A Step-by-Step Tutorial on the NAND Function Approximation Using a Bipolar Perceptron

Mario Pascual González

University of Málaga

November 9, 2023

Abstract

This paper presents a detailed account of a task executed for the Intelligent Systems course within the Bioinformatics degree at the University of Málaga, where the objective was to demonstrate the learning dynamics of a single-layer perceptron by approximating the functionality of the NAND gate. The NAND gate, an elemental component in digital systems, known for its universality in computing, serves as an ideal candidate for such a study due to its clear and simple logical operation. By simulating this gate, we not only reinforced the fundamental concepts underpinning electronic logic gates but also provided a platform for understanding the learning mechanisms of more intricate neural networks.

The perceptron implemented in this task was of a single-layer variety, with two inputs and a bipolar output, using the sign function as the activation. It was meticulously trained to mirror the NAND logic, with its evolution over iterations captured visually to illustrate the perceptron's adaptability and the incremental refinement of its decision-making capability.

This study underscores the importance of foundational models like the perceptron in illuminating the pathway to understanding complex neural network architectures and their learning behaviors, offering invaluable insights into the initial steps of neural computation.

Keywords: Perceptron, NAND Gate, Single-Layer Neural Model, Computational Logic

1 Introduction

In the landscape of digital electronics, the NAND gate occupies a position of critical importance due to its universal functionality; any Boolean function can be constructed exclusively with NAND gates, highlighting their capacity for forming the foundational building blocks of digital systems [1].

The operation of the NAND gate is so fundamental that it is often dubbed as the "universal gate," playing a pivotal role in the synthesis of arithmetic logic units, memory storage devices, and a myriad of other digital circuits. Consequently, the ability to model and understand the NAND gate's operation through various computational approaches not only solidifies the core principles of electronic design but also bridges the cognitive gap between logical theory and practical application.

This work presents a compelling exploration of the single-layer perceptron's ability to approximate the functionality of a NAND gate, thereby serving as a microcosm of the perceptron's potential in emulating logical operations and decision-making processes [2].

By delving into the perceptron learning algorithm's iterative refinement of decision boundaries, we demonstrate the perceptron's capacity to categorize and process information in a binary fashion akin to electronic gates.

This analogy transcends a mere academic exercise, unraveling the perceptron's prospective applications in complex problem-solving and its potential as a tool for innovation in bioinformatics, Machine Learning (ML), and Artificial Intelligence (AI).

2 Background

2.1 Perceptron

In the realm of Neural Networks (NN), the perceptron is recognized as the simplest type of Artificial Neuron (AN). The iteration in this study leverages a Single-Layer Perceptron (SLP), which is the fundamental unit of a neural network, consisting of input neurons connected to an output node via a series of weights and a bias. This configuration forms the basis of a perceptron designed to model binary functions, where its learning capacity is limited to problems that are linearly separable [3].

The perceptron employed here is designed to process two input values, denoted as x_1 and x_2, which are real numbers representing features in a dataset. These inputs are fed into a linear combination with corresponding weights w_1 and w_2, summed together with a bias term b. The mathematical representation of the

perceptron's output y before activation is given by the linear combination:

$$y_{in} = w_1 x_1 + w_2 x_2 + b$$

Following the linear summation, the output is passed through an activation function. For the perceptron in this study, the activation function is the sign function, defined as:

$$sign(y_{in}) = \begin{cases} -1 & \text{if } y_{in} < 0 \\ 1 & \text{otherwise} \end{cases}$$

The activation function transforms the weighted input into a binary output that corresponds to the binary states of a neuron: active or inactive, designated by the bipolar states {-1, 1}. In the context of digital logic, this bipolar output is analogous to the binary states of a logic gate, with -1 representing a logical 0 and 1 representing a logical 1.

