SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Sri Lanka Institute of Information Technology

Faculty of Computing

IT2011 - Artificial Intelligence and Machine Learning

Dr.Lakmini Abeywardhana

Year 02 and Semester 01

Lecture 7

Neural Networks and Deep Learning

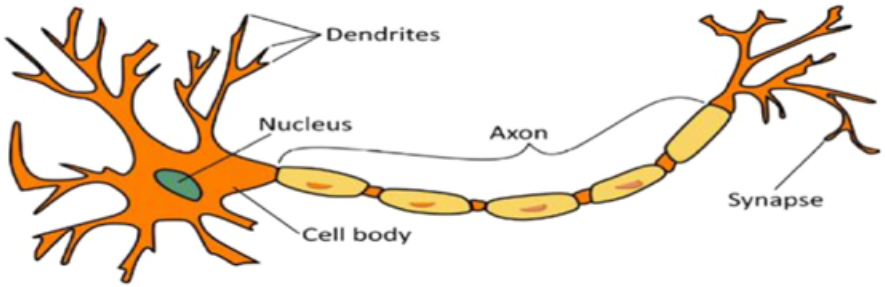
What is Deep Learning?

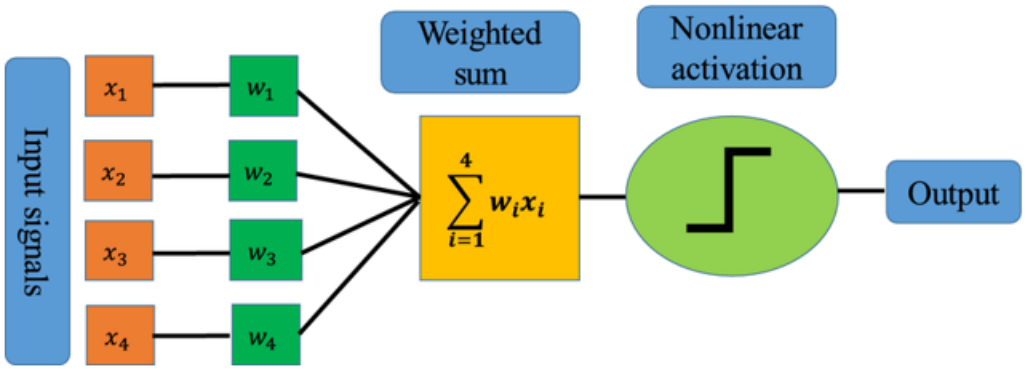
- Deep Learning is a way for computers to learn patterns and make decisions, inspired by how the human brain works.
- **Part of Machine Learning (ML):**
  - Traditional ML $\rightarrow$ Algorithms need you to manually select features (e.g., detecting edges in an image).
  - Deep Learning $\rightarrow$ Learns features automatically (e.g., finds eyes, nose, and face without explicit coding).

- Think of traditional ML as cooking with a recipe (steps are predefined).
- Deep Learning is like a chef who experiments and learns to cook by tasting, improving over time.

Biological Inspiration

- Neural networks are inspired by the brain's structure:
- Neurons receive signals through dendrites, process them, and transmit signals through axons.



