Modelling and Training Printed Neuromorphic Circuits

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc. Michael Hefenbrock

aus Offenburg

Tag der mündlichen Prüfung:	11.02.2022
Erster Gutachter:	Prof. Dr. Michael Beigl, KIT
Zweiter Gutachter:	Prof. Dr. Jasmin Aghassi-Hagmann, KIT
Dritter Gutachter:	Prof. Dr. Mehdi Baradaran Tahoori, KIT

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen - die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Offenburg, 05.05.2022
Michael Hefenbrock

Acknowledgment

This work would have not been possible without the help and support of many people. First of all, I would like to express my deepest gratitude towards my parents and my brother for being supportive of my endeavor to pursue graduation.

Furthermore, I want to thank my advisor Prof. Dr. Michael Beigl for providing feedback on this work and several topics leading up to it. In the same way, I would like to thank Dr. Till Riedel for several discussions which helped to broadening my view on several topics in computer science.

I want to thank my friends, colleagues and co-authors of the MERAGEM graduate school. In particular, I would like to thank Dennis D. Weller, Farhan Rasheed and Ahmet T. Erozan for our joint research and insightful discussion. Dennis D. Weller also performed the majority of the necessary laboratory and circuit simulation tasks that enabled this work.

I also want to acknowledge the coordinators of graduate school, Prof. Dr. Jasmin Aghassi-Hagmann and Prof. Dr. Mehdi B. Tahoori. Prof. Aghassi-Hagmann aided me with personal and scientific advice throughout the time of my PhD, and Prof. Tahoori tirelessly provided rich and invaluable feedback on our joint research papers. He also provided me with many opportunities for joint collaborative research with his group at the Chair of Dependable Nano Computing (CDNC) at the Karlsruhe Institute of Technology (KIT).

Finally, I would like to thank the Ministry of Science, Research and Art of the state of Baden-Württemberg which supported this work and most of my research in the form of the MERAGEM doctoral program.

ABSTRACT

Printed electronics is an emerging, complementary technology to classical silicon electronics. In contrast to silicon technology, which is tailored towards high integration and high performance computing with small feature sizes, printed electronics displays less performance and exhibits larger feature sizes. However, it also offers great advantages, such as lower cost per area through additive manufacturing, fabrication on flexible and soft substrates, as well as biodegradability. Moreover, digital printing techniques like inkjet printing enable on-demand and point-of-use fabrication. Through the availability of affordable printers, digitally printed electronics may, much like 3D printing, enable custom fabrication in the future. Finally, through the possibility to cheaply fabricate smart devices, printed electronics is increasingly recognized as a key driver of the Internet of Things (IoT).

Unfortunately, to leverage these benefits, several challenges need to be overcome. For example, the realisation of classical digital designs is often infeasible due to their extensive component demands and required integration densities. This characteristic may be especially limiting for small, customized smart devices intended for the processing of sensory data in IoT applications. Moreover, the additive manufacturing processes of printed electronics exhibit substantial variations, which make the reliable fabrication of functional circuits difficult.

To address these challenges, adapting concepts from neuromorphic computing to printed electronics could be a promising approach. Neuromorphic computing is a brain-inspired computing paradigm with neuron-like hardware primitives. As these components operate analog, they could process sensor data directly, while also offering lower footprints than digital circuitry. In addition, due to the inherent parallelism of their computations, they may compensate for the high latencies in printed electronics. This allows to perform lightweight computations directly on the printed devices, avoiding the need for energy intensive and possibly insecure transmission. Hence, using printed neuromorphic circuits may be an excellent solution for intelligent and customized near sensor processing in Internet of Things (IoT) applications. The design of neuromorphic circuits can be viewed as a machine learning task similar to training artificial neural networks. This additionally has several benefits over classical manual solutions. For example, it offers an on-demand design solution in line with the on-demand and customized fabrication capabilities of printed electronics. Moreover, variations in the fabrication process can be addressed in the learning process to increase the fabrication yield.

Motivated by this potential, the aim of this thesis is to explore the training of printed neuromorphic circuits. Additionally, a data-driven approach for modelling the variations of printed components is described. In detail, the following topics are addressed.

Printed neural networks

To employ and train printed Neuromorphic Circuits (NCs), models of the main hardware primitives need to be devised. Through these component models, a model for a printed NC can be expressed. This model is referred to as printed Neural Network (pNN). To train pNNs, gradient-based learning methods from classical artificial neural networks can be adapted. However, several aspects of the training procedure need to be modified to improve the learning process and ensure the feasibility of the final design. Namely, the loss function, the update rule and the initialization strategy for the learnable parameters. To verify the approximation capabilities of pNNs, they are trained and evaluated on several benchmark datasets.

A data-driven framework for variation modelling

Printed components and devices exhibit high variations in production. Variation analysis and variation modelling of printed devices and components are thus imperative for realizing functional printed circuits. However, modelling printed components is difficult as the underlying physical relationships are complicated and only partially understood. If the device behaviour cannot be fully described in a physical model, often empirical or semi-empirical approaches are employed. Unfortunately, the parameter distribution of such models may not display simple and classical distributions shapes such as normal distributions. To deal with such effects, a flexible, data-driven framework is proposed. The framework is then used to develop a variation model for a printed electrolyte-gated transistor.

Variation-aware training for printed neural networks

To combat the large variations of printed circuits and reduce their impact on printed NCs, variation models for the components are developed. Based on these variation models, a variation-aware training procedure for pNNs is proposed. Here, the training objective is modified to minimize the expected loss under the anticipated variations. When employed, it is able to improve the robustness of trained pNNs to component variations on benchmark datasets.

Post-fabrication tuning for printed neuromorphic circuits

Depending on the variation of the fabrication process, the yield of printed neuromorphic circuits may not be sufficient. This may in part be due to incorrectly printed resistors, resulting in faulty conductance values. In this case, the unique feature of additive manufacturing can be leveraged to partly restore the intended functionality. To this end, this work describes a post-fabrication procedure to systematically restore the operation of a printed NCs. The procedure is evaluated on benchmark datasets under component variations.

ZUSAMMENFASSUNG

Gedruckte Elektronik ist eine neuartige, komplementäre Technologie zur klassischen Siliziumelektronik. Im Gegensatz zur Siliziumtechnologie, die auf hohe Integration und hohe Rechenleistung bei kleinen Strukturgrößen zugeschnitten ist, bietet die gedruckte Elektronik weniger Performanz und weist größere Strukturen auf. Sie bietet jedoch auch große Vorteile, wie zum Beispiel niedrigere Kosten pro Fläche durch additive Fertigung, die Herstellung auf mechanisch flexiblen Substraten, sowie die biologische Abbaubarkeit der Komponenten. Außerdem ermöglichen digitale Drucktechniken, wie der Tintenstrahldruck, eine Herstellung nach Bedarf direkt am Einsatzort. Durch die Verfügbarkeit von kostengünstigen Druckern könnte digital gedruckte Elektronik, ähnlich wie der 3D-Druck, in Zukunft eine individuelle Fertigung von Schaltungen ermöglichen. Weiterhin wird die gedruckte Elektronik durch die Möglichkeit smarte Geräte kostengünstig herzustellen zunehmend als eine der wichtigsten Triebkräfte des Internets der Dinge (IoT) gesehen.

Um diese Potenziale zu realisieren, müssen jedoch einige Herausforderungen bewältigt werden. So ist beispielsweise die Verwendung klassischer, digitaler Designs aufgrund des hohen Komponentenbedarfs und der erforderlichen Integrationsdichte oft nicht realisierbar. Dies könnte insbesondere eine Einschränkung für kleine, individuelle, intelligente Geräte sein, die für die Verarbeitung von Sensordaten in IoT-Anwendungen vorgesehen sind. Weiterhin weist der additive Herstellungsprozess in der gedruckten Elektronik erhebliche Schwankungen auf, die eine verlässliche Fertigung funktionierender Komponenten erschweren.

Um mit diesen Herausforderungen umzugehen, bietet die Übernahme von Konzepten aus dem neuromorphen Rechnen einen vielversprechenden Ansatz. Neuromorphes Rechnen ist ein vom Gehirn inspiriertes Rechenparadigma mit neuronenähnlichen Hardwarebausteinen. Da diese Komponenten analog arbeiten, könnten sie Sensordaten direkt verarbeiten, wobei sie zusätzlich einen geringeren Platzbedarf aufweisen. Zusätzlich könnte durch die inhärent parallelen Berechnungen die hohen Latenzen der gedruckten Komponenten teilweise kompensiert werden. Dies ermöglicht leichtgewichtige Berechnungen direkt auf den gedruckten Geräten durchzuführen, ohne dass eine energieintensive und möglicherweise unsichere Übertragung notwendig wird. Dieser Ansatz bietet daher eine vielversprechende Lösung für eine intelligente, sensornahe Verarbeitung in IoT-Anwendungen. Der Entwurf neuromorpher Schaltungen kann als ein Problem des maschinellen Lernens, ähnlich dem Training künstlicher neuronaler Netze, gesehen werden. Dies hat weitere Vorteile gegenüber manuellem Design. Beispielsweise bietet es eine Entwurfslösung mit der es möglich ist ein Design bei Bedarf, passend zur Fertigung nach Bedarf in der gedruckten Elektronik, zu erzeugen. Zusätzlich können Variationen in der Herstellung bereits im Lernprozess berücksichtigt werden um die Ausbeute der Fertigung zu erhöhen.

Motiviert durch dieses Potenzial ist das Ziel dieser Arbeit die Erforschung des Trainings von gedruckten neuromorphen Schaltungen. Weiterhin wird ein datengetriebener Modellierungsansatz beschrieben, um Variationen gedruckter, elektronischer Komponenten zu modellieren. Im Einzelnen werden die folgenden Themen behandelt.

Gedruckte neuronale Netze

Um gedruckte neuromorphe Schaltungen einzusetzen und zu trainieren, werden Modelle ihrer Grundkomponenten benötigt. Mit Hilfe dieser Modelle kann daraufhin ein Modell der gedruckten neuromorphen Schaltung erzeugt werden. Dieses Modell wird als gedrucktes neuronales Netz bezeichnet. Um gedruckte neuronale Netze zu trainieren, können gradientenbasierte Lernverfahren, wie sie im Training klassischer neuronalen Netze verwendet werden, adaptiert werden. Zur Verbesserung des Lernprozesses und um zu garantieren, dass das finale Design valide ist, müssen mehrere Schritte des Trainingsprozesses angepasst werden. Hierzu gehören beispielsweise die Verlustfunktion, die Update-Regel und die Initialisierung der lernbaren Parameter. Zur Verifizierung der Approximationseigenschaften der gedruckten neuronalen Netze werden diese auf mehreren Benchmarkdatensätzen trainiert und evaluiert.

Ein datengetriebener Ansatz zur Entwicklung von Variationsmodellen

Gedruckte Bauelemente und Komponenten weisen in der Produktion hohe Schwankungen auf. Variationsanalyse und Variationsmodellierung von gedruckten Bauelementen und Komponenten sind daher für die Realisierung funktionaler gedruckter Schaltungen unerlässlich. Die Modellierung von gedruckten Komponenten ist jedoch schwierig, da die zugrunde liegenden physikalischen Zusammenhänge kompliziert, und bisher nur teilweise verstanden sind. Falls das Verhalten eines Bauelements nicht vollständig durch ein physikalisches Modell beschrieben werden kann, werden häufig empirische oder teilweise empirische Ansätze verwendet. Leider weisen die Parameterverteilungen solcher Modelle oft keine klassischen Verteilungsfunktionen wie zum Beispiel Normalverteilungen auf. Um mit solchen Effekten umzugehen, wird ein flexibler, datengetriebener Ansatz vorgeschlagen. Dieser wird daraufhin verwendet um ein Variationsmodell für einen gedruckten Transistor zu entwickeln.

Trainieren von gedruckten neuronalen Netzen unter Variation

Um mit den großen Variationen bei der Fertigung gedruckter Schaltungen umzugehen, und ihren Einfluss auf gedruckte neuromorphe Schaltungen zu mildern, werden Variationsmodelle ihrer Kompontenten entwickelt. Basierend auf diesen Variationsmodellen wird ein angepasster Trainingsansatz für gedruckte neuronale Netze entwickelt. Hierbei wird die Zielfunktion des Trainings dahingehend modifiziert den erwarteten Verlust unter der antizipierten Variation zu minimieren. Durch den Einsatz dieses Verfahrens ist es möglich, die Robustheit trainierter gedruckter neuronaler Netze gegen Komponentenvariationen auf Benchmarkdatensätzen zu erhöhen.

Ein Nachbearbeitungsverfahren für gedruckte neuromorphe Schaltungen

Je nach Schwankung des Herstellungsprozesses ist die Ausbeute von gedruckten neuromorphen Schaltungen gering. Dies kann zum Teil auf ungenau gedruckte Widerstände zurückzuführen sein. Um diese Produktionsfehler zu korrigieren kann eine einzigartige Eigenschaft der additiven Fertigung genutzt werden. Diese ermöglicht es, die ursprünglich beabsichtigte Funktionalität (teilweise) wiederherzustellen. Um dies zu erreichen, wird in dieser Arbeit ein Verfahren zur systematischen Wiederherstellung der Funktion einer gedruckten neuromorphen Schaltungen vorgestellt. Hierfür werden die vorab eingeführten gedruckten neuronalen Netze verwendet. Das Verfahren wird ebenfalls an einer Reihe von Benchmark-Datensätzen unter Komponentenvariationen evaluiert.

Remarks on notation and mathematical formulations

The notation used in this work is largely influenced by the book of Christopher Bishop [13]. Scalar values are usually denoted by lowercase, italic characters, e.g., λ, α, s. To distinguish vectors from scalars, the former are displayed as bold lowercase letters, e.g. $\mathbf{x}$, $\mathbf{w}$, while matrices are displayed in bold uppercase letters, e.g., $\mathbf{\Sigma}$, $\mathbf{W}$. The transpose of a vector (or matrix) $\mathbf{x}$ is denoted by $\mathbf{x}^\top$. Vectors are defined as column vectors, e.g., $\mathbf{x} = [x_1, x_2, \cdots]^\top$. If not specified otherwise, all scalars, vectors and matrices consist of real numbers from $\mathbb{R}$. Interval ranges of variables are expressed through $x \in [a, b)$, denoting $a \leq x < b$.

Sets of variables, e.g. $\{\theta_n \mid n = 1, \cdots, N\}$, are often collected in vectors, e.g. $\boldsymbol{\theta}$, for brevity of notation. The vector $\mathbf{1}$ and the matrix $\mathbf{I}$ respectively denote a vector of ones and the identity matrix. The dimensions should thereby be clear from the context. When operations such as sums are performed over the entries of a vector (or a set), the endpoint of the sum is often left unspecified, e.g., $\mathbf{w}^\top\mathbf{x} = \sum_i w_i x_i$ or $\sum_i \theta_i = \boldsymbol{\theta}^\top\mathbf{1}$. This is done for increased readability and to avoid the definition of infrequently used variables.

At several points in this work, parameterized functions are introduced, e.g., a function $\psi(\mathbf{x}, \boldsymbol{\theta})$, where the parameters $\boldsymbol{\theta}$ are found through optimization routines. To distinguish between the inputs $\mathbf{x}$ and the parameters $\boldsymbol{\theta}$, the latter are written as subscripts to the function, e.g., $\psi_{\boldsymbol{\theta}}(\mathbf{x})$. Additionally, when a one-dimensional function $\phi(x) : \mathbb{R}^1 \to \mathbb{R}^1$ is applied to a vector or matrix, the operation should be understood as an element-wise application of ϕ. Consequently, also order relations like $\leq$ are used on numbers and on vectors. In the latter case, they refer to an element-wise operation. In some equations, the indicator function $\mathbb{1}_{\{\cdot\}}$ is used, which will be defined as 1 if the respective condition is met and 0 otherwise. As with other functions defined for scalars, $\mathbb{1}_{\{\cdot\}}$ applies in an element-wise fashion to vectors and matrices. The rest of the expressions should be clear from the context.

Finally, many problems in this work are formulated as optimisation problems. Most of the time, this is done to signify an intention and reveal an approach to tackle the problem. Hence, finding a "good" solution for these problems will often suffice. To find these solutions, gradient-based techniques are frequently applied even in cases where the respective function is not differentiable at every point in the input domain. Such situations are mostly due to the use of hinge functions, i.e., $(\cdot)^+ = \max\{0, \cdot\}$ or indicator functions $\mathbb{1}_{\{\cdot\}}$. Although differentiability is commonly not assured, continuity for the relevant parameters is usually given. Nevertheless, the application of gradient-based methods may be seen as heuristically motivated.

Employed software

The implementation of all components was done using *python* with the help of several libraries. For fitting linear models and Gaussian mixtures, the *scikit-learn* library [118] was used. General optimization problems for parameter fitting of nonlinear models were solved using *scipy* [149] (optimize). Statistical calculations such as statistical tests were also performed using the *scipy* (stats) package. All data was handled using *pandas* [106] and *numpy* [61] and (printed) neural networks were implemented using *pytorch* [117]. The plots and illustrations were generated using *matplotlib* [73], and flowcharts were created using Microsoft Powerpoint 2016.

Previously published results

Several contents of this work have already been published as parts of collaborative work.

- Chapter 3 is mainly based on [156], where the printed neural network model and training algorithm were published. They are also briefly discussed in [154, Sec. 4.1] alongside the hardware primitives for printed neuromorphic computing. Compared to both sources, the chapter provides more in-depth explanations of the modelling and training steps with several additional illustrations.
- Chapter 4 has been published in [126] and is also discussed in [124, Sec. 3.4]. Compared to the publication, the chapter present a more in-depth explanation of the variation modelling and puts less emphasize on the technology of the printed electrolyte-gated transistor for which a model was developed.
- Chapter 5 is also based on [156]. Compared to the paper, it presents the development of the circuit component models in detail through the procedure described in Chapter 4. Additionally, the experiments are extended considerably and also involve an additional dataset, variation levels and graphs, alongside a more detailed analysis of the results.
- Chapter 6 is accepted for publication as [67] by the time of writing this thesis. Compared to the submitted work, the chapter provides a more detailed description of the method and extended experiments.

Table of Contents

List of Figures

List of Tables

Acronyms

AIC Akaike Information Criterion. 45

BIC Bayesian Information Criterion. 45, 49, 51

EGT Electrolyte-Gated Transistor. xxi, 22, 23, 46–48, 51, 57, 90

GMM Gaussian Mixture Model. 40, 41, 43–45, 49, 51, 57, 60

IoT Internet of Things. vii, 1, 5, 6, 16, 19, 22, 34, 38, 89

MaA Measuring-aware Accuracy. xviii, xxi, 32, 33, 63–74, 83–86

NC Neuromorphic Circuit. viii, 2, 3, 5, 8, 12, 14, 16, 17, 19–24, 26, 28, 30, 34–36, 53–57, 74, 75, 77, 78, 80, 81, 83, 87, 90, 91

pNN printed Neural Network. viii, xxi, 2, 19, 22, 23, 25–28, 30, 32–36, 53, 55, 58–60, 62, 66, 67, 73–75, 77, 79, 80, 83, 85, 87, 89–91

1 Introduction

The Internet of Things (IoT) [25] is seen as a major technological revolution. According to the vision, sensors and electronics are embedded as a core component in everyday objects. This allows them to be connected seamlessly, communicate, and exhibit responsiveness to contextual changes, promising radical innovations for industrial applications as well as the customer experience. For example, industries like retail and agriculture may be able to track individual products through smart labels [130] and verify their authenticity. This is not only limited to expensive items, but even disposable, low-cost consumer goods such as milk boxes, bandages or even single pills could be enhanced with electronics in the future [110]. This may allow to assess the quality of products in terms of freshness or spoilage [130] and dynamically adjust their prices on a per-product basis. Smart clothes could measure health indicators [151] and detect body odor, while smart cups [8] may display the correct drinking temperature for coffee, tea or other beverages, allowing to completely rethink the interaction with simple, everyday objects.

Printed electronics can be seen as a major enabler of this revolution. Even though printed electronics is not as efficient as its silicon counterpart, it promises a low-cost fabrication of possibly disposable electronic components on various substrates [130]. Aside from replication printing techniques aiming for the mass production of standardized devices such as displays or batteries [33, Sec. 4.3], jet-printing technologies [33, Sec. 4.2] offer interesting capabilities in the context of IoT. Since their manufacturing tools are comparably affordable, they hold the potential of a democratisation of manufacturing technology, allowing the on-demand fabrication of low-cost, custom circuitry, anywhere and by anyone. In combination with 3D printing and functional inks, individuals may be able to create their own smart devices with desired geometries and material properties [130].

Unfortunately, printed electronics also offers several drawbacks. Among them are high latencies and comparably large footprints [14]. This is especially true for classical, digital designs [110], which may therefore often be unsuitable. Hence, realizing custom sensor processing circuitry in printed electronics will not be straightforward. While analog designs compare more favourably [154, Sec. 6.1], their design is often more complicated and most circuits are designed by experts with many years of experience in the field. Especially when designing circuits for novel technologies such as printed electronics, a thorough understanding of the underlying physics is required. Additionally, immature processes often lead to substantial variations which require especially robust designs. Unfortunately, proper design rules are not yet established. Hence, to enable the vision of truly customized, ubiquitous smart devices, flexible, automatic modelling and design solutions are imperative.

To tackle these problems, this work proposes the use of neuromorphic computing as a computation paradigm. This promises several benefits. Firstly, neuromorphic components can operate analog and thus directly process sensory data [156]. Additionally, the distributed, parallel character of the computations may allow to compensate for the high latencies in printed

electronics [67]. Finally, the use of neuromorphic components allows to formulate the design search problem as an optimisation problem similar to training artificial neural networks. Here, variations can be considered in the training procedure to obtain robust designs [156].

1.1 Objective and summary of contributions

The aim of this thesis is to explore the modelling and training of printed neuromorphic circuits. The contributions of this work can be categorized into the following main topics.

Modelling components of printed neuromorphic circuits

To train printed Neuromorphic Circuits (NCs), models describing their behaviour are required. The behaviour of the circuit thereby depends on the behaviour of its main components, also referred to as hardware primitives. These hardware primitives are the resistor crossbar, the activation function circuit and the inverter function circuit. The resistor crossbar thereby realizes a weighted sum operation and the activation function circuit introduces nonlinear behaviour. Additionally, the inverter function circuit is required to express a notion of negative weights, as these cannot be implemented directly.

In this work, nominal and variation models for these hardware primitives are proposed. The models are tailored towards training algorithms addressing the special characteristics of printed NCs. For example, the ability to fabricate only one of two alternative connections (inverted and noninverted), depending on which one is required for the specific task. Furthermore, the variation models allow to not only assess the quality of printed NCs under variation, but also to consider them in the training procedure.

To model more complex distribution and correlations, a data-driven framework for variation modelling was developed. The framework is not limited to modelling NCs components but can also be used to model the variation in other electronic components or circuits such as printed transistors, e.g., [126]. The models for the individual components can be combined to form a model for a printed neuromorphic circuit. This model is referred to as printed Neural Network (pNN).

Training printed neuromophic circuits

Training printed NCs 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. Based on a model of a printed NC (the pNN), training algorithms similar to those of artificial neural networks can be developed. A suitable training algorithm for printed NCs should be able to respect the characteristics of the technology. Namely, technologically feasible ranges for the conductances of resistors. Additionally, the comparably large variation in fabrication of printed circuits should be accounted for.

To address these characteristics, this work proposes a training algorithm that respects the limited ranges of conductance values. Additionally, anticipated component variations are addressed by a modified training objective and Monte Carlo gradient estimation. Furthermore,

a loss function to guide the training procedure toward desired configurations and an initialisation scheme to improve the learning dynamics are proposed. Finally, a post-fabrication tuning procedure to increase the yield of printed NCs through additive manufacturing is described.

1.2 The structure of this work

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

- Chapter 2 provides a background on printed electronics and its challenges. Then, the concept of automatic circuit design is briefly introduced to provide context to this work. Furthermore, neuromorphic circuits, artificial neural networks and gradient-based learning are presented as additional preliminaries to the following chapters.
- Chapter 3 introduces printed neural networks as a model for learning the functionality of printed neuromorphic circuits. Furthermore, the steps of the training procedure for printed neural networks are discussed in detail, and several trained printed neural networks are evaluated on benchmark datasets.
- Chapter 4 introduces the data-driven variation modelling framework. A special focus is placed on semi-empirical models involving fitting parameters. The framework is evaluated by developing a variation model for a printed electrolyte-gated transistor.
- Chapter 5 directly builds on Chapter 3 and discusses a method for variation-aware training of printed neural networks. For this purpose, variation models for the core components of printed neuromorphic circuits are developed. The framework of Chapter 4 is thereby employed to devise variation models for activation function and inverter circuits. Printed neural networks trained using the proposed procedure are then evaluated under simulated variations.
- Chapter 6 proposes a post-fabrication tuning procedure for neuromorphic circuits based on the printed neural network model from Chapter 3. The procedure allows to systematically reprint parts of the circuit in order to restore the intended operation and is also evaluated under simulated conductance variation.
- Chapter 7 summarizes the work and provides an outlook on future research directions.

Chapters 3 - 6 introduce further preliminaries and background material in a dedicated section and close with a discussion and possible directions for future work.

2 Background

To provide context to this work, the following briefly introduces printed electronics and challenges in designing and modelling printed circuitry. The introduction is followed by brief review of automatic circuit design which is related to the design approach of training neuromorphic circuitry. Then, neuromorphic circuits and the concept of neuromorphic computing, which focuses on a brain-inspired computing paradigm, is described. As one of the underlying concepts for this work is artificial neural networks, they are briefly introduced as a prerequisite. Additionally, gradient-based learning, which is the most common strategy employed to train artificial neural networks, is explained. Finally, the basic hardware primitives of neuromorphic circuits are introduced and the differences between printed and programmable NCs fabricated in conventional technologies are outlined.

2.1 Printed electronics

Printed electronics is an emerging, complementary technology to conventional, silicon-based electronics [101]. Conventional electronics are fabricated through subtractive processes of removing material via lithography and etching. Contrary to that, manufacturing in printed electronics is an additive process where material is only deployed where it is needed [33, Sec. 1.1]. This characteristic leads to different properties of the fabricated circuits. Even though printed electronics cannot challenge conventional electronics in terms of integration density and performance, they exhibit lower fabrication costs (per area) and allow for fabrication on a wide range of different substrates through conductive inks [24]. Most notably, printed circuits can be fabricated on flexible and stretchable substrates such as textile [12, 95], foil or paper [23]. Furthermore, through appropriate choice of materials, nontoxic, bio-degradable and possibly edible electronic components [83, 113, 121] can be fabricated.

These characteristics allow printed electronics to target many novel application domains for which solutions in conventional electronics are either too expensive, or cannot meet the required specifications [123, 110]. Examples include applications for soft sensors [99, 90, 82], soft robotics [28] and e-textiles [12]. Additionally, cheap, disposable and short-term-use electronic products [121] can be utilized for medical and agricultural applications [113, 81], as well as smart packaging [20] and labelling [147]. Especially recently developed low-voltage technologies, such as the printed electrolyte-gated transistors [100, 101], in conjunction with printed batteries [95, 31] or printed energy harvesters [2] may be employed in the context of these applications. For these reasons, printed electronics is increasingly recognized as a key enabler for the IoT [24].