The perceptron's learning algorithm iteratively adjusts the weights and bias based on the difference between the predicted output and the actual label of the input data. The adjustments are guided by the learning rate η, a hyperparameter that dictates the magnitude of the weight update. The learning rule for the perceptron modifies the weights and bias as follows:

$$w_i(new) = w_i(old) + \eta \cdot (y - y_{pred}) \cdot x_i$$

$$b(new) = b(old) + \eta \cdot (y - y_{pred})$$

where y_{pred} is the output predicted by the perceptron before the update, and y is the true label of the current training example.

2.2 NAND Function

The logical NAND function is a fundamental binary operation that outputs a false value (or logical 0) only when all its inputs are true (or logical 1); in all other cases, it outputs a true value (or logical 1). Mathematically, the NAND operation can be expressed as the negation of the AND operation. For a two-input system, the truth table for the NAND function is as follows:

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

In this table, A and B represent the binary inputs, and A NAND B represents the output. When translating this binary logic to the bipolar system used by our perceptron model, where the binary values $0, 1$ correspond to the bipolar values $-1, 1$ respectively, the NAND function's bipolar output can be represented with the following modified truth table:

A'	B'	A' NAND B'
-1	-1	1
-1	1	1
1	-1	1
1	1	-1

In this bipolar representation, A' and B' are the bipolar inputs, and A' NAND B' is the bipolar output. The conversion from binary to bipolar forms aligns with the perceptron's activation states and facilitates the application of linear classification methods used in machine learning.

To mathematically define the NAND function for the bipolar inputs, let's consider input x_1 and x_2 taking values from the set {-1, 1}. The output y using the sign function can be given by the inequality that if both x_1 and x_2 are 1, the output is -1; otherwise, the output is 1. We can express this relationship as:

$$y = \text{sign}(-x_1 \cdot x_2)$$

The sign function ensures that only when both inputs are positive, the output becomes negative. This expression is a direct translation of the NAND logic into the perceptron learning context and sets the stage for implementing a perceptron that imitates the NAND gate's functionality.

3 Methodology

3.1 NAND Data Generation

The methodology for the simulation of the NAND gate using a perceptron begins with the generation of appropriate training data.
The input data consists of all possible pairs of bipolar binary inputs that a two-input NAND gate can have. In the bipolar system, the binary values 0 and 1 are represented as -1 and 1, respectively, aligning with the perceptron's activation states.
The output for each input pair is determined by the NAND gate's truth table. The output is 1 for all cases except when both inputs are 1, which corresponds to an output of -1.

```
inputs = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
outputs = [1, 1, 1, -1]
```

A pandas DataFrame NAND is constructed using these lists, where 'Input A' and 'Input B' are designated as the DataFrame columns. An additional column 'Output' stores the bipolar outputs corresponding to each input pair.

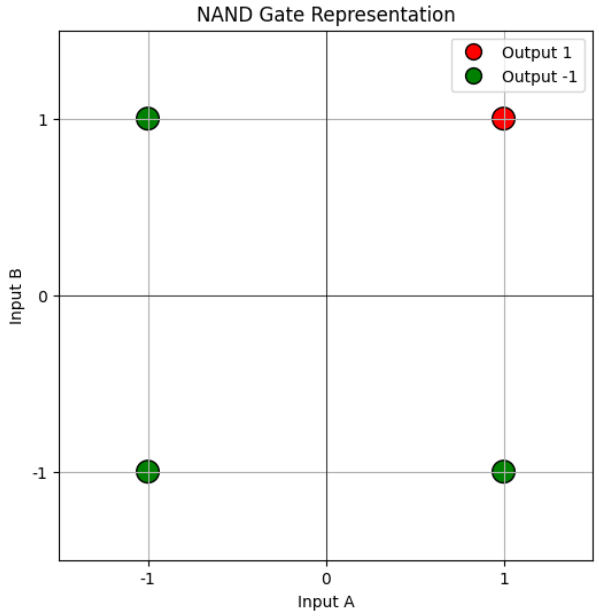

Figure 1: NAND input data for the SLP model implemented

3.2 Perceptron workflow

The training workflow of the perceptron model is sequential and iterative. The perceptron is presented with one training example at a time, which consists of a pair of bipolar inputs from the NAND gate's truth table. For each training instance, the perceptron performs the following steps:

1. *Calculation of the Predicted Output.* The perceptron computes the weighted sum of the inputs and adds a bias term. The predicted output, y_{pred}, is then passed through the sign activation function to yield a binary output in the bipolar scheme {-1, 1}.