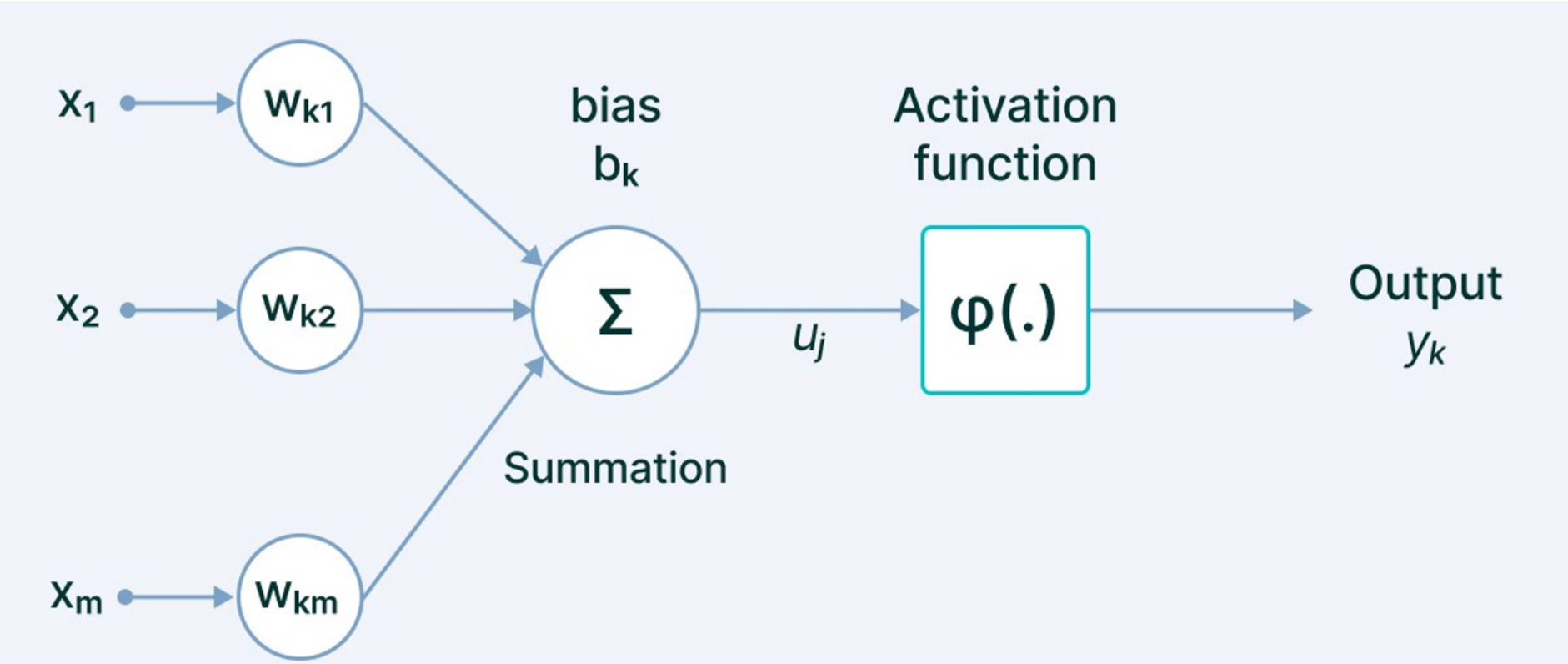


Introduction to Neural Networks

What is a Neural Network?

- A neural network is a computational model inspired by the way biological neural networks in the human brain process information.
- It is made up of layers of interconnected nodes or "neurons."

Neuron


x1
xm
wk1
wk2
wkm
bias
bk
Activation
function
Σ
φ(.)
uj
Output
yk
Summation

Basic Terminology

- **Neuron (Node)**: Smallest processing unit.
- **Layer**:
  - Input Layer $\rightarrow$ raw data enters.
  - Hidden Layer(s) $\rightarrow$ where learning happens.
  - Output Layer $\rightarrow$ final prediction.
- **Weights**: Strength of connection between neurons.
- **Bias**: Adjustment term.
- **Activation Function**: Adds non-linearity (so network can solve complex problems).

Classification Problems

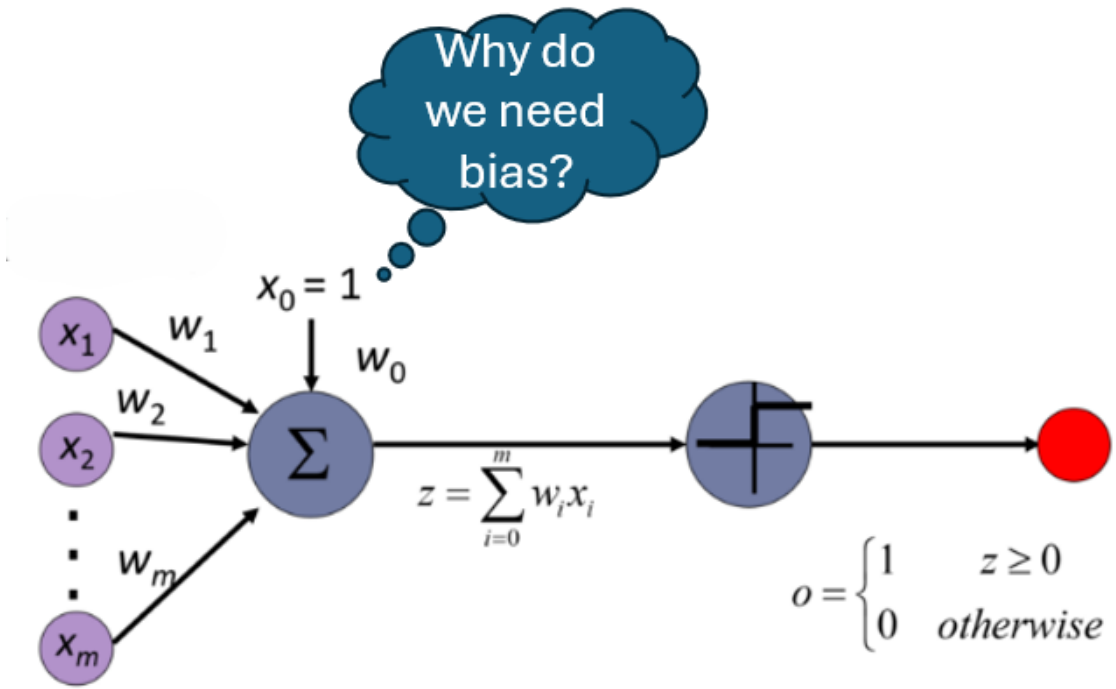
- Solving a classification problem means, identifying to which category a given example belongs.
- Binary classification – two class labels
  - Disease/no disease, spam/not spam, buy/do not buy ...
- Multi-class classification – more than two class labels
  - Hand-written character recognition, Face recognition, ...

