Karlsruhe Institute of Technology

Institut für Telematik
Pervasive Computing Systems / TECO
Prof. Dr.-Ing. Michael Beigl

Learnable Nonlinear Circuit for printed Neuromorphic Circuits

Lernbare nichtlineare Schaltungen für gedruckte neuromorphe Schaltungen

Master's Thesis
by

Zhidong Yang

Department of Informatics

Responsible Supervisor:	Prof. Dr. Michael Beigl
Supervising Staff:	Haibin Zhao

Project Period: 01/07/2022 – 30/12/2022

Karlsruhe Institute of Technology

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 30. 12 2022

Zhidong Yang

Acknowledgement

I would like to take this opportunity to thank all the people who contributed to this master thesis.

Firstly, I would like to thank my supervisors Haibin Zhao, who always provided helpful advice and guidance throughout my master thesis. Whenever I needed assistance of any kind, he is patient and never hesitated to help me out. For that, I am truly grateful. I would also like to thank my professor Prof. Dr. Michael Beigl.

Finally, I would like to give my special thanks to my family and friends, who give me help, moral support and affection.

Abstract

With the rapid growth of emerging fields such as the Internet of Things and wearable devices, more requirements such as flexibility, low-cost and non-toxicity are posed. However, these advanced properties are often beyond the reach of conventional rigid silicon technology, which make printed electronics increasingly recognized as a key driver of these fields. With neuromorphic computing, printed neuromorphic circuits offer not only the aforementioned properties, but also their learning-based design process offers high optimization efficiency. Generally, printed neuromorphic circuits express their functionality through resistor crossbars to emulate weighted-sum operation, and nonlinear subcircuits to express activation functions. The values of the former are usually learned, while the latter are predefined and considered fixed in training for all tasks. This work focuses on the nonlinear subcircuits of printed neuromorphic circuits and explore an approach to make the components of nonlinear subcircuits optimizable. We conduct experiments to study the effectiveness of this approach. The preliminary experiment shows that, with this approach, the accuracy and robustness (standard deviation) of printed neuromorphic circuits can be improved by 10.7% and 84.8% respectively under 5% variation of circuit components, and 19.2% and 75.5% respectively under 10% variation of circuit components.

Keywords: printed electronics, neuromorphic computing, learnable nonlinear circuit

Zusammenfassung

Mit dem schnellen Wachstum aufstrebender Bereiche wie dem Internet der Dinge und tragbaren Geräten werden immer mehr Anforderungen wie Flexibilität, niedrige Kosten und Ungiftigkeit gestellt. Diese fortschrittlichen Eigenschaften sind jedoch oft außerhalb der Reichweite herkömmlicher starrer Silizium-basierte Technologien, wodurch die gedruckte Elektronik zunehmend als wichtiger Treiber dieser Bereiche anerkannt wird. Denn beim Neuromorphic Computing bieten gedruckte neuromorphe Schaltungen nicht nur die oben genannten Eigenschaften, sondern bieten durch ihren lern-basierten Entwurfsprozess eine hohe Optimierungseffizienz. Im Allgemeinen drücken gedruckte neuromorphe Schaltungen ihre Funktionalität durch Widerstandskreuzschienen aus, um eine Operation mit gewichteter Summe zu emulieren, und nichtlineare Teilschaltungen, um Aktivierungsfunktionen auszudrücken. Die Werte der ersteren werden in der Regel erlernt, während die letzteren vordefiniert sind und im Training für alle Aufgaben als fest gelten. Diese Arbeit konzentriert sich auf die nichtlinearen Teilschaltkreise von gedruckten neuromorphen Schaltkreisen und untersucht einen Ansatz dafür die Komponenten nichtlinearer Teilschaltungen optimierbar. Während wir zusätzliche Flexibilität der auszudrückenden Funktionalität bieten, führen wir Experimente durch, um die Wirksamkeit dieser Methode zu untersuchen. Das Experiment zeigt, dass mit diesem Ansatz die Genauigkeit und Robustheit (Standardabweichung) von gedruckten neuromorphen Schaltungen um 10.7 % bzw. 84.8 % unter 5 % Variation der Schaltungskomponenten, und um 19,2 % bzw. 75,5 % unter 10 % Variation der Schaltungskomponenten verbessert werden kann.

Schlüsselwörter: gedruckte Elektronik, neuromorphes Rechnen, lernbare nichtlineare Schaltungen

Abbreviations

ANN	artificial neural network
Inv	negative weight circuit
NC	neuromorphic circuit
NN	neural network
PE	printed electronics
pNN	printed neural network
ptanh	circuit for tanh-like activation function

Contents

List of Figures

List of Tables

1. Introduction

Internet of Things (IoT) [12] is an emerging field, including ubiquitous computing, commodity sensors, increasingly powerful embedded systems, and machine learning [26]. In recent years, the significant trend of IoT is the explosive growth of devices connected and controlled by the Internet [1]. These devices, which promise revolutionary advances for both industrial applications and the customer experience, enable common things to be seamlessly connected, communicate, and display responsiveness to contextual changes [1]. For example, retail may be able to track specific products through smart labels and verify their authenticity using smart labels [49].

With IoT infrastructures stepping into our daily lives, non-toxic and low-cost electronics are in high demand [31], and there is a growing attention in printed electronics (PE) for fast-moving goods and wearable technologiesIoT. Because it promises a low-cost fabrication of possibly disposable electronic components on various substrates [49] [35]. Additionally, due to the additive manufacturing of PE, it offers high level customization [53]. There are different PE fabrication processes for PE such as jet-printing technology and replicate printing techniques(e.g., screen, flexography and gravure printing) [53]. These processes promise the on-demand fabrication of low-cost, custom circuitry, anywhere and by anyone. In combination with 3D printing and functional inks, people may be able to create their own smart devices with desired geometries and material properties in the future [49].

However, PE suffers from several drawbacks. Among them are mainly high latencies and comparably large footprints [9]. This is especially

true for classical and digital designs, because Boolean digital logic designs would lead to substantial area overhead, low performance, and high power consumption. Consequently the application requirements cannot be met [43]. Additionally, some factors, such as fabrication error of PE [58] and run-time degradation through usage (aging) [60], can aösl influence the performance of printed circuits.

To address these issues, biological-inspired neuromorphic computing is developed and can be leveraged as a suitable computing paradigm for PE. Firstly, neuromorphic computing system can directly process sensory data without converting them to digital signal [58]. Additionally, the use of neuromorphic computing allows formulating the circuit design as an optimization problem similar to training artificial neural networks. Moreover, variations in PE can be considered in the training to obtain robust designs [58]. For example, [58] proposed an inverter-based printed neuromophic circuit, consisting of negative weight circuits, crossbar circuits, and tanh-like nonlinear circuits. It also proposed a design algorithm for the circuits, which is aware of the variation of resistances in the crossbar. Similarly, [60] investigated the aging of crossbar in neuromorphic circuits and proposed an adapted training objective to mitigate aging effects.

However, these works focus mainly on the resistors of the crossbars as learnable parameters, which correspond to weights in neural networks (NNs), whereas the nonlinear components are always predefined and fixed. Since the additive manufacturing property of PE offers high customization level, the design and optimization of nonlinear subcircuits should be also considered. In this work, we investigate the nonlinear subcircuits, namely ptanh and negative weights circuit (Inv) with intention to make their characteristic learnable, i.e., the nonlinear subcircuits' parameterization can be learned together with the values for crossbars resistances (i.e., weights) in training. We refer to this optimizable nonlinear subcircuits as *learnable nonlinear subcircuit*. Additionally, we also explore learning flexibility, namely different training frameworks and learning strategies, to study the performance of printed neuromorphic circuits (NCs).

1.1 Objective

The main aim of this work is to explore the modelling of the nonlinear subcircuits of printed NCs, and integrate the obtained model into the

design process of printed NCs. Additionally, due to the differentiable model of the nonlinear circuits, the fabrication error can also taken into account in the the design process. Furthermore, we explore different training frameworks and learning strategies to study the performance of printed NC. The contributions of this work can be summarized into the following points:

- **Modelling of nonlinear subcircuits of printed neuromorphic circuits**

  To design printed NCs, the model of printed NCs is required. The printed NCs consists of three hardware primitives, namely resistor crossbar, ptanh and Inv. The resistor crossbar realizes a weighted-sum operation (weighted-sum in NN), and the ptanh realizes nonlinear transformation (activation function in NN). Additionally, Inv are required to express a notion of negative weights, as these cannot be implemented directly, more details see subsubsection 2.1.6.2.

  In this work, the process of modelling of ptanh and Inv are introduced. and we obtain NN-based surrogate models for nonlinear subcircuits. The models for three hardware primitives can be assembled to form a model for a printed NC. This model is referred to as printed neural network (pNN). In this way, training a pNN refers to design a printed NC, i.e., the components of nonlinear subcircuits can be optimized in the design of printed NCs.

- **Learning flexibility of printed neuromorphic circuit**

  In this work, we also explore learning flexibility, namely different training frameworks and learning strategies, for printed neuromorphic circuit. There are three training frameworks, namely neuron-level, layer-level, net-layer. Neuron-level means that each neuron has independent surrogate models for ptanh and Inv. Layer-level means that the same surrogate models for ptanh and Inv are shared by all neurons in a layer. Net-level means that the same surrogate models for ptanh and Inv are shared across all neurons. Learning strategies refers to parameters of crossbar, ptanh and Inv are updated simultaneously or alternatively in the training of pNNs.

- **Variation-aware training of printed neuromorphic circuit with nonlinear subcircuits**

Training pNN refers to finding configurations of the adjustable components of the NC to realize a desired functionality. Hence, training can be seen as an automatic design solution for circuits composed of hardware primitives of printed NCs. Variation-aware training refers to taking the fabrication errors of printed components into account during the training for pNNs.

In this work we extend variation-aware training of pNN with the consideration of nonlinear subcircuits. Then we conduct experiments to prove the feasibility of nonlinear subcircuits. Additionally, we explore different training frameworks and learning strategies to study the performance of pNN.

1.2 Outline

The rest of this work is structured in the following chapters:

Chapter 2 provides a background on PE and its challenges. Afterwards, the concept of automatic circuit design is briefly introduced to provide context of this work. Furthermore, NCs, artificial neural networks, gradient-based learning and related work are presented as additional preliminaries to the following chapters.

Chapter 3 introduces the process of modelling of ptanh and Inv, and the integration of these two models into pNN. Additionally, we extend variation-aware training of pNN with the consideration of learnable nonlinear subcircuits.

Chapter 4 introduces the design of experiments for printed NC in different considerations.

Chapter 5 introduces the experiments, which are conducted The framework of Chapter 4 in order to study the performance of pNN with integration of the model of ptanh and Inv into pNN.

Chapter 6 draws conclusions for this work and proposes some suggestions for future work.

2. Background & Related Work

This chapter introduces the background and related work as preliminary for this work. Firstly printed electronics and challenges in designing and modelling printed circuitry are briefly introduced. the followings is a review of automatic circuit design, which is related to the design strategy of training neuromorphic circuitry. Artificial neural network (ANN), as a necessity for this work, is presented, and subsequent introduction are neuromorphic circuit and the concept of neuromorphic computing based on brain-inspired computing paradigm. Additionally, the basic hardware primitives of NC, which work like the key components in ANN, as well as gradient-based learning, which is the most common strategy employed to train ANN, is explained. Finally, the related work of pNN and learnable activation function are outlined.

2.1 Background

2.1.1 Printed electronics

Printed electronics is a new and complementary technology to traditional, silicon-based electronics [40]. Traditional electronics are fabricated using subtractive processes such as lithography and etching, which require the removal of material. Printing electronics, on the other hand, are fabricated with additive manufacturing, i.e., it uses material only where it is needed and provides high level customization [53]. Additionally, low-voltage technologies are developed, such as the printed electrolyte-gated transistors, which make the fabrication costs of printed electronics decreases [39]. Furthermore, printed

electronics can be fabricated with nontoxic materials [49]. Due to its unique properties, printed electronics are able to address a wide range of novel application domains, where conventional electronics can't meet the special requirements [43]. Examples are applications for soft sensors [36]. These factors, such as stretchable, non-toxic, flexible, and low-cost, have led to printed electronics being acknowledged as a promising candidate with the rapid development of IoT [11].

There are different process for the fabrication of printed electronics, and they can be broadly divided into duplicate printing and digital printing. In duplicate printing, such as screen and gravure printing, process, a costly mask or masterplate are required. However, it enables a high throughput and volume production [53]. On the contrary, Digital printing, such as inkjet or aerosol jet printing, doesn't require a mask. Through the nozzle, droplets of conductive ink are ejected onto the substrate to print out the device. This characteristic of digital printing bound its scalability, while it allows for the fabrication of each component and circuit on demand without substantial setup costs [53]. In the future, digital printing would allow the cheap, on-demand fabrication of customized smart devices for home users, which could be popular in the fields of IoT [49].

However, there are some challenges to be overcome. For example, the variations of printed electronics are generally much higher than those of traditional electronics [11], and the solution to these variations need to developed for designing functional circuits [46]. Additionally, because the underlying physics are frequently unknown, only semi-empirical models for devices have been developed far [39]. It is possible, that near sensor processing applications frequently arise in the fields of IoT in the future [23]. Hence, a new and improved computing paradigms should be explored to process sensor data properly [57], although classical digital design, which is yet often not very efficient [9] and requires high device counts [57], is theoretically feasible for such applications.

These factors have prompted the investigation of bespoke analog designs, which allow for the implementation of near sensor processing circuits with a reduced size and power footprint without simultaneously the decrement of performance [57]. Additionally, the design for these circuits might take the specialized requirements and characteristics of the technology, such as the feasibility of certain device types,

into account, Hence, specialized learning algorithms are needed to be developed [23]. Finding a circuit configuration through these algorithms can be seen as an automatic design solution.

