Pratham_LAB1

September 22, 2024

1 Lab1

1.1 Submitted by Pratham M (2347138)

Defining the perceptron

[11]:
```
import numpy as np


class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=10):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    # ReLU activation function
    def relu(self, z):
        return np.maximum(0, z)

    # Derivative of ReLU
    def relu_derivative(self, z):
        return np.where(z > 0, 1, 0)

    #Train the perceptron using gradient descent
    def train(self, X, y, random_weights=True):
        n_samples, n_features = X.shape

        if random_weights:
            self.weights = np.random.rand(n_features)
        else:
            self.weights = np.array([0.5, 0.5])
        self.bias = 0.0

        for epoch in range(self.epochs):
            for idx, x_i in enumerate(X):

                linear_output = np.dot(x_i, self.weights) + self.bias
```

```
                predicted = self.relu(linear_output)
                error = y[idx] - predicted
                gradient = error * self.relu_derivative(predicted)
                self.weights += self.learning_rate * gradient * x_i
                self.bias += self.learning_rate * gradient

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return self.relu(linear_output)

perceptron = Perceptron(learning_rate=0.1, epochs=100)
```

AND

Truth Table for AND Gate:

Input 1

Input 2

Output

0

0

0

0

1

0

1

0

0

1

1

1

[12]:
```
# AND Gate Implementation
X_and = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y_and = np.array([0, 0, 0, 1])  # AND gate outputs

perceptron_and = PerceptronSigmoid(learning_rate=0.1, epochs=100)
perceptron_and.train(X_and, y_and, random_weights=True)

# Test the AND gate perceptron
print("AND Gate Results:")
```

```
correct_predictions = 0
for x, target in zip(X_and, y_and):
    output = perceptron_and.predict(x)
    predicted = 1 if round(output) >= 0.5 else 0  # Convert to binary output
    print(f"Input: {x} - Predicted: {predicted} - Actual: {target}")
    if predicted == target:
        correct_predictions += 1
accuracy_or = correct_predictions / len(y_and) * 100
print(f"AND Gate Accuracy: {accuracy_or:.2f}%\n")
```

```
AND Gate Results:
Input: [0 0] - Predicted: 0 - Actual: 0
Input: [0 1] - Predicted: 0 - Actual: 0
Input: [1 0] - Predicted: 0 - Actual: 0
Input: [1 1] - Predicted: 1 - Actual: 1
AND Gate Accuracy: 100.00%
```

Questions - AND Gate

How do the weights and bias values change during training for the AND gate?

- Initially, the weights and bias are random. As the perceptron encounters training errors, it updates them based on the difference between predicted and actual output (the error), using the learning rate to control the step size. ##### Can the perceptron successfully learn the AND logic with a linear decision boundary?
- Yes, the AND gate is linearly separable, so a Single Layer Perceptron can successfully learn to classify it with a linear decision boundary.

OR Gate

Truth Table for OR Gate

Input 1

Input 2

Output

0

0

0

0

1

1

1

0

1

1

1

1

[13]:
```python
# OR Gate Implementation
X_or = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y_or = np.array([0, 1, 1, 1])  # OR gate outputs

perceptron_or = PerceptronSigmoid(learning_rate=0.1, epochs=100)
perceptron_or.train(X_or, y_or, random_weights=True)

# Test the OR gate perceptron
print("OR Gate Results:")
correct_predictions = 0
for x, target in zip(X_or, y_or):
    output = perceptron_or.predict(x)
    predicted = 1 if round(output) >= 0.5 else 0  # Convert to binary output
    print(f"Input: {x} - Predicted: {predicted} - Actual: {target}")
    if predicted == target:
        correct_predictions += 1
accuracy_or = correct_predictions / len(y_or) * 100
print(f"OR Gate Accuracy: {accuracy_or:.2f}%\n")
```

```
OR Gate Results:
Input: [0 0] - Predicted: 1 - Actual: 0
Input: [0 1] - Predicted: 1 - Actual: 1
Input: [1 0] - Predicted: 1 - Actual: 1
Input: [1 1] - Predicted: 1 - Actual: 1
OR Gate Accuracy: 75.00%
```

Questions - OR Gate

What changes in the perceptron's weights are necessary to represent the OR gate logic?

