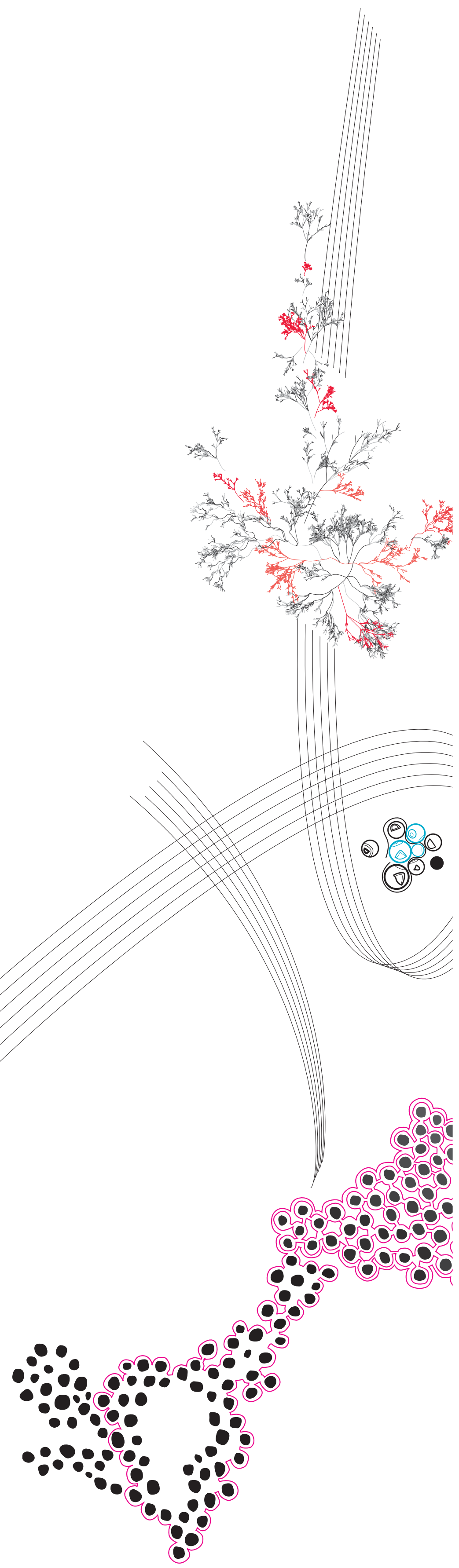
BSc Thesis Applied Mathematics



A minimal nanoneuron capable of nonlinear classification

Thai Ha Bui

Supervisor: prof.dr.ir. B.J. Geurts, prof.dr.ir. M. van Keulen

August, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

First of all, I would like to thank my mom for being incredibly supportive both mentally and financially during my Bachelor studies, without her, I would not be here. I would like to give special thanks to my supervisors prof.dr.ir. Bernard Geurts for fruitful discussions and guidance during my thesis, and prof.dr.ir. Maurice van Keulen for helpful counselling about neural networks. I would also like to thank Hans-Christian Ruiz Euler, he was very excited to see someone working on his previous work and did not hesitate to provide advice when I contacted him. My work is a result of many meetings with other researchers and students who worked on this journey before me, namely prof. Wilfred van der Wiel, Marlon Backer and Bram de Wilde, so thank you to all of you as well. I would also like to express my appreciation for the University of Twente for providing me with a great academic environment and preparing me for writing this thesis. Lastly, I am grateful for the connections I have made during my years in Enschede, these people have inspired me tremendously in many ways.

Throughout the work on this project, I have gained valuable insights into academic research and what it means to work in an academic environment. While three months is insufficient to encapsulate the entirety of a research career, it provided a good introduction. I realized that coding is a fundamental component of any future endeavor I undertake. However, I noticed a significant amount of time could be saved if researchers adhered to a consistent coding framework, which is often not the case. This observation is probably not one of my own, but it emphasized the importance of structured and efficient coding practices.

I also learned the importance of independence in research. Being proactive in seeking guidance and gathering necessary materials enabled me to navigate and understand large codebases effectively. The advancements in AI have also been beneficial, helping me search for information and work more efficiently. While AI accelerated my workflow, the most crucial aspects of research, like formulating hypotheses, designing experiments, interpreting results, still require human input and cannot be fully automated, yet. If one falsely relies on AI and 'dreams on' with it, at least with the current state of AI, I believe it could bring harm to science.

Furthermore, I am grateful for the strong mathematical foundation I possess. It allowed me to grasp the underlying context of problems from different perspectives. This experience has gave me a peek in research and prepared me for future academic and professional pursuits.

Contents

A minimal nanoneuron capable of nonlinear classification

Thai Ha Bui

August, 2024

Abstract

This paper addresses the challenge of optimizing neuromorphic computing hardware by investigating Dopant Network Processing Units (DNPUs). To circumvent the costs and labor-intensive nature of direct fabrication of these devices, we leverage computer simulations and machine learning techniques. Utilizing results from Theuws, we train a surrogate model via machine learning to study the behavior of the smallest theoretically feasible DNPU. Our analysis focuses on the device's stability, robustness, and its ability to implement Boolean functions, with an emphasis on the non-linear XOR gate. The results demonstrate that DNPUs can efficiently perform complex computations with minimal size parameters, promising their potential for energy-efficient neural network hardware.

Keywords: neuromorphic computing, dopant network, machine learning, nanoelectric device, ai hardware, kinetic monte carlo, neural network, neural network emulation

1 Introduction

Recent AI advances have raised demands for better hardware that would best accommodate AI models. We are particularly interested in energy efficiency and computational speed. There have been numerous attempts in developing hardware specialized for AI. Neuromorphic computing (NC) is an approach to designing computer architecture with inspiration from human brain, which is not only extremely energy efficient, but also incredibly spatially effective. NC was first mentioned in the works of Carver Mead, he asserted in his paper [10] that *biological solutions are many orders of magnitude more effective than those we have been able to implement using digital methods.*

My research focuses on exploring Dopant Network Processing Units (DNPUs), which are nanoelectrical devices that exhibit unique properties that make them suitable for neural network applications. Traditional neural networks rely on numerous interconnected nodes, each performing simple computations and collectively solving complex tasks. The DNPU unit is particularly fascinating due to its nonlinear behavior, which is a result of the disordered dopant network [18]. Recently, these devices developed capable of classifying linearly non-separable data [3], signaling their potential as robust AI hardware. Another study has shown that the key benefit of DNPUs, when applied in hardware, is the substantial reduction in the number of parameters and operations needed, which leads to increased computational efficiency [15].

The conceptualization and creation of such devices requires experimenting with a large amount of parameters, thus direct physical fabrication is cost-prohibitive and labour-intensive. Instead, we can leverage computer simulations to circumvent these challenges.

In this paper, I investigate the effects of size on the performance of the device. I will use results found by Theuws [19], train a surrogate model from simulated data using machine learning techniques and study the behaviour of the smallest DNPU theoretically possible. A throughout analysis of the stability and robustness of the model will be discussed.

1.1 Literature review

Neuromorphic Computing: Neuromorphic computing has emerged as a revolutionary approach to designing computer architecture by drawing inspiration from the human brain, which exhibits exceptional energy efficiency and spatial effectiveness. This concept, first introduced by Carver Mead, asserts that biological solutions far surpass the efficiency of digital methods in various computational tasks [10].

Hopping-Transport Mechanism and Reconfigurable Logic: The hopping-transport mechanism in disordered dopant networks has been identified as a key enabler for reconfigurable logic in DNPUs. Tertilt et al. [18] studies this mechanism through kinetic Monte Carlo simulations, revealing temperature-dependent current-voltage characteristics and the successful artificial evolution of basic Boolean logic gates. The hopping-transport mechanism allows for subtle control over local electrostatic potentials and current flow within the dopant network, enabling the reconfiguration of logic gates. The stochasticity of this hopping-transport mechanism is crucial in the ability of DNPUs to solve nonlinear tasks.

Dopant Network Processing Units (DNPUs): While existing studies have significantly advanced our understanding of DNPUs and their applications, there are several limitations and areas that require further exploration. For instance, the study by Chen et al. [3] and Ruiz-Euler et al. [15] demonstrated the potential of DNPUs in classifying linearly non-separable data, but they primarily focused on larger scale DNPUs and did not explore the limits of miniaturization.

The research by Theuws [19] provided insights into the minimal size parameters required for functional DNPUs, establishing a foundational understanding of their physical properties. However, it lacked an extensive evaluation of the stability and robustness of these minimal configurations in practical applications.

This thesis builds upon these studies by specifically targeting the smallest possible DNPU size and evaluating its performance through kinetic Monte Carlo simulations and machine learning models. By addressing the size effects on DNPUs when minimized, this research aims to contribute a comprehensive understanding of the practical viability and efficiency of minimal DNPUs in neuromorphic computing.

1.2 Overview of paper

The structure of this paper begins with the Preliminaries section 2, which covers foundational concepts, including an overview of Dopant Network Processing Units (DNPUs), the Kinetic Monte Carlo (KMC) simulation method used to model the behavior of the minimal DNPU found my [19], deep learning with feedfoward neural network and linear separability. Next, section 3 delves into the robustness of the KMC simulation and validates its convergence properties to ensure it accurately reflects physical systems. Section 4 describes the creation and training of a surrogate model using machine learning techniques to emulate the DNPU's functionality, including the network architecture and performance evaluation. Following this, section 5 explores the potential of our DNPU to implement basic Boolean functions, focusing on the non-linear XOR gate, and details the network architecture and cost function used for training. Finally, Section 6 discusses the findings, including the implications of the DNPU size on performance, evaluate the experimental feasibility of the surrogate model's predictions, and offer insights into potential work for future research and development.

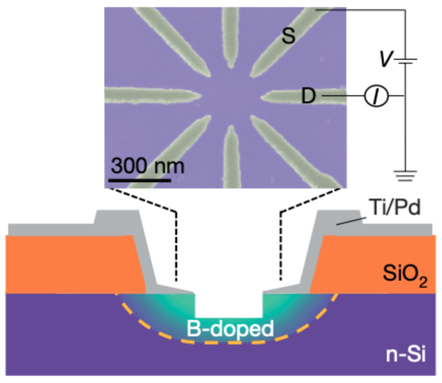
2 Preliminaries

2.1 Dopant Network Processing Units

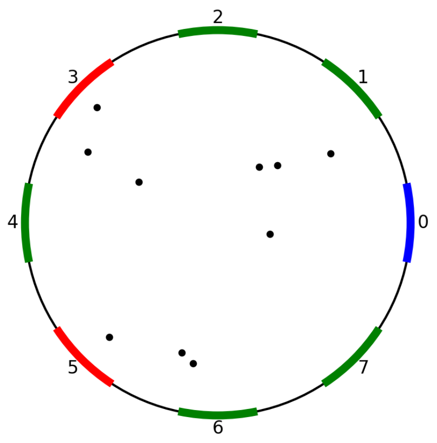
2.1.1 The device

We consider a device (as depicted in Figure 1A), where a silicon substrate ($n-Si$) is doped with p-type material, in our case, with Boron (B-doped). This doping process introduces 'holes' in the semiconductor, which are essentially the absence of electrons in the silicon lattice structure. In semiconductor physics, a hole acts as a positive charge carrier. When an electron in the valence band gains enough energy, it can jump to the conduction band, leaving behind a vacancy in the valence band. This vacancy, or 'hole,' can move through the lattice as neighboring electrons move to fill the vacancy, effectively allowing current to flow. A layer of silicon dioxide (SiO_2) serves as an insulating layer to prevent leakage of charges and to separate the substrate from the processing layers above. And lastly, Titanium/Palladium (Ti/Pd) contacts which serves as electrodes.