2.1.2 Automatic circuit design

The goal of automatic circuit design is to finding a design for a circuit that meets a specific functionality, and the functionality is given through input-output relationships or a behavioural description, which refers to finding an appropriate component topology and dimensions [54]. The Topology refers to deciding which components to connect, while sizing refers to determining the appropriate parameterization of the components, such as the width of a transistor or the conductance value of a resistor [33]. Further solutions may involve the derivation of the circuit layout, including component placement and routing, and automatic circuit design techniques have been broadly classified into knowledge-based and optimisation-based approaches [7]. The first to emerge are knowledge-based techniques that rely on prebuilt design plans [28], and optimisation-based approaches can be further categorized into equation-based, simulation-based, and learning-based approaches [51].

Equation-based techniques [41] optimize the circuit and its components using a set of equations that describe the circuit and its components. As a result, the equations are obtained either analytically and automatically using symbolic analysis tools [7], or through circuit simulation fitting [15]. with these equations evaluations are usually completed more quickly, and numerical solvers, such as interior point methods [25], are applied to tackle the design problem.

Circuit simulations are used in simulation-based approaches to evaluate the performance of a circuit design. Such approaches often leverage evolutionary algorithms [7], or more recently Bayesian optimisation, which are primarily concerned with component sizing [38]. Evolutionary approaches can integrate topology selection with component scaling. Furthermore, they can meet a variety of different design requirements, and meanwhile have minimal limits [24]. Additionally, several simulation-based approaches apply predictive models, such as SVMs [7], to rule out unfavorable solutions in order to limit the number of costly necessary circuit simulations.

Learning-based approaches are based on reinforcement learning [56]. They are more sample efficient since they require fewer circuit simula-

tions than methods with evolutionary algorithm. Furthermore, similar to evolutionary approaches, Reinforcement learning-based methods can perform topology search and component scaling [23].

In contrast to these approaches, this work employs neuromorphic computing with printed NC, see Section 2.1.4. Instead of traditional digital or analog components, printed neural networks [58] are represented by model equations and learned in the same way of ANN, see Section 2.1.3. As a result, the technique in this study may be viewed through classical automatic circuit design as an equation-based optimisation strategy, where circuit design is analogous to ANN. After training, the components, which are similar to the weight in NN, can be mapped to printed neuromorphic circuit.

2.1.3 Artificial neural networks

Artificial neural networks, namely also ANNs, are computing systems inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called artificial neurons, which model the neurons in a biological brain [2]. It can be understood as directed computation graphs with weighted edges (see Figure 2.1), which expresses a function $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ parameterized with $\boldsymbol{\theta}$, and $\boldsymbol{\theta}$ an integration of parameters.

As Figure 2.1 shows, the edges are viewed as connections, and the nodes are called neurons. Furthermore, all neurons can be organized in so-called layers l, and the inputs shared by all neurons in the same layer are same. The computation performed by a neuron j is a weighted sum of all incoming connections i with weights w_{ij}^l plus an additional bias value b_j^l. Finally, a nonlinear function $\phi(\cdot)$ called activation function is applied, and the neuron model equation can described by

$$\text{Neuron}_j^l(\mathbf{x}) = \phi\left(\sum x_i w_{ij}^l + b_j^l\right)$$

where $\mathbf{x} = [x_1, x_2, \ldots]^T$ is considered to be a vector of inputs x_i. Regarding to activation functions, Rectifiers [29] are widely used for simplicity and speed in modern NNs [34]. The basic requirement for an activation function is nonlinearity and differentiability, which are essential for the successful application of gradient-based approaches to update the parameters of NN [20].

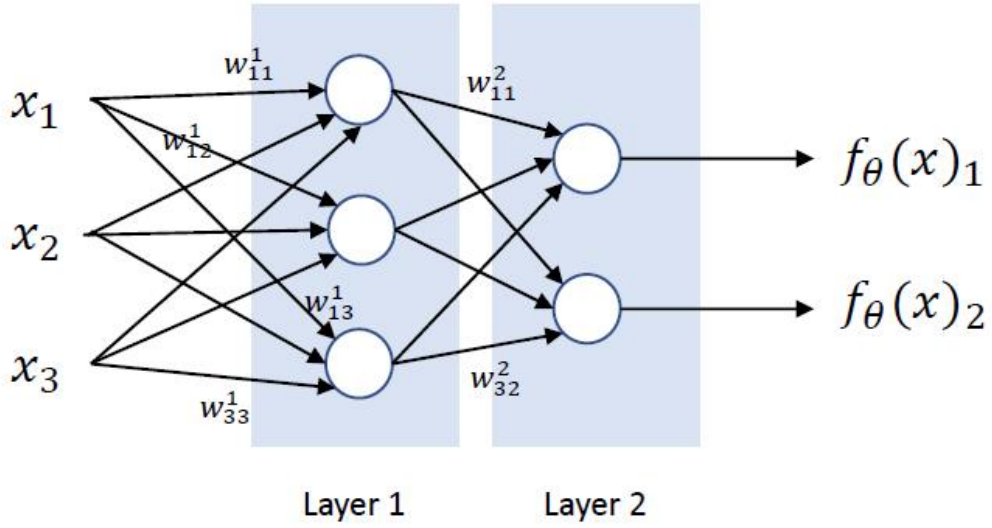


Figure 2.1: A fully connected artificial (feed-forward) neural network mapping three inputs to two outputs. In the layer l, for each node j, all incoming connections i are multiplied with their respective weights w_{ij}^l and summed up with an additional bias b_j^l (omitted in the figure). Afterwards, a nonlinear activation function is applied to the result which forms the neuron activation.

NN are also further implemented and discussed in terms of layers, and a layer denotes the combined computations of several neurons with shared inputs and can be described by

$$\text{Layer}^l(\mathbf{x}) = \phi\left(\mathbf{x}\mathbf{W}^l + \mathbf{b}^l\right)$$

The weights of the neurons in the layer are summarized in the columns of the matrix $\mathbf{W}^l$, the biases are stacked in the vector $\mathbf{b}^l$, and the activation function can be seen as an element-wise operation. Through composing multiple layers, the expression of a NN can be described with a set of equations as

$$f_\theta(\mathbf{x}) = \left(Layer^L \circ \cdots \circ Layer^l \circ \cdots \circ Layer^1\right)(\mathbf{x})$$

Note that the activation function around the most outer layer is optional or could be defined as linear. Additionally, each layer may have a different activation function. In the following, the parameters of NN are summarized in the vector $\boldsymbol{\theta} = \{\mathbf{W}^l, \mathbf{b}^l \mid l = 1, \cdots, L\}$ for brevity and NN is simply denoted as the function $f_\theta(\mathbf{x})$.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) \mid \mathbf{x}_n \in \mathbb{X}, \mathbf{y}_n \in \mathbb{Y}, n = 1, \cdots, N\}$ of input $\mathbf{x}_n$ and desired outputs $\mathbf{y}_n$, a loss function $l(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$ can be defined. The loss function measures a distance of the output of the

network $f_{\boldsymbol{\theta}}(\mathbf{x}_n)$ for a training instance $\mathbf{x}_n$ to the desired output $\mathbf{y}_n$ [20]. Typical choices for the loss function are, e.g., the mean squared error (MSE) for regression, or the cross-entropy (CE) for classification tasks. In a classification setting, which is exclusively considered in this work, $\mathbf{y}_n$ refer to class labels and are usually mapped to natural numbers, i.e., $\mathbb{Y} \in \mathbb{N}$, and $\mathbb{X} \in \mathbb{R}^m$ with $m = \dim(\mathbf{x})$. NN is then build to have as many outputs as the classes. Consequently, each class can be associated with an index of NN output (vector). Most commonly, the loss function is used to define the loss $L(\boldsymbol{\theta})$ over all training instances of $\mathcal{D}$ for a given parameterization $\boldsymbol{\theta}$ as

$$L(\boldsymbol{\theta}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} l(\mathbf{x}, y, \boldsymbol{\theta})$$

Analogously, the loss $L(\boldsymbol{\theta})$ can also be defined as the average of the $l(\mathbf{x}, y, \boldsymbol{\theta})$ for all $\mathcal{D}$. The improvement of the loss $L(\boldsymbol{\theta})$ can be measured either on the training data or on a separate validation dataset.

2.1.4 Neuromophic circuit

Neuromorphic circuit [42] named NC refers to circuit that use brain-like computing concepts. Recently, most publications on neuromorphic computing share one or more of the following characteristics, i.e., non-von-Neumann architectures, analog or mixed-signal computation and spiking behaviour similar to biological neurons [10] [23].

The first characteristic illustrates the contrast between neuromorphic computing and traditional von-Neuman computing. In the process of von-Neuman computing, data is read from memory, computations are performed on the central processing unit, and the result is written back. In contrast, computations in neuromorphic computing are decentralized. In the process of brain-inspired computing, units behave as neurons, and calculations are performed directly and parallel in memory. This process eliminates the so-called "memory-wall" and enhances the speed of certain computations while simultaneously reduces power consumption [6].

The second characteristic is deal with analog computing. Instead of digitizing the signal and constructing it from fundamental AND and OR operations, alternative hardware primitive circuits are employed to implement operations [23]. Most notably, Kirchoff's law enables the achievement of a zero-cost addition [42], and weighted sums are

realized through resistor crossbars [27]. Additionally, analogue input signals from sensors in analogue computing can be processed directly without converting them to digital signals, which reduces time and power consumption [58].

With respect to biological motivation, NCs sometimes attempt to add a time-dependent element, such as charge buildup over time, and spiking neural networks take this into account [23].

In this work, NCs mimic ANN, and the third characteristic is not considered meanwhile the other two characteristics are applied. Such NCs are easier to construct and train, and outperform hardware implementations of spiking neural networks in almost all aspects, including lower latency and reducing the consumption of power and area [17]. Hence, neuromorphic circuits based on artificial neural networks are considered suitable for real-world applications [23].

2.1.5 Gradient-based learning

Gradient descent is an optimisation algorithm for unconstrained optimisation problems with differentiable objective functions. In the training of NN, it is assumed that only the loss function $L(\boldsymbol{\theta})$ should be minimized.

The intuition behind gradient descent can be derived from the Taylor expansion of $L(\boldsymbol{\theta})$ at a point $\boldsymbol{\theta}^{(t)}$ in a direction $\boldsymbol{d}$, i.e.,

$$L(\boldsymbol{\theta}^{(t)} + \alpha \cdot \boldsymbol{d}) = L(\boldsymbol{\theta}^{(t)}) + \alpha \cdot \boldsymbol{d}^T \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) + O(\alpha^2)$$

For $\alpha \in \mathbb{R}^+$ small enough($\alpha \approx 0$), the $O(\alpha^2)$ part can be neglected and the expression reveals a characterization of a descent direction for $\boldsymbol{d}$, i.e., if $\boldsymbol{d}$ is chosen such that $\boldsymbol{d}^T \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \leq 0$, then $L(\boldsymbol{\theta}^{(t)} + \alpha \cdot \boldsymbol{d}) \leq L(\boldsymbol{\theta}^{(t)})$. More specially, the value of the function $L(\boldsymbol{\theta})$ can be decreased when $\boldsymbol{\theta}^{(t)}$ moves in the direction $\boldsymbol{d}$.

A recommendation for the descent direction is $\boldsymbol{d} = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})$, which leads to the gradient descent update rule for $\boldsymbol{\theta}$ with

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})$$

The parameter $\alpha^{(t)} \in \mathbb{R}^+$ refer to as learning rate or step size at iteration t, and is often simply a small and constant value, or chosen through one of various heuristics. More recent techniques are based on gradient descent that use knowledge about earlier gradients in the

update algorithm, such as Adam [32]. In the training of NN, the update rule can be stopped either after a fixed number of iterations, or if the loss $L(\boldsymbol{\theta})$ doesn't improve any more over a fixed number of epochs, such as early-stopping [45].

Another important aspect to be considered is the initialization of weights, i.e., the starting points $\boldsymbol{\theta}^{(0)}$. Especially when applying gradient-based learning to optimize NN, the initially chosen $\boldsymbol{\theta}^{(0)}$ (initialization) can greatly influence the success of learning [18]. Because unsuitable initializations may lead to unfavourable propagation dynamics, which hamper the learning process. For instance, suppose the initializations in a layer mapping nearly all inputs to a saturation region of the activation function, which means $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \approx 0$ (vanishing gradients), the loss $L(\boldsymbol{\theta})$ thereby decrease very slowly, and the updates doesn't work. To avoid this situation, multiple strategies have been developed to find good parameter initialization, e.g., the weights are usually drawn uniformly or normally distributed around zero [20]. Additionally, the choice of activation function can also influence the learning process [21], as it influences the standard deviation of the layer outputs. For the computation of the gradient in NN, backpropagation algorithm are widely applied because of its efficiency [50], and it is combined with automatic differentiation techniques [44].

