**Practical No 1**

import numpy as np

import matplotlib.pyplot as plt

# Define activation functions

def linear(x):

    return x

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def tanh(x):

    return np.tanh(x)

def relu(x):

    return np.maximum(0, x)

# Generate input values

x = np.linspace(-5, 5, 400)

# Compute activation function values

y\_functions = {

    "Linear": linear(x),

    "Sigmoid": sigmoid(x),

    "Tanh": tanh(x),

    "ReLU": relu(x)

}

# Colors for plots

colors = ["blue", "red", "green", "purple"]

# Create subplots

plt.figure(figsize=(10, 6))

for i, (name, y) in enumerate(y\_functions.items(), start=1):

    plt.subplot(2, 2, i)  # Arrange in 2 rows, 2 columns

    plt.plot(x, y, label=name, color=colors[i-1], linewidth=2)

    plt.title(name, fontsize=12, fontweight="bold")

    plt.axhline(y=0, color='black', linestyle='dashed', linewidth=0.7)

    plt.axvline(x=0, color='black', linestyle='dashed', linewidth=0.7)

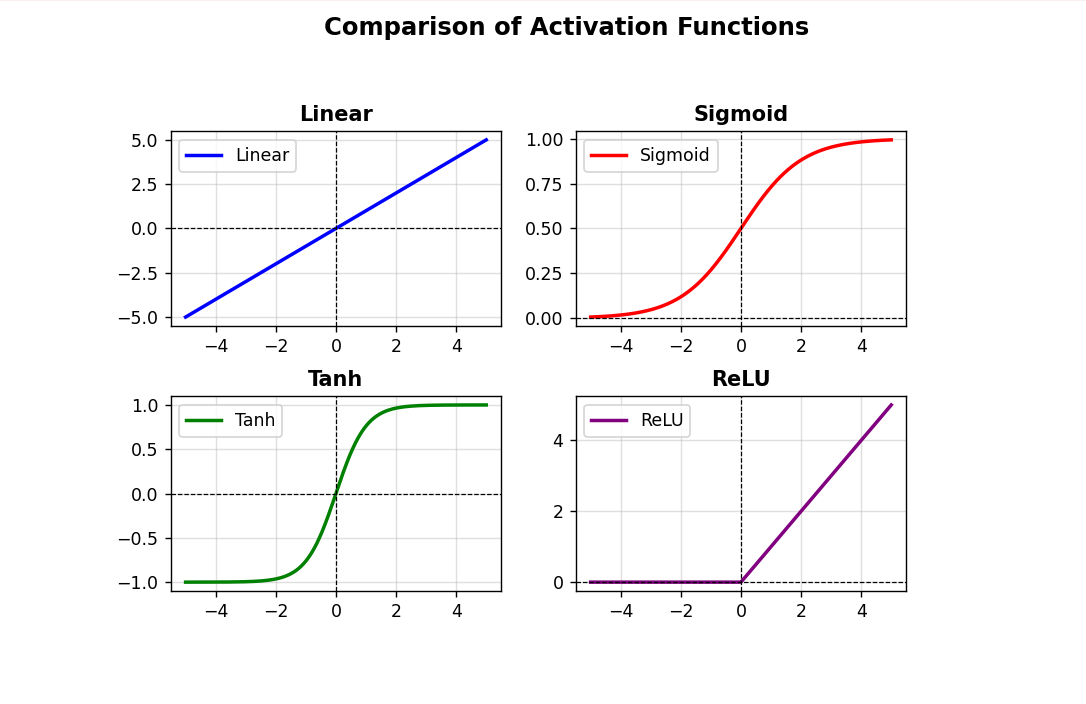
    plt.grid(alpha=0.4)

    plt.legend()

plt.suptitle("Comparison of Activation Functions", fontsize=14, fontweight="bold")

plt.tight\_layout(rect=[0, 0, 1, 0.95])  # Adjust layout

plt.show()