There are several processes for the fabrication of printed electronics. They can be broadly categorized into replicate printing and digital printing techniques. Replicate printing techniques (e.g., screen, flexography and gravure printing), similar to conventional electronics, require the fabrication of a mask or masterplate which is usually expensive. However, after fabrication

of the mask, replicate printing techniques are generally scalable and allow for an high volume and high throughput production [33, Sec. 4.3]. Contrary to replicate printing, digital printing technologies (e.g., inkjet or aerosol jet printing) require no mask. Here, devices are printed by ejecting droplets of conductive ink through nozzles onto the substrate. Even though this makes digital printing not as scalable as replicate printing processes, it allows for a custom fabrication of each component and circuit without substantial setup costs [33, Sec. 4.2].

Especially the characteristics of digital printing are in stark contrast to conventional industrial electronics manufacturing, where electronics are usually fabricated centralized and at large scale. Similar to 3D printing for manufacturing, this unique characteristic of digital printing technologies gives rise to the vision of "democratizing fabrication" [130]. A scenario in which anyone can fabricate electronics at point-of-use [24]. In the context of IoT, this would allow the cheap, on-demand fabrication of customized smart devices for home users in the future [130].

Challenges in modelling and designing printed circuitry

To realize this vision, many challenges must be overcome [24]. For example, unique challenges arise in circuit design for printed electronics and process design kits are still underdeveloped [124, Ch. 2]. Additionally, the vast possibilities of combining substrates and ink compositions require new models to be developed for simulations. However, since the underlying physics is often not fully understood, mostly semi-empirical models for devices have been developed so far, see [100]. Moreover, the variations of devices are generally much higher than those of conventional electronics [24], and accurately modelling these variations is imperative for designing functional circuits [124, Sec. 1.2.1].

In the future, near sensor processing applications can be expected to frequently arise in the context of IoT applications. To effectively process such data, new and optimized computing paradigms employing machine learning models may be required [154, Ch. 4]. Even though classical digital design is theoretically feasible for such applications, it is often not very efficient [14] and requires high device counts [110] [154, Sec. 6.2].

For this reason, bespoke (tailored) analog designs have been explored, allowing to realize near sensor processing circuitry with lower footprint in area and power, while also offering higher performance [154, Sec. 6.2]. Designs for these circuits could be devised on-demand through appropriate learning algorithms. Since these algorithms need to respect the specialized requirements and characteristics of the technology, such as the feasibility of certain device types, specialized learning algorithms are required. Finding a circuit configuration through these algorithms may be seen as an automatic design solution.

2.2 Automatic circuit design

The idea of automatic circuit design or automatic circuit synthesis is to find a design for a circuit that satisfies a certain functionality. The functionality is thereby given through input-output relationships or a behavioural description [141]. The two parts of a design usually relate to finding a suitable topology and sizing of the component. Finding the topology refers to choosing which components to connect, while sizing relates to finding a suitable parameterization, e.g. widths for transistors or conductance values for resistors, of the components [86]. Further

solutions may also include the derivation of the circuit layout, including component placement and routing [62, 7].

The techniques for automatic circuit design have been broadly characterized into knowledge-based and optimisation-based approaches [7, 136]. According to [7], knowledge-based approaches were the first to appear and leverage prebuilt design plans, see [76, 161]. Optimisation-based approaches utilize optimisation techniques to obtain solutions to the design problem. They can be further categorized into equation-based, simulation-based, and learning-based approaches [136].

Equation-based approaches, see [104, 62, 116, 68, 35], perform the optimisation based on a set of equations describing the circuit and the components. The equations are thereby derived either analytically, automatically through symbolic analysis tools [7], or through the fitting of circuit simulations, e.g. [35]. Through the use of these equations, evaluations tend to be faster, and numerical solvers, such as interior point methods [68], are used to solve the design problem.

Simulation-based approaches use circuit simulations to evaluate the performance of a configuration. Such approaches often leverage evolutionary algorithms, see [86, 7, 148, 30, 59], or more recently also Bayesian optimisation, see [97, 65, 72, 162]. The latter approaches thereby mostly focus on component sizing. The ability to combine topology selection and component sizing is one of the main advantages when using evolutionary approaches. Furthermore, they allow to include various other design constraints and have few limitations [68]. To reduce the number of costly circuit simulations that have to be performed, several simulations-based methods also employ predictive models, such as SVMs [7] or neural networks [59], to rule out unfavourable solutions. More recently, learning-based approaches leveraging reinforcement learning have been promoted [150, 136, 160], and claim to be more sample efficient, i.e, requiring less circuit simulations than evolutionary algorithm-based methods. Similar to evolutionary approaches, reinforcement learning-based methods can also perform topology search and component sizing jointly.

In contrast to these approaches, this work employs neuromorphic computing hardware primitives (Section 2.6) instead of classical digital or analog components. The components will be described through model equations referred to as printed neural networks (Section 3.2.1) and learned similar to artificial neural networks. Hence, through the lens of classical automatic circuit design, the approach in this work may be seen as an equation-based optimisation approach, where circuit design becomes similar to neural network training (see Section 2.5). After training, the components can be mapped to neuromorphic hardware primitives that will be introduced in the following.

2.3 Neuromorphic circuits

The terms neuromorphic computing and neuromorphic circuits [107] refers to circuits that utilize ideas of brain-like computing. Recently published works on neuromorphic computing thereby share one or multiple of the following attributes: 1) non-von-Neumann architectures, 2) analog or mixed-signal computation and 3) spiking behaviour as in biological neurons [16].

The first aspect emphasizes the contrast of neuromorphic computing to classical von-Neuman computing. In von-Neuman computing the data is read from the memory, computations are

carried out in the (central) processing unit, and the result is written back. In neuromorphic computing, computations are performed decentralized, in brain-inspired computation units referred to as neurons. Through this, computations are directly performed in memory and are inherently parallel, avoiding the so-called "memory-wall". These characteristics are often cited as being better suited for high performance computing [71], increasing the speed of certain computations and also lowering the power consumption [5].

The second aspect is analog computing. Here, instead of digitizing the signal and building it out of basic AND and OR operations, different hardware primitives, i.e., "neurons" are used to implement operations. Most notably, a zero-cost addition can be achieved through *Kirchoff's law* [107], and weighted sums are realized through resistor crossbars, see [71]. A further benefit of analog computing is that analog input signals from sensors do not need to be converted to digital signals through additional circuitry, but can be directly processed.

Thirdly, to follow the true, biological inspiration, neuromorphic circuits often try to incorporate a time dependent aspect, i.e., the accumulation of charges over time. This is considered in so-called spiking neural networks.

In this work, NCs trying to mimic artificial neural networks (see Section 2.4) are discussed. Hence, the aspect of time dependence and charge accumulation (as in spiking neural networks) is not considered. The first and the second aspect however apply. Such NCs are easier to realize and train, and surpass hardware realisations of spiking neural networks in almost all aspects, offering better performance (delay), and requiring lower power and area [41]. Since printed electronics already performs comparably weak regarding these aspects [24], neuromorphic circuits based on artificial neural networks, as discussed in the following, may be considered more suitable for real-world applications.

2.4 Artificial neural networks

This section briefly introduces fully connected (feedforward) artificial neural networks. Contrary to spiking neural networks, which try to reflect the firing behaviour of biological neurons as close as possible, artificial neural networks are only loosely inspired by biological neurons. They can be best understood as directed computation graphs with weighted edges (see Figure 2.1) expressing a function $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ parameterized with $\boldsymbol{\theta}$.

The edges are referred to as connections, and the nodes are called neurons. Furthermore, all neurons can be organized in so-called layers l. All neurons in the same layer share the same inputs. The computation performed by a neuron j is a weighted sum of all incoming connections i with weights w_{ij} plus some additional scalar value b_j called bias. Finally, a nonlinear function $\phi(\cdot)$ called activation function is applied, leading to the neuron model equation

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

where $\mathbf{x} = [x_1, x_2, \cdots]^\top$ is considered to be a vector of inputs x_i. Various kinds of functions can be employed as activation functions. Historically, sigmoidal-functions were widely used, but are often replaced by rectifiers [77], such as ReLUs, for simplicity and speed [87] in modern neural networks [56, Sec. 6.1]. However, other functions may perform just as well [56, Sec. 6.3.3].

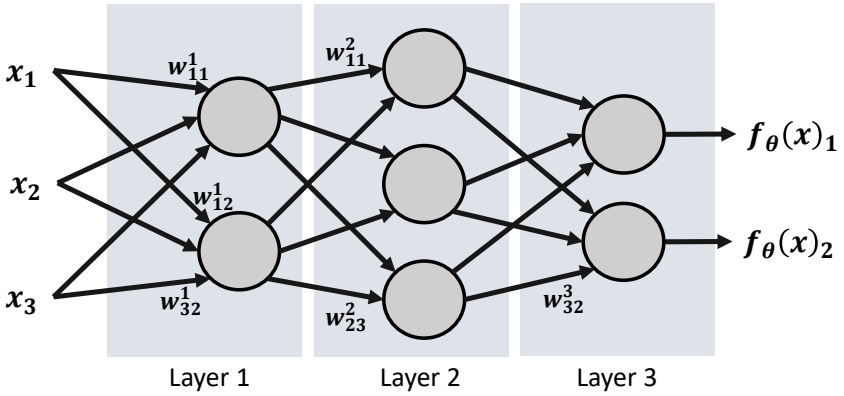


Figure 2.1: A fully connected artificial (feedforward) neural network mapping three inputs to two outputs. In the layer l, for each node j, all incoming connections i are multiplied with their respective weights w^l_{ij} and summed up with an additional bias b^l_j (omitted in the figure). Afterwards,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

Altogether, the main requirement for an activation function is nonlinearity[1] and differentiability (almost everywhere). The differentiability is mainly required to successfully apply gradient-based methods [56, Sec. 4.3] for finding the parameters of the neural network.