Perceptron: The First Neural Network

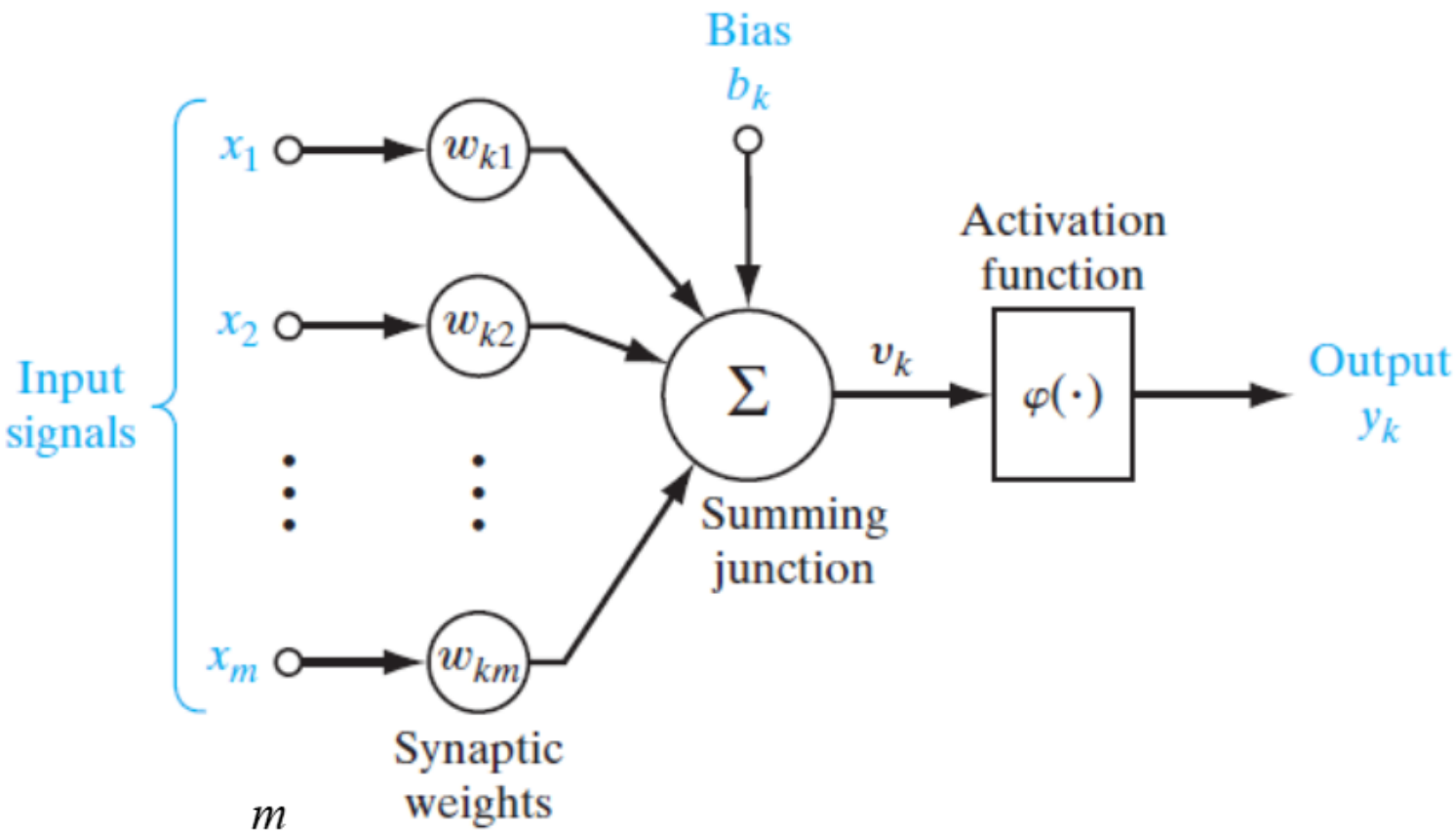
- The perceptron is one of the earliest types of artificial neurons introduced by Frank Rosenblatt in 1958.
- It is a **binary classifier** that determines whether an input belongs to one of two classes.
- A perceptron takes several binary or real-valued inputs, applies weights to them, sums the results, adds a bias, and then passes the result through an activation function (usually a step function).
- **Mathematical Representation**: y = 1 if (w · x + b) 0, else 0

What is a Perceptron?

- A **perceptron** is the simplest form of a neural network.
- It has:
  - **Inputs** (like exam marks, height, etc.)
  - **Weights** (importance of each input)
  - **Bias** (adjusts output up/down)
  - **Activation Function** (decides output: YES/NO)



Model of a neuron

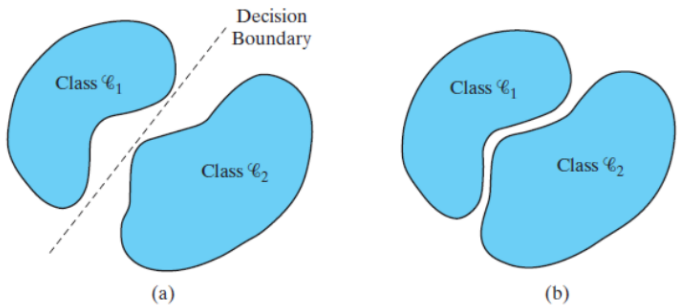


$$u_k = \sum_{j=1}^{m} w_{kj} x_k$$

$$y_k = \phi(u_k + b_k)$$

Perceptron

The perceptron is the simplest form of a neural network used for the classification of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).



•**Mathematical Representation:** y = 1 if (w · x + b) ≥ 0, else 0

- The two classes C_1 and C_2 are sufficiently separated from each other for us to draw a hyperplane (in this case, a straight line) as the decision boundary.
- If, however, the two classes C_1 and C_2 are allowed to move too close to each other, as in (b) above, they become nonlinearly separable, a situation that is beyond the computing capability of the perceptron.