- The weights will adjust to reflect that as long as one of the inputs is 1, the output should be 1. Thus, the weights tend to be positive enough to push the linear combination above the activation threshold in the presence of 1s.

How does the linear decision boundary look for the OR gate classification?

- The decision boundary separates the inputs (0,1), (1,0), and (1,1) from (0,0), representing a linear decision surface where any non-zero input leads to a positive output.

AND-NOT Gate

Truth Table for AND-NOT Gate

Input 1

Input 2

Output

0

0

0

0

1

0

1

0

1

1

1

0

[14]:
```
# AND-NOT Gate Implementation
X_andnot = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y_andnot = np.array([0, 0, 1, 0])  # AND-NOT gate outputs

perceptron_andnot = PerceptronSigmoid(learning_rate=0.1, epochs=100)
perceptron_andnot.train(X_andnot, y_andnot, random_weights=True)

# Test the AND-NOT gate perceptron
print("AND-NOT Gate Results:")
correct_predictions = 0
for x, target in zip(X_andnot, y_andnot):
    output = perceptron_andnot.predict(x)
    predicted = 1 if round(output) >= 0.5 else 0  # Convert to binary output
    print(f"Input: {x} - Predicted: {predicted} - Actual: {target}")
    if predicted == target:
        correct_predictions += 1
accuracy_andnot = correct_predictions / len(y_andnot) * 100
print(f"AND-NOT Gate Accuracy: {accuracy_andnot:.2f}%\n")
```

```
AND-NOT Gate Results:
Input: [0 0] - Predicted: 0 - Actual: 0
Input: [0 1] - Predicted: 0 - Actual: 0
Input: [1 0] - Predicted: 0 - Actual: 1
Input: [1 1] - Predicted: 0 - Actual: 0
AND-NOT Gate Accuracy: 75.00%
```

Questions - AND-NOT

What is the perceptron's weight configuration after training for the AND-NOT gate?

- The weights will adjust such that the perceptron responds only when the first input is 1 and the second input is 0, reflecting the AND-NOT condition. ##### How does the perceptron handle cases where both inputs are 1 or 0?
- The perceptron outputs 0 for both these cases, as required by the AND-NOT logic

XOR Gate

Truth Table for XOR Gate

Input 1

Input 2

Output

0

0

0

0

1

1

1

0

1

1

1

0

[15]:
```
# XOR Gate Implementation
X_xor = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y_xor = np.array([0, 1, 1, 0])  # XOR gate outputs

perceptron_xor = PerceptronSigmoid(learning_rate=0.1, epochs=100)
perceptron_xor.train(X_xor, y_xor, random_weights=True)

# Test the XOR gate perceptron
print("XOR Gate Results:")
correct_predictions = 0
for x, target in zip(X_xor, y_xor):
    output = perceptron_xor.predict(x)
    predicted = 1 if round(output) >= 0.5 else 0  # Convert to binary output
```

```
    print(f"Input: {x} - Predicted: {predicted} - Actual: {target}")
    if predicted == target:
        correct_predictions += 1
accuracy_xor = correct_predictions / len(y_xor) * 100
print(f"XOR Gate Accuracy: {accuracy_xor:.2f}%\n")
```

```
XOR Gate Results:
Input: [0 0] - Predicted: 0 - Actual: 0
Input: [0 1] - Predicted: 0 - Actual: 1
Input: [1 0] - Predicted: 1 - Actual: 1
Input: [1 1] - Predicted: 1 - Actual: 0
XOR Gate Accuracy: 50.00%
```

Observe and discuss the perceptron's performance in this scenario.

The perceptron will struggle with the XOR gate because XOR is not linearly separable (no single straight line can separate the classes). A single-layer perceptron can only solve linearly separable problems like AND, OR, and AND-NOT. To solve XOR, you'd need a more complex model, such as a multi-layer perceptron (MLP) with non-linear activation functions.

Questions - XOR Gate

Why does the Single Layer Perceptron struggle to classify the XOR gate?

- XOR is not linearly separable, meaning it cannot be correctly classified by a Single Layer Perceptron because its decision boundary is non-linear.

What modifications can be made to the neural network model to handle the XOR gate correctly?

- A Multi-Layer Perceptron (MLP) with at least one hidden layer can successfully classify XOR by learning non-linear decision boundarie