Viewing neural networks as connected neurons reveals the analogy to biological neurons, however, this view is less useful for applications. Here, neural network are usually implemented and discussed in terms of layers. A layer thereby denotes the combined computations of several neurons with shared inputs and can be described by

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

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 $\boldsymbol{b}^l$, and the activation function should be seen as an element-wise operation. Through composing multiple layers, a closed-form expression of a neural network can be written as

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \left(\text{Layer}^L \circ \cdots \circ \text{Layer}^l \circ \cdots \circ \text{Layer}^2 \circ \text{Layer}^1\right)(\mathbf{x}). \tag{2.1}$$

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 the network are summarized in the set/vector $\boldsymbol{\theta} = \{\mathbf{W}^l, \boldsymbol{b}^l \mid l = 1, \cdots, L\}$ for brevity and the network is simply denoted as the function $f_{\boldsymbol{\theta}}(\mathbf{x})$.

The network given by (2.1) is called a fully connected (feedforward) neural network. Neural networks of this type are able to approximate a wide range of functions (details see [56, Sec. 6.4.1]) with arbitrary precision. Note that there are also other networks such as recurrent networks, see [56, Ch. 10], which cannot be described as easily. However, these types of networks are not considered in this work. Therefore, all neural networks in the following explicitly denote artificial feedforward neural networks and are simply referred to as neural networks.

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 inputs $\mathbf{x}_n$ 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

[1] With linear activation functions the neural network can only express linear functions [56, Sec. 6.1].

of the output of the network $f_{\boldsymbol{\theta}}(\mathbf{x}_n)$ for a training instance $\mathbf{x}_n$ to the desired output $\mathbf{y}_n$ [56, Sec. 6.2]. 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. $\mathbb{Y} \subseteq \mathbb{N}$, and $\mathbb{X} \subseteq \mathbb{R}^m$ with $m = \dim(\mathbf{x})$. The neural network is then build to have as many outputs as there are classes.[2] Consequently, each class can then be associated with an index of the neural network output (vector). Formally, $f_{\boldsymbol{\theta}}(\mathbf{x})_i$ denotes the i-th output. 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

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

Analogously, the loss $L(\boldsymbol{\theta})$ can also be defined as the average of the $l(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$ for all $\mathcal{D}$.

Aside from a low mismatch between the actual and the intended output of the network, additional criteria regarding the parameterization $\boldsymbol{\theta}$ of the network $f_{\boldsymbol{\theta}}(\mathbf{x})$ are desired sometimes. Such preferences can be expressed through so-called penalty functions. For example, the generalisation[3] of the network can often be improved by decreasing the magnitude of unimportant weights. This can be achieved through *weight decay* [56, Sec. 5.2.2], which penalizes the euclidean norm of the weight vectors. Although penalty functions are mostly employed for regularisation and improving the generalisation error, they can also be used to express more general constraints or preferences.

Having defined a loss $L(\boldsymbol{\theta})$ [and possibly a penalty function $P(\boldsymbol{\theta})$], the neural network $f_{\boldsymbol{\theta}}(\mathbf{x})$ can be trained/learned. Training a neural network can be understood as trying to find a (good enough) solution for the optimisation problem

$$\boldsymbol{\theta}^\star = \operatorname*{argmin}_{\boldsymbol{\theta}} \quad L(\boldsymbol{\theta}) + \lambda \cdot P(\boldsymbol{\theta}), \tag{2.2}$$

where $\lambda \in \mathbb{R}^+$ is a hyper-parameter to scale the influence of the penalty $P(\boldsymbol{\theta})$. After a parameter vector $\boldsymbol{\theta}^\star$ is found, the neural network $f_{\boldsymbol{\theta}^\star}(\mathbf{x})$ should yield good results when predicting $\mathbf{y}_n$ from inputs $\mathbf{x}_n$. However, since (2.2) cannot be solved analytically and the loss $L(\boldsymbol{\theta})$ is usually nonconvex, one may only hope to find good solutions through iterative procedures. One such iterative procedure is gradient descent [56, Sec. 4.3] which will be discussed in the next section.

[2] Sometimes only $k-1$ outputs are chosen to encode k classes. This is especially common in two class settings, where only one output is required as the intended output class can be decided based on a threshold value. Besides having less output neurons, there are also other advantages of the $k-1$ encoding such as less correlated outputs. However, in this work, the k outputs for k classes encoding is used. The reasons for this is that the k output encoding allows to obtain the intended output class via a simple argmax on the network output which should be easier to realize in hardware.

[3] The generalisation refers to the predictive performance of the neural network on unseen data.

2.5 Gradient-based learning

Gradient descent is an optimisation algorithm for unconstrained optimisation problems with differentiable objective functions. For the application of training neural networks, it is assumed in the following that only the loss $L(\boldsymbol{\theta})$ should be minimized. However, adding a penalty term does not hurt the generality.

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.,

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

For some $\alpha \in \mathbb{R}^+$ small enough, (certainly for $\alpha \to 0$) the $\mathcal{O}(\alpha^2)$ part can be neglected and the expression reveals a characterisation of a descent direction for $\boldsymbol{d}$. If $\boldsymbol{d}$ is chosen such that $\boldsymbol{d}^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \leq 0$, then $L(\boldsymbol{\theta}^{(t)} + \alpha \cdot \boldsymbol{d}) \leq L(\boldsymbol{\theta}^{(t)})$ (see [11, Sec. 1.2]). In other words, the value of the function $L(\boldsymbol{\theta})$ can be decreased when moving from $\boldsymbol{\theta}^{(t)}$ in the direction $\boldsymbol{d}$.

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

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}). \tag{2.3}$$

The parameter $\alpha^{(t)} \in \mathbb{R}^+$ is referred to as learning rate or step size at iteration t, and is chosen through one of various heuristics (often simply as a small and constant value) in practice. Also, more modern methods were developed on the basis of gradient descent which leverage information of previous gradients in the update rule such as Momentum (see [56, Sec. 8.3.2]) or Adam [84]. Applying such techniques for learning neural networks is often summarized under the term gradient-based learning [56, Sec. 6.2]. More generically, these procedures can be seen as updating the parameters of the neural network in each step t through some small, weighted modification $\Delta\boldsymbol{\theta}^{(t)}$, i.e.,

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \alpha^{(t)} \Delta\boldsymbol{\theta}^{(t)}.$$

The update $\Delta\boldsymbol{\theta}^{(t)}$ is thereby chosen to decrease the loss function $l(\mathbf{x}, y, \boldsymbol{\theta})$ (at least in expectation) in each iteration t for an appropriate choice of $\alpha^{(t)}$. In the context of neural network learning, the procedure can either be stopped after a fixed number of iterations, or if the loss $L(\boldsymbol{\theta}^{(t)})$ does not improve anymore over several updates. The improvement of the loss $L(\boldsymbol{\theta}^{(t)})$ can thereby be measured either on the training data or on a separate validation dataset.

What remains to be chosen is the starting points $\boldsymbol{\theta}^{(0)}$. Especially when applying gradient-based learning to optimize neural networks, the initially chosen $\boldsymbol{\theta}^{(0)}$ (initialization) can greatly influence the success of learning [55]. One reason for this is that unsuitable initializations may lead to unfavourable propagation dynamics which hinder the learning process. For example, if the initialisation procedure leads to a layer mapping almost all inputs to a saturation region of the activation function, $\boldsymbol{\theta}^{(0)}$ can lie on a plateau with $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(0)}) \approx 0$ (vanishing gradients). Here, update rules like (2.3) only decrease the loss $L(\boldsymbol{\theta})$ very slowly.

To deal with such effects, multiple strategies have been developed to find good heuristics for parameter initialisation. In these schemes, the weights are usually drawn uniformly or normally distributed around zero [56, Sec. 8.4], where the scale is related to the number of inputs and/or output connections [55]. Additionally, schemes may vary based on the activation function [66], as it influences the standard deviation of the layer outputs.

Since the computation of the gradient is an integral part of gradient-based methods, it needs to be computed efficiently. For computational graphs such as neural networks, this can be done through the backpropagation algorithm [132] in conjunction with automatic differentiation techniques, see [117].

To summarize, through gradient-based learning, parameterized models can be optimized to behave in a way specified by the training objective, i.e., the loss function. Gradient-based learning is thereby not limited to neural networks with weights and biases as described in Section 2.4. Also, models for printed neuromorphic circuitry, if expressed appropriately, may qualify for gradient-based learning. The main requirement is for the models to be composed of parameterized, modular components, which express differentiable computations.

Finally, it should be noted that gradient descent may also be used for problems that technically do not permit its application, e.g., if the function is not differentiable (everywhere). However, in these cases, the use of the method may be seen as a heuristic, and success cannot be guaranteed.

2.6 Hardware primitives for neuromorphic computing

Since the NCs considered in this work draw inspiration from artificial neural networks (see Section 2.4), their fundamental buildings blocks reflect the same core components, i.e., neurons constructed from an activation function $\phi(\cdot)$ and the weighted sum operation $\sum_i x_i w_i + b$. From these neurons, arbitrary sized NCs can be built by stacking and connecting layers of neurons as seen before. Depending on the technology and availability of certain components, such as analog-to-digital converters and sense amplifiers, different implementations have been proposed for the activation function and the weighted sum operation. The components described in the following can be understood as analog, inverter-based NCs similar to [5] and have been previously described in detail in [154, Sec. 4.1].

