
Assignment 6

Question 1: Write a Program
Implement the following activation functions and their derivatives:
1. Sigmoid
2. Tanh
3. ReLU
4. Leaky ReLU (with alpha=0.1)
5. ELU (with alpha=1.0)

Then create a visualization that shows: - Each activation function - Their derivatives - How they respond to inputs in the range [-5, 5]

(Activation functions are mathematical operations applied to the output of a neuron that introduce non-linearity into neural networks. They determine whether a neuron should be activated ("fired") based on its inputs.)

```
# TODO: Implement the activation functions
def sigmoid(x):
    # Your code here
    pass

def sigmoid_derivative(x):
    # Your code here
    pass

def tanh(x):
    # Your code here
    pass

def tanh_derivative(x):
    # Your code here
    pass

def relu(x):
    # Your code here
    pass

def relu_derivative(x):
    # Your code here
    pass
```

```
def leaky_relu(x, alpha=0.1):
    # Your code here
    pass

def leaky_relu_derivative(x, alpha=0.1):
    # Your code here
    pass

def elu(x, alpha=1.0):
    # Your code here
    pass

def elu_derivative(x, alpha=1.0):
    # Your code here
    pass

# TODO: Create a function to plot the activation functions and their derivatives
def plot_activation_functions():
    # Your code here
    pass

# Run your code to generate the plots
if __name__ == "__main__":
    plot_activation_functions()
```

Implement Perceptron for the AND Gate

How the Perceptron Works

A perceptron is the simplest form of artificial neural network. For the AND gate:

1. **Inputs: Two binary inputs (0 or 1)**
2. **Output: Single binary output (0 or 1)**
3. **Learning Rule: Adjusts weights based on error between predicted and actual output**

AND Gate Truth Table

Input 1	Input 2	Output
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

Key Components of the Implementation

1. **Weights and Bias: The perceptron learns two weights (one for each input) and a bias term**
2. **Activation Function: A simple step function that returns 1 if the weighted sum is positive, 0 otherwise**
3. **Training Algorithm:**
    - **For each input pattern, calculate the predicted output**
    - **Compare with the expected output and compute error**
    - **Adjust weights and bias if there's an error**
    - **Repeat until convergence (no errors) or max iterations reached**
4. **Decision Boundary: The perceptron creates a linear boundary that separates the input space**