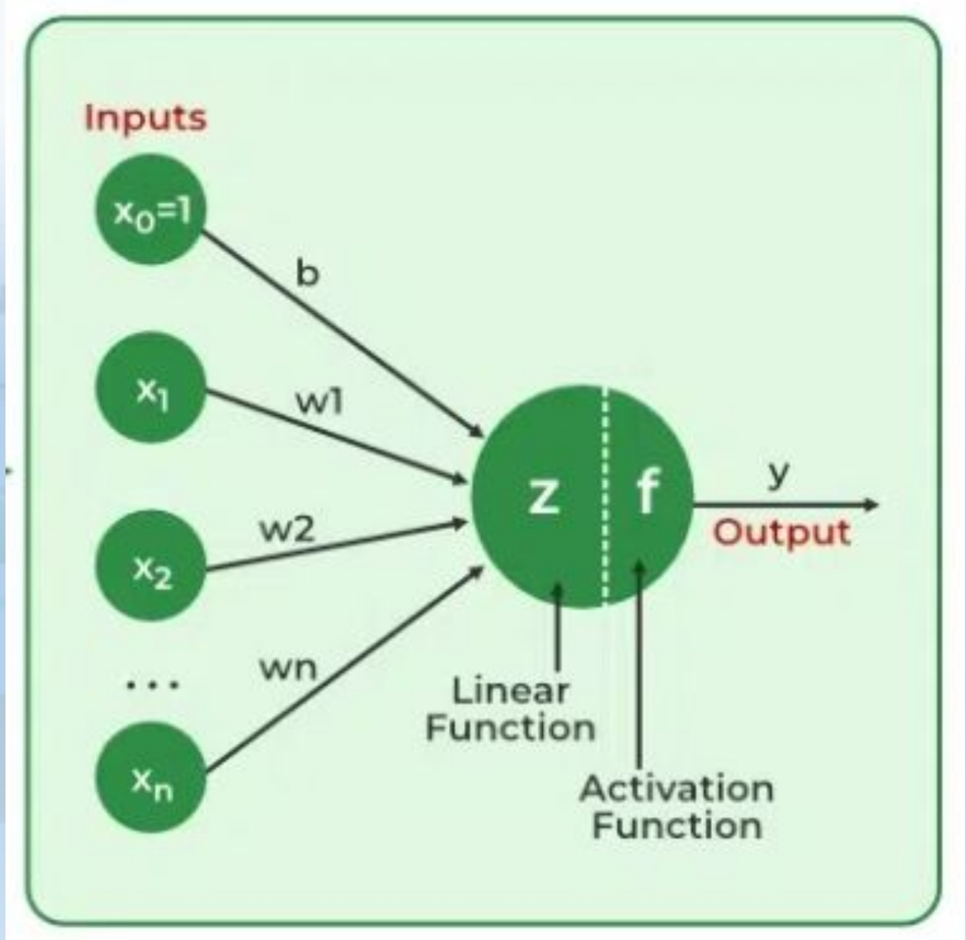
Deep Learning

Neural Network

Gananath Bhuyan
Assistant Professor
School of Computer Engineering
KIIT DU, Bhubaneswar

Neural Network

- **Neurons**: The basic units that receive inputs, each neuron is governed by a threshold and an activation function.
- **Connections**: Links between neurons that carry information, regulated by weights and biases.
- **Weights and Biases**: These parameters determine the strength and influence of connections.
- **Propagation Functions**: Mechanisms that help process and transfer data across layers of neurons.
- **Learning Rule**: The method that adjusts weights and biases over time to improve accuracy.



How does a neural network work?