The network that we study in this paper can be imagined as in Figure 1B.



(A)



(B)

FIGURE 1: (A) Schematic of the device. Figure adapted from [6]. (B) Schematic of the network. Black dots as dopants (10). Red bars represent input electrodes, blue bar is output electrode and green bars are control electrodes. The numbers are indices of each electrode.

In this paper, we will be exploring the functionality of a DNPU with 10 dopants, 0 donors and circle design with radius 16.77nm. These are the theoretical results obtained in the work of Theuws [19], which are the lowerbound of size parameters, such that the system still works within the physical properties we desire.

2.1.2 Hopping mechanism

In doped semiconductors, charge carriers such as electrons move between localized states, or impurity sites, through a process known as *hopping.* This movement is essential for charge transport, especially in disordered systems where traditional band conduction is inefficient. Hopping occurs when an electron at one impurity site gains sufficient energy to move to a neighboring site. This process is influenced by the spatial separation between the sites and the energy difference between the initial and final states.

The rate at which an electron hops from one site to another is described by the transition rate. This rate is crucial for understanding the electrical properties of doped semiconductors and can be modeled using the Miller-Abrahams formula. This formula takes into account both the distance between impurity sites and the energy difference between the states.

Definition 2.1. *The transition rate Γ_{ij} between two impurity sites i and j is defined as:*

$$\Gamma_{ij} = \begin{cases} \nu_0 \exp\left(-\left(\frac{2r_{ij}}{a_B} + \frac{\Delta E_{ij}}{k_B T}\right)\right) & \text{if } \Delta E_{ij} > 0 \\ \nu_0 \exp\left(-\frac{2r_{ij}}{a_B}\right) & \text{if } \Delta E_{ij} \leq 0 \end{cases} \tag{1}$$

where:

- *ν_0 is the attempt frequency, representing the frequency of attempts to hop.*
- *r_{ij} is the distance between impurity sites i and j.*
- *a_B is the Bohr constant.*
- *$\Delta E_{ij} = E_j - E_i$ is the energy difference between sites j and i.*
- *k_B is the Boltzmann constant.*
- *T is the temperature.*

The calculations of the energy difference ΔE_{ij} between pairs of hopping sites are described in [1].

2.2 Kinetic Monte Carlo simulation

Kinetic Monte Carlo (KMC) simulation is a computational algorithm used to simulate the time evolution of systems where events occur at rates determined by underlying physical processes. In the context of charge transport in disordered materials like our system, KMC simulations are particularly useful for capturing the stochastic nature of electron hopping between localized states, or impurity sites.

At each time step [1] in the KMC simulation of the DNPU, the following steps are performed:

[1] The use of the exponential distribution in line 10 is due to the memoryless property of continuous-time Markov processes. If transitions occur at a total rate $\lambda = \sum_{k,l} \Gamma_{kl}$, the waiting time Δt until the next event follows an exponential distribution. For details, see [14].

Algorithm 1 Monte Carlo Step (MCS)

```
1: Input: Current state of the system (voltages of 7 input electrodes, current of output
   electrode, initial occupation states, geometric parameters, initial energy potentials)
2: Output: Updated state of the system, time increment
3: procedure MonteCarloStep
4:     for each possible transition from site $i$ to site $j$ do
5:         Calculate transition rate $\Gamma_{ij}$ defined in 2.1
6:     end for
7:     Normalize the transition rates to obtain the probability $P_{ij}$ of each transition:
           $P_{ij} = \frac{\Gamma_{ij}}{\sum_{k,l} \Gamma_{kl}}$
8:     Select one transition based on the probabilities $P_{ij}$.
9:     Update the system state (electrical field and energy differences) to reflect the chosen
   transition.
10:    Draw a random number $\Delta t$ from an exponential distribution with parameter
   $\sum_{k,l} \Gamma_{kl}$:
           $\Delta t \sim \text{Exponential}(\sum_{k,l} \Gamma_{kl})$
11:    Advance the simulation time by $\Delta t$.
12: end procedure
```

This process is repeated for a large number of iterations to simulate the time evolution of the system [1]. In our simulations, a *sample* consists of seven input electrodes and the corresponding output current after running the simulation for a specified number of KMC steps.

A Monte Carlo simulation developed by Wilde [4] and extended by Becker [1] is utilized to model the stochastic behavior of the DNPU in Figure 1B. The analysis of convergence of this algorithm is given in section 3.

2.3 Machine Learning

In this subsection, we introduce fundamental concepts in machine learning, particularly focusing on how these concepts are applied to develop a surrogate model for DNPUs in Section 4. This explanation aims to provide a clear understanding for readers who may not be familiar with artificial intelligence (AI) techniques. For a deep understanding of the topic I recommend Chapter 5 of [5].

Machine learning is a subset of artificial intelligence where algorithms learn from data to make predictions or decisions without being explicitly programmed for specific tasks [5]. The core components of a machine learning model are described in Table 1 with application to a surrogate model of DNPU in this paper.
The model mentioned in Table 1 used to map the inputs to output in our case is the a feedforward neural network, explained in section 2.3.1. The training process is explained in section 2.3.2.

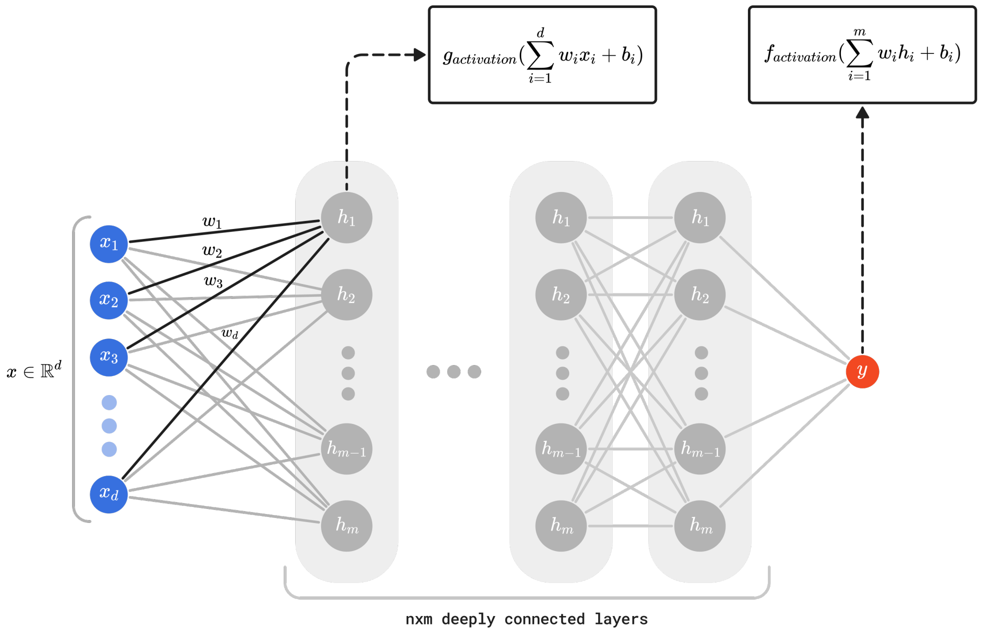
Name	**Description**	**In our study**
Data	The examples or observations the model learns from.	Input-output pairs generated from KMC simulations.
Features	The input variables used to make predictions.	The 7 input electrodes configurations.
Labels	The output variables that the model aims to predict.	The output currents from the DNPU simulations.
Model	The mathematical structure (such as a neural network) that maps inputs to outputs.	A deep feedforward neural network used as a surrogate model.
Training	The process of adjusting the model's parameters to minimize the difference between predicted and actual outputs.	The optimization process using the Adam optimizer to minimize the MSE loss.

Table 1: Core components of a machine learning model and their application in our study.

2.3.1 Feedforward Neural Networks

A neural network is a type of machine learning model inspired by the human brain. The basic structure is visualized in Figure 2, it consists of layers of interconnected nodes (neurons) divided into three kinds: input units, hidden units and output units. In the standard architecture, each layer is fully-connected, i.e. each neuron in each layer is connected to all neurons in the previous layer. Each connection between neurons has a weight w_i, which determines the strength of the connection, and each neuron has a bias term b_i. When data flows forward through each hidden layer (hence the name feedforward neural network), each neuron sums over all neurons from previous layer with their respective weight and bias and then the applies an activation function $g_{activation}$ to the weighted sum of inputs plus bias. This activation function [2] introduces non-linearity that enables the network to model complex relationships between inputs and outputs. The output neuron performs similar computation as the ones in the hidden layers but the choice of activation function $f_{activate}$ here is usually different than $g_{activation}$, it depends on the type of problem being solved by the network. For instance, in this paper, no activation function is applied in the output neuron since the goal of our network is to emulate the behavior of a DNPU, the output is expected to be raw values that represent the current output given a set of electrodes.

[2]The Rectified Linear Unit (ReLU) $g(z) = max(0, z)$ is usually used due to its simplicity and effectiveness in the vanishing gradient problem [12].



