Neural network representation of Boolean circuits

Shouvik Sur

2026-10-09

Table of contents

1 Neural representations of 2-bit logic gates **1**
1.1 SLP representation of Linear Threshold Functions . . . 2
1.1.1 AND . . . 2
1.1.2 OR . . . 3
1.1.3 NAND . . . 3
1.1.4 NOR . . . 4
1.2 SLP cannot represent XOR . . . 4
1.2.1 A minimal MLP for XOR . . . 5
1.2.2 MLP vs. universal gate representations . . . 5

2 Neural representations of 3-bit logic gates **6**

1 Neural representations of 2-bit logic gates

Most standard 2-bit gates can be represented by a single layer perceptron (SLP). These include the AND, OR, NAND, and NOR gates. This can be understood geometrically by considering the 2-bit dataset $\mathcal{S} = \{0,1\}^2$ as coordinates of the corners of a square. These four gates classify $\mathcal{S}$ by "drawing" *one* decision boundary.

By contrast, the classification performed by XOR will require two decision boundaries. Consequently, XOR cannot be represented by an SLP and the minimal neural network requires one hidden layer with two neurons. This results in a depth 2 neural circuit for representing and learning XOR.

Interestingly, since NAND is a universal gate, it is possible to represent XOR by a circuit of four NAND gates. This circuit of logical gates, however, has depth 3. Here, we show how to negotiate

this mismatch of circuit depth, and in the process develop insights into compressing logical-gate circuits to match the minimal MLP implementations of the same function. We anticipate such compressions are maximal for tasks that are "hard" for logical gates but "easy" for MLPs due to the inherent non-linearity of the latter.

1.1 SLP representation of Linear Threshold Functions

A linear threshold function (LTF) performs a linear transformation on its inputs and outputs a binary result based on a threshold condition. `AND`, `OR`, `NAND`, and `NOR` gates are examples of LTFs. In order to verify the statement, let us obtain the functional representation of these gates. We will consider a function $f(\vec{x}) = \Theta(\vec{w} \cdot \vec{x} + b)$ with $\vec{w}$ and b being parameters, Θ the Heaviside step function, and the two dimensional vector $\vec{x} \in \mathcal{S}$. Note that $f(\vec{x})$ is an SLP with a single neuron and two inputs, characterized by the weights w_1 and w_2, and the bias b. By utilizing the truth table of the respective gates, we can place restrictions on $\vec{w}$ and b as follows.

1.1.1 `AND`

Truth table:

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Thus, $f(\vec{x})$ must satisfy

- $f((0,0)) < 0 \Rightarrow b < 0$.
- $f((0,1)) < 0 \Rightarrow w_2 + b < 0 \Rightarrow w_2 < -b$.
- $f((1,0)) < 0 \Rightarrow w_1 < -b$.
- $f((1,1)) > 0 \Rightarrow w_1 + w_2 > -b$.

Since these conditions are mutually compatible, `AND` can be represented by the two-neuron SLP $f(\vec{x})$. Therefore, a simple functional representation would correspond to $\vec{w} = (1,1)$ and $b = -3/2$, which implies $f(\vec{x}) = \Theta(x_1 + x_2 - 3/2)$.

1.1.2 OR

Truth table:

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Thus, $f(\vec{x})$ must satisfy

- $f((0,0)) < 0 \Rightarrow b < 0$.
- $f((0,1)) > 0 \Rightarrow w_2 > -b$.
- $f((1,0)) > 0 \Rightarrow w_1 > -b$.
- $f((1,1)) > 0 \Rightarrow w_1 + w_2 > -b$.

These conditions are trivially satisfied and a simple functional representation would correspond to $\vec{w} = (1,1)$ and $b = -1/2$, which implies $f(\vec{x}) = \Theta(x_1 + x_2 - 1/2)$.

1.1.3 NAND

Truth table:

x_1	x_2	$\overline{x_1 \wedge x_2} = \bar{x}_1 \vee \bar{x}_2$
0	0	1
0	1	1
1	0	1
1	1	0

Thus, $f(\vec{x})$ must satisfy

- $f((0,0)) > 0 \Rightarrow b > 0$.
- $f((0,1)) > 0 \Rightarrow w_2 > -b$.
- $f((1,0)) > 0 \Rightarrow w_1 > -b$.
- $f((1,1)) < 0 \Rightarrow w_1 + w_2 < -b$.

Satisfied. A simple functional representation would correspond to $\vec{w} = (-1,-1)$ and $b = 3/2$, which implies $f(\vec{x}) = \Theta(-x_1 - x_2 + 3/2)$.

1.1.4 NOR

Truth table:

x_1	x_2	$\overline{x_1 \vee x_2} = \bar{x}_1 \wedge \bar{x}_2$
0	0	1
0	1	0
1	0	0
1	1	0

Thus, $f(\vec{x})$ must satisfy

- $f((0,0)) > 0 \Rightarrow b > 0$.
- $f((0,1)) < 0 \Rightarrow w_2 < -b$.
- $f((1,0)) < 0 \Rightarrow w_1 < -b$.
- $f((1,1)) < 0 \Rightarrow w_1 + w_2 < -b$.