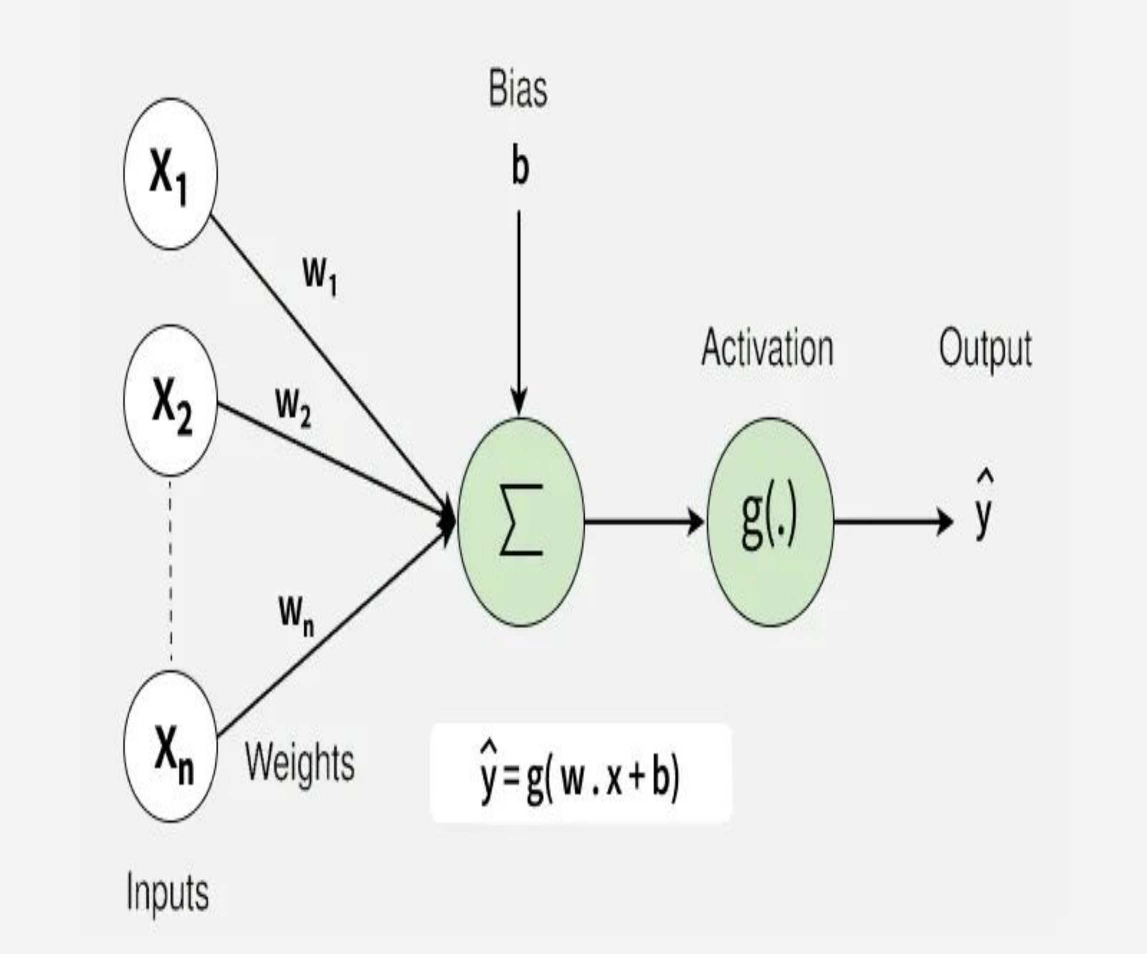
1. Forward Propagation
   A. Linear Transformation
   B. Activation

$$z = w_1 x_1 + w_2 x_2 + .. + w_n x_n + b$$

$$z = \sum_{i}^{n} w_i x_i + b$$

$$y' = g(z)$$

2. Backpropagation
   A. Loss Calculation
   B. Gradient Calculation
   C. Weight Update
3. Iteration



Example:

Email: "Congratulations! You have won a cash prize. Click here to claim your reward."

1. Create a feature vector:

"congratulations","cash","offer" ➪ [1, 1, 0]

2. Defining the network:

Input: 3 neurons (I1, I2,I3), Hidden: 2 neuron (H1, H2), output : 1 neuron (Out)

weights :

IP➪Hidden: [0.6, -0.3, 0.2][-0.4, 0.9, 0.1]

Hidden➪output : [-0.8, 0.6]

3. Processing at hidden layer:

a. Linear Transfer:

Input vector = [1, 1, 0]

H1 : (1×0.6) + (1×−0.3) + (0×0.2) = 0.6 − 0.3 + 0 = 0.3

H2 : (1×−0.4) + (1×0.9) + (0×0.1) = −0.4 + 0.9 = 0.5

b. Activation: (ReLU) ➪ max(0,x)

H1 : max (0, 0.3) = 0.3

H2 : max (0, 0.5) = 0.5

4. Processing at Output:

a. Linear Transfer:

Input = [0.3, 0.5]

Out: (0.3×-0.8) + (0.5×0.6) = -0.24 + 0.3 = 0.06

b. Activation: (Sigmoid) ➪ $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$\frac{1}{1 + e^{-0.06}} = 0.515$$

Deep Feed Forward Network

- multilayer perceptrons (MLPs)
- The goal of a feedforward network is to approximate some function f^* without any looping back to the previous layer.
- Directed acyclic graph.
- $y = f^*(x)$ ⇨ if y is an image then y is the class of it (dog/cat)
- $y = f(x;\theta)$ ⇨ $y = f(x;w,b)$ ⇨ $y = wx + b$
- $y = (f^n ......(f^2 (f^1 (x)))....)$

Activation Functions

- It is a mathematical function applied to the output of a neuron.
- They introduce non-linearity into the network enabling it to learn and model complex data patterns.
- Without this non-linearity feature a neural network would behave like a linear regression model no matter how many layers it has.
- **ReLU**(x) (Rectified Linear Unit):

$$ReLU(x) = max(x, 0)$$

- **Tanh**:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Sigmoid**:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Activation Functions

- They introduce non-linearity into the network enabling it to learn and model complex data patterns.
- ReLU(x) (Rectified Linear Unit):

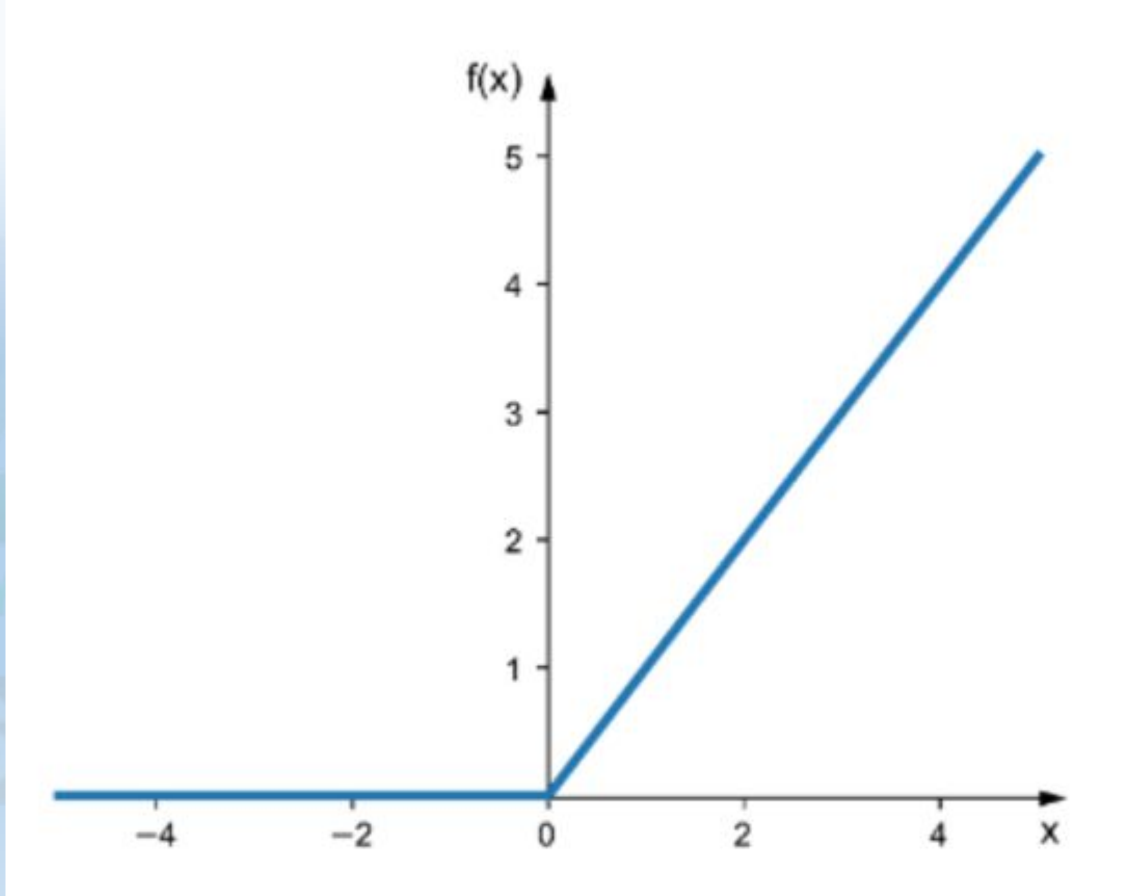
$$ReLU(x) = max(x, 0)$$

- Tanh:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Activation Functions

- They introduce non-linearity into the network enabling it to learn and model complex data patterns.
- ReLU(x) (Rectified Linear Unit):