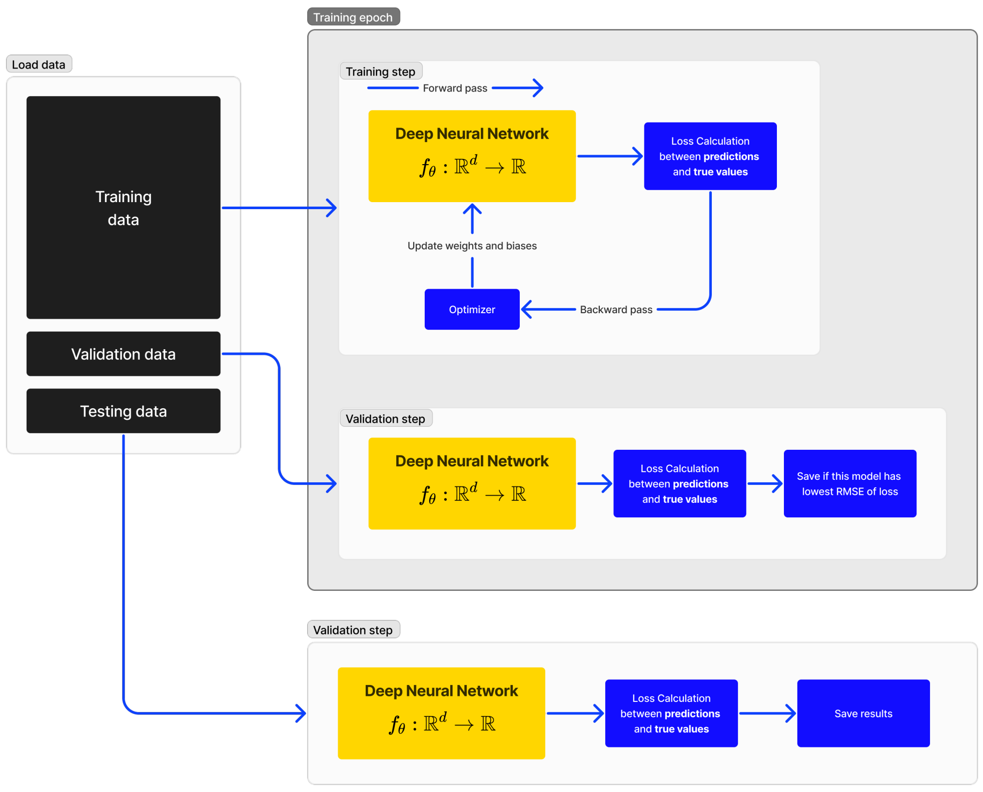
$g_{activation}(\sum_{i=1}^{d} w_i x_i + b_i)$
$f_{activation}(\sum_{i=1}^{m} w_i h_i + b_i)$
$x \in \mathbb{R}^d$
x_1
x_2
x_3
x_d
w_1
w_2
w_3
w_d
h_1
h_2
h_{m-1}
h_m
y
nxm deeply connected layers

FIGURE 2: Visualization of a neural network. The blue neurons is the input layer, grey neurons form hidden layers and red neuron is the output layer.

For more detailed explanation of Feedforward Neural Networks, refer to Chapter 21.1 of [17].

2.3.2 Training

Training a neural network involves adjusting the model's parameters to minimize the error between the predicted and actual outputs. The goal is to train a network $f_\theta : \mathbb{R}^d \to \mathbb{R}$ parameterized by θ, which represents the collective weights W and biases b of the neural network used in the SM, to approximate the mapping from inputs $\boldsymbol{x}_i$ to output y_i. This can be formulated as an optimization problem searching for $\min_\theta \mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{Y})$, where $\mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{Y})$ is a loss function (sometimes refered to as a criterion). The core steps in training a feedforward neural network, as applied in our study, are outlined below with a high level visualization in Figure 3.



Training epoch
Load data
Training data
Validation data
Testing data
Training step
Forward pass
Deep Neural Network
$f_\theta : \mathbb{R}^d \to \mathbb{R}$
Loss Calculation between predictions and true values
Update weights and biases
Optimizer
Backward pass
Validation step
Deep Neural Network
$f_\theta : \mathbb{R}^d \to \mathbb{R}$
Loss Calculation between predictions and true values
Save if this model has lowest RMSE of loss
Validation step
Deep Neural Network
$f_\theta : \mathbb{R}^d \to \mathbb{R}$
Loss Calculation between predictions and true values
Save results

FIGURE 3: High level flowchart of training process of a deep neural network.

Data Preparation: The training process begins with the preparation of data. We generate a dataset of input-output pairs using Kinetic Monte Carlo (KMC) simulations. These pairs consist of the configurations of the seven input electrodes and the corresponding output currents. Let $X = \{\boldsymbol{x}_i\}_{i=1}^N$ represent the set of input configurations, where $\boldsymbol{x}_i \in \mathbb{R}^d$, in our case $d = 7$, and the number of samples $N = 100,000$. Similarly, let $Y = \{y_i\}_{i=1}^N$ represent the output currents, where $y_i \in \mathbb{R}$. The samples are split into three parts used for training, validation and testing, all described below.

Model Initialization: We define the architecture of our deep neural network (DNN) $f_\theta : \mathbb{R}^d \to \mathbb{R}$, parameterized by θ, which includes the weights W and biases b of the network. The network comprises several layers, each containing a specific number of neurons, with activation functions applied to introduce non-linearity.

Forward Pass: During each training iteration, the input data $\boldsymbol{x}_i$ is passed through the network to produce a predicted output $y_i = f_\theta(\boldsymbol{x}_i)$.

Loss Calculation: The $\mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{Y})$ is a performance measure between the predicted outputs and the actual outputs. The choice of this function depends on the goal of the network. Additionally, $\mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{Y})$ has to be differentiable with respect to θ to allow gradient-based optimization methods to work.

Backward Pass: The gradients of the loss function with respect to the network parameters θ are calculated using backpropagation. These gradients indicate how θ should be adjusted to reduce the loss.

Parameter Update: The network parameters θ are updated using the Adam [8] optimizer, which is a variant of gradient descent. This optimizer adjusts the learning rate adaptively for each parameter, facilitating efficient and effective training:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta; X, Y)$$

where η is the learning rate, which controls the size of the steps the optimizer takes to adjust the weights and biases of the network. Choosing the right learning rate is crucial: a rate too high can cause the training process to converge too quickly to a suboptimal solution, or even diverge, while a rate too low can make the training process excessively slow. In our study, we use η from [16] and experiment around that value.

Validation: After each training epoch, the performance of the network is evaluated on a validation set. This helps in monitoring overfitting and underfitting. The model achieving the lowest Root Mean Squared Error (RMSE) on the validation set is selected as the final trained model.

Number of Epochs: The training process is typically run for a predefined number of epochs. An epoch is a complete pass through the entire training dataset. The number of epochs determines how many times the learning algorithm will work through the entire training dataset.

Convergence and testing: The training process continues for a predefined number of epochs or until the loss converges to a satisfactory level. The model performance is tested on a testing set, which has never been used during the training process. This provides a measure of how well the model generalizes to new, unseen data, ensuring that the model's predictions are accurate and does not overfit to the training data.

2.4 Linear separability

In this subsection, we introduce definitions related to linear separability in the context of machine learning for binary classification tasks. We also discuss the XOR problem as an example of a non-linearly separable dataset.

Definition 2.2 (**Linear Separability**). *A dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0, 1\}\}$ is said to be* ***linearly separable*** *if there exists a vector $\mathbf{w} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$ such that:*

$$\mathbf{w}^T \mathbf{x}_i + b > 0 \quad \textit{if} \quad y_i = 1$$

$$\mathbf{w}^T \mathbf{x}_i + b < 0 \quad \textit{if} \quad y_i = 0$$

for all $(\mathbf{x}_i, y_i) \in \mathcal{D}$.

Definition 2.3. *A* ***hyperplane*** *in an n-dimensional space is a flat affine subspace of dimension $n - 1$. It can be described by the linear equation:*

$$\mathbf{w}^T \mathbf{x} + b = 0$$

where **w** *is a normal vector to the hyperplane and b is a bias term.*

From the definitions 2.2 and 2.3, it is easy to see that in an n-dimensional space, a hyperplane is an $(n-1)$-dimensional subspace that divides the space into two half-spaces. Linear separability means that we can find such a hyperplane [3] that separates all data points of one class from all data points of the other class without any overlap. Figure 4 provides visual representations of linear separability in 1D, and 2D, where the hyperplane is a point and line, respectively.

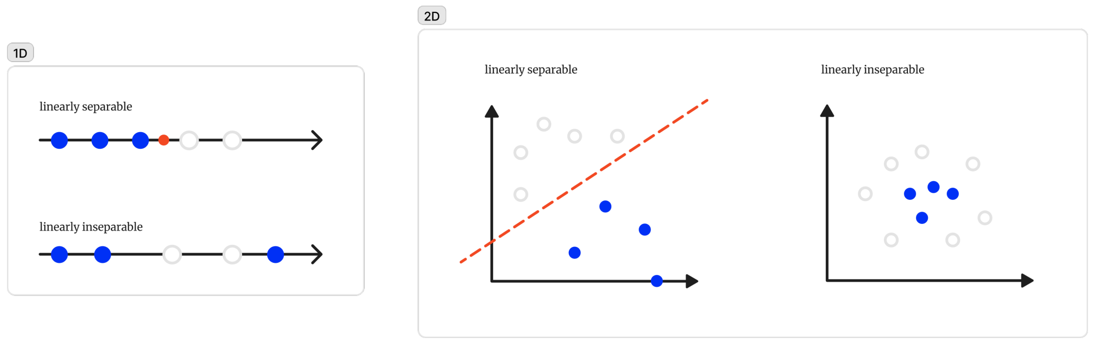


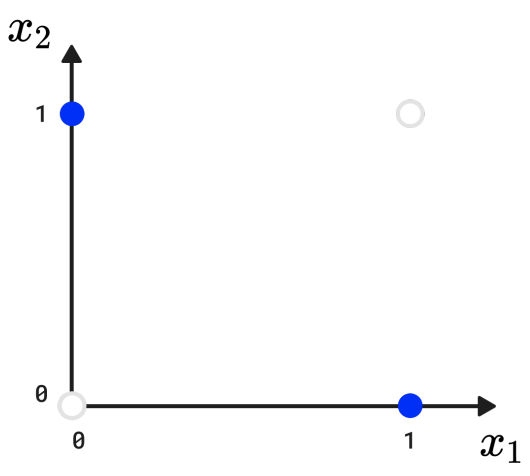
FIGURE 4: Linear separability in one-dimensional and two-dimensional spaces, where red dot and line are hyperplanes of their respective space that linearly divide the dataset. Datapoints are colored same if they belong to the same class.

The XOR problem (Figure 5) is a classic example of a dataset that is not linearly separable. In this dataset:

- The points (0,0) and (1,1) belong to one class (0).
- The points (0,1) and (1,0) belong to another class (1).

If we try to separate these points using a line, we find it impossible to do so without misclassifying at least one point. This is because the classes are interwoven in such a way that no single straight line can separate them. We will prove this observation formally in the next theorem.

[3]These are usually called *decision boundary*.



x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 5: Two-dimensional space plane of points induced by *XOR* function.

Theorem 2.1. *The dataset induced by XOR function is not linearly separable.*

Proof. Consider the points $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$. Assume there exists a linear boundary defined by $\mathbf{w}^T \cdot \mathbf{x} + b = 0$ that separates these points such that (0,0), (1,1) belong to the same class and (0,1), (1,0) belong to the same class. In other words, the following must hold:

$$\mathbf{w}^T \cdot (0,0) + b \leq 0 \quad \text{and} \quad \mathbf{w}^T \cdot (1,1) + b \leq 0 \tag{2}$$
$$\mathbf{w}^T \cdot (0,1) + b > 0 \quad \text{and} \quad \mathbf{w}^T \cdot (1,0) + b > 0 \tag{3}$$

Let $\mathbf{w} = (w_1, w_2)$, we obtain from (2) and (3):

$$\mathbf{w}^T \cdot (0,0) + b = b \leq 0 \tag{4}$$
$$\mathbf{w}^T \cdot (1,1) + b = w_1 + w_2 + b \leq 0 \iff w_1 + w_2 \leq -b \tag{5}$$
$$\mathbf{w}^T \cdot (0,1) + b = w_2 + b > 0 \tag{6}$$
$$\mathbf{w}^T \cdot (1,0) + b = w_1 + b > 0 \tag{7}$$

Adding (6) and (7), we get:

$$w_1 + w_2 + 2b > 0 \iff w_1 + w_2 > -2b \tag{8}$$

From (5) and (8) $\Rightarrow -b > -2b \iff b > 0$. This is a contradiction with (4).

Therefore, no single vector $\mathbf{w}$ and bias b can satisfy these inequalities simultaneously. Thus, no linear boundary can perfectly separate the XOR data points, proving that XOR is not linearly separable. $\square$

For more information on linear separability, refer to [2].

3 Simulation Analysis

This section discusses topics regarding the convergence to true device and robustness of the simulation developed by [1].

We start by stating a theorem for the conditions to converge to an actual physical system of a KMC simulation and argue that our system indeed fulfills these conditions, therefore after a sufficient number of steps it will converge to true behaviour of the corresponding real-world system.

Theorem 3.1 (Convergence of Kinetic Monte Carlo Simulations)**.** *Let X_{t_n} be a sequence of states using transition probabilities $\mathbb{P}(X_{t_{n+1}} \mid X_{t_n})$ computed by the KMC algorithm, which computes a conditional probability distribution that depends only on the current state of the system, that is:*

$$\mathbb{P}\{X_{t_{n+1}} = x_{n+1} \mid X_{t_n} = x_n, \ldots, X_{t_0} = x_0\} = \mathbb{P}\{X_{t_{n+1}} = x_{n+1} \mid X_{t_n} = x_n\}.$$

Assume that the Markov chain defined by the KMC algorithm is ***irreducible***, ***aperiodic***, *and* ***positive recurrent***. *Let $X_{true}(t)$ be the true state of the system. Then, for a sufficiently large number of Monte Carlo steps, X_{t_n} converges in distribution to $X_{true}(t)$.*

Proof. By the properties of Markov chains, the ergodic theorem and the law of large numbers, we know that if the Markov chain is irreducible, aperiodic, and positive recurrent, for sufficiently large n, the distribution of X_{t_n} will approximate π, the stationary distribution [14].

Since $X_{\text{true}}(t)$ represents the true state of the system and assuming the physical system's dynamics align with the stationary distribution π, we have:

$$\pi = \mathbb{P}(X_{\text{true}}(t) = x).$$

Therefore, as the number of Monte Carlo steps increases, the distribution of X_{t_n} converges to the stationary distribution π. This implies:

$$X_{t_n} \xrightarrow{d} X_{\text{true}}(t).$$

This completes the proof. □

To prove the convergence of our KMC, we need to establish that the Markov chain defined by the KMC algorithm is irreducible, aperiodic, and positive recurrent. In each Monte Carlo Step 0, transitions are considered from each site i to every possible site j with transition rates Γ_{ij} (Definition 2.1). So every state is reachable from any other state because every possible transition is considered [4], making the chain irreducible. Transitions are chosen based on normalized probabilities P_{ij}. The randomness in choosing Δt from an exponential distribution with parameter $\sum_{k,l} \Gamma_{kl}$ adds variability to the time intervals between transitions, which ensures aperiodicity. Finally, given that transition rates Γ_{ij} are positive and finite, the expected return time to any state is finite, thus KMC is positive recurrent. This is because the system is continuously updated and every state has a non-zero probability of being revisited within a finite number of steps. Given these conditions,

[4]Transition rates must be non-zero. This is guaranteed in our system since ν_0 is non-zero and r_{ij} is finite, with T greater than zero, so the exponential term in (1) is positive.

the KMC simulation converges to the true dynamical behavior of the system under study based on theorem 3.1. In the next subsection, we provide a simple statistical analysis of the robustness of the simulation.

3.1 Robustness of the simulation

An important parameter of the KMC simulation is the number of KMC steps (defined in 0) in one simulation. As the more steps we take, the longer time we observe the behavior in our system and hence the closer we get to its equilibrium state. The probability density plot in Figure 6 displays the distribution of uncertainties in output currents of 4000 samples using three different numbers of KMC steps: 10^3, 10^4, and 10^5. The uncertainties are directly derived using the KMC simulation [1]. The results indicate that as the number of KMC steps increases, the distributions of uncertainties become more concentrated around lower values, and the mean uncertainties decrease, demonstrating that simulations with more steps yield more robust and consistent results with lower variance.

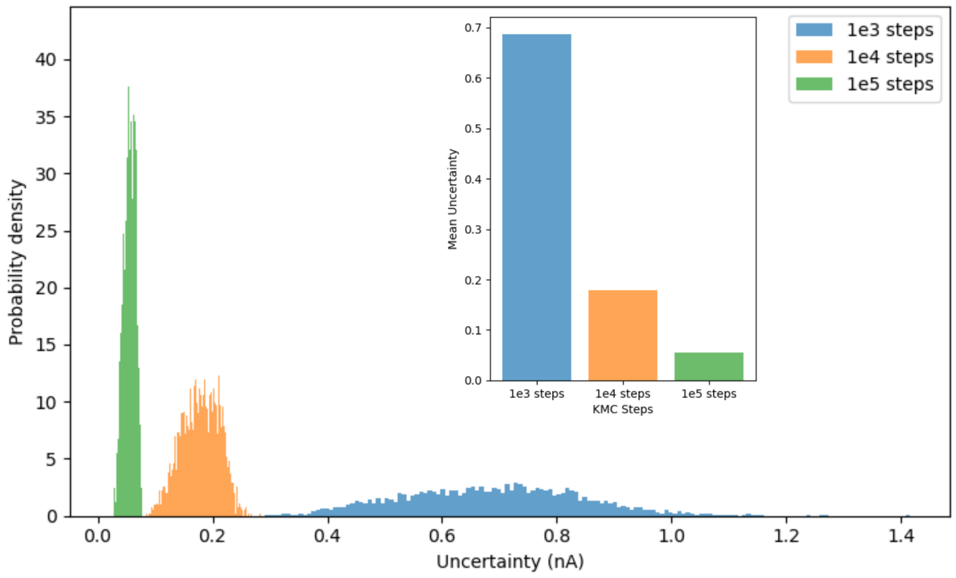


FIGURE 6: Distribution of uncertainties of simulations with different KMC steps, the x-axis shows the uncertainty values, the y-axis indicates the probability density. The inset bar chart compares the mean uncertainties.

While simulation with 10^5 KMC steps yields the lowest mean uncertainty and most concentrated distribution, the improvement over 10^4 steps is marginal compared to the significant increase in computational time required. The mean uncertainty for 10^4 steps is already substantially lower than for 10^3 steps, and its distribution shows much reduced variance. The additional reduction in uncertainty from 10^4 to 10^5 steps is relatively small. Thus, 10^4 KMC steps is used in this paper, providing sufficiently low uncertainty and consistent results to train the SM while being computationally feasible.

In Table 2, we provide additional output currents statistics of the main dataset of 100,000 samples used to train the SM with 10^4 steps.

Statistic	**Current (nA)**	**Uncertainty (nA)**
Mean	-0.21	0.06
Median	-1.08	0.06
Standard Deviation	4.36	0.01
Variance	19.05	0.00
Min	-7.36	0.02
Max	7.36	0.09

TABLE 2: Uncertainty statistics of output current.

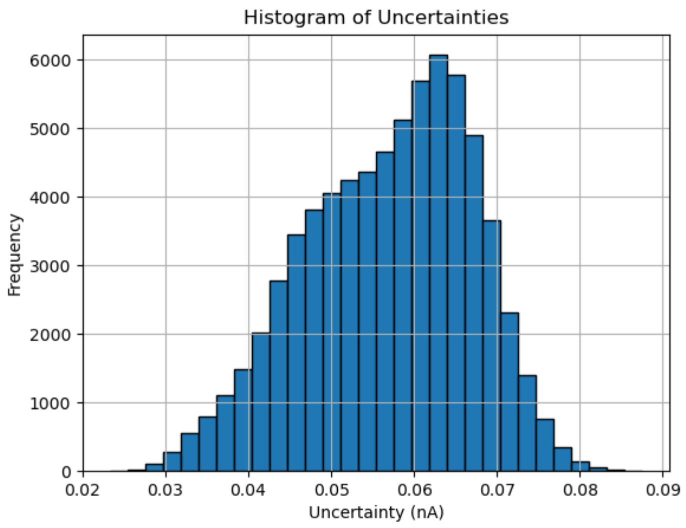


FIGURE 7: Histogram of Uncertainties

The relatively small values of the uncertainty statistics (Table 2) with higher uncertainty with significantly lower frequency (Figure 7), particularly the mean and standard deviation of the uncertainty, indicate that the simulation results are consistent and reliable. The narrow range of uncertainty (0.02 nA to 0.09 nA) further supports the robustness of the simulation outcomes.

As the number of samples increases, we expect the statistical metrics to converge to their true values (Theorem 3.1). In other words, the mean and standard deviation should stabilize, indicating that additional samples do not significantly alter the results. The consistency in the simulation output currents provides confidence in the validity of the modeled behavior of the DNPU device.

4 Surrogate model

The simulation of [1] comes with various algorithms to find boolean functions that yield accurate results but requires a lot of computations. For example, finding the right control electrodes for a XOR gate can take up to 90 hours. This is because the algorithm run a large amount of KMC steps. With a SM that emulates the DNPU well, we can use machine learning techniques to find the right set of control electrodes for the XOR problem in much less time [16].

This section includes the methods used to model the DNPU using artificial intelligence. We begin with a definition of the SM, explain the design choices behind the network used to realize this SM in Section 4.1, and discusses the results in Section 4.2. A sensitivity analysis is provided in Section 4.3 to further investigate the trained SM.

Definition 4.1. *A **surrogate model** (SM) [16] is an artificial neural network (ANN) trained to emulate the functionality of a complex system or device, such as a nanoelectronic device.*

We will train the SM using a fully-connected feedforward neural network trained using simulated data from the KMC simulation (training proccess explained as in Section 2.3). After completing the training, validation, and testing steps, the final output is a SM that accurately emulates the behavior of the DNPU based on the simulated data. This model is capable of predicting the output current for given input electrode configurations with high accuracy, significantly reducing the computation time required compared to direct KMC simulations.

4.1 Network architecture

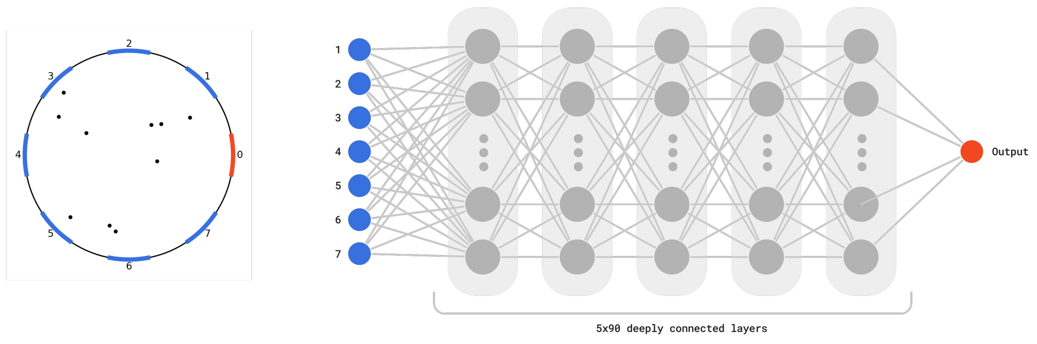


FIGURE 8: Visualization of the network architecture. Blue regions depict 7 input electrodes and red region is the output electrode in hardware setup (left) and SM (right).

We define $f_\theta : \mathbb{R}^7 \to \mathbb{R}$ as fully connected deep neural network illustrated in Figure 8. The network begins with 7 input electrodes, each represented as nodes $\boldsymbol{x}_i \in \mathbb{R}^7$ in the input layer. The network has a single output electrode, which produces the final output current $y_i \in \mathbb{R}$ based on the input configurations. The core of the network [5] consists of five hidden layers, each comprising 90 deeply connected neurons. For each neuron in the hidden layers, the ReLU function $g(z) = max(0, z)$ is used as activation function and no activation function is applied on the output neuron.

The loss function, $\mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{Y})$, used in the training process is the MSE loss, defined as:

[5]The choices are replicate of the study by Ruiz Euler [16].

$$\mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{N} \sum_{i=1}^{N} (f_\theta(\boldsymbol{x}_i) - y_i)^2 \tag{9}$$

, which is clearly differentiable with respect to θ.

To find the parameter set θ that minimizes equation (9), we employ backpropagation to compute the gradients of the loss function with respect to each parameter in the network. These gradients are then used to update the parameters iteratively using gradient descent algorithm Adam [8] for 3,000 epochs with a learning rate of 10^{-5} and a mini-batch size of 128. Additionally, batch normalization was used to normalize the activations of each layer. This technique helps in accelerating the training process and improving the performance and stability of the network [7].

The package developed by BRAINS [16] was used to implement the discussed network. This package comes with classes built on top of PyTorch [13] that allows for further implementation and extension of the trained SM.

4.2 Results

Figure 9 shows the RMSE in nanoamperes (nA) over 3,000 training epochs. Initially, the RMSE decreases rapidly for both sets, indicating that the model is effectively capturing the underlying patterns in the data.

As training progresses, the RMSE for both training and validation sets continues to decline but at a slower rate, eventually stabilizing, which signifies the convergence of the model. The stabilization of RMSE values indicates that the model parameters have reached a point where further training results in minimal improvements in performance. This plateau suggests that the model has learned the optimal parameters to generalize well on unseen data.

The final test loss (in green) $\approx$ 0.234 nA, further supports the model's robustness and accuracy in predicting outputs for new, unseen data. Although the RMSE does not decrease to zero, this is expected in practical machine learning applications due to inherent noise in the data and the complexity of the model. A non-zero RMSE indicates that while the model performs well, it does not overfit to the noise present in the training data, maintaining a balance between bias and variance. This result reflects a well-trained model that is capable of generalizing its learned patterns to new, unseen data rather than merely memorizing the training set.

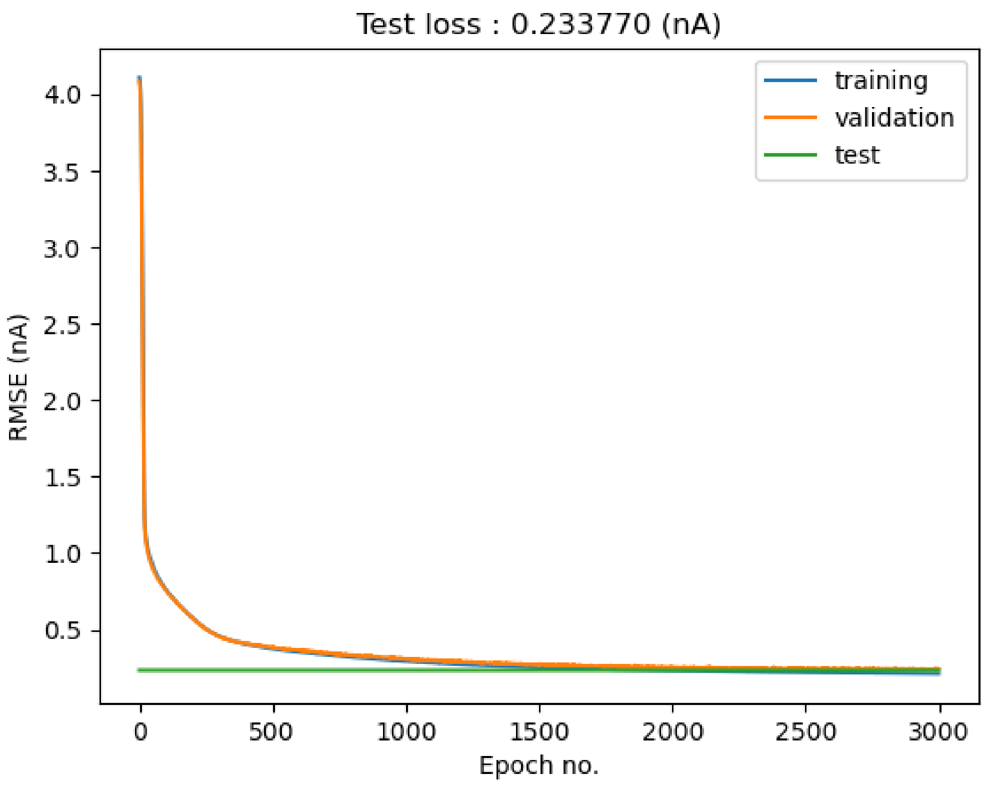


FIGURE 9: RMSE in nanoamperes (nA) across 3000 training epochs for the training, validation, and test datasets.

Figure 10 presents a scatter plot comparing all true outputs with their corresponding predicted outputs from the test set. Each point represents a single data sample, with the x-axis indicating the true output value and the y-axis showing the predicted output value. The red dashed line represents the ideal scenario where the predicted outputs perfectly match the true outputs (i.e., a 45-degree line). This plot provides a comprehensive view of the model's performance across the entire dataset. The concentration of data points along the red dashed line indicates a high level of agreement between the true and predicted outputs. The closer the points are to this line, the more accurate the model's predictions are. This plot helps in identifying any systematic biases or trends where the model might consistently overpredict or underpredict. The dense clustering of points around the line confirms that the surrogate model is making accurate predictions consistently.

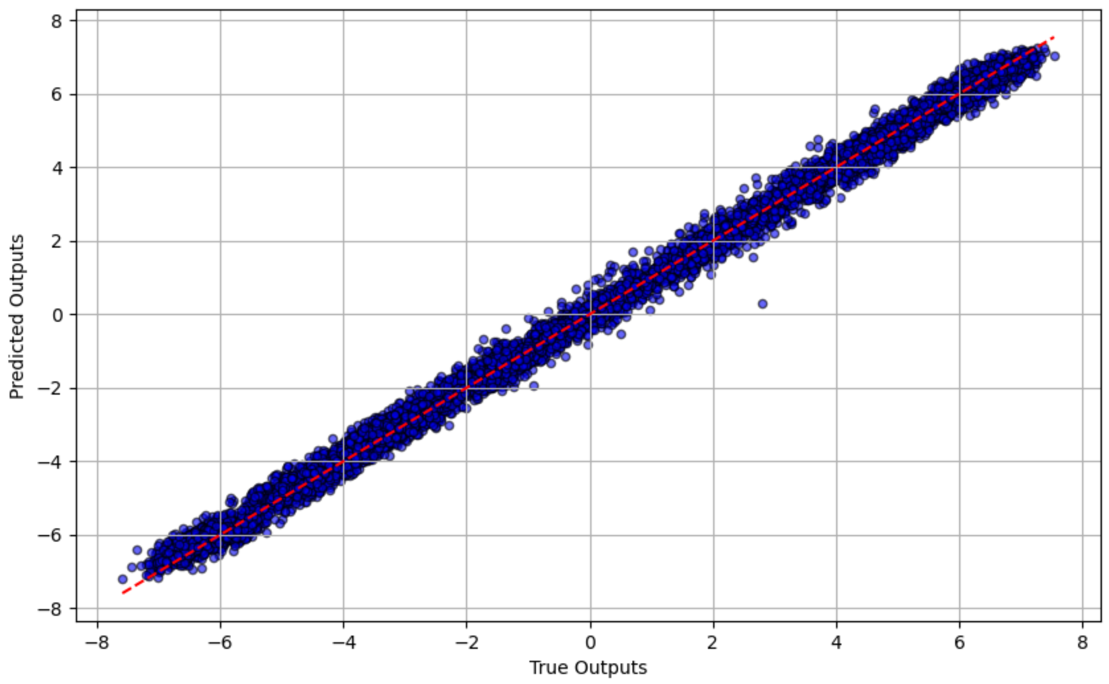


FIGURE 10: All data points comparison between true and predicted values.

4.3 Sensitivity Analysis

To further investigate the robustness of the model, we conduct a sensitivity analysis on multiple surrogate models trained on same, and different data.

Definition 4.2. *The* ***sensitivity*** S_{m,j,δ_p} *for each model* $m \in \{1, 2, \dots, M\}$*, feature* j*, and perturbation* δ_p *is calculated as the mean absolute difference between the base prediction* $\hat{y}_i$ *and the perturbed prediction* $\hat{y}_i^{(m,j,\delta_p)}$*, i.e.:*

$$S_{m,j,\delta_p} = \frac{1}{N} \sum_{i=1}^{N} \left| \hat{y}_i^{(m,j,\delta_p)} - \hat{y}_i \right| \tag{10}$$

We systematically perturb each of the 7 input features and observe the output. We perform this analysis on 100 random inputs with perturbations magnitudes (-0.2, -0.5, 0.2, 0.5) one at a time to each input feature, i.e. for each input feature, we individually apply -0.2, -0.5, 0.2, and 0.5, observing the model's output for each perturbation. This analysis provides a quantitative measure of how sensitive each model's output is to changes in each input feature. Higher sensitivity values indicate that the model's output is more affected by variations in the corresponding input feature.

The final output is a three-dimensional array $\mathbf{S} \in \mathbb{R}^{M \times d \times P}$ containing the sensitivity values for each model, feature, and perturbation magnitude. We plot sensitivities in Figure 11 for each input and observe a close sensitivity similarities between all models. To clarify, models 1 and 2 were trained on the same dataset of 100,000 samples while models 3 and 4 were trained on different datasets (also of 100,000 samples).

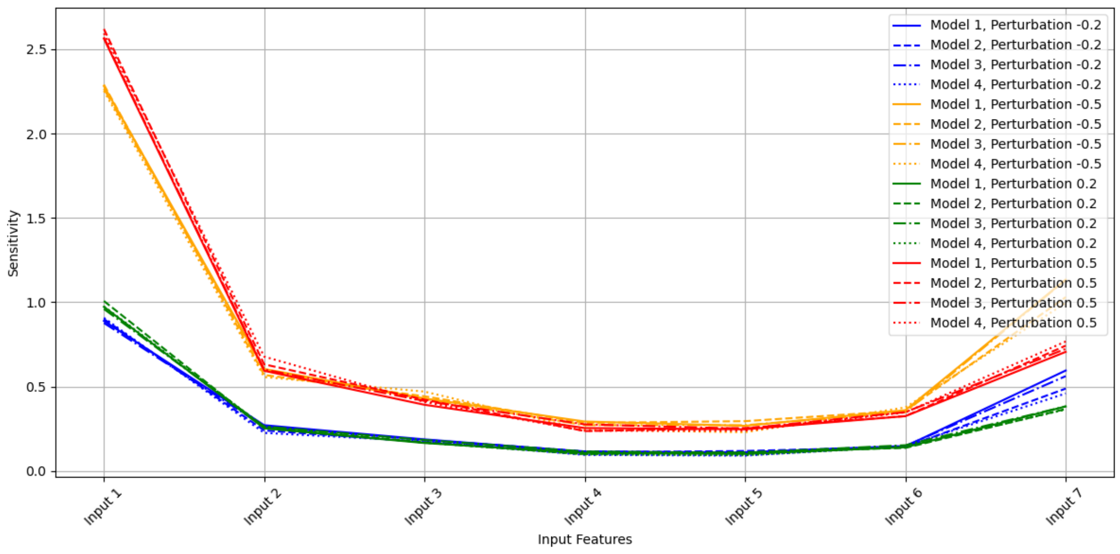
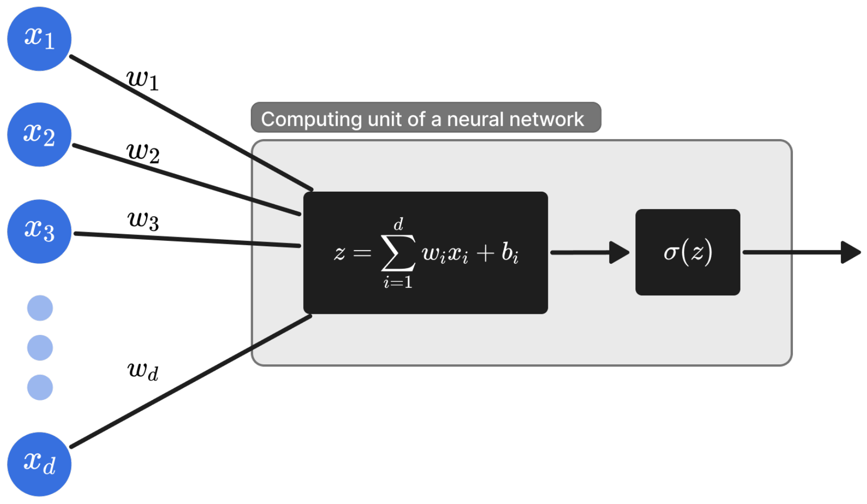


FIGURE 11: The plot displays the sensitivity of each input feature for four different models under various perturbations. Each line represents the sensitivity of one model to one perturbation magnitude.

From this analysis, we can not only say that the models have similar sensitivity profiles, it also reveals that input 1 and 7 have the most significant impact on the surrogate models' output. This pattern can be interpreted by considering the physical and electrical characteristics of the input and control electrodes in the DNPU. The high sensitivity of input 1 and 7 suggests that it has a strong coupling with the output electrode, likely due to its proximity or direct influence on the charge distribution within the dopant network (Figure 1B). On the other hands, inputs with the smallest sensitivity are likely farther from the critical regions of the dopant network that directly affect the output electrode, hence their perturbations result in minimal changes to the output. In conclusion, the models align well with the physical expectations of the DNPU, demonstrating that the surrogate model accurately captures the sensitivity dynamics inherent in the device's architecture.

5 Finding boolean gates

At the heart of a neural network, we have a computing unit, or a neuron (Figure 12), which is usually combined with other computing units to create a neural layer, multiple of these layers create a neural network that performs deep learning and exhibit intelligence in machines [9].



x_1
x_2
x_3
x_d
w_1
w_2
w_3
w_d
Computing unit of a neural network
$z = \sum_{i=1}^{d} w_i x_i + b_i$
$\sigma(z)$

FIGURE 12: Schematic of a single computing unit in a neural network. x_i's represent inputs and σ represent the activation function, usually the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

These computing units are not that powerful, as they succeed in linear classifications, but fail in nonlinear ones, such as the XOR boolean problem [6] (Figure 13). The output of such unit is a linear combination of their inputs:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{11}$$

here x_i are the input values, w_i are the weights, b is the bias, σ is the activation function. This makes it impossible to separate classes that are not linearly separable, like XOR.

[6]For proof of dataset induced by XOR function being nonlinearly separable, see Theorem 2.1.

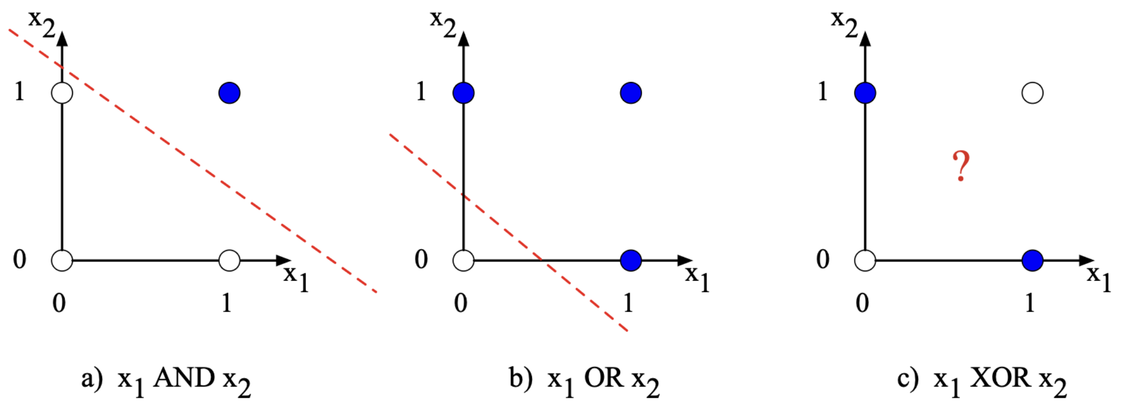


FIGURE 13: Linear separability for AND, OR and XOR functions. Blue dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The red lines in a) and b) represent possible linear decision boundaries. Figure styled after [17].

While the XOR gate cannot be realized by a single unit, a layered network of these units can. The non-linear decision boundary of XOR can be realized with a neural network consisting of 1 hidden layer with 2 neurons [5]. In [16], they successfully demonstrated the XOR function on a single DNPU with radius of about 150nm. This section aims to explore the implementation of the XOR gate using a much smaller DNPU with radius 16.77nm (results found in [19]). The goal is to demonstrate, using a SM, that even with a minimized hardware footprint, the DNPU retains its ability to solve non-linearly separable problems such as XOR effectively.

5.1 Network architecture

The primary objective of our network is to determine a set of five control voltages applied to electrodes 1, 2, 4, 6, 7 in Figure 1B that will allow the DNPU to correctly classify the Boolean inputs 00, 01, 10, and 11. The network architecture utilizes a single-layered model based on the SM as defined in 4.1. This SM functions as a neuron. During this process, the hidden layers of the SM are frozen to preserve the learned functionality of the DNPU. The goal is to train this DNPU-based neuron to classify basic Boolean functions such as AND, OR, and XOR, with particular emphasis on the XOR function due to its non-linear separability.

To achieve this, pairs representing Boolean variables (00, 01, 10, 11) are fed into the input electrodes of the SM, i.e. inputs 3 and 5 in Figure 1B. To facilitate comparison with the results from [16], we assign a current value of -0.6 nA to class 0 and 1.2 nA to class 1. The network architecture includes 7 input electrodes and 1 output electrode, as illustrated in Figure 14. Throughout training, each input pair is frozen while control electrodes are iteratively adjusted to minimize the error between predicted and target outputs.

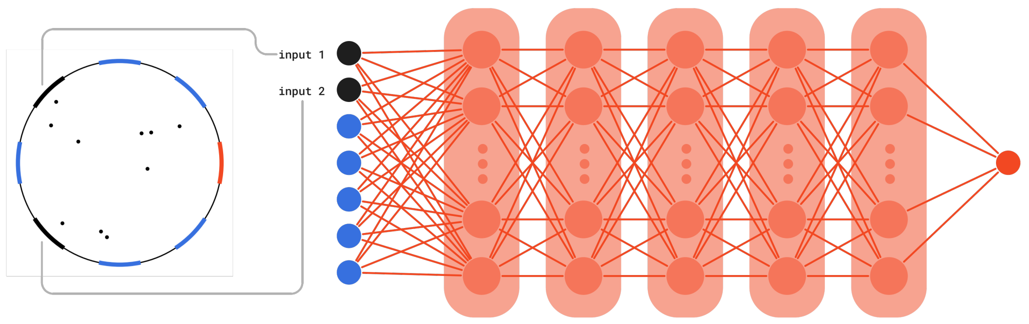


FIGURE 14: Network architecture for training a boolean functionality of a SM. The weights of the hidden layers are frozen (in red) to preserve the learned functionality of the DNPU, as well as the input pair (in black). Only the control electrodes (blue inputs) are learnable parameters.

The training process for obtaining the boolean gate is depicted in Figure 15. Binary input pairs (00, 01, 10, 11) are fed to the SM together with an initially random set of five control voltages. During the forward pass, the SM processes these inputs through multiple layers to produce output currents. It is helpful to conceptualize the SM as a black box functioning like a DNPU, but with the advantage that we can perform backpropagation since it is fundamentally a neural network. The SM generates a dataset of output currents, which are then divided into classes. For the XOR function, input pairs 00 and 11 are classified as class 0, while input pairs 01 and 10 are classified as class 1. The loss function is calculated to maximize the separation between these classes.

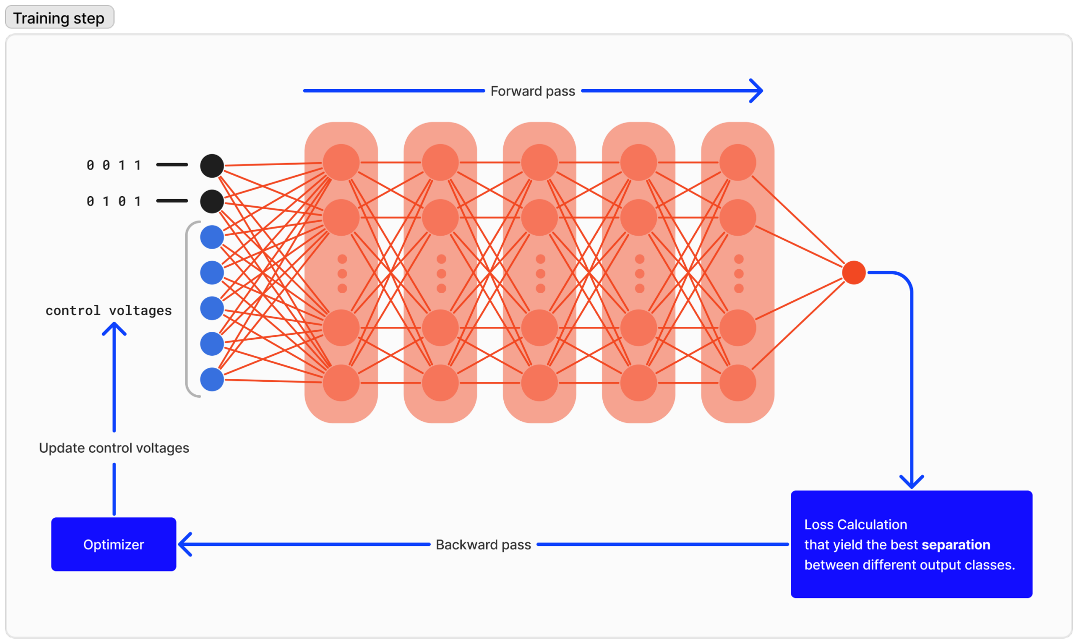


FIGURE 15: Training process for obtaining the boolean gate. The binary input pairs (00, 01, 10, 11) and a set of 5 control voltages create a dataset that is fed to the SM. The optimizer updates the control voltages based on the loss calculation, which aims to maximize the separation between different output classes.

Gradient descent algorithm Adam [8] is employed with a custom loss function (Section 5.1.1) for 600 epochs and learning rate $\eta = 0.08$ (replicate of [16]) to optimize the control electrodes.

5.1.1 Cost function

The design of the cost function heavily depends on the output type, which in our case is the truth table for the correct Boolean function. The performance of the DNPU as a Boolean gate can be quantified by the separation between the outputs classified as 1 and 0. The larger the separation, the more robust gate. In this section, we develop a cost function that demonstrates better extrinsic evaluation than the one proposed in [16].

Definition 5.1. *Let I_i be current class of the logic values $i = \{0, 1\}$. The **current separation** y_{sep} is the minimum difference between $\max I_0$ and $\min I_1$, i.e. $y_{sep} = \max I_0 - \min I_1$.*

We propose the following cost function to train the nanoneuron for Boolean classification:

$$\mathcal{L}(y, z) = \mathcal{L}_{\text{BCE}}(y, z) + \lambda \cdot \mathcal{L}_{\text{sep}}, \tag{12}$$

where $\mathcal{L}_{\text{BCE}}$ is the Binary Cross-Entropy loss, λ is a hyperparameter that controls the trade-off between classification accuracy and current separation and $\mathcal{L}_{\text{sep}}$ is the separation loss defined as follows:

$$\mathcal{L}_{\text{BCE}}(y, z) = -\frac{1}{n} \sum_{i=1}^{n} \left[z_i \log(y_i) + (1 - z_i) \log(1 - y_i) \right], \tag{13}$$

where $y_i = \sigma(I_i)$ is the predicted probability obtained by applying the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ to the output current I_i, and z_i is the target binary label.

The separation loss $\mathcal{L}_{\text{sep}}$ is defined as:

$$\mathcal{L}_{\text{sep}} = \max(0, \beta - y_{\text{sep}}), \tag{14}$$

where β is a desired separation threshold.

We proceed to show that $\mathcal{L}$ defined in (12) is a valid loss function by showing it is differentiable. $\mathcal{L}_{\text{BCE}}$ is a well-established loss function for binary classification tasks. It is differentiable and its minimization directly correlates with maximizing the probability of correct classification [5].

We show that the separation loss component $\lambda \cdot \mathcal{L}_{\text{sep}}$ is differentiable and ensures adequate separation between the current classes.

Theorem 5.1. *$\mathcal{L}_{sep}$ is differentiable with respect to the control parameters of the DNPU.*

Proof. The function has piecewise definition:

$$\mathcal{L}_{\text{sep}} = \begin{cases} 0 & \text{if } y_{\text{sep}} \geq \beta \\ \beta - y_{\text{sep}} & \text{if } y_{\text{sep}} < \beta \end{cases}$$

When $y_{\text{sep}} \geq \beta, \mathcal{L}_{\text{sep}} = 0$. The derivative with respect to any control parameter is zero in this region. When $y_{\text{sep}} < \beta, \mathcal{L}_{\text{sep}} = \beta - y_{\text{sep}}$. We need to analyze the differentiability of $\beta - y_{\text{sep}}$.

The function $\beta - y_{\text{sep}}$ is linear in y_{sep}, and hence, its derivative with respect to any control parameter x is:

$$\frac{\partial \mathcal{L}_{\text{sep}}}{\partial x} = -\frac{\partial y_{\text{sep}}}{\partial x} = -\left(\frac{\partial \min I_1}{\partial x} - \frac{\partial \max I_0}{\partial x}\right).$$

here $I_{1,\text{min}}$ and $I_{0,\text{max}}$ are differentiable with respect to the control parameters as they are outputs of the SM, which is a differentiable neural network model. □

Theorem 5.2. *The cost function $\mathcal{L}(y, z)$ is differentiable with respect to the control parameters of the DNPU.*

Proof. The Binary Cross-Entropy loss $\mathcal{L}_{\text{BCE}}$ is known to be differentiable with respect to its inputs, as it involves logarithmic and linear operations, which are differentiable functions. The separation loss $\mathcal{L}_{\text{sep}}$ is also differentiable according to Theorem 5.1. Therefore, the combined cost function $\mathcal{L}(y, z)$, being a sum of differentiable functions, is differentiable with respect to its inputs. □

Now that we have established (12) to be a valid loss function, we compare the its performance on training the SM for XOR function to the loss function proposed in [16].

- The *Correlation Sigmoid Loss Function* proposed in [16] is formulated as follows:

$$\mathcal{L}_{\text{CorrSig}}(y,z) = (1-\rho(y,z))/\sigma\left(\frac{(y_{\text{sep}}-q)}{p}\right), q=3, p=5$$

  where $\rho(y,z)$ is the Pearson correlation coefficient, and $\sigma(x)=\frac{1}{1+e^{-x}}$ is the sigmoid function.

- The *Piecewise Linear Loss Function* defined in (12), the desired separation threshold $\beta=2$ and $\lambda=1$ were chosen as these values demonstrated the quickest convergence during the experiments.

The training losses for both loss functions over 600 epochs of same training data and learning rate are plotted in Figure 16. The Piecewise Linear Loss Function demonstrates rapid convergence and maintains lower, more stable loss values throughout the training period. In contrast, the Correlation Sigmoid Loss Function shows significant fluctuations and ultimately stabilizes at a higher loss value.

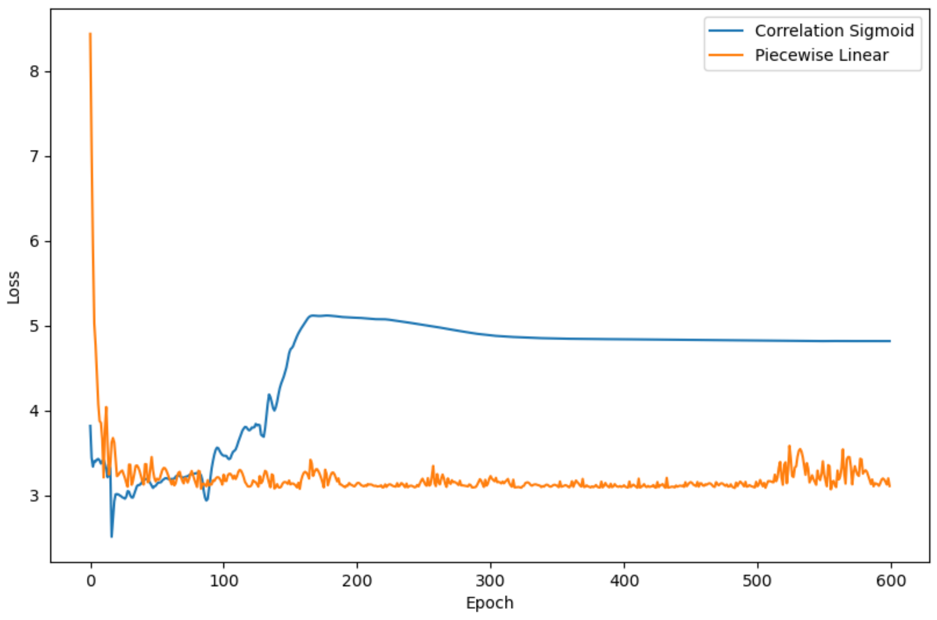


FIGURE 16: Training losses comparison over 600 epochs of learning XOR function. The Piecewise Linear Loss Function (orange) achieves lower and more stable loss values compared to the Correlation Sigmoid Loss Function (blue).