Satisfied. A simple functional representation would correspond to $\vec{w} = (-1,-1)$ and $b = 1/2$, which implies $f(\vec{x}) = \Theta(-x_1 - x_2 + 1/2)$.

1.2 SLP cannot represent XOR

The truth table of XOR

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

constrains the parameters of the SLP/LTF as follows

- $f((0,0)) < 0 \Rightarrow b < 0$,
- $f((0,1)) > 0 \Rightarrow w_2 > -b$,
- $f((1,0)) > 0 \Rightarrow w_1 > -b$,
- $f((1,1)) < 0 \Rightarrow w_1 + w_2 < -b$.

These conditions are mutually incompatible, which implies $f(\vec{x})$ cannot represent XOR. This necessitates the introduction of a hidden layer in the neural network, promoting the SLP to an MLP.

1.2.1 A minimal MLP for XOR

The weights of the hidden layer with two neurons is encoded by the 2×2 matrix $\hat{W}^{(1)}$, and the bias by $\vec{b}^{(1)}$, such that the logit is given by $\vec{z}^{(1)} = \hat{W}^{(1)}\vec{x} + \vec{b}^{(1)}$. The output layer operates on $x_i^{(1)} = \Theta(z_i^{(1)})$ and produces the logit $z^{(2)} = \hat{W}^{(2)}\vec{x}^{(1)} + b^{(2)}$, where $\hat{W}^{(2)}$ is a 1×2 matrix (i.e. a row vector). Therefore, the final output is

$$f(\vec{x}) = \Theta\left(\sum_{i=1,2} W_i^{(2)}\Theta\left(W_{i1}^{(1)}x_1 + W_{i2}^{(1)}x_2 + b_n^{(1)}\right) + b^{(2)}\right) \tag{1}$$

A solution for the parameters of the MLP that allows $f(\vec{x})$ to represent XOR is

$$\hat{W}^{(1)} = \begin{bmatrix} 2 & 2 \\ 1/2 & 1/2 \end{bmatrix}; \qquad \vec{b}^{(1)} = \begin{bmatrix} -1 \\ -3/4 \end{bmatrix}; \qquad \hat{W}^{(2)} = \begin{bmatrix} 3/2 & -1 \end{bmatrix}; \qquad b^{(2)} = -1 \tag{2}$$

which leads to

$$f(\vec{x}) = \Theta\left(\frac{3}{2}\Theta\left(2(x_1 + x_2) - 1\right) - \Theta\left((x_1 + x_2)/2 - 3/4\right) - 1\right)$$

In PyTorch this MLP is given by

```
class XORModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.main=nn.Sequential(
            nn.Linear(2, 2),
            nn.ReLU(),
            nn.Linear(2, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.main(x)
```

1.2.2 MLP vs. universal gate representations

Four NAND gates represent XOR as

$$x_1 \oplus x_2 = \overline{y_1 \wedge y_2} \tag{3}$$

where $y_n = \overline{\overline{x_1 \wedge x_2} \wedge x_n}$. The maximum order of nesting of the Boolean operations in the expression of $x_1 \oplus x_2$ corresponds to the number of layers. Naively, since each NAND gate

correspond to an LTF and each LTF corresponds to a layer of an MLP, it would appear that one needs a three-layered MLP to represent `XOR`.

The redundancy is eliminated by noticing that y_n is, in fact, representable by a single LTF, not a nested LTF. To see this, we simplify y_n as

$$\begin{aligned} y_n &= \overline{\overline{x_1 \wedge x_2} \wedge x_n} \\ &= \overline{(\bar{x}_1 \wedge x_n) \wedge (\bar{x}_2 \wedge x_n)} \\ &= \overline{(\bar{x}_2 \wedge x_1)}\ \delta_{n,1} + \overline{(\bar{x}_1 \wedge x_2)}\ \delta_{n,2} \\ &= (x_2 \vee \bar{x}_1)\ \delta_{n,1} + (x_1 \vee \bar{x}_2)\ \delta_{n,2} \\ &= f(x_2 - x_1 + 1/2)\ \delta_{n,1} + f(x_1 - x_2 + 1/2)\ \delta_{n,2}. \end{aligned} \tag{4}$$

Therefore, applying the reduction in Equation 4 to Equation 3 we find that `XOR` indeed requires a single nesting of LTFs and the circuit depth matches that obtained with the MLP representation of `XOR`.

2 Neural representations of 3-bit logic gates

Generalizing beyond the 2-bit operations, here, we consider the `Toffoli` and `Fredkin` gates. These gates are not representable by a single LTF, and, thus, are higher dimensional generalizations of `XOR`. However, these gates are *invertible* as they have three inputs and three outputs, unlike `XOR`. We will see that this feature introduces additional complexity in their representations.

[to be continued ...]