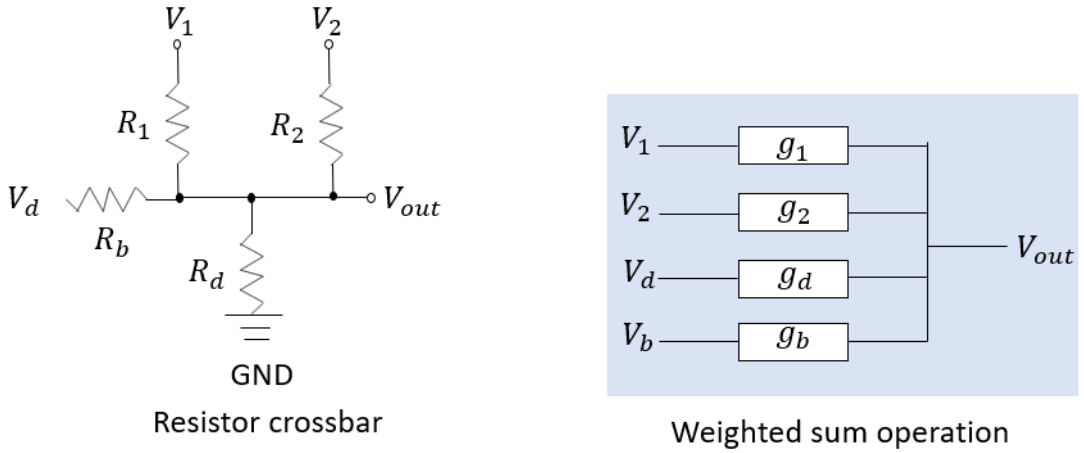
In conclusion, parameterized models can be optimized using gradient-based learning to behave in a way that is determined by the loss function. Additionally, proper choices for initialization of weights and activation function are the important parts for gradient-based learning. If printed neuromorphic circuit can map into ANN, we can also use gradient-based learning to optimize printed neuromorphic circuit.

2.1.6 Hardware primitives for neuromorphic computing

Since the NC in this work draws inspiration from ANN, the functionality of the fundamental components of NC should be same as the core components of ANN, i.e., neurons constructed from an activation function $\phi(\cdot)$ and the weighted sum operation $\sum_i x_i w_i + b$. Different implementations of printed NC have been proposed for the activation function and the weighted sum operation, and the components of NCs [58] in this work are described in the following.

2.1.6.1 Weighted-sum crossbars

For the weighted-sum $\sum_i x_i w_i + b$, most works utilize resistance crossbars [59], where the input and output signals are represented by voltages (see Figure 2.2).



(a) (b)

Figure 2.2: A weighted sum operation of a neuron realized through a resistor crossbar.

Through Kirchhoff's law and Ohm's law, the (output) voltage V_{out} can be calculated as

$$\sum_i I_i + I_b + I_d = 0 \Leftrightarrow$$

$$(V_b - V_{out}) \cdot g_b + (V_d - V_{out}) \cdot g_d + \sum_i (V_i - V_{out}) \cdot g_i = 0 \Leftrightarrow$$

$$\begin{aligned} V_{\text{out}} = & \underbrace{V_b \cdot \frac{g_b}{g_b + g_d + \sum_j g_j}}_{b} + \underbrace{V_d \cdot \frac{g_d}{g_b + g_d + \sum_j g_j}}_{V_d=0} \\ & + \sum_i \underbrace{V_i \cdot \frac{g_i}{g_b + g_d + \sum_j g_j}}_{x_i \cdot w_i} \end{aligned} \tag{2.1}$$

For simplicity and brevity of notation, conductance values g_i is used instead of resistance values, i.e., $g_i = R_i^{-1}$. The analogy to the weighted sum operation can be readily seen by interpreting x_i as V_i, and a weight w_i is a function of all conductances of the crossbar resistors and given by

$$w_i = \frac{g_i}{g_b + g_d + \sum_j g_j}$$

The bias b is realized by the product of V_b and the fraction of g_b of the sum of all conductances of the crossbar. For simplicity of the resulting circuit, V_b will be chosen as a fixed value (assuming $1V$ in the following), and the bias b behaves thereby like a weight w_b with a fixed input $V_b = 1V$. The values of the parameters w_i and b are bound to the range of $[0, 1]$ and are coupled through the denominator $g_b + g_d + \sum_i g_i$. Since this coupling would reduce the effective number of free parameters of the neuron by one, it is relaxed via an additional pseudo-input connection $V_d = 0V$ (ground) with a conductance g_d. Additionally, technological limitations have to be considered, For example, the range of feasible conductance values $g_i \in [g_{min}, g_{max}]$.

As is described in Section 2.1.5, that proper initialization of the parameters can be essential for the success of neural network training [19]. In this work, we take θ related to conductance as learnable parameters (weight in NN), i.e., $g = |\theta|$. Equation 2.1 shows, that the scale of the resulting network weights w_i is already directly related to the number of inputs through the coupling constraint of the crossbar. We can therefore simply initialize the surrogate conductances θ_i, i.e., the starting points $\theta_i^{(0)}$, uniformly around zero with a constant deviation. Additionally, it should be initially set to the highest possible value $\theta_d^{(0)} = g_{max}$ in order to allow for maximum decoupling.

2.1.6.2 Negative weights circuit

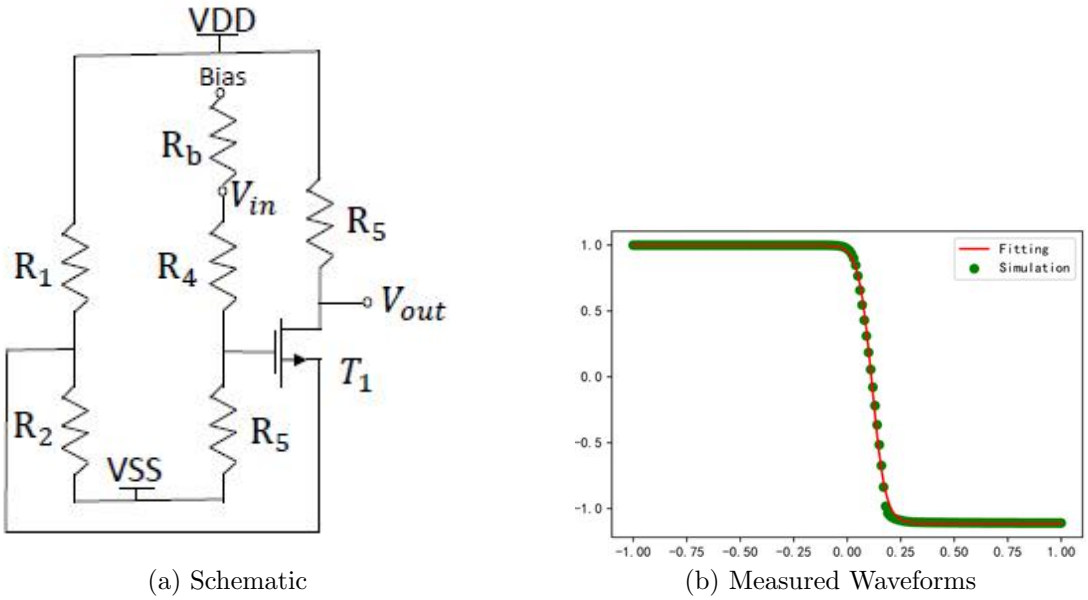
Since conductances g_i can only be nonnegative, the weights w_i (see Equation 2.1) are also restricted to nonnegative values. Hence, a concept of negative weights is necessary to address this problem, and the idea is to change the corresponding input x_i of w_i, i.e., the product of x_i with a negative weight is expressed through $x_i(-w_i) = (-x_i)w_i$. Unfortunately, $-x_i$ cannot be directly generated so far and only be approximately implemented by inverting the signal x_i through an appropriate circuit. One such circuit, namely negative weight circuit (Inv), has been realized in printed electronics [58]. The figure (a) in Figure 2.3 represents the schematic of a circuit for a negative tanh-like transformation, and the red line of figure (b) in Figure 2.3 represents the form of responding tanh-like function. Hence, the characteristic curve of the circuit can be approximately described by a modified negative tanh-like function $inv(\cdot)$, i.e.,

$$V_{out} = \text{inv}(V_{in}) = -(\eta_1 + \eta_2 \cdot \text{inv}((V_{in} - \eta_3) \cdot \eta_4)) \tag{2.2}$$

where V_{in} and V_{out} are the input and output voltages of the negative tanh-like circuit, and the auxiliary parameters $\boldsymbol{\eta} = [\eta_1, \eta_2, \eta_3, \eta_4]$ modifies the tanh function in translating and scaling. Obviously, $\boldsymbol{\eta}$ is determined by the resistors R and the electrolyte-gated transistor T. The design parameters of the transistors are its width W and length L [59]. In this way, the function of Inv is then described by

$$V_{out} = \text{inv}_{\boldsymbol{\eta}(\boldsymbol{p})}(V_{in})$$

where $\boldsymbol{p} = [R_1, R_2, R_3, R_4, R_5, W, L]$ summarizes all design parameters in this circuit. Hence, the notion of negative weights is expressed through $x_i(-w_i) = (-x_i)w_i \approx inv(x_i)w_i$. However, since $inv(x_i)$ only represents an approximation to $-x_i$, the characteristic of $inv(x_i)$ has to be accounted for when modelling and training printed NCs. In general, calculating $\boldsymbol{\eta}$ directly from $\boldsymbol{p}$ is sophisticated, therefore, previous works only focused on the conductances g_i in the crossbar and used a fixed $inv(\cdot)$. In this work, the behaviour of this nonlinear circuit, i.e., $\boldsymbol{\eta(p)}$, will be investigated and parameterized in terms of its parameters $\boldsymbol{p}$, which enables its design optimization. We refer to this optimizable nonlinear circuit as *learnable nonlinear circuit*.



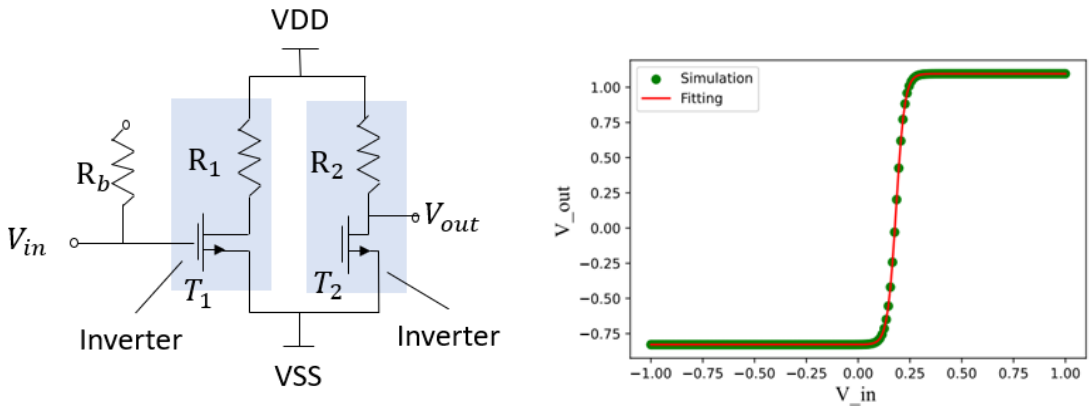
(a) Schematic (b) Measured Waveforms

Figure 2.3: (a) The schematic of the proposed negative weight circuit. (b) Contains the simulated and fitted negative tanh-like function of negative weight circuit.

2.1.6.3 Circuit for tanh-like activation function

The activation function is the second essential part of a NN after the weighted-sum operation. As is explained in Section 2.1.3, that the

basic requirement for an activation function is nonlinearity and differentiability. Many of the current methods aim to resemble well-known functions such as ReLU or tanh. The latter was similarly realized in printed electronics and referred to as ptanh [58]. The left figure in Figure 2.4 represents the schematic of a circuit for a tanh-like transformation and the red line of right figure in Figure 2.4 represents the form of corresponding modified tanh-like function. Hence, the characteristic curve of the circuit can be approximately described by a modified tanh-like function $ptanh(\cdot)$, i.e.,



(a) Schematic (b) Measured Waveforms

Figure 2.4: (a) The schematic of the circuit for realizing tanh-like function. (b) Contains the simulated and fitted transfer function of the circuit for realizing tanh-like function.

$$V_{out} = \text{ptanh}(V_{in}) = \eta_1 + \eta_2 \cdot \tanh((V_{in} - \eta_3) \cdot \eta_4) \tag{2.3}$$

where V_{in} and V_{out} are the input and output voltages of the tanh-like circuit, and the parameter $\boldsymbol{\eta} = [\eta_1, \eta_2, \eta_3, \eta_4]$ modifies the tanh function in translating and scaling. Obviously, $\boldsymbol{\eta}$ is also determined by the resistors R and the electrolyte-gated transistor T, and the design parameters of the transistors are its width W and length L [59]. In this way, function of ptanh is then described by

$$V_{out} = \text{ptanh}_{\boldsymbol{\eta}(\boldsymbol{p})}(V_{in})$$

In this work for simplicity, we use the same circuit as Inv with inverted VDD and VSS to achieve the ptanh. Analogous to the Inv, we will also bring the components in this circuit to be optimizable. To distinguish the parameters from Inv, we introduce the superscript ACT and INV, i.e., $\boldsymbol{p}^{ACT}$ and $\boldsymbol{p}^{INV}$

This activation function exhibits regions where the absolute value of the strength is increased, i.e., there are x such that $|ptanh(x)| > |x|$.

These areas enable signal magnitude to be maintained or even increased while enhancing signal separation. Only the inverter-based ptanh activation function is taken into consideration in the following because signal separation is crucial for measuring, differentiating, and comparing the signal strengths at the multiple outputs of the NCs [23]. In Section 2.1.5, vanishing gradients can occur because of the combination of initialization and activation function. For example, when here $V_{in} > 2.0$, there is fast no gradients, i.e., the derivative of ptanh is zero, then gradient-based learning doesn't work. Hence, the initialization should make the derivative of ptanh is the highest, i.e., here the output of activation function at the begin of training should be zero and benefits the training dynamics at the beginning of the training process. Assuming that the other θ have been initialized already (see subsubsection 2.1.6.1), this can be achieved by reformulating the activation function $ptanh(\cdot)$, and the initialization of the surrogate conductance is thus given by

$$\theta_b = \frac{ptanh^{-1}(0)}{1 - ptanh^{-1}(0)}(\sum_i |\theta_i| + |\theta_d|)$$

2.2 Related work

The related work is mainly deal with printed neural network. The high variations in printed structures are a challenge of designing printed circuits [11], and these variations have an impact on printed devices' electrical characteristics, which in turn affects the outputs of circuits. If ignored, the printed NCs can show substantially worse accuracy than initially expected. To make NCs more robust to various variations and faults, various approaches have been proposed. The majority of research focuses on changing the parameters of printed NCs using either additive or multiplicative variation or noise [52] [13] [37]. More recently, timing faults have also been considered [16] [60]. In this work, a variation-aware training method [23][58] is applied and here some important contents are introduced.

2.2.1 Loss function

In the training of NN, the loss function guides the learning process and expresses the favorability of a set of parameters $\boldsymbol{\theta}$. For pNN, there is a modelling assumption for loss function, i.e., a sufficient signal separation should be encouraged to be able to clearly measure and

distinguish different outputs [23]. More specifically, the value of the correct output of the network $f_{\boldsymbol{\theta}}(\mathbf{x})$ should exceed T and all other outputs should return values below 0. To express these conditions, a loss function inspired by a multi-class-hinge-loss [14] is constructed by

$$L(\boldsymbol{\theta}) = \sum_{(\mathbf{x})\in\mathcal{D}} l(\mathbf{x}, y, \boldsymbol{\theta}) \tag{2.4}$$

with

$$l(\mathbf{x}, y, \boldsymbol{\theta}) = (m + T - f_{\boldsymbol{\theta}}(\mathbf{x}))^+ + (m + \min_{i\neq y} f_{\boldsymbol{\theta}}(\mathbf{x})_i)^+$$

Here, $T \in \mathbb{R}^+$ denotes an implementation-specific measuring threshold, $m \in \mathbb{R}^+$ is a user-defined parameter called margin and $(\cdot)^+ = max\{0, \cdots\}$. Using this loss function, a positive loss is incurred if the output activation of the neuron associated with the correct class $f_{\boldsymbol{\theta}}(\mathbf{x})$ does not surpass $m + T$ (margin and threshold), or the activation of any other network output $f_{\boldsymbol{\theta}}(\mathbf{x})_{j\neq y}$ is bigger than $-m$. The parameter T should be set to a value above which a signal is measurable and can be clearly distinguished from 0. The margin m represents a hyperparameter and can be used to further improve the separation [23]. In this work, uniformly with [23] [60], we use $m = 0.3$ and $T = 0.1$.

2.2.2 Gradient-based learning for feasible conductance

A pNN can be trained using gradient-based learning as described in Section 2.1.5, and we use backpropagation [50] in this work. The surrogate conductances $\boldsymbol{\theta}$ in Equation 2.4 represent the learnable parameters. However, the values of the surrogate conductances must be restricted in order to guarantee that the values of all surrogate conductances are eventually fabricable. Hence, we need to reformulate the process of gradient-based learning.

The range of feasible conductance values is $|g| \in [g_{min}, g_{max}] \cup \{0\}$ (0 relates to not printing) [23], i.e., $|\theta| \in [g_{min}, g_{max}] \cup \{0\}$. To deal with the infeasible region $[-g_{min}, g_{min}]$, we employ the method used in another related work [60]. Firstly, we directly project the parameters $\theta \in [-g_{min}, g_{min}]$ to 0 in the forward pass of training, because zero-valued surrogate conductances don't need to be printed, which reduces the amount of material needed and the amount of time needed for fabrication. However, this would result in the gradient descent obtaining a zero gradient for the parameters in the corresponding regions. So we leverage the gradient of infeasible region with an estima-

tor, namely straight-through estimator [8], to perform backpropagation. More specifically, in the forward pass of training, the function of straight-through estimator is described by

$$\theta_{ste} = \begin{cases} 0 & |\theta| < g_{min} \\ sign(\theta) \cdot g_{max} & |\theta| > g_{max} \\ \theta & otherwise \end{cases} \tag{2.5}$$

while in the backward pass of training the gradient of a surrogate conductance θ (w.r.t. θ_{ste}) is 1, i.e., $\nabla_{\theta_{ste}} = 1$. Through the use of the straight-through estimator, backpropagation can be applied for training and all θ remain feasible throughout the training.

2.2.3 Variation model for printed conductance

A resistor's conductance state changes due to deviations of the geometrical parameters such as length or area, or variations of the material-related quantity expressed as the conductivity. If an industrial printing process with perfect alignment of printed resistors is assumed, geometrical changes can be disregarded, leaving only variations in the material's conductivity caused by variations in the material's properties or by variations in the environment's temperature and humidity [23]. This presumption leads to the formulation of a conductance variation model with a variation level ϵ by

$$g = \overline{g} \cdot (1 + r) \quad \text{with} \quad r \sim p(r) \tag{2.6}$$

where $\overline{g}$ denotes the intended conductance and r is a random variable drawn from the distribution $p(r)$. Since the conductance variation is assumed independent, the joint variation model for the conductances is simply the product of the individual distributions.

2.2.4 Variation-aware training

Variation-aware training means, that the fabrication errors are taken into account by using the derived variation models in the training of pNN. In the training of classic neural network, the loss function $L(\boldsymbol{\theta})$ is minimized with respect to the parameters $\boldsymbol{\theta}$. However, given variations model of conductances, the surrogate conductances $\boldsymbol{\theta}$ should be considered as random variables. Consequently, the loss $L(\boldsymbol{\theta})$ should be also considered as a random variable, and we should reformulate the loss function with expected loss w.r.t the random variable $\boldsymbol{\theta}$.

The distribution is given by the variation models (for a set variation level ε_θ) and influence which parameters are sampled. Depending on which nominal values are aimed for, different samples of $\boldsymbol{\theta}$ are obtained, i.e. $\boldsymbol{\varepsilon_\theta \theta}$. the training objective therefore becomes

$$min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\varepsilon}_\theta}[L(\boldsymbol{\theta})] = \int L(\boldsymbol{\varepsilon_\theta \theta}, \boldsymbol{x}, \boldsymbol{y}) p(\boldsymbol{\varepsilon}_\theta) d\boldsymbol{\varepsilon}_\theta \tag{2.7}$$

Nevertheless, since the objective function in Equation 2.7 (and also its gradients) will usually not have an analytical solution, approximations via, e.g., Monte Carlo estimates through samples from $\boldsymbol{\varepsilon_\theta \theta}$ have to be used. Hence, estimating the objective function for a given ε_θ through Monte Carlo samples is straightforward, With these reformulations, the expected loss can be estimated through

$$\mathbb{E}_{\boldsymbol{\varepsilon}_\theta}[L(\boldsymbol{\theta})] \approx \frac{1}{N} \sum_{n=1}^{N} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\varepsilon}_\theta^{(n)} \boldsymbol{\theta})$$

with $\boldsymbol{\varepsilon}_\theta^{(n)} \sim p(\boldsymbol{\varepsilon}_\theta)$, where N is the number of samples drawn from their distribution in each epoch. In pNN, surrogate conductances θ_i is used as learnable parameter [58] [23], and only the conductances of resistance crossbars are considered as learnable parameter, meanwhile the other parts of pNN is predefined and fixed [58].

2.2.5 Learnable activation function

There are many researches and papers, show that learnable activation function can improve the performance of neural network. For example, Parametric Rectified Linear Unit (PReLU) [22], which achieve 4.94% top-5 test error on the ImageNet 2012 classification dataset. SPLASH units [55], which simultaneously improve the accuracy of deep neural networks while also improving their robustness to adversarial attacks, and S-shaped rectified linear activation unit (SReLU) [30]. To our knowledge, there is currently no design framework for printed neuromorphics that take the components of Inv and ptanh as learnable parameter.

3. Learnable nonlinear circuit

In this section, we extend the learnable parameters in pNNs by the parameters of nonlinear subcircuit. More specifically, we introduce differentiable, NN-based surrogate models for the nonlinear subcircuit, and integrate them into the pNN model (w.r.t their constraints). Additionally, we modify the variation-aware training of pNNs with the proposed learnable nonlinear circuits.

3.1 Modelling of nonlinear suncircuits

Modelling of nonlinear circuitry means to find the mapping from the circuit components $\boldsymbol{p}$ to the parameters $\boldsymbol{\eta}$ in the tanh-like function. This mapping should be differentiable, because we use back-propagation in the training of pNN. With the inspiration of NC we decide to choose NN as surrogate models to approximate this mapping.

As shown in Figure 3.1, to model the nonlinear subcircuit, the design space of circuit components should be firstly defined. Subsequently, the database that describes how the circuit behavior is established. Finally, an NN is trained as the approximator from $\boldsymbol{p}$ to $\boldsymbol{\eta}$. Here we briefly introduce the process of modeling of Inv. For ptanh we use the same circuit as Inv with inverted VDD and VSS, so the process of modelling of ptanh is analogous to Inv.

3.1.1 Design space

In accordance with printing technology and circuit design experience, we empirically limited the design parameters in the range $[p_{min}, p_{max}]$.

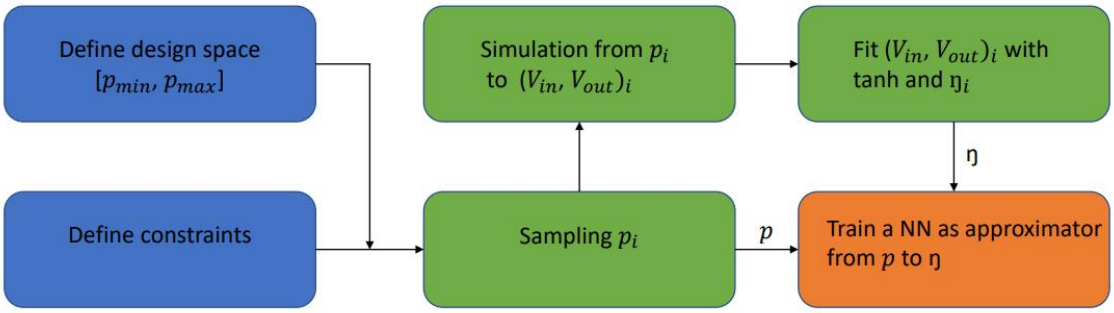


Figure 3.1: Pipeline for modeling of nonlinear circuits. The blue boxes indicate the data preparation, the green boxes denote the process of building database, and the orange box refers to the model approximation.

Moreover, to keep the characteristic curve likely to a tanh-like function, two inequality constraints are introduced. All constraints are reported in Table 3.1. It is to be noted that if we vary the ratio of the resistance parameters, we have one less degree of freedom to learn the circuit. However, we do not consider the ratio of the resistances, as the absolute values of the voltage divider resistances matters most. The resistance R_1 and R_3 must be sufficiently smaller than R_2 and R_4 respectively, otherwise the voltage divider cannot meet the assumption of a constant ratio due to the connections with surrounding circuit elements. The factors how much one resistance is greater than the other resistances for correct learning are found empirically by performing sweep analysis of all the resistance in the simulation tool which leads to the desired behaviour [58]. If we don't change the resistances sufficiently, the functionality will not be substantially affected.

	$R_1(\Omega)$	$R_2(\Omega)$	$R_3(k\Omega)$	$R_4(k\Omega)$	$R_5(k\Omega)$	$W(\mu m)$	$L(\mu m)$
minimal	10	5	10	8	10	200	10
maximal	500	250	500	400	500	800	70
inequality	$R_1 > R_2$		$R_3 > R_4$		-	-	-

Table 3.1: Feasible design space of non-linear circuit

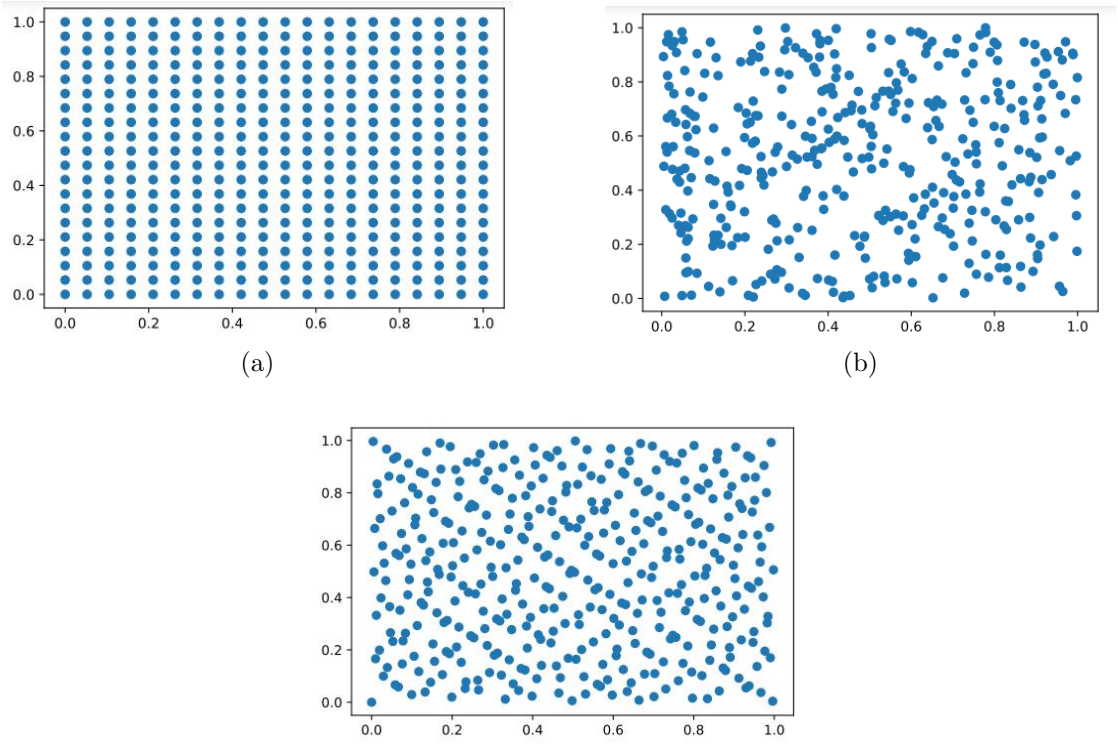
3.1.2 Database

The database consists mainly of the design parameters $\boldsymbol{p}$ and the parameters $\boldsymbol{\eta}$ in tanh-like functions. To obtain a set of $\boldsymbol{p}$ that are representative enough for the whole feasible design space, the sampling way should be considered.

