In [3]:
```
import numpy as np
import pandas as pd
```

In [4]:
```
# Load your CSV file with input features and target labels
data = pd.read_csv("C:/Users/aryan/OneDrive/Documents/archive/2020-2021.csv")
```

In [5]:
```
# Identify non-numeric columns and preprocess them (adjust column names as needed)
non_numeric_columns = data.select_dtypes(exclude=[np.number]).columns
data = pd.get_dummies(data, columns=non_numeric_columns, drop_first=True)
```

In [6]:
```
# Extract input features and target labels
X = data.iloc[:, :-1].values.astype(float)  # Convert input features to float
y = data.iloc[:, -1].values.astype(float)   # Convert target labels to float
```

In [7]:
```
# Define activation functions
def logarithmic(x):w
    return np.log(1 + x)

def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -700, 700)))

def ReLU(x):
    return np.maximum(0, x)

def tanh(x):
    return np.tanh(x)
```

In [8]:
```
# Define the delta rule for updating weights
def delta_rule(weights, learning_rate, error, inputs):
    return weights + learning_rate * error * inputs
```

In [9]:
```
# 1-Layer Perceptron
def one_layer_perceptron(X, y, activation_function, learning_rate, epochs):
    # Initialize weights and bias
    num_features = X.shape[1]
    weights = np.random.rand(num_features)
    bias = np.random.rand()

    for epoch in range(epochs):
        for i in range(X.shape[0]):

            weighted_sum = np.dot(X[i], weights) + bias

            output = activation_function(weighted_sum)

            error = y[i] - output

            # Update weights and bias using the delta rule
            weights = delta_rule(weights, learning_rate, error, X[i])
            bias += learning_rate * error

    return weights, bias
```

In [10]:
```
# 2-Layer Perceptron
def two_layer_perceptron(X, y, activation_function, learning_rate, epochs):
    num_input_features = X.shape[1]
    num_hidden_units = 4  # Adjust as needed
    num_output_units = 1

    hidden_weights = np.random.rand(num_input_features, num_hidden_units)
    hidden_bias = np.random.rand(num_hidden_units)

    output_weights = np.random.rand(num_hidden_units, num_output_units)
    output_bias = np.random.rand(num_output_units)

    for epoch in range(epochs):
        for i in range(X.shape[0]):

            hidden_layer_input = np.dot(X[i], hidden_weights) + hidden_bias
            hidden_layer_output = activation_function(hidden_layer_input)

            output_layer_input = np.dot(hidden_layer_output, output_weights) + output
            output_layer_output = activation_function(output_layer_input)

            output_error = y[i] - output_layer_output

            delta_output = output_error * (output_layer_output * (1 - output_layer_ou
            output_weights += learning_rate * np.outer(hidden_layer_output, delta_out
            output_bias += learning_rate * delta_output

            delta_hidden = np.dot(delta_output, output_weights.T) * (hidden_layer_out
            hidden_weights += learning_rate * np.outer(X[i], delta_hidden)
            hidden_bias += learning_rate * delta_hidden

    return hidden_weights, hidden_bias, output_weights, output_bias
```

In [11]:
```
learning_rate = 0.1
epochs = 1000
weights, bias = one_layer_perceptron(X, y, sigmoid, learning_rate, epochs)

learning_rate = 0.1
epochs = 1000
hidden_weights, hidden_bias, output_weights, output_bias = two_layer_perceptron(X, y,
```

In [13]:
```
# Calculate accuracy
accuracy = np.mean(predictions == y)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 100.00%
```

In [17]:
```
import numpy as np
import pandas as pd

# Load your CSV file with input features and target labels
data = pd.read_csv("C:/Users/aryan/OneDrive/Documents/archive/2020-2021.csv")

noise_level = 0.05
y_noisy = y.copy()
num_samples = len(y)

num_noisy_samples = int(noise_level * num_samples)
indices_to_change = np.random.choice(num_samples, num_noisy_samples, replace=False)
y_noisy[indices_to_change] = 1 - y_noisy[indices_to_change]  # Flip the labels

accuracy = np.mean(predictions == y_noisy)
print(f"Noisy Accuracy: {accuracy * 100:.2f}%")
```

```
Noisy Accuracy: 95.00%
```

In [ ]: