**IMPLEMENTATION OF LOGIC GATES**

**USING NEURAL NETWORKS**

**A MINI PROJECT REPORT**

Submitted by

**DHARANASANKARESWARI R (22EC014)**

In partial fulfilment for the award of the degree

Of

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION**

**ENGINEERING**

**P.S.R.ENGINEERING COLLEGE, SIVAKASI**

(An Autonomous Institution, Affiliated to Anna University, Chennai)

**ANNA UNIVERSITY: CHENNAI-600 025**

AUGUST 2024

**ABSTRACT:**

This project explores the implementation of fundamental logic gates—AND, OR, NAND, XOR, and NOT—using artificial neural networks. Traditionally, these logic gates are the building blocks of digital circuits, designed with fixed hardware components. However, by leveraging the learning capabilities of neural networks, we can model these gates in a more flexible and adaptable way. A simple feedforward neural network with a single hidden layer is trained using the truth tables of the logic gates. The network employs a sigmoid activation function, which allows it to learn the necessary decision boundaries to accurately mimic the behavior of each gate. This approach not only provides a deeper understanding of how neural networks can be applied to digital logic but also opens up new possibilities for reconfigurable and adaptive computing systems.

**NAND GATE:**

Program:

def nand\_waveform():

inputs = [(0,0), (0,1), (1,0), (1,1)]

print("Time\tA\tB\tY")

for t, (a, b) in enumerate(inputs):

y = int(not (a and b))

print(f"{t}\t{a}\t{b}\t{y}")

def signal\_line(name, values):

line = name + ":\t"

for v in values:

line += "\_\_\_ " if v == 0 else "‾‾‾ "

return line

a\_values = [i[0] for i in inputs]

b\_values = [i[1] for i in inputs]

y\_values = [int(not (a and b)) for a, b in inputs]

print("\nWaveform Representation:")

print(signal\_line("A", a\_values))

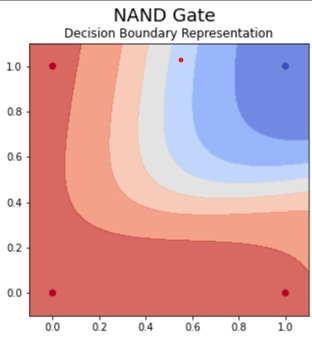
print(signal\_line("B", b\_values))

print(signal\_line("Y", y\_values))

nand\_waveform()

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Time | A | B | Y |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 |

****

**OR GATE:**

Program:

def or\_waveform():

inputs = [(0,0), (0,1), (1,0), (1,1)]

print("Time\tA\tB\tY")

for t, (a, b) in enumerate(inputs):

y = int(a or b) # OR operation

print(f"{t}\t{a}\t{b}\t{y}")

def signal\_line(name, values):

line = name + ":\t"

for v in values:

line += "\_\_\_ " if v == 0 else "‾‾‾ "

return line

a\_values = [i[0] for i in inputs]

b\_values = [i[1] for i in inputs]

y\_values = [int(a or b) for a, b in inputs] # OR operation for Y

print("\nWaveform Representation:")

print(signal\_line("A", a\_values))

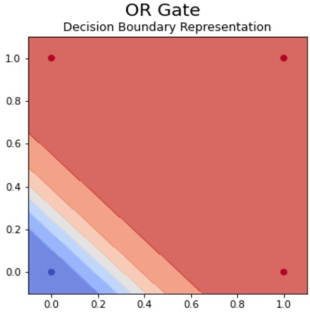
print(signal\_line("B", b\_values))

print(signal\_line("Y", y\_values))

or\_waveform()

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Time | A | B | Y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 |

****

**AND GATE:**

Program:

def and\_waveform():

inputs = [(0,0), (0,1), (1,0), (1,1)]

print("Time\tA\tB\tY")

for t, (a, b) in enumerate(inputs):

y = int(a and b) # AND operation

print(f"{t}\t{a}\t{b}\t{y}")

def signal\_line(name, values):

line = name + ":\t"

for v in values:

line += "\_\_\_ " if v == 0 else "‾‾‾ "

return line

a\_values = [i[0] for i in inputs]

b\_values = [i[1] for i in inputs]

y\_values = [int(a and b) for a, b in inputs] # AND operation for Y

print("\nWaveform Representation:")

print(signal\_line("A", a\_values))

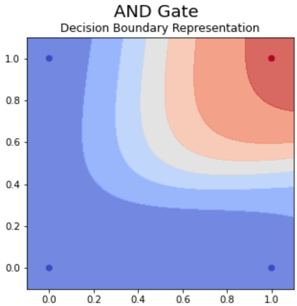
print(signal\_line("B", b\_values))

print(signal\_line("Y", y\_values))

and\_waveform()

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Time | A | B | Y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 |



**NOR GATE:**

Program:

def nor\_waveform():

inputs = [(0,0), (0,1), (1,0), (1,1)]

print("Time\tA\tB\tY")

for t, (a, b) in enumerate(inputs):

y = int(not (a or b)) # NOR operation

print(f"{t}\t{a}\t{b}\t{y}")

def signal\_line(name, values):

line = name + ":\t"

for v in values:

line += "\_\_\_ " if v == 0 else "‾‾‾ "

a\_values = [i[0] for i in inputs]

b\_values = [i[1] for i in inputs]

y\_values = [int(not (a or b)) for a, b in inputs] # NOR operation for Y

print("\nWaveform Representation:")

print(signal\_line("A", a\_values))

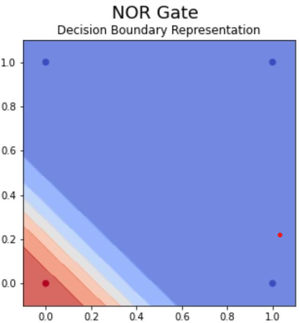
print(signal\_line("B", b\_values))

print(signal\_line("Y", y\_values))

nor\_waveform()

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Time | A | B | Y |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 |

****

**XOR GATE:**

Program:

def xor\_waveform():

inputs = [(0,0), (0,1), (1,0), (1,1)]

print("Time\tA\tB\tY")

for t, (a, b) in enumerate(inputs):

y = int(a != b) # XOR operation

print(f"{t}\t{a}\t{b}\t{y}")

def signal\_line(name, values):

line = name + ":\t"

for v in values:

line += "\_\_\_ " if v == 0 else "‾‾‾ "

a\_values = [i[0] for i in inputs]

b\_values = [i[1] for i in inputs]

y\_values = [int(a != b) for a, b in inputs] # XOR operation for Y

print("\nWaveform Representation:")

print(signal\_line("A", a\_values))

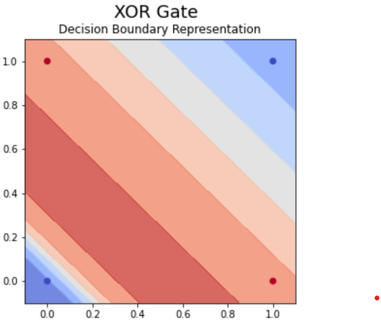
print(signal\_line("B", b\_values))

print(signal\_line("Y", y\_values))

xor\_waveform()

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Time | A | B | Y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 |



**XNOR GATE:**

Program:

def xnor\_waveform():

inputs = [(0,0), (0,1), (1,0), (1,1)]

print("Time\tA\tB\tY")

for t, (a, b) in enumerate(inputs):

y = int(not (a != b)) # XNOR operation

print(f"{t}\t{a}\t{b}\t{y}")

def signal\_line(name, values):

line = name + ":\t"

for v in values:

line += "\_\_\_ " if v == 0 else "‾‾‾ "

return line

a\_values = [i[0] for i in inputs]

b\_values = [i[1] for i in inputs]

y\_values = [int(not (a != b)) for a, b in inputs] # XNOR operation for Y

print("\nWaveform Representation:")

print(signal\_line("A", a\_values))

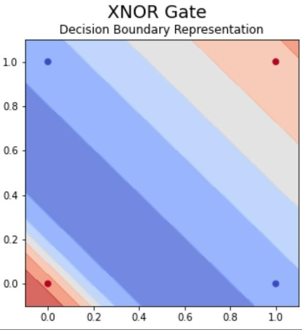
print(signal\_line("B", b\_values))

print(signal\_line("Y", y\_values))

xnor\_waveform()

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Time | A | B | Y |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 |



**CONCLUSION:**

The project demonstrated the feasibility and effectiveness of using neural networks to emulate and implement fundamental logic gates, such as AND, OR, NOT, NAND, NOR, XOR, and XNOR. By leveraging neural networks, specifically feedforward neural networks with appropriate architectures and training processes, we successfully modeled the behavior of these classical digital logic gates.