2. *Comparison and Update.* The perceptron's predicted output is compared with the expected output. If the predicted output matches the expected output, the model's parameters (weights and bias) remain unchanged. However, if there is a discrepancy, indicating an error in the prediction, the perceptron's weights and biases are updated following the previously stipulated learning rule.

3.3 Perceptron fitting to NAND function

The initial weights assigned to the perceptron are $w_1 = -0.00537144$, $w_2 = -0.00890953$, and the initializing bias term is $b = -0.00950643$. A learning rate proximate to zero is imperative to ensure meticulous training of the perceptron, thereby preventing the oversight of significant fitting outcomes. The learning rate selected for this purpose is $\eta = 0.01$.

Iteration 1. The input data is $A' = [-1, -1]$, the expected output is $y = 1$. First, we calculate the perceptron raw output before applying the activation function:

$$y_{in} = (A_1 \times w_1) + (A_2 \times w_2) + b =$$

$$(-1 \times -0.00537144) + (-1 \times -0.00890953) - 0.00950643$$

$$y_{in} = 0.00477454$$

Running the activation function $sign(y_{in})$ through that value would result in $y_{pred} = 1$. The prediction matches the expected output, so no update is made.

The decision boundary for this iteration is finally included.

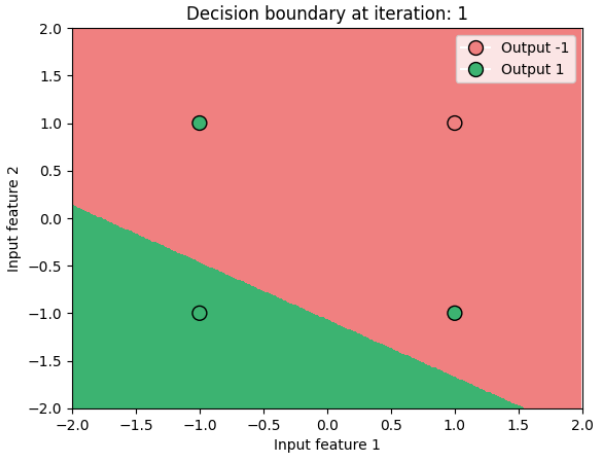

Figure 2: Perceptron decision boundary for iteration 1

Iteration 2. The input data is $A' = [-1, 1]$, the expected output is $y = 1$. First, we calculate the perceptron raw output before applying the activation function:

$$y_{in} = (A_1 \times w_1) + (A_2 \times w_2) + b =$$

$$(-1 \times -0.00537144) + (1 \times -0.00890953) - 0.00950643$$

$$y_{in} = -0.01304452$$

Running the activation function $sign(y_{in})$ through that value would result in $y_{pred} = -1$. The prediction does not match the expected output, so we update the weights and bias:

$$w_1(new) = w_1(old) + \eta \times (y - y_{pred}) \times A'_1 =$$

$$= -0.00537144 + 0.01 \times (1 - (-1)) \times (-1)$$

$$w_1(new) = -0.00537144 - 0.02 = -0.02537144$$

$$w_2(new) = w_2(old) + \eta \times (y - y_{pred}) \times A'_2 =$$

$$= -0.00890953 + 0.01 \times (1 - (-1)) \times (1)$$

$$w_2(new) = -0.00890953 + 0.02 = 0.01109047$$

$$b(new) = b(old) + \eta \times (y - y_{pred}) =$$

$$= -0.00950643 + 0.01 \times (1 - (-1))$$

$$b(new) = -0.00950643 + 0.02 = 0.01049357$$

Now that the weights and biases have been updated, the decision boundary of the perceptron has changed. The new decision boundary is included bellow.