- **Sampling ways**

Sampling is a process in statistical analysis where researchers take a predetermined number of observations from a larger population. In this work, there are four sampling ways introduced and discussed to find the suitable values of circuitry components.

Grid search sampling is based on sampling with equidistant, i.e. the value is sampled with equal distance in each step. Although the distribution of sampling value is evenly, in some situation many sampling values have same information, i.e. the values are not informative. For example, in figure 3.2a many sampling value have same x-value or y-value.



(a) (b)

(c)

Figure 3.2: Three sampling way, which sample 400 points from a square of length 1. (a) displays the result of Grid search sampling. Figure (b) displays the result of Monte Carlo sampling. (c) displays the result of Quasi Monte Carlo sampling.

Monte-Carlo sampling is based on random sampling. The basic principle of the Monte Carlo method is that the probability of the event can be estimated with the frequency of numerous tests. When the sample capacity is large enough, the frequency of the incident is the probability. Although Monte-Carlo sampling is a random sampling way and the results can be informative, the distribution of results is not evenly. There is a example to explain it(see 3.2b).

Mixed grid-search & Monte-Carlo Sampling combines the two aforementioned sampling ways. In this work it means, that a part of values are sampled by grid search sampling meanwhile the other part are sampled by Monte-Carlo sampling. Although this way takes the advantages of two sampling ways into account, it can't promise the aforementioned effect, i.e, the distribution of sampled values is evenly and the sampled values are informative.

In contrast to Monte Carlo method, the Quasi-Monte Carlo method solves problems using low-discrepancy sequences (also called quasi-random sequences or sub-random sequences) [3], which takes the distribution of sampled value into account. More specially, not only this way bases on random sampling, but also the distribution of sampled value is evenly. there is a example to explain it(see 3.2c).

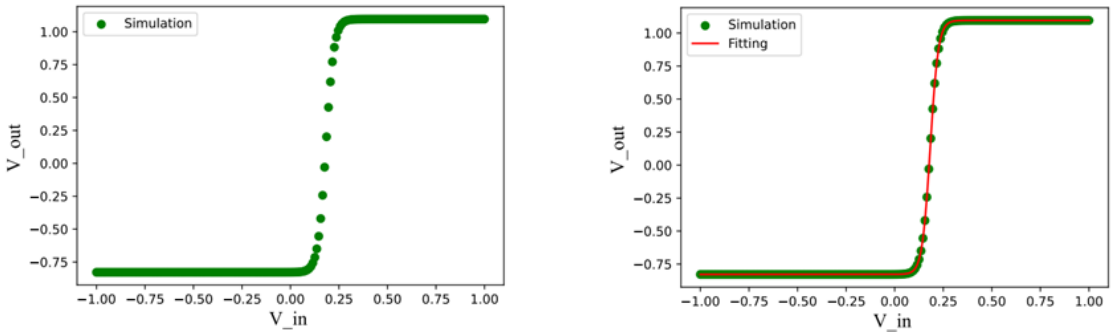
In this work, we don't know the concrete distribution of values of circuitry components, i.e., we don't know which values are useful. Based on this premise, we should sample the different values as many as possible, and the range of sampling values should be as wide as possible, i.e., the sampling values should be informative and have high margin. As the pictures of sampling ways show(see Figure 3.2), Quasi Monte-Carlo sampling meets these requirements.

Hence, We employ the Quasi Monte-Carlo sampling to draw 10000 points in the feasible design space, which are denoted by $\boldsymbol{p}_i, i = 1, ..., 10000$. Afterwards, we use with Cadence Virtuoso [4] for SPICE simulation based on a prior developed printed Process Design Kit (pPDK) [48] to simulate the input and output voltages $(V_{in}, V_{out})_i$. it is to be noted, that besides input and output voltages, there is a bias voltage which is to be considered for proper functioning of the circuit. In theory, we can use any DC bias voltage but in practical, we have some restrictions. The bias pin is connected to resistors which are connected to the EGT. If an EGT is applied to a voltage higher than 2V (it is found empirically), the electrolyte degrades and the EGT can be destroyed accidentally. So a relative voltage drop at the EGT (from Drain to Source or from Gate to Source) exceeding 2V has to be prevented. However, we use zero bias voltage for simplicity. In simulation, we perform DC sweep of input from 2V to -2V with linear step size of 0.01, and we capture the data for the transfer characteris-

tic of input and output voltages $(V_{in}, V_{out})_i$. To extract $\boldsymbol{\eta}_i$, we fit the simulated data $(V_{in}, V_{out})_i$ with Equation 2.2 with minimal Euclidean distance [5], e.g.,

$$\boldsymbol{\eta}^* = \arg\min_{\boldsymbol{\eta}} \|\text{inv}_{\boldsymbol{\eta}}(V_{in}) - V_{out}\|_2$$

The green points in Figure 3.3(a) exemplifies a simulation result with a certain $\boldsymbol{p}_i$. The right curve in Figure 3.3(b) is the corresponding fitted tanh-like curve. By now, we obtain numerous design parameters $\boldsymbol{p}_i$ and their corresponding parameters in tanh-like functions $\boldsymbol{\eta}_i$. In the next step, an NN-based approximator to imply the transformation from $\boldsymbol{p}_i$ to $\boldsymbol{\eta}_i$ will be build.



(a) Simulation (b) Fitting

Figure 3.3: (a) illustrates an example of the simulation of ptanh. (b) Contains the simulation value from (a) and the corresponding fitting function of ptanh.

3.1.3 NN-based Approximator

Inv contains printed electrolyte-gated field-effect transistors [47], and the relationship between $\boldsymbol{p}_i$ and $\boldsymbol{\eta}_i$ sophisticated. Based on existing knowledge, an accurate model cannot be built. Hence, we propose to use an NN as the approximator to approximate the behavior of Inv. Neural networks are mathematical approximation model consist of simple primitive operations(weighted-sum and activation function), meanwhile has high ability to approximate designed functionalities. Given input X and corresponding output y and in order to get the correct y, we need to find a specific expression between X and y, which can be done by training the NN. In this work, the approximator provide a differentiable mapping from $\boldsymbol{p}_i$ to $\boldsymbol{\eta}_i$.

In summary, firstly we define feasible design space and constraints for the circuit components $\boldsymbol{p}$ of Inv. Subsequently, a set of representative values of these circuit components are obtained with Monte-Carlo

sampling, then with these sampling values the simulation of Inv are carried out to get the input and output voltages, and the input-output curves are parameterized with tanh-like model. Finally the transformation from component values to the parameters in the input-output curves are modelled with a NN-based approximator.

3.2 Constraints of nonlinear subcircuits in pNN

To integrate the NN-based approximator into the pNN and make the design parameters $\boldsymbol{p}$ optimizable, We introduce new learnable parameters $\boldsymbol{\rho}$ for the pNNs. To simplify the inequality constraints, we don't directly R_1 and R_3 learnable, but their corresponding difference $dR_1 = R_1 - R_2$ and $dR_3 = R_3 - R_4$. thus,

$$\boldsymbol{\rho} = [R_2, R_4, R_5, W, L, dR_1, dR_3]$$

where the seven parameters have the own feasible range, so we clip them into the feasible range by a straight through estimator [60], i.e.,

$$\boldsymbol{\rho} := clip(\boldsymbol{\rho})$$

After this, We transform them into the value of component with $R_1 = dR_1 + R_2$ and $R_3 = dR_3 + R_4$, i.e.,

$$\boldsymbol{p} = [R_1, R_2, R_3, R_4, R_5, W, L]$$

Since the element R_1 and R_3 are inferred, We clip them into the feasible range by a straight through estimator. After forcing all design parameters in feasible space, physical quantities $\boldsymbol{p}$ is extended to $\boldsymbol{p}'$ by the ratios $k_1 = R_2/R_1, k_2 = R_3/R_4, k_3 = W/L$(explanations see Section 4.1), i.e.,

$$\boldsymbol{p} \mapsto \boldsymbol{p}' = [R_1, R_2, R_3, R_4, R_5, W, L, k_1, k_2, k_3]$$

Then $\boldsymbol{p}'$ is normalized to $\tilde{\boldsymbol{p}'}$ by p_{min} and p_{max}. The yield $\tilde{\boldsymbol{p}'}$ will be put into the NN-based approximator, i.e.,

$$\tilde{\boldsymbol{\eta}} = \text{Approximator}(\tilde{\boldsymbol{p}'})$$

Finally, $\tilde{\boldsymbol{\eta}}$ is used to build the tanh-like function after denormalization. There is a picture (Figure 3.4) which shows the flowchart for the processing of the learnable parameters for an approximator of the nonlinear subcircuit in the pNN.

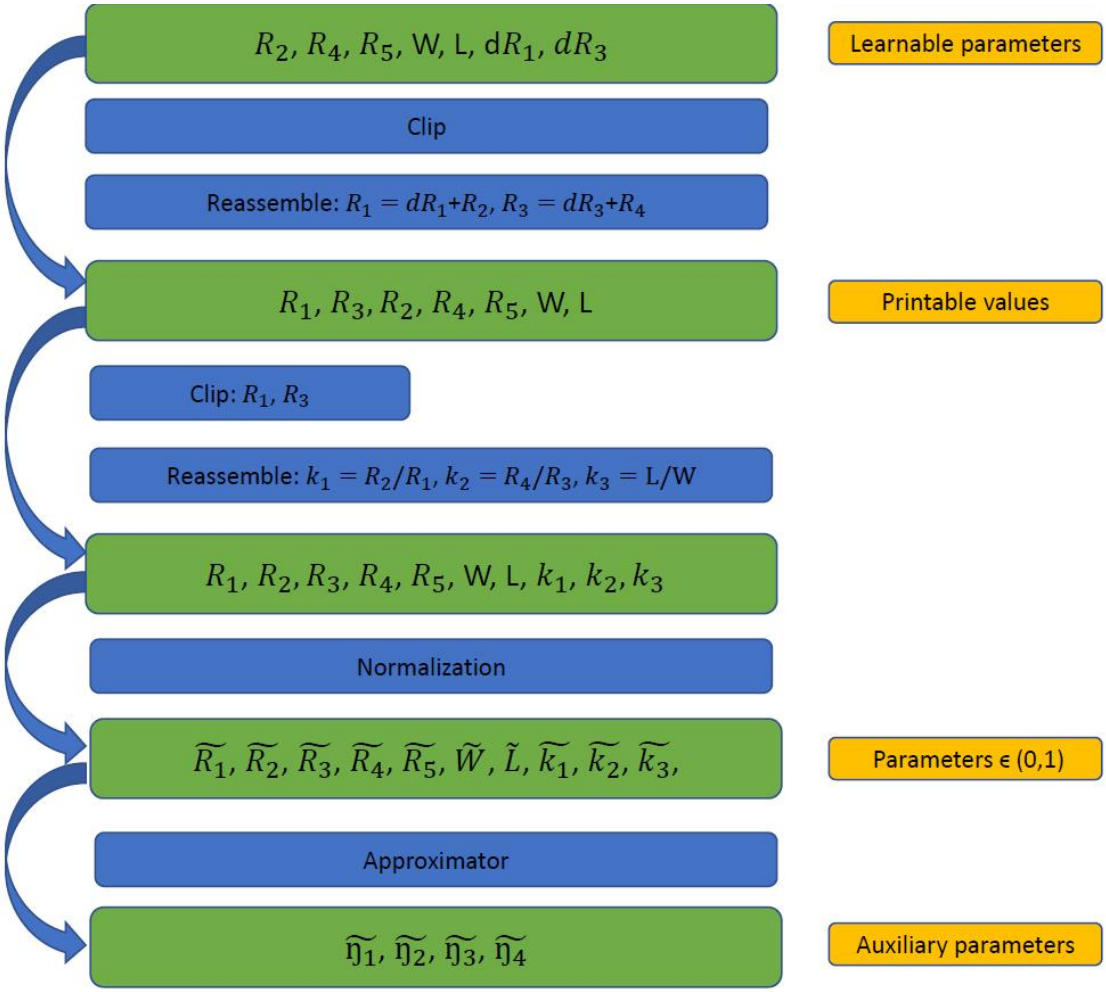


Figure 3.4: Flowchart for the processing of the learnable parameters for an approximator of the nonlinear subcircuit in the pNN.

3.3 Variation-aware training with nonlinear subcircuits

Variation-aware training refers to taking the fabrication error of printed components into account during the training of pNNs. In Section 2.2.4 only the surrogate conductance $\boldsymbol{\theta}$ of crossbars are considered. However, in this work the components of nonlinear circuits are also considered. In nonvariation-aware training, the loss function L depends on the training data $(\mathbf{x}, \mathbf{y})$ and the learnable parameters, i.e., the surrogate conductance $\boldsymbol{\theta}$ and the $\boldsymbol{\rho}$ for nonlinear circuits. Therefore, the loss function can be denoted as $L(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\rho})$. In the following text, I notate nonvariation-aware training as *nominal* training.

In variation-aware training of this work, the parameters $\boldsymbol{\theta}$ and the $\boldsymbol{\rho}$ are both modelled as stochastic variables $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$ and $\boldsymbol{\rho} \sim p(\boldsymbol{\rho})$. For simplicity, I model them as $\boldsymbol{\varepsilon}_\theta \boldsymbol{\theta}$ and $\boldsymbol{\varepsilon}_\rho \boldsymbol{\rho}$ respectively, where each element in $\boldsymbol{\varepsilon}_\theta$ and $\boldsymbol{\varepsilon}_\rho$ follows the same uniform distribution $\boldsymbol{U}\{1 - \varepsilon, 1 + \varepsilon\}$. Here, ε can be chosen to reflect the precision of printing technology. In this situation, the expected value of loss function in the nominal training is minimized, i.e.,

$$L = \mathbb{E}\{L(\mathbf{x}, \mathbf{y}, \boldsymbol{\varepsilon}_\theta \boldsymbol{\theta}, \boldsymbol{\varepsilon}_\rho \boldsymbol{\rho})\} = \int L(\mathbf{x}, \mathbf{y}, \boldsymbol{\varepsilon}_\theta \boldsymbol{\theta}, \boldsymbol{\varepsilon}_\rho \boldsymbol{\rho}) p(\boldsymbol{\varepsilon}_\theta) p(\boldsymbol{\varepsilon}_\rho) \mathrm{d}\boldsymbol{\varepsilon}_\theta \mathrm{d}\boldsymbol{\varepsilon}_\rho$$

However, this integration has no analytical solution, and approximations via, e.g., Monte Carlo estimates through samples have to be used, i.e.,

$$L \approx \frac{1}{N} \sum_{n=1}^{N} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\varepsilon}_\theta^{(n)} \boldsymbol{\theta}, \boldsymbol{\varepsilon}_\rho^{(n)} \boldsymbol{\rho})$$

where $\boldsymbol{\varepsilon}_\theta^{(n)}$ and $\boldsymbol{\varepsilon}_\rho^{(n)}$ are sampled values from their distribution $p(\boldsymbol{\varepsilon}_\theta)$ and $p(\boldsymbol{\varepsilon}_\rho)$. In this regard, the objective of variation-aware training becomes

$$\min_{\boldsymbol{\theta}, \boldsymbol{\rho}} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\rho})$$

Multiple gradient-based algorithms can be employed to solve this optimization problem, such as [32].

4. Design & Implementation

In this chapter, we will introduce the design and implementation from the following parts.

Section 4.1 introduces how to obtain a high-quality NN-based approximator for the nonlinear subcircuits.

Section 4.2 introduces different training frameworks and learning strategies, and explore which framework and strategies are suitable for the training of pNN in this work.

Subsequently, We integrate the obtained approximator into pNN, and apply better training framework and learning strategy in the next parts.

Section 4.3 introduces the design of finding suitable learning rate for the NN-based approximator during the training of pNNs.

Section 4.4 introduces the design of pNNs with NN-based approximator to investigate the effectiveness of NN-based approximator for pNNs.

Section 4.5 introduces the variation-aware training of pNNs with NN-based approximator. Variation-aware means, that the fabrication error of the component in crossbars, ptanh and Inv are taken into account in the training of pNN.

4.1 NN-based approximator

To obtain a high-quality NN-based approximator, we employ data preprocessing and hyperparameter tuning for the NN. The hyperpa-

rameters include learning rate and the number of layers of NN. Empirically, data normalization can significantly improve the performance of a NN, the range of different inputs in Table 3.1 varies greatly up to 6 orders of magnitude difference. Additionally, by scrutinizing the circuit schematic in Figure 2.3, we can find that R_1 and R_2 function as a voltage divider, therefore, the ratio between R_1 and R_2 is an important feature. This is also same for R_3 and R_4. In fact, the ratio between W and L is also crucial [48]. If each parameter is normalized independently, the ratio information will be weakened. Thus, we extend the physical quantities manually with three additional features, i.e.,

$$\boldsymbol{p}' = [R_1, R_2, R_3, R_4, R_5, W, L, k_1, k_2, k_3]$$

where $k_1 = R_2/R_1, k_2 = R_4/R_3, k_3 = L/W$. We normalize the data $\boldsymbol{p}'$ to $\tilde{\boldsymbol{p}}'$ as the input of the NN. Similarly, the target output of the NN is the normalized $\boldsymbol{\eta}$, which is denoted by $\tilde{\boldsymbol{\eta}}$. The minimal and maximal values, namely $\boldsymbol{\eta}_{min}, \boldsymbol{\eta}_{max}, \boldsymbol{p}_{min}, \boldsymbol{p}_{max}$, in the normalization process will be saved for denormalization in the later work.

We randomly split the database into three sets, namely training set(70%), validation set(20%), and test set(10%), and train the NNs with the training set and stop the training with the validation set using early-stopping [45]. Afterwards, we use test set to evaluate all trained NNs and find the best hyperparameter. We use the mean-square-error(MSE) as the evaluation metric. Since numerical optimization problems are frequently affected by the initial points, we utilize different random seeds($s = \{0, ..., 9\}$) to vary the location of the initial solutions. The hyperparameters are the numbers of layers of NN (the range in $[2, 15]$) and the learning rate $lr \in \{0, 0.1, 0.01, 0.001\}$. The results is in Section 5.1.

4.2 Learning flexibility

In this part, we explore learning flexibility, namely different training frameworks and learning strategies, to improve the performance of pNN, which are not considered in previous works. There are three training frameworks listed, namely neuron-level, layer-level, net-layer. Neuron-level means that each neuron has independent NN-based approximator for ptanh and Inv. Layer-level means that the same NN-based approximator for ptanh and Inv are shared by all neurons in a layer. Net-level means that the same NN-based approximator for

ptanh and Inv are shared across all neurons. Figure 4.1 visualizes the different training frameworks. Learning strategies refers to parameters of crossbar, ptanh and Inv are updated simultaneously or alternatively in the training of pNNs. We conduct the experiments to decide which training framework and learning strategy are applied in the final experiments.

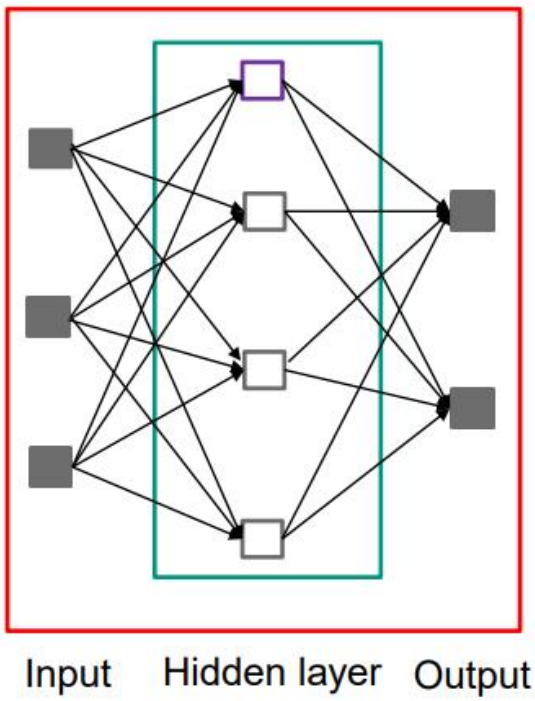


Figure 4.1: Visualization of different training frameworks. Red, green and purple colors denote net-level, layer-level, and neuron-level.

Uniformly with [60], we utilize 13 benchmark classification datasets (see Table 5.3). regarding the topology, we decide $\#input - 3 - \#output$ for pNNs, which is also same as [60]. As preprocessing, We normalize the input data in $[0, 1]$ to simulate the electrical signal measured by sensors. The datasets are then randomly split into training(60%), validation(20%), and test(20%). Additionally, we also use different learning rate, i.e., $lr \in \{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$. Since numerical optimization problems are frequently affected by the initial points, we utilize different random seeds($s = \{0, ..., 9\}$) to vary the location of the initial solutions. There are totally 4680 setups($13 \times 3 \times 2 \times 6 \times 10$, 13 for datasets, 3 for training frameworks, 2 for learning strategies, 6 for learning rate, 10 for seeds). The results is in Section 5.2.

4.3 Hyperparameter tuning for nonlinear subcircuits

Before we conduct the final experiments, we need to find suitable learning rate for learnable nonlinear circuit in the training of pNNs. We integrate the obtained approximator into pNN, and apply better training framework and learning strategy.

Uniformly with [60], we utilize 13 benchmark classification datasets. regarding the topology, we decide $\#input - 3 - \#output$ for pNNs, which is also same as [60]. As preprocessing, we normalize the input data in $[0, 1]$ to simulate the electrical signal measured by sensors. The datasets are then randomly split into training(60%), validation(20%), and test(20%). For $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$, i.e., weights and activation functions, we choose $\alpha_{\boldsymbol{\theta}} = 0.1$ for $\boldsymbol{\theta}$ and $\alpha_{\boldsymbol{\rho}} \in \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05\}$ for $\boldsymbol{\rho}$. we use early-stopping strategy with a patience equaling to 5000 epochs as the stop criterion for the training. Since numerical optimization problems are frequently affected by the initial points, we utilize different random seeds($s = \{0, ..., 9\}$) to vary the location of the initial solutions. There are totally 780 setups($13 \times 6 \times 10$, 13 for datasets, 6 for learning rate, 10 for seeds). The results is in Section 5.3.

4.4 Learnable nonlinear subcircuits

Learnable nonlinear circuit means, that parameters of ptanh and Inv are updated in the training of pNN, i.e., the nonlinear subcircuits can be optimized. In this part we design the final experiments to investigate the contribution of (learnable) nonlinear subcircuits in pNN.

Uniformly with [60], we utilize 13 benchmark classification datasets. regarding the topology, we decide $\#input - 3 - \#output$ for pNNs, which is also same as [60]. As data preprocessing, we normalize the input data in $[0, 1]$ to simulate the electrical signal measured by sensors. The datasets are then randomly split into training (60%), validation (20%), and test (20%) sets.

Since $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$ represent different underlying content, i.e., weights and activation functions. we choose different learning rates for them, namely $\alpha_{\boldsymbol{\theta}} = 0.1$ for $\boldsymbol{\theta}$ and $\alpha_{\boldsymbol{\rho}}$ for $\boldsymbol{\rho}$ ($\alpha_{\boldsymbol{\rho}} = 0$ refers to nonlearnable nonlinear circuit). we use early-stopping strategy with a patience equaling to 5000 epochs as the stop criterion for the training. Since numerical optimization problems are frequently affected by the initial points, we utilize different random seeds ($s = \{0, ..., 9\}$) to vary the location of the initial solutions. The results is in Section 5.4.

4.5 Variation-aware training

Section 3.3 introduces variation-aware Training for pNNs, which take only the resistances of crossbars as learnable parameters into account. In this work, the parameters of ptanh and Inv are also considered as learnable parameters. We apply the same variation model with uniform distribution for the resistance and geometries of transistor of ptanh and Inv. In this part, we design the final experiments to investigate the effectiveness of variation-aware training with learnable nonlinear subcircuits.

Uniformly with [60], we utilize 13 benchmark classification datasets. regarding the topology, we decide $\#input - 3 - \#output$ for pNNs, which is also same as [60]. As data preprocessing, we normalize the input data in $[0, 1]$ to simulate the electrical signal measured by sensors. The datasets are then randomly split into training (60%), validation (20%), and test (20%).

Since $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$ represent different underlying content, i.e., weights and activation functions. we choose different learning rates for them, namely $\alpha_{\boldsymbol{\theta}} = 0.1$ for $\boldsymbol{\theta}$ and $\alpha_{\boldsymbol{\rho}}$ for $\boldsymbol{\rho}$. we use early-stopping strategy with a patience equaling to 5000 epochs as the stop criterion for the training. Regarding the variation, we select $\varepsilon \in 0\%, 5\%, 10\%$ (0% refers to nominal training). For Monte-Carlo approximation, we select $N_{train} = 10$. Since numerical optimization problems are frequently affected by the initial points, we utilize different random seeds ($s = \{0, ..., 9\}$) to vary the location of the initial solutions. The results is in Section 5.5.

5. Evaluation

For artificial neural networks, training relates to finding a network to express a certain desired behaviour. The desired behaviour is usually characterized by a dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=0}^{N}$ of pairs of input-output relationships. After training, i.e., finding suitable parameters, the network should be able to identify the correct y_n for a given input $\mathbf{x}_n$. This can be achieved by minimizing the loss function with respect to the parameters of the network.

However, in contrast to artificial neural networks, the parameters of the pNN have to respect the technology constraints. Hence, these constraints have to be considered in the training procedure. For example, the fabrication of the respective NC is only possible if the conductance g_i of the pNN satisfy $g_i \in [g_{min}, g_{max}] \cup \{0\}$ (see Section 2.2.2), and the initialization of parameters should also be considered (see subsubsection 2.1.6.1). This chapter presents the results of experiments which are described in Chapter 4.

Section 5.1 introduces the results of experiments (Section 4.1) for NN-based approximator. The experiments' aim are to find high-quality NN-based approximator for the final experiments.

Section 5.2 introduces the results of experiments (Section 4.2) for training frameworks and learning strategies. The experiments' aim are to find suitable training framework and learning strategy for the final experiments.