2.6.1 Realizing weighted sums through resistance crossbars

The first fundamental operation to implement is the weighted sum. For this purpose, most works utilize resistance crossbars, see [71, 5, 157], where the input and output signals are represented by voltages (see Figure 2.2). The connections thereby form a classical "Y" or

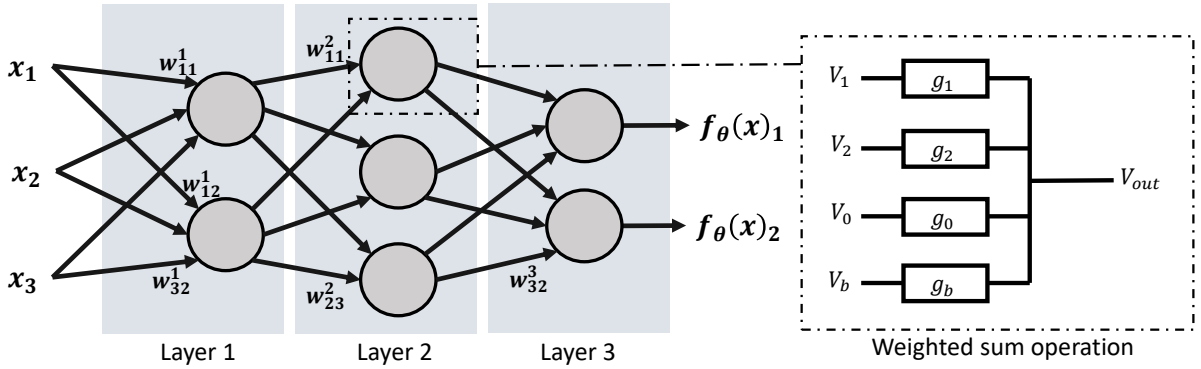


Figure 2.2: A weighted sum operation of a neuron realized through a resistor crossbar.

"star" circuit with a resistor in each branch.

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

$$\begin{aligned}
\sum_i I_i = 0 \iff \\
(V_b - V_{out}) \cdot g_b + \sum_i (V_i - V_{out}) \cdot g_i = 0 \iff \\
V_b \cdot g_b + \sum_i V_i \cdot g_i - V_{out} \left(g_b + \sum_i g_i \right) = 0 \iff \\
V_{out} = \underbrace{V_b \cdot \frac{g_b}{g_b + \sum_j g_j}}_{b} + \sum_i \underbrace{V_i \cdot \frac{g_i}{g_b + \sum_j g_j}}_{x_i \cdot w_i}.
\end{aligned}$$

For simplicity and brevity of notation, everything is expressed in terms of conductance values g_i instead of resistance values ($g_i = R_i^{-1}$). The analogy to the weighted sum operation can be readily seen by interpreting x_i as V_i. In this case, a weight w_i is a function of all conductances of the crossbar resistors (see also [154, Appendix B]) and given by

$$w_i = \frac{g_i}{g_b + \sum_i g_i}. \tag{2.4}$$

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 (voltage divider). For simplicity of the resulting circuit, V_b will be chosen as a fixed value (assuming 1V in the following). Through this, the bias b behaves like a weight w_b with a fixed input $V_b = 1\text{V}$.

Evidently, the parameters expressed through the crossbar are not as flexible as the general weights and biases in the artificial neural network. Most notably, the values of the parameters w_i and b are bound to the range of $[0, 1]$ and are coupled through the denominator $g_b + \sum_i g_i$ in (2.4) which leads to $b + \sum_i w_i = 1$. Since this coupling would reduce the effective number of free parameters of the neuron by one, it is relaxed through an additional pseudo-input connection $V_0 = 0\text{V}$ (ground) with a conductance g_0 (see Figure 2.2). Through this additional input, the other weights and the bias only have to satisfy the inequality $b + \sum_{i>0} w_i \leq 1$ instead of the more strict equality in absence of g_0. Aside from the constraints of the crossbar, also technological limitations have to be considered. These are mainly concerned with the range of feasible conductance values $g_i \in [g_{min}, g_{max}]$ in the technology.

For brevity of notation, all conductances of a neuron are summarized in the vector $\boldsymbol{g}$ and its entries will be referred to as g_i. Consequently, the input vector $\mathbf{x}$ to each neuron is augmented with two additional entries $\mathbf{x} \leftarrow [\mathbf{x}, 0, 1]^\top$. The conductances connected to these inputs then represent g_0 and g_b respectively. For a given neuron, the entries of the weight vector $\mathbf{w}$ are then given by

$$w_i = \frac{g_i}{\sum_j g_j} = \frac{g_i}{\boldsymbol{g}^\top \mathbf{1}}. \tag{2.5}$$

Note that due to the input augmentation, $\mathbf{w}$ also contains the bias b as an entry. Analogously to neural networks, the computations of multiple neurons can be summarized in layers computing multiple weighted sums $\mathbf{xW}$ with weight matrices given by

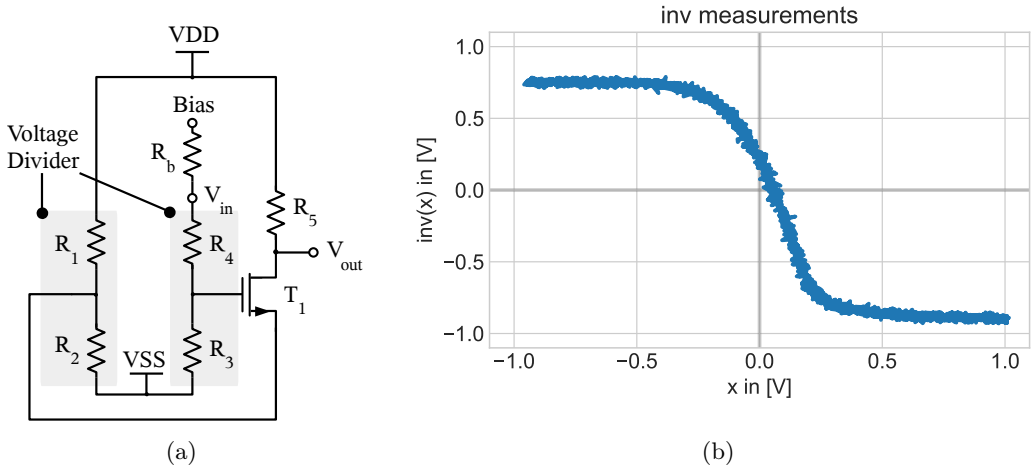
$$\mathbf{W} = \mathbf{G}\,\mathrm{diag}(\mathbf{G}^\top \mathbf{1})^{-1}.$$

Here, the matrix $\mathbf{G}$ summarizes the conductance vectors of each printed neuron in its columns, and diag$(\cdot)$ extends its input vector to a diagonal matrix.

2.6.2 Representing negative weights through inverted inputs

Since conductances g_i can only be nonnegative, the weights w_i formed through ratios of conductance values in (2.5) are also restricted to nonnegative values. However, to be able to solve tasks relevant for applications, a notion of negative weights is required.

As this cannot be implemented through conductances, the idea is to instead invert the respective input x_i to w_i. Hence, 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 and may only be approximately implemented by inverting the signal x_i through an appropriate circuit, see [5]. One such circuit has been realized in printed electronics (see Figure 2.3a). Measured outputs of the respective circuit can be seen in Figure 2.3b. Hence, the notion of negative weights is expressed through $x_i(-w_i) = (-x_i)w_i \approx \text{inv}(x_i)w_i$.



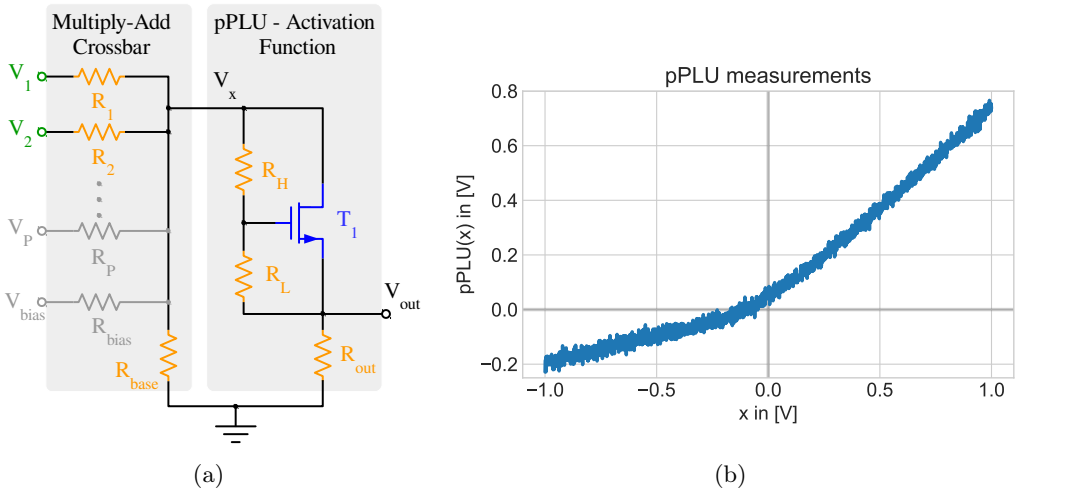
(a) (b)

Figure 2.3: Figure (a) displays the circuit for the input inverter. Figure (b) shows measurements of the output voltage. Figure (a) was created by Dennis D. Weller and adopted with permission.

To summarize, in case of a positive weight w_i, the input x_i can be directly used, while negative weight w_i require additional circuitry. The circuitry thereby approximates $-x_i$ through inv(x_i). However, since inv(x_i) only represents an approximation to $-x_i$, the characteristic of inv(x) has to be accounted for when modelling and training printed NCs (see Section 3.2.2).

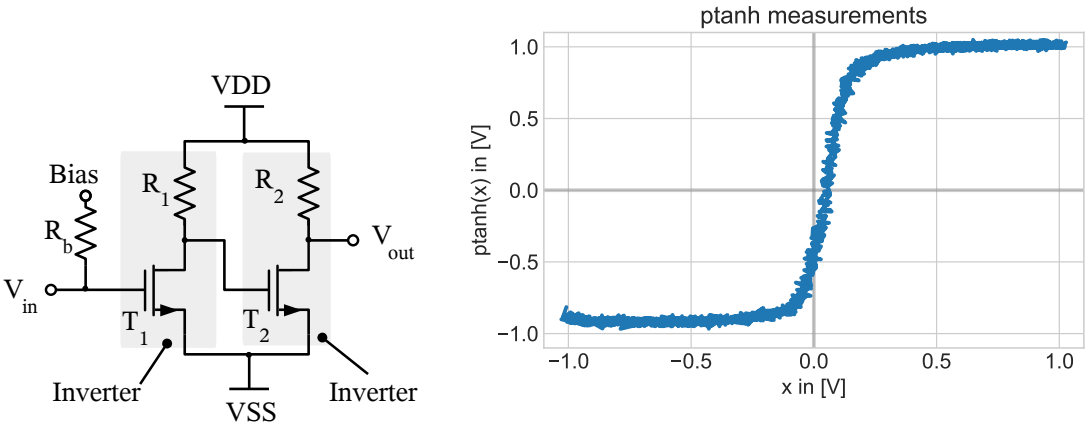
2.6.3 Activation functions for printed neuromorphic circuits

Besides the weighted sum operation, the activation function represents the second fundamental component of a neural network. As the main requirements for the activation function are smoothness and nonlinearity, various nonlinear circuits could be used to represent activation functions. Nevertheless, existing solutions often try to approximate classical functions such as tanh or ReLU. For example, the pPLU [157] represents a ReLU-like activation function, while [5] describes an inverter-based activation function of sigmoidal shape. The latter was similarly realized in printed electronics and referred to as ptanh [156].



(a) (b)

Figure 2.4: The printed linear unit activation function [157]. Figure (a) shows the circuit for a neuron with a pPLU activation function. Figure (b) shows measurement data of the pPLU circuit. Figure (a) was adopted from [157].



(a) (b)

Figure 2.5: The inverter-based tanh activation function circuit. Figure (a) shows the ptanh circuit. Figure (b) shows measurements from the circuit. Figure (a) was provided by Dennis D. Weller and adopted with permission.

Among the activation functions realized in printed electronics, the ptanh can be seen as more favourable. The reason for this is the modelling of the pPLU, as $\mathrm{pPLU(x)} = \max\{0.2x, 0.75x\}$ in [157]. As can be seen, the magnitude of the signal decreases in both, the positive and the negative region due to the multiplication with coefficients smaller than one. Consequently, $|\,\mathrm{pPLU}(x)| < |x|$ for all x. This is critical since the weights formed by the resistor crossbar in (2.5) are also smaller than one. Hence, the signal successively shrinks through the computations in each layer. This may not allow for a sufficient (measurable) signal separation at the output of the network.

Contrary to the pPLU, the inverter-based ptanh activation function (see [154, Sec. 4.1.3]) exhibits regions where the absolute value of the strength is increased through the activation, i.e., there are x such that $|\,\mathrm{ptanh}(x)| > |x|$. These regions allow to preserve or possibly increase the signal magnitude and improve the signal separation. Since the signal separation is essential

to measure, differentiate and compare the signal strengths at the different outputs of the NCs, only the inverter-based ptanh activation function is considered in the following.

2.7 Conventional versus inkjet-printed neuromorphic circuits

Although the overall designs of the NC components can be used for both, printed and conventional silicon-based technology[4], the fabrication process, and therefore the choice of the devices differs. Due to the subtractive fabrication process, silicon-based technologies have to be fabricated in multiple process steps including the fabrication of masks, which makes low-batch fabrication costly and time consuming. On the other hand, in inkjet-printed electronics, all components can be fabricated on the same printer, on-demand, and with little setup costs. Unfortunately, printed circuits generally also exhibit considerably lower performance and integration density [24]. Naturally, the envisioned applications for printed and silicon-based NCs also differ greatly. NCs fabricated in silicon-based technologies may provide acceleration and power savings for deep learning applications [5, 52]. Contrary to that, printed classifiers [110] and printed NCs aim to realize customized circuitry for near sensor processing in IoT applications [154, Sec. 2.5], that can be designed and fabricated on-demand and at point of use.

Aside from the envisioned applications, the components used for the hardware primitives of printed NCs and silicon-based NCs also differ. Most notably, in silicon-based NCs, the resistor crossbar is generally realized using some form of voltage-programmable resistors, see [71, 5]. On the other hand, in printed electronics, resistors are realized through printing different patterns/geometries of conductive ink. See Figure 2.6 for an example of a printed resistor. While it is possible to tune the conductance of a resistor after fabrication, the conductance value can only be decreased through this procedure. Hence, post-fabrication tuning capabilities of the conductance values (see Chapter 6 for details) are not nearly as flexible as for reprogrammable devices.

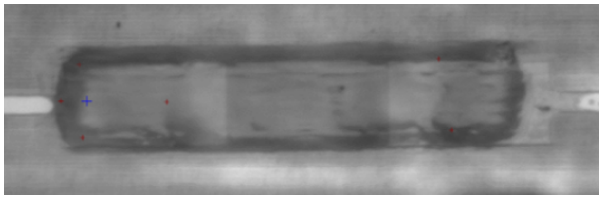


Figure 2.6: A single printed PEDOT:PSS resistor of 200 kΩ printed with two layers of width 200 µm and length 800 µm. The picture was taken by Dennis D. Weller and adopted with permission.

Due to the reprogrammability of silicon-based NCs, the task to be solved does not need to be known before the fabrication of the circuit. However, this also means that a silicon-based NC has to support all inverted and noninverted input connections allowing to program positive and negative weights. This may even become infeasible when possible skip-connections[5] should be considered. Furthermore, since different tasks may require networks of vastly different size, the predetermined network size may often be inadequate for a given task.

[4] The designs in silicon-based technologies may be adapted as inverters could for example be implemented more efficiently through the use of complementary devices, i.e., n-type and p-type transistors.

[5] A skip-connection is a connection from one layer to multiple other layers aside from its immediate successor.

In contrast to silicon components, printed components can be fabricated cheaply, on-demand and task-specific [110]. Through this, the network can be first designed and trained according to the task. Then, the exact number of neurons required for the task can be fabricated. Additionally, since the weights are already known at the time of production, depending on the sign of the weight, only either the inverted or the noninverted connection of each input needs to be fabricated. Thus, only half of the crossbar resistances are required. However, algorithms for training classical silicon-based NCs, such as [5], do not leverage this potential (details see Section 3.1). This calls for the development of training algorithms tailored to harness these savings and address the unique characteristics of printed NCs.

3 Printed Neural Networks

Alongside low-cost fabrication on various substrates, affordable, on-demand fabrication is one of the big promises made by printed electronics. The on-demand fabrication is thereby enabled through digital printing technologies such as inkjet printing. Similar to classical 3D printing for manufacturing, digital printing technologies allow custom circuitry to be fabricated at the point of use. This does not only benefit decentralized, industrial fabrication in small lot sizes, but may also allow for an on-demand realisation of customized smart devices in the future [130].

Unfortunately, the design of printed circuits is challenging and requires substantial effort. Classical, digital designs in printed electronics are often not very efficient. This is due to their large feature sizes (up to three orders of magnitude larger compared to silicon technologies), high latencies, and low device counts [67]. Hence, the implementation of conventional, digital architectures in printed electronics has extremely high hardware overheads. Since analog designs display considerable lower footprints [154, Ch. 6.1], they may offer a promising solution.

However, analog design is usually substantially more challenging, and to truly fulfill the promise of a personalized and customized fabrication [130], a simple and fast design approach is imperative. In line with the vision of cheap, on-demand fabrication, the effort of designing circuits should be similar to fabricating them. This requires design solutions to be lean, and ideally able to generate a readily printable design from a specification of its desired functionality. With respect to the applications in the context of the IoT, many tasks may consist of near sensor processing or simple control. Such tasks can be naturally targeted using machine learning models directly implemented analog and in hardware, see [110].