****

**Practical No 2**

% McCulloch-Pitts Neural Network for ANDNOT Function

% Define input vectors (x1, x2)

inputs = [0 0; 0 1; 1 0; 1 1]; % [x1, x2]

% Define weights and threshold

w = [1 -1]; % Weights for x1 and NOT x2

theta = 1;  % Threshold

% Compute net input and output

net\_input = inputs \* w';

outputs = net\_input >= theta; % Apply step activation function

% Display results

disp('Weights of Neuron:');

disp(['w1 = ' num2str(w(1))]);

disp(['w2 = ' num2str(w(2))]);

disp(['Threshold: Theta = ' num2str(theta)]);

disp(' ');

disp('Output:');

disp(['w1 = ' num2str(w(1))]);

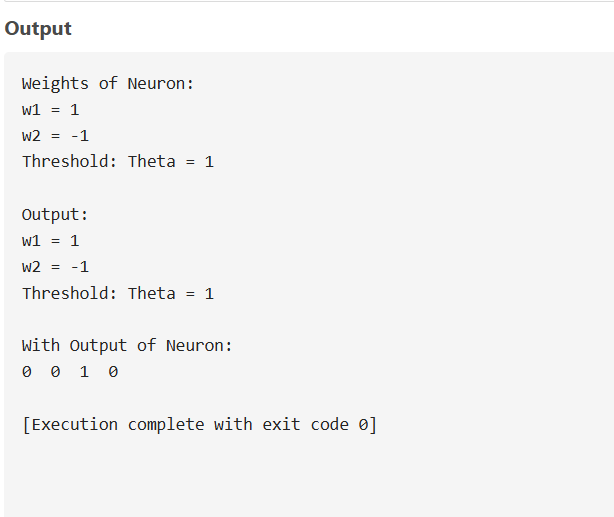
disp(['w2 = ' num2str(w(2))]);

disp(['Threshold: Theta = ' num2str(theta)]);

disp(' ');

disp('With Output of Neuron:');

disp(num2str(outputs'));



**Practical No 3**

import tkinter as tk

from tkinter import ttk

# Function to classify even (1) and odd (0)

def classify\_number(n):

    return 1 if n % 2 == 0 else 0  # Even -> 1, Odd -> 0

# Create main window

root = tk.Tk()

root.title("Perceptron: Even/Odd Classification")

root.geometry("400x350")

root.configure(bg="#2C3E50")  # Dark background

# Style for the Treeview table

style = ttk.Style()

style.theme\_use("clam")  # Use a clean style

style.configure("Treeview",

                background="#ECF0F1",

                foreground="black",

                rowheight=30,

                fieldbackground="#ECF0F1")

style.configure("Treeview.Heading",

                font=("Arial", 12, "bold"),

                background="#2980B9",

                foreground="white")

# Create table using Treeview

columns = ("Number", "ASCII Value", "Even (1) / Odd (0)")

tree = ttk.Treeview(root, columns=columns, show="headings", height=10)

tree.heading("Number", text="Number")

tree.heading("ASCII Value", text="ASCII Value")

tree.heading("Even (1) / Odd (0)", text="Even (1) / Odd (0)")

# Center columns and set width

for col in columns:

    tree.column(col, anchor="center", width=120)

# Insert data into the table

for num in range(10):  # ASCII digits 0-9

    ascii\_val = ord(str(num))

    even\_odd = classify\_number(num)

    tree.insert("", "end", values=(num, ascii\_val, even\_odd))

tree.pack(pady=20, padx=20)

# Styled Exit button

exit\_button = tk.Button(root, text="Exit", command=root.quit,

                        bg="#E74C3C", fg="white", font=("Arial", 12, "bold"),

                        padx=10, pady=5, relief="flat", activebackground="#C0392B")

exit\_button.pack(pady=10)

# Run application

root.mainloop()