Section 5.3 introduces the results of experiments (Section 4.3) for learning rate of learnable nonlinear subcircuits in the training of pNNs.

The experiments' aim are to find a suitable learning rate for learnable nonlinear subcircuits.

Section 5.4 introduces the results of experiments (Section 4.4) without variation. The experiment's aim was to study the performance of the pNN with learnable nonlinear subcircuits.

Section 5.5 introduces the results of experiments (Section 4.5) with variation. The experiments aimed to study the performance of variation-aware training with learnable nonlinear subcircuits.

5.1 Experiment for NN-based approximator

As stated by Section 4.1, we compare the valid loss of each setup, i.e., each setup has one seed, own numbers of layers and own learning rate. It is to be noted that we don't choose the average loss over seed. Because with this average loss we can't find an exact NN as an NN-based approximator. Figure 5.1 shows the validation loss of NNs with increased layers.

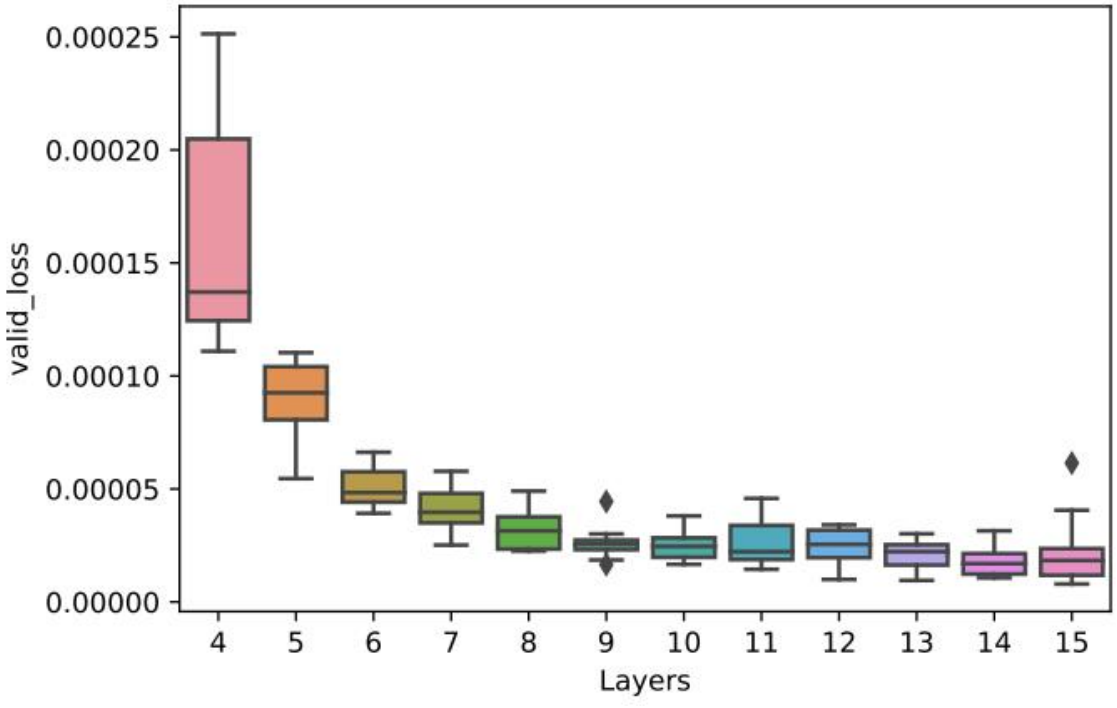


Figure 5.1: Visualization of the results from the validation loss of NNs with increased layers (learning rate $lr = 0.01$). The x-axis and the y-axis refer to the layers of NN and validation loss (ten seeds).

Through the comparison, the best topology of NN-based approximator is 15 layers NN(# neurons: 10-9-9-8-8-7-7-6-6-6-5-5-5-4) with the

learning rate 0.01. Figure 5.2 visualizes the results of three sets from this NN-based approximator. As the picture shows, there is no overfitting in the training according to the loss. Hence, this approximator provides acceptable prediction, and we apply this NN-based approximator in the following experiments.

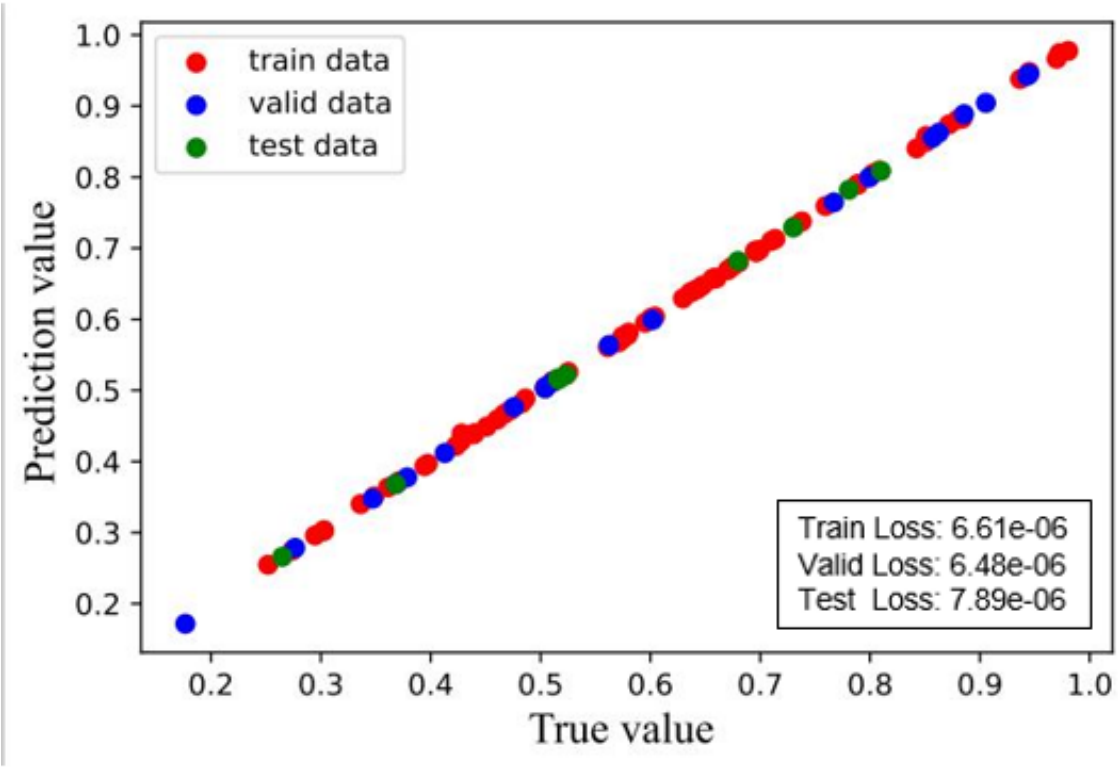


Figure 5.2: Visualization of the results of 100 examples from the NN-based approximator. The x-axis and the y-axis refer to the true value $\boldsymbol{\eta}$ and $\tilde{\boldsymbol{\eta}}$. Red, blue and green colors denote the data from training, validation and test sets.

5.2 Experiment for learning flexibility

According to Section 4.2, we compare the average valid accuracy over related setups to evaluate the effect of training frameworks and learning strategies for the training of pNN.

As a result, for learning strategies the mean accuracy of training simultaneously over all related setups (77.09%) is better than the result of training alternatively(75.09%). Regarding the training frameworks, Table 5.1 shows the average valid accuracy under different training frameworks and learning rates.

It is to be noted that, despite the higher number of learnable parameters, the pNNs with neuron-level and layer-level do not perform significantly better (sometimes even worse) than the pNNs with net-level,

Accuracy / Learning rate / Level	0.0005	0.001	0.005	0.01	0.05	0.1
Neuron-level	0.8625	0.8679	0.8468	0.8042	0.5926	0.5314
Layer-level	0.8694	0.8647	0.8589	0.8210	0.6014	0.5059
Net-level	0.8708	0.8656	0.8572	0.8362	0.6977	0.5424

Table 5.1: The average valid accuracy under different training frameworks and learning rates.

and the training of pNN is time-saving. Hence, we decide to apply the training frameworks of net-level, and the parameters of crossbar, ptanh and Inv in a pNN are updated simultaneously in the next experiments.

5.3 Hyperparameter tuning for nonlinear subcircuits

In accordance with Section 4.3, we evaluate different setups with the valid set, since in the final experiments we will select the best pNNs for each dataset (w.r.t. the loss) on validation sets. The mean accuracy and standard deviation from valid set are calculated and reported under different datasets and learning rate (see Table 5.2). To demonstrate the results from different experiment setups more intuitively, we average the accuracies over all datasets to a scalar for each learning rate. The average values are also reported in this table.

Learning rate / Datasets	0.0001	0.0005	0.001	0.005	0.01	0.05
Acute Inflammation	1.0 ± 0.0	1.0 ± 0.0	0.922 ± 0.165	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
Balance Scale	0.944 ± 0.034	0.965 ± 0.007	0.96 ± 0.0	0.976 ± 0.012	0.969 ± 0.011	0.967 ± 0.008
Breast Cancer Wisconsin	0.97 ± 0.005	0.965 ± 0.009	0.96 ± 0.005	0.97 ± 0.006	0.955 ± 0.009	0.966 ± 0.006
Cardiotocography	0.885 ± 0.05	0.854 ± 0.025	0.853 ± 0.041	0.76 ± 0.033	0.796 ± 0.047	0.812 ± 0.053
Energy Efficiency ($y1$)	0.871 ± 0.019	0.825 ± 0.026	0.806 ± 0.126	0.819 ± 0.037	0.841 ± 0.039	0.809 ± 0.024
Energy Efficiency ($y2$)	0.924 ± 0.025	0.866 ± 0.123	0.842 ± 0.141	0.803 ± 0.171	0.866 ± 0.059	0.844 ± 0.112
Iris	0.986 ± 0.044	0.986 ± 0.029	0.955 ± 0.028	0.976 ± 0.028	0.91 ± 0.065	0.941 ± 0.028
Mammographic Mass	0.833 ± 0.019	0.791 ± 0.079	0.666 ± 0.13	0.827 ± 0.008	0.818 ± 0.013	0.825 ± 0.009
Pendigits	0.474 ± 0.1	0.478 ± 0.043	0.424 ± 0.049	0.209 ± 0.13	0.366 ± 0.077	0.37 ± 0.083
Seeds	0.956 ± 0.015	0.92 ± 0.028	0.915 ± 0.029	0.932 ± 0.022	0.895 ± 0.043	0.898 ± 0.121
Tic-Tac-Toe Endgame	0.909 ± 0.058	0.816 ± 0.02	0.795 ± 0.01	0.819 ± 0.057	0.757 ± 0.041	0.769 ± 0.036
Vertebral Column (2 cl.)	0.867 ± 0.033	0.823 ± 0.034	0.816 ± 0.034	0.726 ± 0.093	0.718 ± 0.046	0.854 ± 0.071
Vertebral Column (3 cl.)	0.859 ± 0.016	0.831 ± 0.013	0.843 ± 0.011	0.744 ± 0.239	0.841 ± 0.008	0.828 ± 0.061
Average	0.883 ± 0.027	0.855 ± 0.034	0.827 ± 0.059	0.812 ± 0.074	0.826 ± 0.025	0.837 ± 0.04

Table 5.2: The mean and standard deviation from valid sets under different datasets and learning rates.

As the table shows, the mean accuracy of $lr = 0.0001$ is better than others. Hence, we decide to take $lr = 0.0001$ as learning rate for learnable nonlinear circuit, which will be used in the next parts.

5.4 Experiment with learnable nonlinear subcircuits

In this part, we investigate the contribution of (learnable) nonlinear subcircuits in pNN. According to Section 4.4, we select the best pNNs over ten seeds (w.r.t the loss) on validation sets, since they will be the ones to printed. Then we evaluate them with the test datasets.

The accuracy from test sets are calculated and reported under different datasets (see Table 5.3). As the baseline, we report the performance of nonlearnable nonlinear circuit. Hence, if the pNN gets worse than this baseline, there is no benefit to considering the learnable nonlinear circuit anymore. To demonstrate the results from different experiment setups more intuitively, we average the accuracies over all datasets to a scalar for each experiment setup. The average values are also reported in this table.

Dataset	**Nominal training**	
	Nonlearnable nonlinear circuit	**Learnable nonlinear circuit**
Acute Inflammation	1.000	1.000
Balance Scale	0.889	0.937
Breast Cancer Wisconsin	0.95	0.971
Cardiotocography	0.894	0.899
Energy Efficiency ($y1$)	0.948	0.950
Energy Efficiency ($y2$)	0.896	0.922
Iris	0.968	0.968
Mammographic Mass	0.813	0.815
Pendigits	0.583	0.620
Seeds	0.884	0.907
Tic-Tac-Toe Endgame	0.740	1.000
Vertebral Column (2 cl.)	0.810	0.825
Vertebral Column (3 cl.)	0.796	0.841
Average	**0.860**	**0.896**

Table 5.3: Result of the experiment with nonlinear circuits under nominal training on 13 benchmark datasets

The table shows, that learnable nonlinear subcircuits provide a contribution to the final improvement in accuracy, and the (mean) accuracy has been increased by 4.20% compare to nonlearnable nonlinear subcircuits.

5.5 Experiment with variation-aware training

In this part, we investigate the effectiveness of variation-aware training with learnable nonlinear subcircuits. According to Section 4.5, we select the best pNNs over ten seeds w.r.t the loss on validation sets,

since they will be the ones to printed. Then we evaluate them with the test datasets. For Monte-Carlo approximation, we select $N_{test} = 100$.