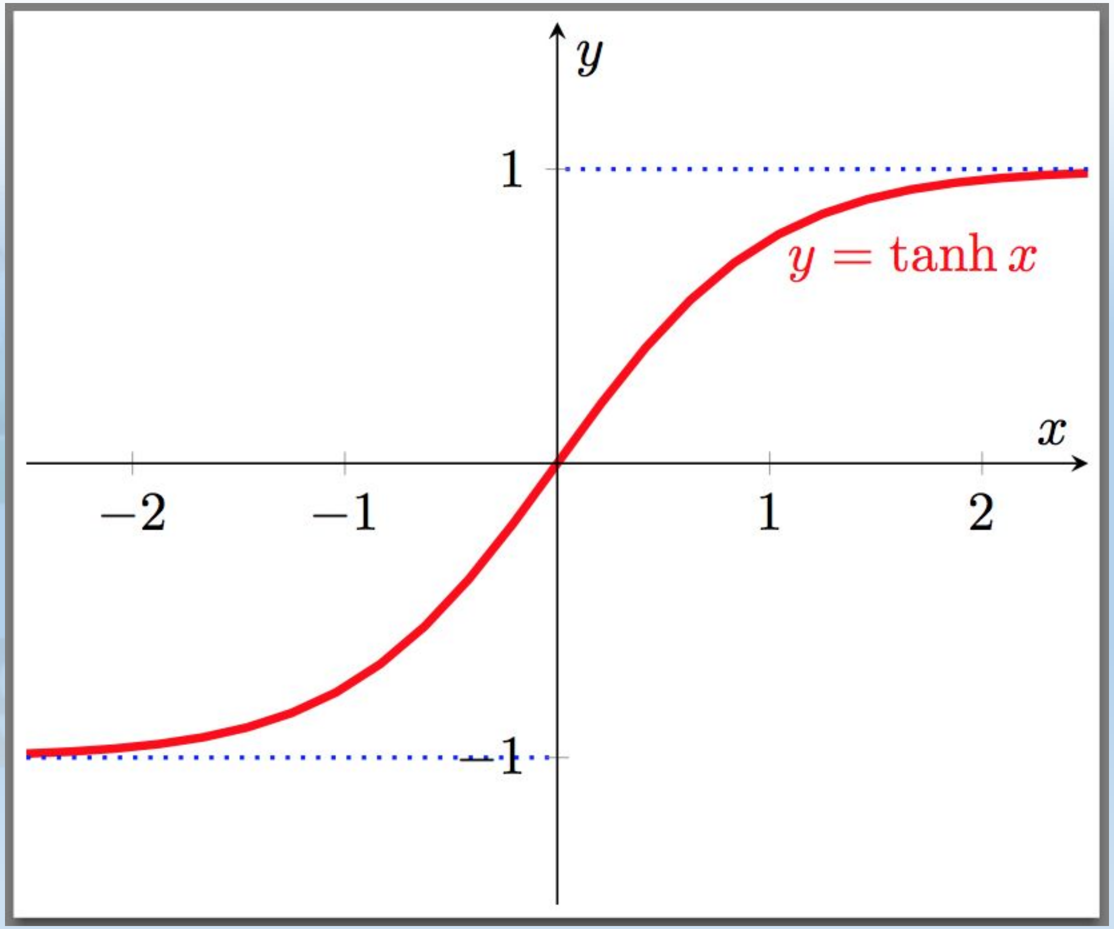
$$ReLU(x) = max(x, 0)$$

- Tanh:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Activation Functions

- They introduce non-linearity into the network enabling it to learn and model complex data patterns.
- ReLU(x) (Rectified Linear Unit):

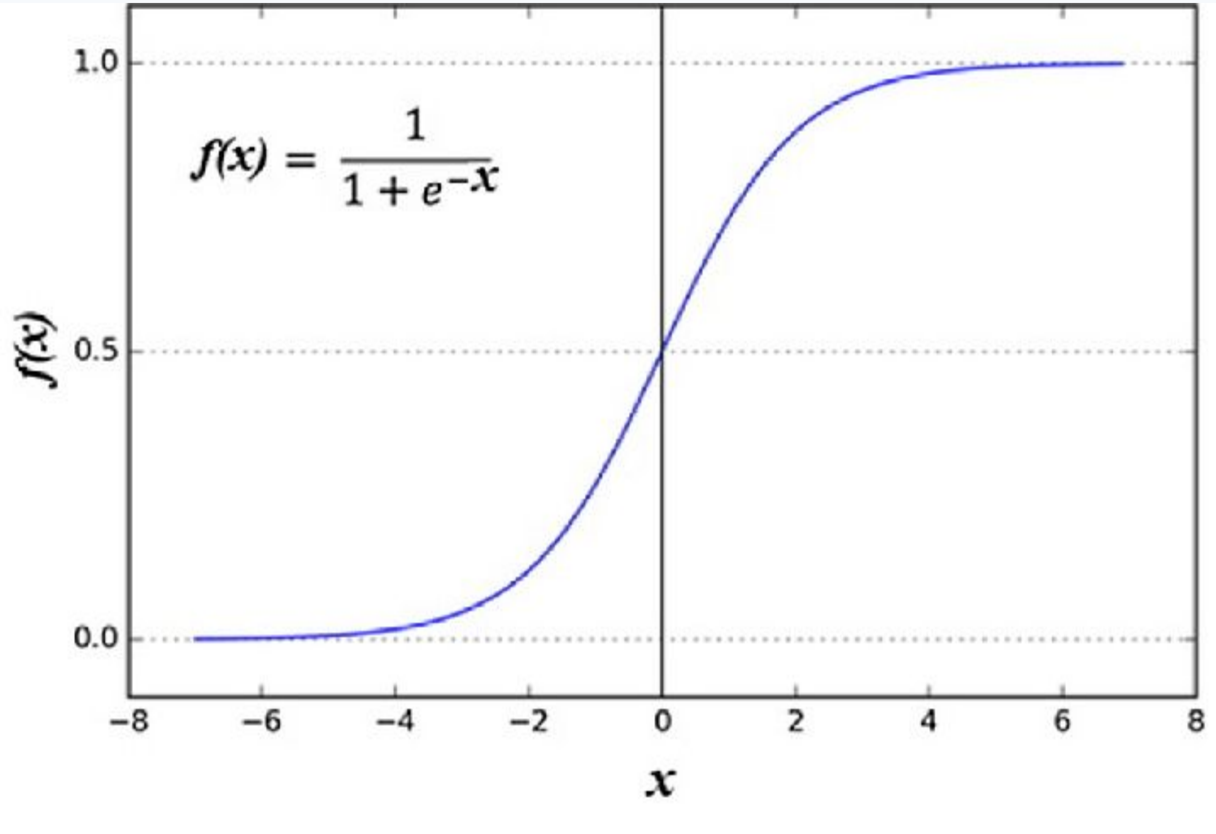
$$ReLU(x) = max(x, 0)$$

- Tanh:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



```
import numpy as np
def relu(x):
    return np.maximum(0, x)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# "congratulations": 1, "cash": 1, "offer": 0
input_vector = np.array([1, 1, 0])
# Step 2: Weights for hidden layer
weights_hidden = np.array([
    [0.6, -0.3, 0.2],   # H1
    [-0.4, 0.9, 0.1]    # H2
])
# Step 3: Weighted sum (hidden layer)
z_hidden = np.dot(weights_hidden, input_vector)  # Shape: (2,)
a_hidden = relu(z_hidden)
# Step 4: Output layer
output_weights = np.array([0.8, -0.6])  # Shape: (2,)
z_output = np.dot(output_weights, a_hidden)
a_output = sigmoid(z_output)
# Step 5: Classification
label = 1 if a_output > 0.5 else 0

print("Input Vector:", input_vector)
print("Hidden Layer Input (z):", z_hidden)
print("Hidden Layer Output (ReLU):", a_hidden)
print("Output Layer Input (z):", z_output)
print("Output (Sigmoid):", a_output)
print("Final Classification: Spam" if label == 1 else "Final Classi
```