Perceptron

- Perceptron consists of a single neuron with adjustable synaptic weights and bias.
- The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt (1958, 1962) for his perceptron brain model.
- Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes.
- The proof of convergence of the algorithm is known as the **perceptron convergence theorem.**

Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where sgn(·) is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

The Batch Perceptron Algorithm

- The previously discussed perceptron convergence algorithm was presented without reference to a cost function. Moreover, the derivation focused on a single-sample correction.
- Introducing the following two will make it generalized.
  - The generalized form of a perceptron cost function
  - use the cost function to formulate a batch version of the perceptron convergence algorithm

Batch Perceptron Algorithm

- We can define a cost function for the perceptron as follows:

$$J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathcal{X}} (-\mathbf{w}^T \mathbf{x}(n) d(n)) \quad \text{where } \mathcal{X} \text{ is the set of samples } \mathbf{x} \textit{ misclassified}$$

- If all samples are correctly classified, then the set X would be empty and J(w) would be zero.
- Differentiate J(w) with respect to w gives the gradient vector

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathcal{X}} (-\mathbf{x}(n) d(n))$$

Batch Perceptron Algorithm cont.

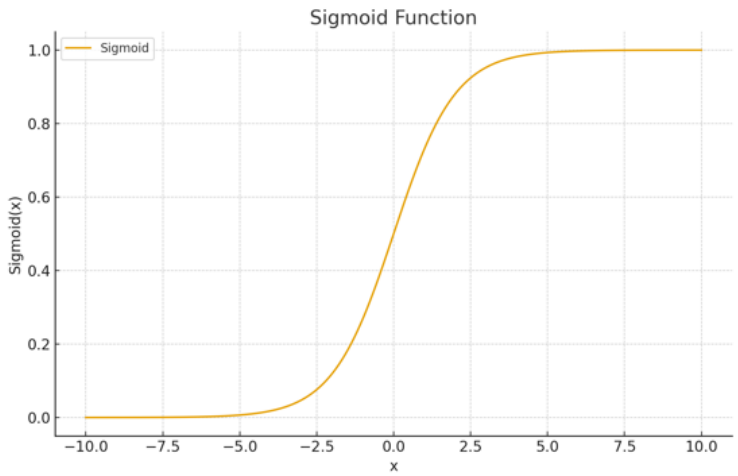
- The weight adjustment method that we discussed above is called the method of steepest descent as the amount of adjustment depends on the steepness of the cost function (i.e. gradient).
- The weight adjustment happens in the direction opposite to the gradient vector.
- The gradient vector is mathematically denoted by $\nabla J(\mathbf{w})$

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathcal{X}} (-\mathbf{x}(n) d(n))$$

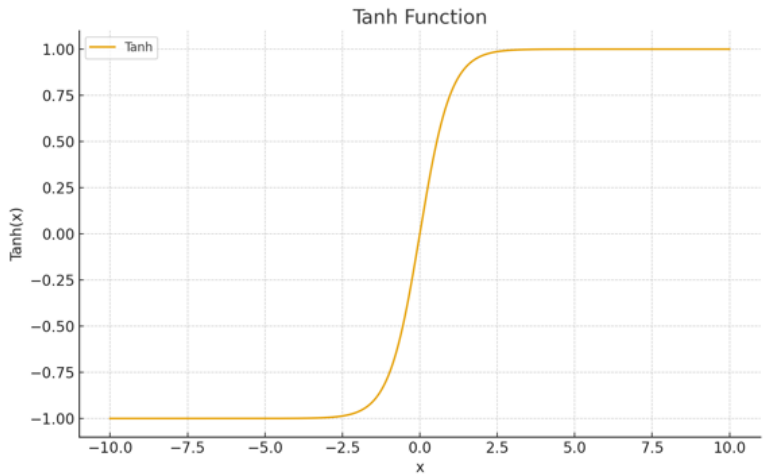
$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n) \nabla J(\mathbf{w}) \\ &= \mathbf{w}(n) + \eta(n) \sum_{\mathbf{x}(n) \in \mathcal{X}} \mathbf{x}(n) d(n) \end{aligned}$$

Common Activation Functions (Sigmoid, Tanh, ReLU, Leaky ReLU)

Sigmoid Function

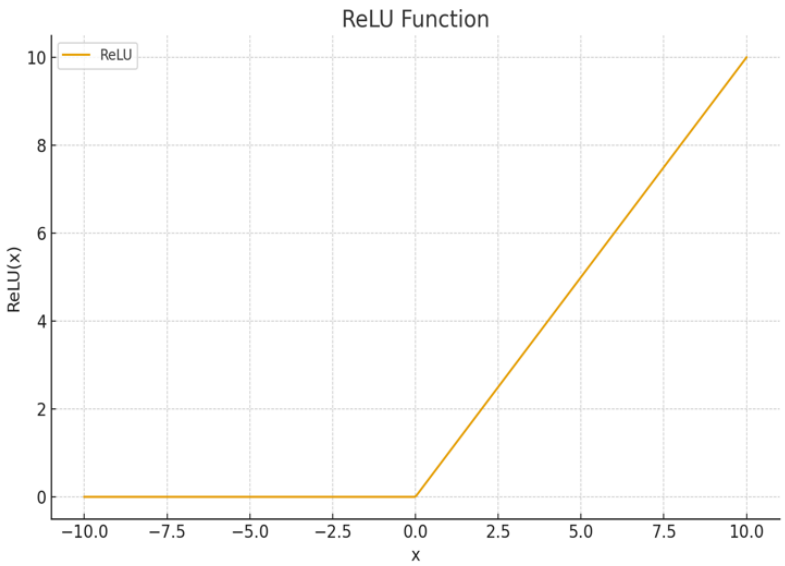


Tanh Function

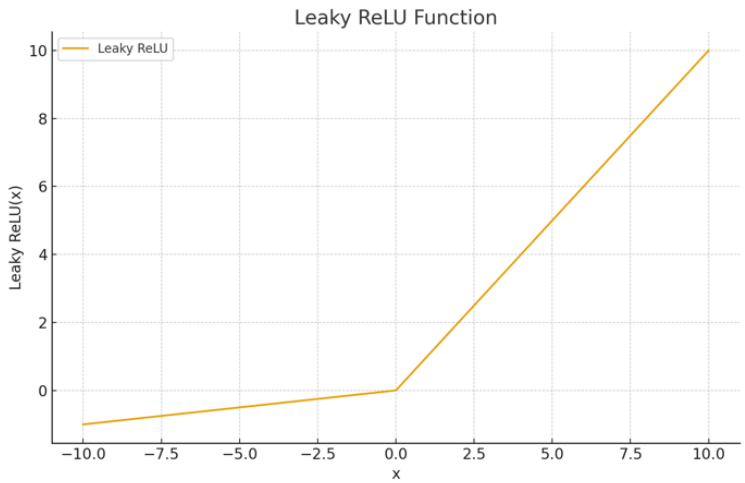


Common Activation Functions (Sigmoid, Tanh, ReLU, Leaky ReLU)

ReLU Function



Leaky ReLU Function



Limitations of Perceptron

- Cannot solve problems where data is **not linearly separable.**
- Example: XOR (Exclusive OR) problem.
- Solution: Use **Multi-Layer Perceptrons (MLP).**

Types of Neural Networks

- **Shallow neural networks** usually have only one hidden layer
  Shallow neural networks are fast and require less processing power than deep neural networks, but they cannot perform as many complex tasks as deep neural networks.
- **Deep neural networks** have multiple hidden layers

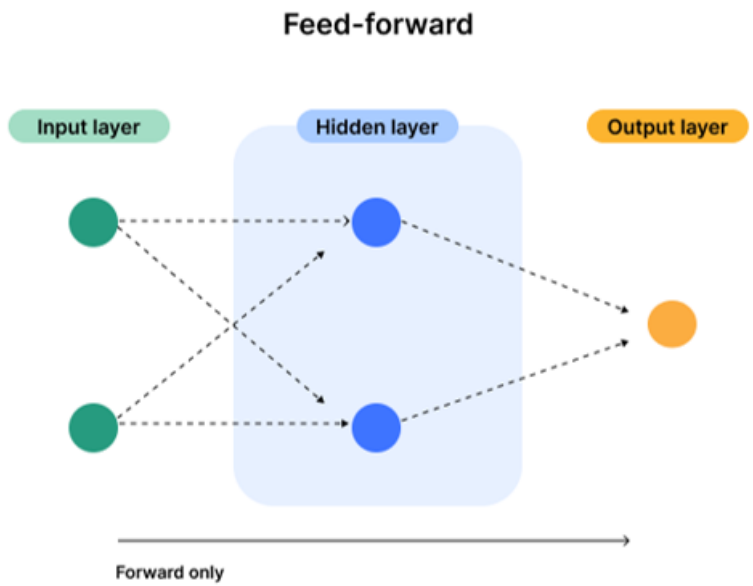
Types of Neural Networks

Feedforward Neural Network (FNN)

- How it works: Information moves in one direction—from input → hidden layers → output.
- Use cases: Basic classification (e.g., predicting if an email is spam or not).
- Key point: It is the simplest type of neural network, no loops.

Convolutional Neural Network (CNN)

- How it works: Uses filters (kernels) to scan through data, especially images, and detect patterns like edges, shapes, or textures.
- Use cases: Image recognition (e.g., recognizing cats vs. dogs), video analysis, medical image analysis.
- Key point: Best suited for image and spatial data.



Types of Neural Networks

Recurrent Neural Network (RNN)

- **How it works**: Has connections that loop back, meaning it remembers information from previous steps.
- **Use cases**: Language modeling, text prediction, speech recognition, time-series forecasting.
- **Key point**: Good for sequential data where the order matters.

Long Short-Term Memory (LSTM) Networks

- **How it works**: A special type of RNN that avoids the problem of "forgetting" long sequences by using memory cells and gates.
- **Use cases**: Machine translation, chatbot responses, predicting stock prices.
- **Key point**: Designed to remember information for longer sequences.

