable parameters:

$$20 + 20 + 15 = \boxed{\text{55 parameters in total.}}$$

This total represents all the weights and biases the model must learn during training — i.e., the parameters updated during backpropagation to minimize loss.