Furthermore, the simplicity of the Piecewise Linear Loss Function makes it easier to implement and interpret, which can be beneficial for further developments and optimizations. Based on these results, we use the Piecewise Linear Loss Function for training the SM for the XOR function.

5.2 Results

In this section, we present the results of our experiments to validate the accuracy of the control voltages determined by the SM for different boolean gates. For each gate, we generated 1,000 samples using KMC simulation for each input pair and the found control electrodes. The predicted values from the SM (blue line) were plotted against the actual simulation values (red dots). Additionally, a separation line was calculated and plotted to distinguish between the logical outputs. The separation line is determined as the average of the highest current value in Group 0 (inputs resulting in logical '0') and the lowest current value in Group 1 (inputs resulting in logical '1') of the simulated currents:

$$\text{separation\_line} = \frac{\max I_0 + \min I_1}{2}.$$

The results (boolean functions performance comparion between SM and KMC simulation) of XOR, AND and OR functions are presented in Figures 17, 18a and 18b, respectively. The found control electrodes corresponding to the results are recorded in Table 3.

The XOR gate

We observe an RMSE of 5.69%, indicating a relatively low prediction error. Notably, the predicted output current for the input pair 01 'overpredicts' the output current, as the actual simulated currents lie below the predicted value [7]. Despite this overprediction, all datapoints in class 0 (input pairs 00 and 11) fall below the separation line at -1.46 nA, while datapoints in class 1 (input pairs 01 and 10) lie above it. This clear division ensures that the separation line effectively distinguishes between the classes (0 and 1) of the XOR function. Consequently, the set of control voltages determined by the SM accurately captures the XOR behavior, validating the model's effectiveness in reproducing the desired logical function.

[7]This suggests that the SM does not perfectly match the KMC simulation results, likely due to the limited amount of simulated data used in our study compared to the 3×10^6 samples used in the study by [16].

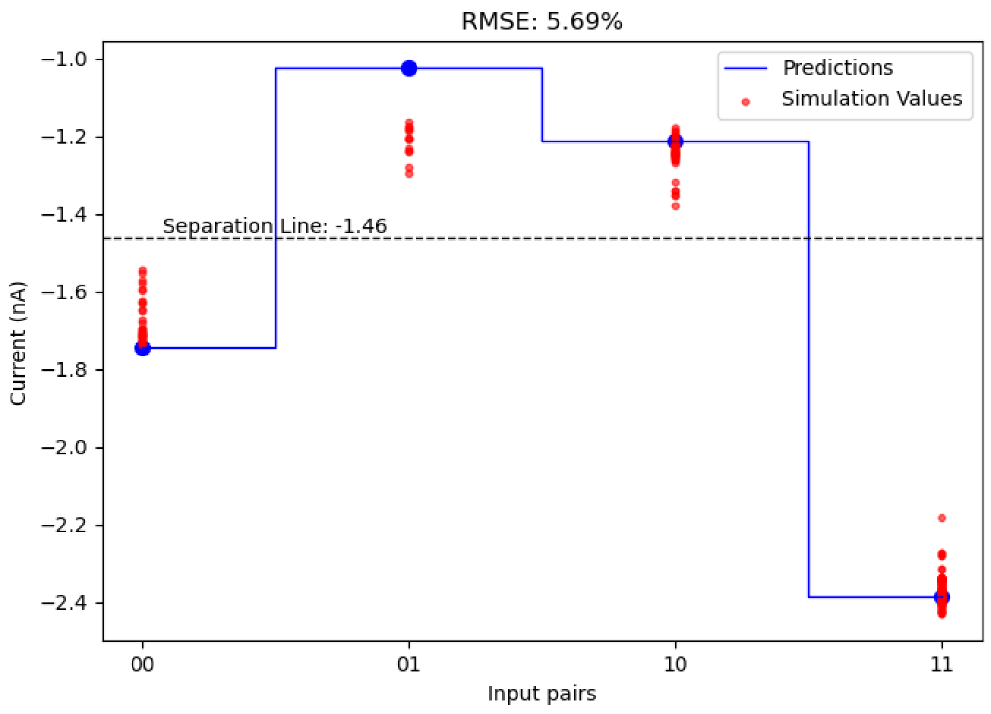
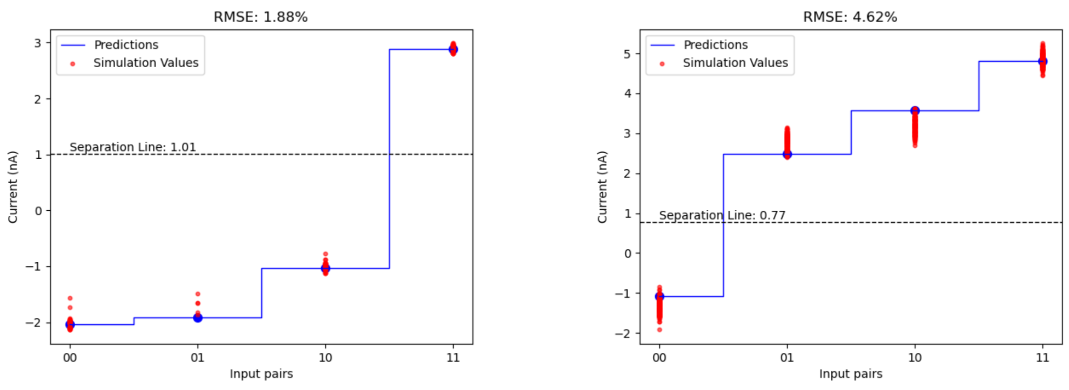


FIGURE 17: Predicted currents (blue line) against the simulation values (red dots) for the XOR function. The x-axis represents the input pairs (00, 01, 10, 11), and the y-axis shows the current in nanoamperes (nA). The dashed line indicates the separation threshold. The provided RMSE is between the SM's predictions and the KMC simulation outputs.



(A) AND function

(B) OR function

FIGURE 18: Comparison of predicted currents and simulation values for the AND and OR functions.

Function	Input 1	Input 2	Input 4	Input 6	Input 7
XOR	-0.2569	-0.5077	0.3191	-0.0151	0.9795
AND	0.4917	-0.5811	-0.4911	0.1593	-0.4207
OR	0.3944	0.1245	-0.0667	0.3826	-0.2614

Table 3: Voltages of control electrodes for Boolean Functions. Input indices correspond to Figure 1B.

Overall, the SM exhibits a high level of accuracy in predicting the output currents for different boolean gates, as evidenced by the relatively low RMSE values for all gates tested. The clear separation lines calculated for each gate confirm that the control voltages determined by the SM effectively differentiate between the logical outputs. Remarkably, the AND and OR gates show a greater separation ($\approx$ 4nA) between classes compared to the XOR gate ($\approx$ 1nA). This increased separation is evidence to the simplicity and linearity of the AND and OR functions, making them easier to model and predict accurately.

6 Conclusion and outlook

This study aimed to investigate whether the minimal size of Dopant Network Processing Units, as suggested by [19], is experimentally viable as a functional DNPU. Our research question was: Can the minimal DNPU size achieve efficient nonlinear classification, specifically implementing an XOR gate?

Our results indicate that the smallest DNPU, with a radius of 16.77 nm and configured with 10 dopants (Figure 1B), can indeed perform complex computations like the XOR function. This conclusion is supported by the surrogate model [16] developed using a deep feedforward neural network and trained using samples generated from Kinetic Monte Carlo simulation [1]. Using a personal computer, our simulations were constrained to 100,000 samples due to computational limitations, resulting in a simulation time of approximately 8 hours. In contrast, larger datasets used in [16] (3×10^6) can enhance accuracy and stability of SMs, emphasizing the benefits of training on extensive datasets.

Future work directions aim to build upon the results of this paper:

1. Neural network of multiple SMs of DNPUs:
   - Explore the scalability of these systems: The next step involves connecting multiple DNPUs to form a network of nanoneurons. This will allow for extensive experimentation on their collective performance and potential as a full-scale neural network for real-world applications.
   - Optimization Techniques: The paper [11] highlights potential methods for optimizing neural networks, such as removing unnecessary connections between neurons. This could further enhance the efficiency and scalability of DNPU-based neural networks.
2. Nanoneurons in Hardware Emulation:
   - The success of these small-sized, energy-efficient DNPUs as nanoneurons highlights their potential in hardware emulation of neural networks. This could

revolutionize neuromorphic computing by enabling more compact and efficient hardware implementations of artificial neural networks.

- These nanoneurons could be integrated into edge devices, providing on-device AI capabilities with minimal power consumption.

In conclusion, our findings affirm the experimental feasibility of the minimal DNPU size for complex computation tasks. Future research should focus on integrating these nanoneurons into larger networks and optimizing their configurations to fully leverage their potential in energy-efficient neuromorphic computing.

References

[1] Marlon Becker. Simulating nano-particle networks to solve classification problems. Master's thesis, Westfälische Wilhelms-Universität Münster, Münster, May 2020.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[3] T. Chen, J. van Gelder, B. van de Ven, et al. Classification with a disordered dopant-atom network in silicon. *Nature*, 577(7790):341–345, 2020.

[4] B. de Wilde. Dopant networks for energy-efficient classification, June 2019.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] T. Hori. Darwin on a chip: Think outside of logic. Master's thesis, University of Twente, 2018.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[8] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. 2015.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[11] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018.

[12] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[14] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, Burlington, MA, USA, ninth edition, 2007.

[15] Hans-Christian Ruiz-Euler, Unai Alegre-Ibarra, Bram van de Ven, Hajo Broersma, Peter A Bobbert, and Wilfred G van der Wiel. Dopant network processing units: towards efficient neural network emulators with high-capacity nanoelectronic nodes. *Neuromorphic Computing and Engineering*, 1(2):1024002, 2021.

[16] H.C. Ruiz Euler, M.N. Boon, J.T. Wildeboer, et al. A deep-learning approach to realizing functionality in nanoelectronic devices. *Nature Nanotechnology*, 15:992–998, 2020.

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2nd edition, 2002.

[18] H. Tertilt, J. Bakker, M. Becker, B. de Wilde, I. Klanberg, B. J. Geurts, W. G. van der Wiel, A. Heuer, and P. A. Bobbert. Hopping-transport mechanism for reconfigurable logic in disordered dopant networks. *Physical Review Applied*, 17(6):064025, 2022.

[19] Gies F.P. Theuws. Neuromorphic computing - the limits of dnpus, July 2022. 10 ECTS BSc Applied Physics Final Project.

A Code reference

The code used for simulations, surrogate model training and finding boolean functions can be found at the following link (https://github.com/spbui00/ml_dnpu).