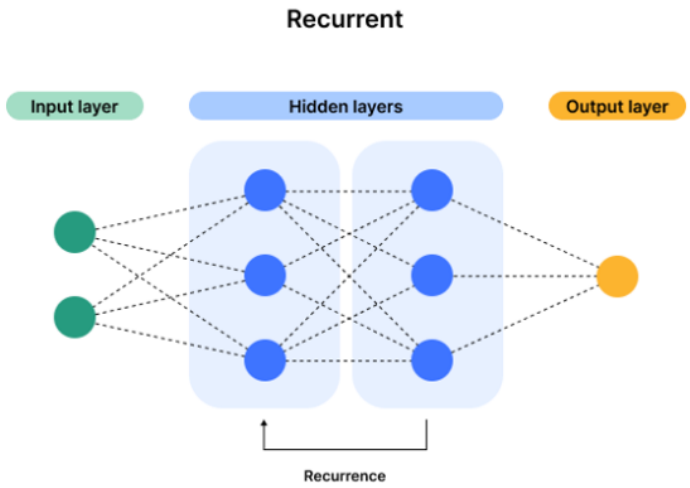


Figure 1: RNN simple cell versus LSTM cell [4]

Types of Neural Networks

Generative Adversarial Networks (GANs)

- **How it works**: Made up of two networks—Generator (creates fake data) and Discriminator (checks if data is real or fake). They compete with each other to improve.
- **Use cases**: Creating realistic images, deepfakes, art, data augmentation.
- **Key point**: Good at generating new, realistic data.

Self-Organizing Maps (SOMs)

- **How it works**: An unsupervised network that maps high-dimensional data into a lower dimension (usually 2D) for visualization.
- **Use cases**: Data clustering, feature mapping, pattern recognition.
- **Key point**: Helps visualize complex data.

Types of Neural Networks

Modular Neural Networks (MNNs)

- **How it works**: Splits a big problem into smaller parts, each solved by a separate network. Their results are combined at the end.
- **Use cases**: Large, complex systems such as robotics or medical diagnosis.
- **Key point**: Breaks down complex problems into manageable modules.

Multi Layer Neural Networks

- A multilayer neural network is an advanced model used in artificial intelligence and machine learning. Think of it as a collection of interconnected perceptrons.
- Unlike a perceptron, which has only one layer of neurons, a multilayer neural network has multiple layers stacked on top of each other. Each layer receives input from the previous layer and applies a mathematical operation called an activation function, such as the sigmoid function.

Multi-Layer Neural Networks (MLP); Structure

Input Layer

- The first layer that takes in raw features from the dataset.
- Example: In image recognition, each pixel value is an input.

Hidden Layers

- Intermediate layers that transform inputs into useful representations.
- Each hidden layer applies **weights + biases + activation functions** to the inputs.
- The deeper the layers, the more abstract patterns the network can learn.

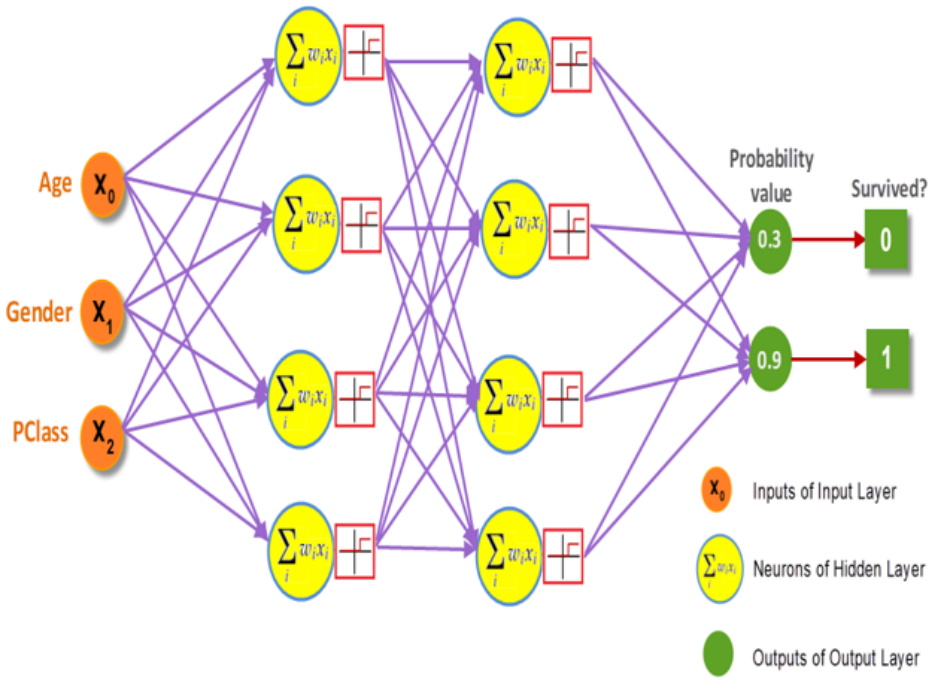
Example (for image recognition):

- **Hidden Layer 1** $\rightarrow$ detects edges (lines, corners).
- **Hidden Layer 2** $\rightarrow$ detects shapes (circles, rectangles).
- **Hidden Layer 3** $\rightarrow$ detects objects (faces, cars, animals).

Multi-Layer Neural Networks (MLP); Structure

Output Layer

- Produces the final decision/prediction.
- Example: Cat vs. Dog classification $\rightarrow$ probabilities like [0.9, 0.1].



Source: https://devskrol.com/2020/11/22/388/

How Neural Networks Learn

Step 1: Forward Propagation

- Data flows from input $\rightarrow$ hidden layers $\rightarrow$ output.
- Example: An image is processed layer by layer, from pixels to object recognition.

Step 2: Loss Function

- Measures error between prediction and actual answer.
- Example: Predicted: "Cat" (0.7), Actual: "Dog".