For pNNs from nominal training, we test them with 5% and 10% variations, respectively. As for pNNs from variation-aware training, we only test the ε that is identical as they were trained. All pNNs are tested with $N_{test} = 100$ Monte-Carlo samples. The mean and standard deviation from $N_{test} = 100$ samples are calculated and reported in the tables. To demonstrate the results from different experiment setups more intuitively, we average the accuracies over all datasets to a scalar for each experiment setup. The average values are also reported in the tables.

The mean accuracy and standard deviation from test sets are calculated and reported under different datasets and 5% variation (see Table 5.4).

Dataset	**Learnable nonlinear circuit**	
	Nominal training	**Variation-aware training**
Acute Inflammation	1.0 ± 0.0	1.0 ± 0.0
Balance Scale	0.911 ± 0.02	0.889 ± 0.004
Breast Cancer Wisconsin	0.933 ± 0.067	0.957 ± 0.013
Cardiotocography	0.686 ± 0.256	0.871 ± 0.005
Energy Efficiency ($y1$)	0.789 ± 0.175	0.939 ± 0.005
Energy Efficiency ($y2$)	0.82 ± 0.108	0.902 ± 0.007
Iris	0.87 ± 0.105	0.93 ± 0.046
Mammographic Mass	0.686 ± 0.118	0.787 ± 0.013
Pendigits	0.35 ± 0.113	0.425 ± 0.028
Seeds	0.888 ± 0.051	0.917 ± 0.029
Tic-Tac-Toe Endgame	0.666 ± 0.208	0.741 ± 0.017
Vertebral Column (2 cl.)	0.708 ± 0.086	0.832 ± 0.026
Vertebral Column (3 cl.)	0.657 ± 0.15	0.835 ± 0.026
Average	$\mathbf{0.766 \pm 0.112}$	$\mathbf{0.848 \pm 0.017}$

Table 5.4: Result of the experiment with variation 5% on 13 benchmark datasets

The mean accuracy and standard deviation from test sets are calculated and reported under different datasets and 10% variation (see Table 5.5).

The tables show, that for most of the 13 benchmark, variation-aware training provide a significant contribution to the final improvement in accuracy and robustness (standard variation) with learnable nonlinear subcircuits. Regarding the (mean) accuracy and (mean) robustness,

Dataset	**Learnable nonlinear circuit**	
	Nominal training	**Variation-aware training**
Acute Inflammation	0.989 ± 0.041	1.0 ± 0.0
Balance Scale	0.879 ± 0.072	0.887 ± 0.009
Breast Cancer Wisconsin	0.788 ± 0.232	0.93 ± 0.067
Cardiotocography	0.536 ± 0.326	0.846 ± 0.019
Energy Efficiency ($y1$)	0.648 ± 0.221	0.915 ± 0.035
Energy Efficiency ($y2$)	0.708 ± 0.199	0.886 ± 0.016
Iris	0.751 ± 0.157	0.819 ± 0.124
Mammographic Mass	0.632 ± 0.12	0.745 ± 0.074
Pendigits	0.241 ± 0.119	0.272 ± 0.012
Seeds	0.787 ± 0.159	0.858 ± 0.037
Tic-Tac-Toe Endgame	0.559 ± 0.182	0.689 ± 0.032
Vertebral Column (2 cl.)	0.689 ± 0.086	0.743 ± 0.043
Vertebral Column (3 cl.)	0.483 ± 0.201	0.758 ± 0.053
Average	$\mathbf{0.668 \pm 0.163}$	$\mathbf{0.796 \pm 0.04}$

Table 5.5: Result of the experiment with variation 10% on 13 benchmark datasets

there are an improvement by 10.7% and 84.8% under 5% variation, 19.2% and 75.5% under 10% variation.

6. Conclusion & Future Work

This final chapter consists of the conclusion as well as an outlook and suggestions to future projects in this field.

6.1 Conclusion

In this thesis, we concentrate on the nonlinear subcircuits of printed neuromophic circuits. By sampling feasible design parameters, simulating and fitting the characteristic curves, and approximating the transformation from physical quantities to the characteristic curves, NN-based models of the nonlinear circuits are generated. After we integrate this model into the pNNs, the transistor sizes and conductances could then be considered as learnable parameters similar to the conductances of the crossbar resistors, which can promise the on-demand fabrication of ptanh and Inv individually, and subsequently enhances the designability of the printed neuromophic circuits greatly. Additionally, considering the manufacturing variation of the components in both crossbars and nonlinear subcircuits, we employ the variation-aware training for pNNs, which aims to minimize the expected loss for pNNs with given variation model. Additionally, we apply different training frameworks and strategies to explore the performance of pNNs.

The preliminary experiment proved that, by introducing these approaches, the accuracy of the pNNs with learnable nonlinear subcircuits improves 4.2%, and the accuracy and robustness of the pNNs with variation-aware training and learnable nonlinear subcircuits are significantly improved (around 10% - 20% in accuracy and 75% - 85%

in robustness, depending on the variations). Hence, we consider this work as an important supplement towards the realization of printed neuromorphic computing systems.

6.2 Outlook

The experience we have gathered, over the course of writing this master thesis, has given us the following ideas that we would like to share for future work:

(1) we can explore better training frameworks and strategies(see Section 4.2) under variation-aware training. In this work, because of limited time we use exhaustive method to find better training frameworks and strategies under variation-unaware training.

(2) There are also further sources of variations, such as input variations, which could be considered. As these variations cannot be influenced by the learnable parameters, they simply represent training with noisy training data and do not fundamentally complicate the procedure.

(3) Aside from variations, other aspects that degrade the performance of printed NCs may be anticipated in training. One of these could be accuracy degradation due to aging of the components. Aging models of the parameters [60] can be combined with learnable nonlinear subcircuits to explore the performance of printed NCs.

(4) We can do ablation study to study the influence of the learnable nonlinear circuit and variation-aware training, i.e., solely change $\alpha_{\boldsymbol{\rho}}$ to 0 and observe the change in results; and (b) study the effect of variation-aware training, i.e., both nominal training and variation-aware training on pNNs with learnable nonlinear circuit.

Bibliography

[1] https://spectrum.ieee.org/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated. Last accessed: 2016-08-15.

[2] https://en.wikipedia.org/wiki/Artificial_neural_network.

[3] https://en.wikipedia.org/wiki/Low-discrepancy_sequence.

[4] https://www.cadence.com.

[5] https://en.wikipedia.org/wiki/Euclidean_distance.

[6] Mohammad Ansari et al. "PHAX: Physical Characteristics Aware Ex-Situ Training Framework for Inverter-Based Memristive Neuromorphic Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.8 (2018), pp. 1602–1613.

[7] Manuel Barros, Jorge Guilherme, and Nuno Horta. "Analog circuits optimization based on evolutionary computation techniques". In: (2010).

[8] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.* 2013.

[9] Nathaniel Bleier et al. "Printed Microprocessors". In: (2020), pp. 213–226.

[10] Sumon Kumar Bose, Jyotibdha Acharya, and Arindam Basu. *Is my Neural Network Neuromorphic? Taxonomy, Recent Trends and Future Directions in Neuromorphic Engineering.* 2020.

[11] Joseph S. Chang, Antonio F. Facchetti, and Robert Reuss. "A Circuits and Systems Perspective of Organic/Printed Electronics: Review, Challenges, and Contemporary and Emerging Design Approaches". In: (2017), pp. 7–26.

[12] Hakima Chaouchi. "Introduction to the Internet of Things". In: 2013.

[13] Ching-Tai Chin et al. "Training techniques to obtain fault-tolerant neural networks". In: *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing.* 1994, pp. 360–369.

[14] Koby Crammer and Yoram Singer. "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines". In: *J. Mach. Learn. Res.* 2 (2001), pp. 265–292.

[15] W. Daems, G. Gielen, and W. Sansen. "An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits". In: 2002.

[16] Jiacnao Deng et al. "Retraining-based timing error mitigation for hardware neural networks". In: 2015, pp. 593–596.

[17] Zidong Du et al. “Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches”. In: *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, pp. 494–507.

[18] Xavier Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Journal of Machine Learning Research - Proceedings Track* 9 (Jan. 2010), pp. 249–256.

[19] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *AISTATS*. 2010.

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

[21] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[22] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[23] Michael Hefenbrock. “Modelling and Training Printed Neuromorphic Circuits”. PhD thesis. Karlsruher Institut für Technologie (KIT), 2022.

[24] M.delM. Hershenson, S.P. Boyd, and T.H. Lee. “Optimal design of a CMOS op-amp via geometric programming”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20.1 (2001), pp. 1–21.

[25] Kevin Ho, Chi-sing Leung, and John Sum. “On Weight-Noise-Injection Training”. In: 2009, pp. 919–926.

[26] Junyan Hu et al. “Fault-tolerant cooperative navigation of networked UAV swarms for forest fire monitoring”. In: (2022), p. 107494.

[27] Miao Hu et al. “Hardware realization of BSB recall function using memristor crossbar arrays”. In: *DAC Design Automation Conference 2012*. 2012, pp. 498–503.

[28] Nuttorn Jangkrajarng et al. “IPRAIL—intellectual property reuse-based analog IC layout automation”. In: (2003).

[29] Kevin Jarrett et al. “What is the best multi-stage architecture for object recognition?” In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 2146–2153.

[30] Xiaojie Jin et al. *Deep Learning with S-shaped Rectified Linear Activation Units*. 2015.

[31] Saleem Khan, Leandro Lorenzelli, and Ravinder S. Dahiya. “Technologies for Printing Sensors and Electronics Over Large Flexible Substrates: A Review”. In: *IEEE Sensors Journal* 15 (2015), pp. 3164–3185.

[32] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014.

[33] John Koza et al. “Automated Design Of Both The Topology And Sizing Of Analog Electrical Circuits Using Genetic Programming”. In: (1998).

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. 2012, 1097–1105.

[35] Isidoro Ibanez Labiano and Akram Alomainy. "Flexible inkjet-printed graphene antenna on Kapton". In: 6.2 (2021), p. 025010.

[36] Yeongjun Lee et al. "Stretchable organic optoelectronic sensorimotor synapse". In: (2018).

[37] Chi-Sing Leung, Wai Yan Wan, and Ruibin Feng. "A Regularizer Approach for RBF Networks Under the Concurrent Weight Failure Situation". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.6 (2017), pp. 1360–1372.

[38] Wenlong Lyu et al. "An Efficient Bayesian Optimization Approach for Automated Optimization of Analog Circuits". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.6 (2018), pp. 1954–1967.

[39] Gabriel Cadilha Marques et al. "Electrolyte-Gated FETs Based on Oxide Semiconductors: Fabrication and Modeling". In: (2017), pp. 279–285.

[40] Gabriel Cadilha Marques et al. "From silicon to printed electronics: A coherent modeling and design flow approach based on printed electrolyte gated FETs". In: 2018, pp. 658–663.

[41] P.C. Maulik and L.R. Carley. "Automating analog circuit design using constrained optimization techniques". In: 1991, pp. 390–393.

[42] C. Mead. "Neuromorphic electronic systems". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636.

[43] Muhammad Husnain Mubarik et al. "Printed machine learning classifiers". In: 2020, pp. 73–87.

[44] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019.

[45] Lutz Prechelt. "Automatic early stopping using cross validation: quantifying the criteria". In: *Neural Networks* 11.4 (1998), pp. 761–767. ISSN: 0893-6080.

[46] Farhan Rasheed. "Compact Modeling and Physical Design Automation of Inkjet-Printed Electronics Technology". PhD thesis. Karlsruher Institut für Technologie (KIT), 2020.

[47] Farhan Rasheed et al. "A Smooth EKV-Based DC Model for Accurate Simulation of Printed Transistors and Their Process Variations". In: *IEEE Transactions on Electron Devices* 65.2 (2018), pp. 667–673.

[48] Farhan Rasheed et al. "Variability Modeling for Printed Inorganic Electrolyte-Gated Transistors and Circuits". In: *IEEE Transactions on Electron Devices* 66.1 (2019), pp. 146–152.

[49] Paulo Rosa, António Câmara, and Cristina Gouveia. "The Potential of Printed Electronics and Personal Fabrication in Driving the Internet of Things". In: *Open J. Internet Things* (2015), pp. 16–36.

[50] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[51] Keertana Settaluri et al. "AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs". In: 2020, pp. 490–495.

[52] D. Simon. "Distributed fault tolerance in optimal interpolative nets". In: *IEEE Transactions on Neural Networks* 12.6 (2001), pp. 1348–1357.

[53] John Wiley & Sons. "Printed electronics: Materials, technologies and applications". In: 2016.

[54] G. Stehr et al. "Initial sizing of analog integrated circuits by centering within topology-given implicit specifications". In: 2003, pp. 241–246.

[55] Mohammadamin Tavakoli, Forest Agostinelli, and Pierre Baldi. *SPLASH: Learnable Activation Functions for Improving Accuracy and Adversarial Robustness.* 2020.

[56] Hanrui Wang et al. *Learning to Design Circuits.* 2018.

[57] Dennis Weller. "Digital and Analog Computing Paradigms in Printed Electronics". PhD thesis. Karlsruher Institut für Technologie (KIT), 2021.

[58] Dennis Weller et al. "Realization and training of an inverter-based printed neuromorphic computing system". In: *Scientific Reports* (2021).

[59] Dennis D. Weller et al. "Programmable Neuromorphic Circuit based on Printed Electrolyte-Gated Transistors". In: *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC).* 2020, pp. 446–451.

[60] Haibin Zhao et al. "Aging-Aware Training for Printed Neuromorphic Circuits". In: *International Conference on Computer Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ICCAD 2022. 2022.