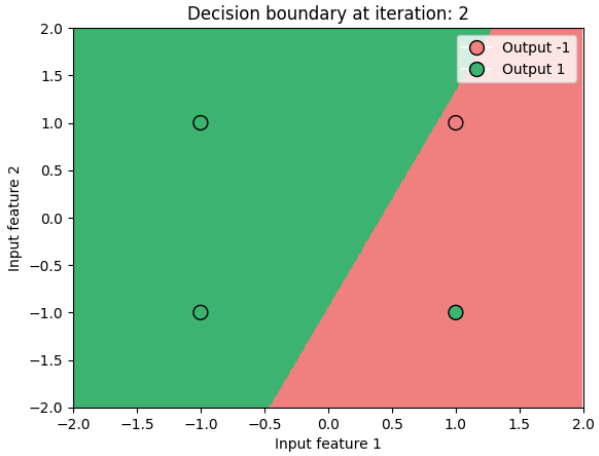

Figure 3: Perceptron decision boundary for iteration 2

Iteration 3. The input data is $A' = [1, -1]$, the expected output is $y = 1$. First, we calculate the perceptron raw output before applying the activation function:

$$y_{in} = (A_1 \times w_1) + (A_2 \times w_2) + b =$$

$$(1 \times -0.02537144) + (-1 \times 0.01109047) + 0.01049357$$

$$y_{in} = -0.02596834$$

Running the activation function $sign(y_{in})$ through that value would result in $y_{pred} = -1$. The prediction does not match the expected output, so we update the weights and bias:

$$w_1(new) = w_1(old) + \eta \times (y - y_{pred}) \times A'_1 =$$

$$= -0.02537144 + 0.01 \times (1 - (-1)) \times (1)$$

$$w_1(new) = -0.02537144 + 0.02 = -0.00537144$$

$$w_2(new) = w_2(old) + \eta \times (y - y_{pred}) \times A'_2 =$$

$$= 0.01109047 + 0.01 \times (1 - (-1)) \times (-1)$$

$$w_2(new) = 0.01109047 - 0.02 = -0.00890953$$

$$b(new) = b(old) + \eta \times (y - y_{pred}) =$$

$$= 0.01049357 + 0.01 \times (1 - (-1))$$

$$b(new) = 0.01049357 + 0.02 = 0.03049357$$

Upon observation of the mathematical demonstration detailing the weight and bias updates we may notice that the weights are consistent with the initial values, while the bias has undergone modification. Consequently, this indicates that the decision boundary retains its original configuration post the initial iteration; however, there is a translational shift, potentially rendering it imperceptible.

This decision boundary is included bellow:

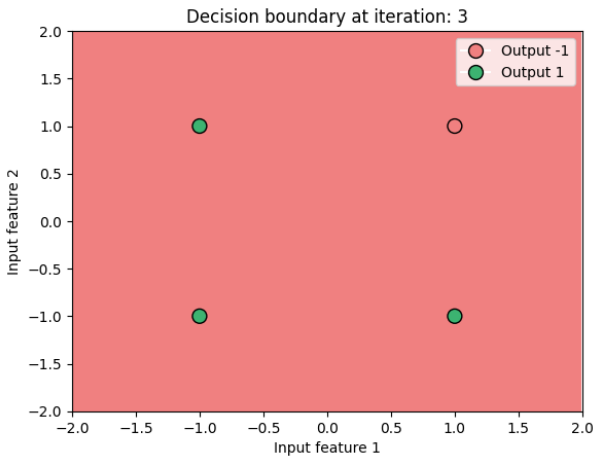

Figure 4: Perceptron decision boundary for iteration 3

Iteration 4. The input data is $A' = [1, 1]$, the expected output is $y = -1$. First, we calculate the perceptron raw output before applying the activation function:

$$y_{in} = (A_1 \times w_1) + (A_2 \times w_2) + b =$$

$$(1 \times -0.00537144) + (1 \times -0.00890953) + 0.03049357$$

$$y_{in} = 0.01621260$$

Running the activation function $sign(y_{in})$ through that value would result in $y_{pred} = -1$. The prediction does not match the expected output, so we update the weights and bias:

$$w_1(new) = w_1(old) + \eta \times (y - y_{pred}) \times A'_1$$

$$w_1(new) = -0.00537144 + 0.01 \times (-1 - 1) \times 1$$

$$w_1(new) = -0.00537144 - 0.02 = -0.02537144$$

$$w_2(new) = w_2(old) + \eta \times (y - y_{pred}) \times A'_2$$

$$w_2(new) = -0.00890953 + 0.01 \times (-1 - 1) \times 1$$

$$w_2(new) = -0.00890953 - 0.02 = -0.02890953$$

$$b(new) = b(old) + \eta \times (y - y_{pred})$$

$$b(new) = 0.03049357 + 0.01 \times (-1 - 1)$$

$$b(new) = 0.03049357 - 0.02 = 0.01049357$$

Now that the weights and biases have been updated, the decision boundary of the perceptron has changed. The new decision boundary is included bellow.

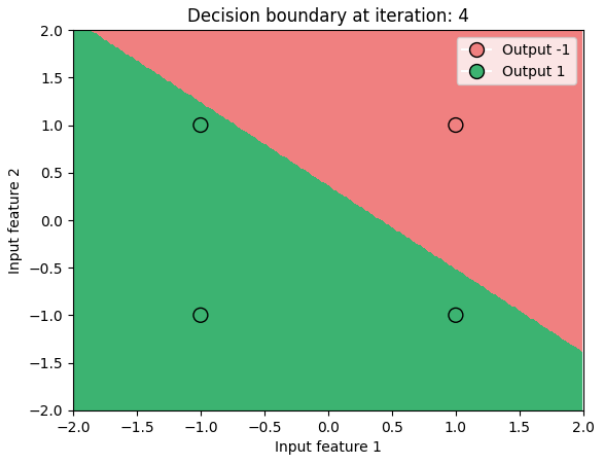

Figure 5: Final decision boundary for the NAND function

4 Results

The perceptron learning algorithm was tasked with modeling the NAND logical function, a cornerstone in digital computing, by iteratively adjusting its decision boundary to correctly classify input pairs. Over successive iterations, the algorithm refined the perceptron's parameters, aiming to align the decision boundary with the expected outputs of the NAND operation in a bipolar scheme.

The final iteration, as depicted in Figure 5, demonstrates a perfectly delineated decision boundary that successfully segregates the input space into two distinct regions, each corresponding to one of the NAND function's output classes.

5 Discussion

The progression of the perceptron's learning dynamics over the course of four iterations is both enlightening and demonstrative of the adaptability inherent within even the simplest of neural network models. Initially, the decision boundary is incorrectly positioned, misclassifying certain inputs (Iteration 1). This misplacement is characteristic of the early stages of perceptron training, where the randomly initialized parameters are yet to be refined through exposure to the training data.

As training progresses to Iteration 2, the decision boundary undergoes a significant shift. This abrupt transition is a direct consequence of the perceptron adjusting its weights and bias in response to the misclassified input pairs.

By Iteration 3, an interesting development occurs—the decision boundary has now moved to a position where all inputs are classified as the same class. This iteration underscores a critical phase in the learning process where the model, after overcompensating for errors in previous iterations, begins to consolidate its understanding of the classification task, albeit not perfectly.