Step 3: Backpropagation

- Error is sent backward to update weights.
- Uses **Gradient Descent** to minimize error step by step.

Step 4: Training

- Repeat forward + backward many times until the network learns.

forward propagation (make a prediction)

Idea

- data flows layer→layer. each neuron does a weighted sum, adds a bias, then applies an activation (e.g., ReLU/sigmoid). the outputs of one layer become the inputs to the next.

Math (for layer l)

- weights & bias: $W^{(l)}, b^{(l)}$
- pre-activation: $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$
- activation: $a^{(l)} = f(z^{(l)})$
- **output layer:**
  - binary classification: $\hat{y} = \sigma(z)$
  - multi-class: $\hat{y} = \text{softmax}(z)$

loss function (measure "how wrong")

Idea

- compare prediction $\hat{y}$ with true label y. The loss is a single number to minimize.

Common choices

- **binary cross-entropy (BCE)** for 2 classes:
  $$L = -\left[y \log(\hat{y}) + (1-y) \log(1-\hat{y})\right]$$
- **categorical cross-entropy (CE)** for K classes:
  $L = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$ (with one-hot y)
- **mse** (regression): $L = \frac{1}{n} \sum (\hat{y} - y)^2$

Loss optimization

$$W^* = \arg\min_W \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

- W: the weights (parameters) of the neural network.
- $f(x^{(i)}; W)$: the model's prediction for input $x^{(i)}$ using weights W.
- $y^{(i)}$: the true label for input $x^{(i)}$.
- $\mathcal{L}(\cdot, \cdot)$: the loss function (e.g., mean squared error, cross-entropy) measuring prediction error.
- $\frac{1}{n} \sum_{i=1}^{n}$ ...: the average loss over all training examples.

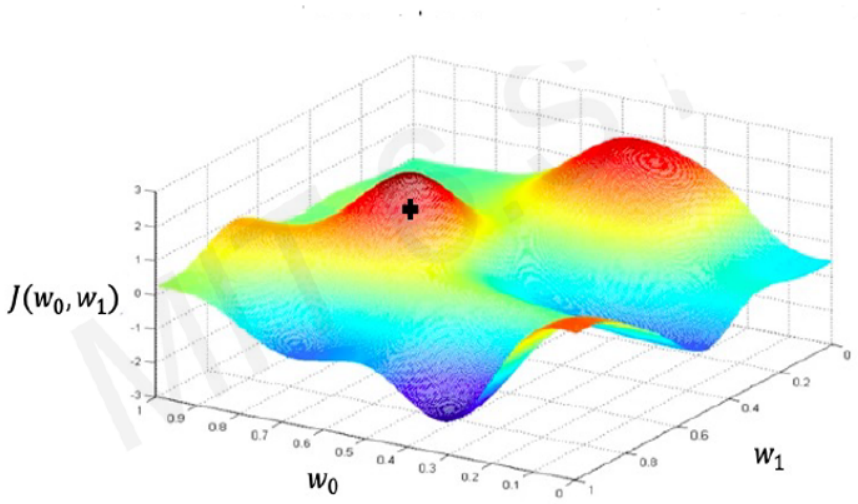
$$W^* = \arg\min_W J(W)$$

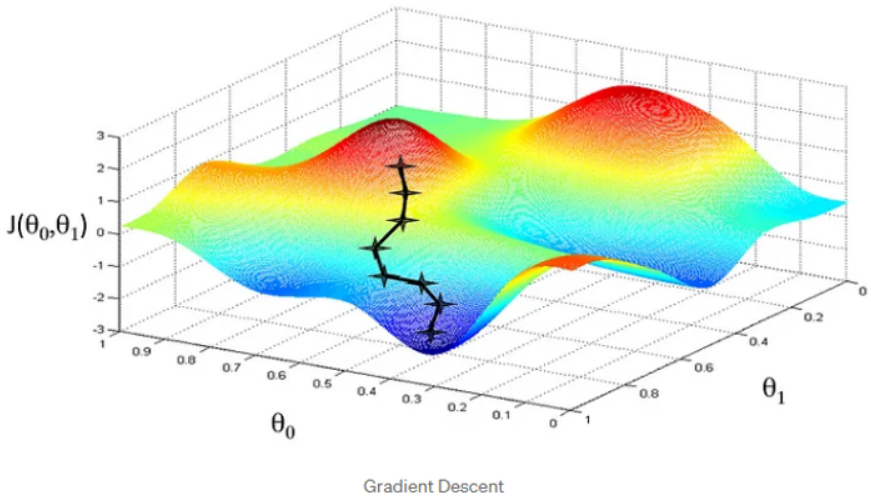
Here, $J(W)$ is just shorthand for the cost function (the average loss above).

So this equation is a simplified representation of the same idea: *choose the weights W that minimize the cost function.*

Training a neural network is about adjusting weights so the model's predictions are as close as possible to the true outputs, by minimizing the loss (cost) function.

Gradient Descent





Gradient Descent

Update the parameters using the partial derivatives of the loss function with respect to the parameters (W and b).