This exemplifies the perceptron's tendency to oscillate before stabilizing, especially when dealing with really small datasets where each input has a substantial impact on the learning process.

The final iteration, Iteration 4, presents a successful learning outcome, with the decision boundary correctly classifying all input pairs. This accomplishment not only signifies the perceptron's ability to internalize the logic of the NAND gate but also exemplifies the convergence properties of the perceptron learning algorithm for linearly separable functions.

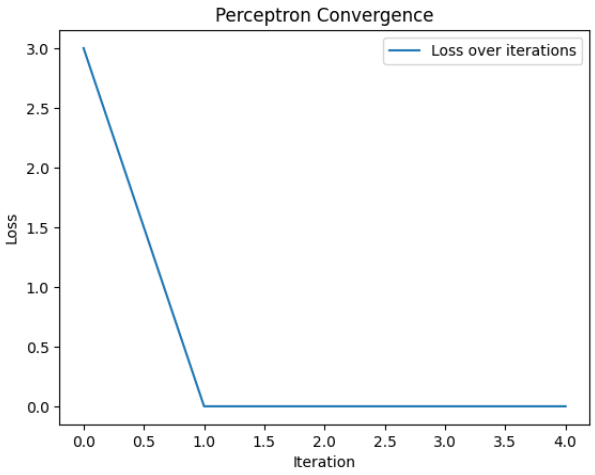

Figure 6: Loss function for 5 epochs

In order to check if the perceptron's loss function converges after one iteration for the initial weights, bias and learning rate conditions, we will run the perceptron algorithm over the same NAND dataset for five epochs.

The Figure 6 backs up our calculations by showing that after just a single complete iteration over the dataset, the perceptron achieves an optimal state, classifying all samples according to the NAND logic with no further loss.

The plot underscores the perceptron's efficiency in learning linearly separable functions, where, once the correct decision boundary is established, the model requires no additional adjustments to maintain its accuracy. This behavior validates the single-layer perceptron's capacity as a robust yet simple binary classifier, capable of swiftly converging to a solution within the confines of a clearly defined logical function like NAND.

6 Conclusion

The observed fluctuations in the decision boundary position across iterations highlight the iterative nature of the learning process within neural networks.

The results of this study serve as a microcosm for the perceptron's broader application to binary classification tasks. This emphasizes the importance of understanding learning dynamics, the potential need for hyperparameter tuning, and the effective simplicity of the perceptron in capturing linear relationships within data, which, believe it or not, is the foundational

bedrock upon which more complex neural network architectures are built.

7 Appendix

The complete code for the project, including the Jupyter notebook and associated visualizations, is available for review and further experimentation. It can be found in the GitHub repository at: https://github.com/Mawio02/NAND-Function-Single-Layer-Perceptron-fitting. Readers are encouraged to download, fork, or contribute to the repository to build upon the work presented in this study.

Acknowledgements

I extend my gratitude to Dr. Ezequiel López Rubio and Dr. Enrique Domínguez Merino for their invaluable guidance and insightful teachings during the writing of this project for the Intelligent Systems course. I am truly appreciative of their patience and commitment to fostering a profound learning experience.

References

[1] M. M. Mano and M. D. Ciletti, *Digital Design*, 3rd ed. Prentice Hall, 2001, chapter 2 covers the importance of the NAND gate and its universality in digital circuits.

[2] M. Minsky and S. Papert, "Perceptrons: An introduction to computational geometry," 1969, the foundational text by Minsky and Papert discusses the capabilities and limits of perceptrons, including their application to logical functions.

[3] M.-A. Mendez Lucero, R.-M. Karampatsis, E. Bojorquez Gallardo, and V. Belle, "Signal perceptron: On the identifiability of boolean function spaces and beyond," *Frontiers in Artificial Intelligence*, vol. 5, 2022. [Online]. Available: https://doi.org/10.3389/frai.2022.770254