Gradient Descent algorithm

Algorithm

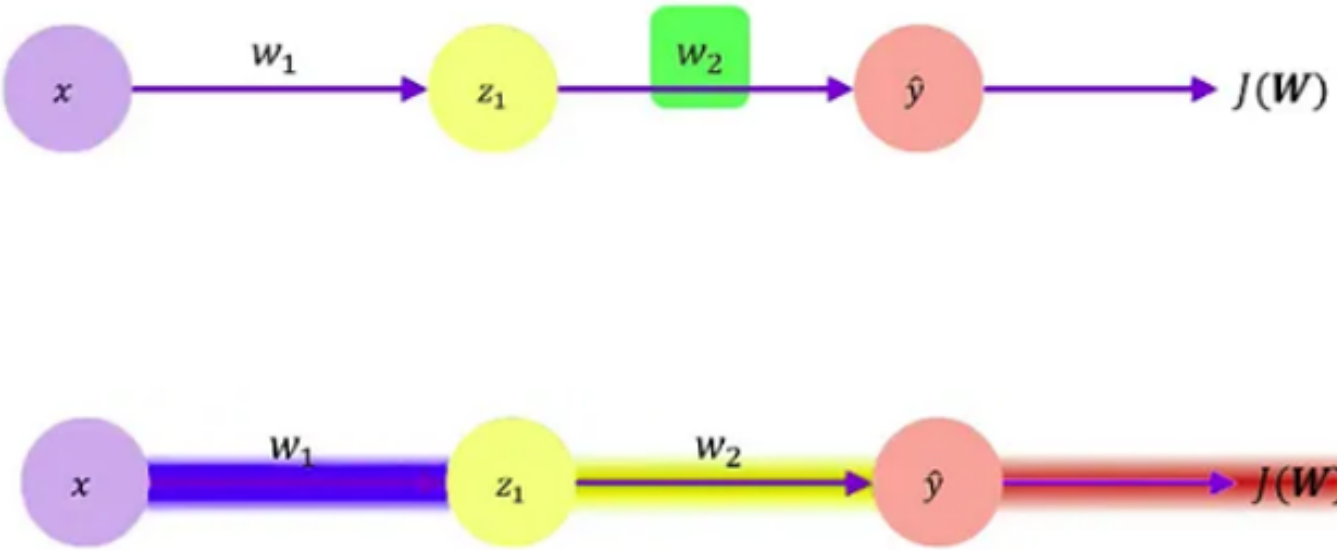
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Where:

- W = current weight
- η = learning rate (step size)
- $\frac{\partial L}{\partial W}$ = gradient of the loss w.r.t the weight

Source: https://medium.com/analytics-vidhya/neural-networks-part-2-building-neural-networks-understanding-gradient-descent-145718e91270

backpropagation (compute gradients)



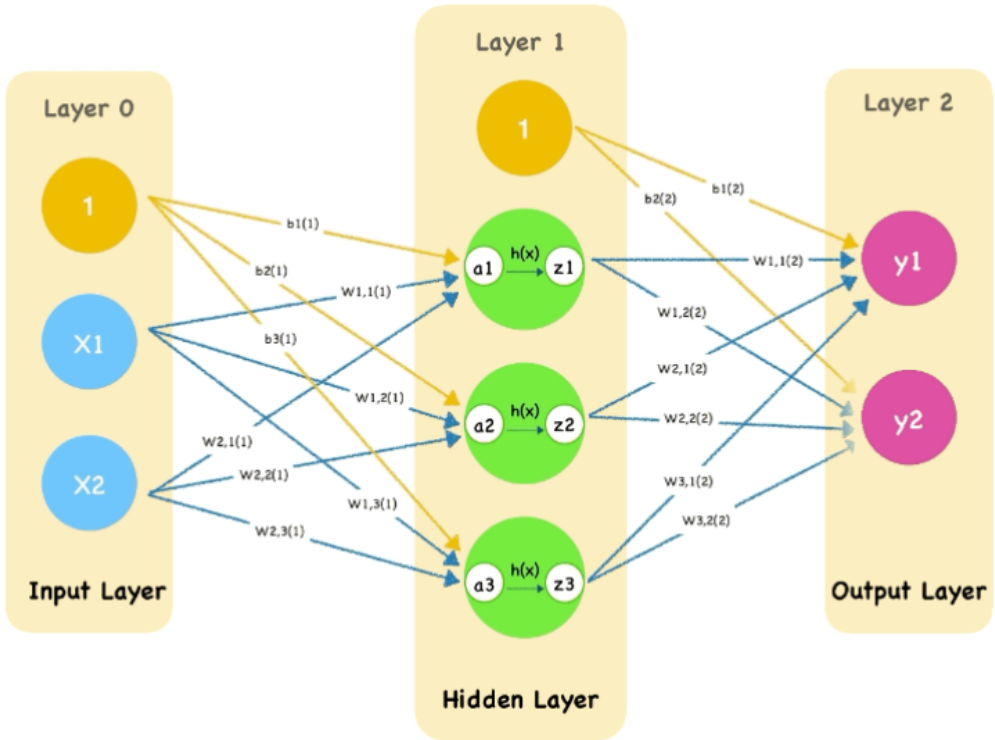
$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Backpropgate all these gradients from output to input

Source: https://medium.com/analytics-vidhya/neural-networks-part-3-understanding-back-propagation-learning-rates-3482a981a2f0

How a Neural Network Learns

- **Forward Propagation**
  - Input data is passed through each layer
  - Weighted sums and activation functions determine neuron outputs



Source:https://yogayu.github.io/DeepLearningCourse/03/ForwardPropagation.html

Thank you