Following this idea, this chapter describes an automatic design (training) approach for printed neuromorphic circuitry. Here, the desired functionality of a design can be expressed as training data for a (printed) neural network model serving as a blueprint for a (printed) neuromorphic circuit. By training the neural network model, a design for a printed NC can be obtained without substantial manual design effort. The learned printed Neural Network (pNN) can then be mapped to respective hardware primitives of printed NCs, see [156]. Since the circuits designed this way should be readily printable, this concept paves the way to the on-demand realisation of customized smart devices. What is more, the use of the neuromorphic computing paradigm may also offer compelling solutions to other design challenges in printed electronics. For example, the slow processing speeds of printed components may be compensated by the inherent parallelism of neuromorphic computations [154, Sec. 4.1].

The rest of the chapter is outlined as follows: First, the related work on training similar neuromorphic circuitry is reviewed. Afterwards, the pNN, providing the model equations for learning printed NC, is derived. Then, the training procedure for pNNs is described. Finally, the training procedure is evaluated by training several pNNs on a set of benchmark datasets. The chapter closes with a discussion of the results and possible directions for future work. The results and methods discussed in this chapter are based on collaborative work that led to [156], and have also partially been reported in [154, Sec. 4.1].

3.1 Related work on training neuromorphic circuits

Due to the characteristics of the components described in Section 2.6, such as limited ranges of realizable conductances or the limitation to only realize negative weights approximately, it is not possible to simply map arbitrary artificial neural networks to NC components. Hence, an adapted training procedure is required.

In general, the process of training a NCs can be seen as finding the circuit or circuit configuration that approximates a desired functionality. For example, for a given programmable silicon-based NC, the target is to find a set of conductances values to which the crossbar resistors should be programmed, see [159, 5, 52, 51, 63, 6]. For printed NCs, where the circuit is not fixed beforehand, training can additionally involve choosing an architecture, i.e., the number of neurons and layers to print, as well as their connectivity.

In programmable, silicon-based NCs, training can often be tackled through on-device methods, e.g. [63, 6]. In on-device training, the true system response is directly used as a signal to adjust the conductances of the NC. However, this is not possible for printed NCs. Here, after initial fabrication, the conductances can only be changed in a limited fashion.[1] Hence, only model-based off-device training methods, that try to find circuit configurations based on a model of the circuit, can be employed.

The challenges of off-device training can be divided into two parts. Firstly, an accurate model for the NC behaviour is required. Secondly, based on the model, a training algorithm needs to be devised. The training algorithm should thereby find a configuration of (technologically) feasible conductances, for which the model exhibits the desired behaviour. Only a few works addressed the training of NCs with components similar to the ones described in Section 2.6.

In [157], a printed NC was proposed alongside a training algorithm. The NC is thereby modelled as a classical artificial neural network with a pPLU activation function (see Figure 2.4). However, the training approach is not limited to this activation function. To ensure feasibility of the design, the approach tries to respect the crossbar constraints, i.e., $\sum_i w_i \leq 1$. First, the classic backpropagation [132] algorithm is used to obtain updates for the weights and biases. Then, the parameter vectors (weights and bias) of each neuron are projected to feasible values by solving an optimization problem.

Although the crossbar constraints can be guaranteed through this procedure, the technological feasibility of the full weight range of $[0, 1]$ is not taken into account. In other words, not all weights in the range of $[0, 1]$ may be realizable through conductance value configurations. Furthermore, the work focuses explicitly on NCs with positive weights. This greatly limits the applicability. Finally, from the computation point of view, the training approach is rather expensive, as an optimization problem has to be solved for every neuron in every update step.

The training approach developed in [5] is also similarly used by [52, 51, 145] and was developed for inverter-based NC (with memristor crossbars). It incorporates both, normal and inverted inputs (expressing negative weights). The weighted sum operation is thereby expressed through

$$\sum_i x_i \cdot w_i^p + \mathrm{inv}(x_i) \cdot w_i^n.$$

[1] It is possible to adjust the weights after the initial fabrication by printing on top of the previously printed resistor. However, this procedure only allows to increase the conductivity and is therefore not suitable for the initial training. For further details on the post-fabrication tuning capabilities of printed NC see Chapter 6.

Here, the w^p and w^n denote positive and negative weights connected to the noninverted and inverted input respectively.

To accurately model the NC behaviour, both, the positive and negative weights are formed through the conductances of separate resistor crossbars and can be calculated using (2.5). In training, the technology constraints, such as limited ranges of feasible conductance values, are expressed through appropriate transformations of learnable parameters. For example, $g_i \in [g_{min}, g_{max}]$ are realized through learnable parameters $\theta_i \in \mathbb{R}$ that are mapped to (g_{min}, g_{max}) using an appropriately shifted and scaled sigmoid function, e.g.,

$$g_i = (g_{max} - g_{min}) \cdot \text{sigmoid}(\theta_i) + g_{min}$$

The parameters θ_i forming the conductances g_i can then be learned using the classic backpropagation algorithm.

Generally, this method is also applicable to train printed inverter-based NCs as discussed in this work. However, since the approach was developed for programmable (memristor) crossbars, it assumes a given NC with both inverted and noninverted inputs connected and the presence of the respective resistors in both paths. It is reasonable to assume that only a subset of these components and connections is required. For example, having either the inverted or the noninverted connection and the associated resistor of each input should be sufficient. Then, the on-demand fabrication capabilities of inkjet printing can be exploited to only fabricate the required components and connections.

Unfortunately, it is usually not possible to subsume the sum of the noninverted component $x \cdot w^p$ and the inverted component $\text{inv}(x_i) \cdot w^n$ of an input x into a single connection. More specifically, there exists no w such that for all values of x and arbitrary w^n and w^p, either $x \cdot w = x \cdot w^p + \text{inv}(x) \cdot w^n$ or $\text{inv}(x) \cdot w = x \cdot w^p + \text{inv}(x) \cdot w^n$. Only in the special case where either w^n of w^p is zero such a solution can be found. This indicates that there exists no simple way to map the parameters obtained by [5] to a design that only uses either connection. In other words, training printed NCs using [5] does not permit to leverage the savings associated with on-demand printing. Consequently, using this approach for training printed NCs would lead to wasteful designs, where at least half of the resistors and connections could be omitted.

To conclude, on-device training is not possible for printed NCs since they do not offer reprogrammability. Hence, only model-based off-device training can be employed. Off-device training requires an accurate model of the NC and a training algorithm that is able to respect technology constraints. Training printed NCs is similar to the training silicon-based NCs with programmable crossbars since they share common types of constraints. However, printed circuits can be fabricated customized and on-demand. This allows to fabricate only the required components, which saves material, production time, as well as area and power. Unfortunately, these saving cannot be harnessed when printed NCs are trained through training algorithms for programmable NCs. Thus, modified training algorithms and models are required that, aside from finding technologically feasible conductance values, allow to express the connectivity to inverted and noninverted inputs.

3.2 A model for learning printed neuromorphic circuits

As the first step towards an off-device method for training printed NCs, a suitable model of the circuit (components) is required. The model should be able to reflect the response of a printed NC to inputs and allow to derive a configuration of feasible conductances through training. The feasibility thereby relates to the technology constraints which permit only conductance values $g \in [g_{min}, g_{max}] \cup \{0\}$ to be printed. Here, the conductance value of $g = 0$ relates to not printing the respective resistor. Note that being able to effectively realize $g = 0$ (through not printing) also represents a fundamental difference between programmable and printed NCs. Additionally, the model has to be able to reflect the decision between using either the inverted or the noninverted version of each input.

The development of the model is discussed in several steps. First, the model assumptions are stated. Then, data-driven models for the circuits of the activation function and the inverter function are developed. Based on the models for these components, model equations for the printed neuron and printed Neural Network (pNN) layer are devised. These then form the fundamental blocks of pNNs, which serve as models for training printed NCs.

3.2.1 Modelling assumptions

For developing the equations of the pNN model, the following assumptions are made.

1. The inputs x_i are in the range of $[0, 1]$.
2. All conductance values $g_i \in [g_{min}, g_{max}] \cup \{0\}$ can be fabricated.
3. The outputs of the pNN can be considered distinguishable (measurable) if one output displays an activation exceeding a threshold T and all other outputs return values below 0.
4. The behaviour of the inverter and activation function circuit can be modelled accurately through a closed-form expression.
5. The behaviour of the printed NC can be modelled by combining the model equations for its components.

The first assumption relates to the input voltage range which is assumed to be $[0, 1]$. This is motivated by the operating voltage ranges of printed Electrolyte-Gated Transistors (EGTs) [100], and also represents a suitable choice for printed IoT devices. Furthermore, the circuits for the printed activation and inverter functions (see Section 2.6) have also been realized using printed EGTs, see [154, Ch. 4.1] [156]. Hence, all components operate on a similar range of inputs.

The second assumption essentially states the technologically feasible range of conductances. Note that even though any possible conductance in the specified range may theoretically be achievable, printed electronics exhibit substantial variations [24]. These variations usually do not permit realizing desired conductance values exactly. However, the procedure proposed in this chapter should first be evaluated by its ability to find suitable ideal conductances. Dealing with variations will be addressed in Chapter 5.

The third assumption is motivated by the requirement to properly assess the correct output of the NC. For this, a clearly measurable difference between the correct and the incorrect outputs of the NC is required. This is expressed through the threshold T (e.g., 100 mV). It is assumed that if the output surpasses T and all other outputs are below 0, the signals can be

distinguished.

The fourth assumption states that sufficiently accurate, closed-form models of the behaviour of the nonlinear circuit components, i.e., the activation function and the inverter circuit, can be obtained. This assumption is similarly made in [5], where the outputs of these circuits are fitted based on simulation data. The capabilities of modelling these circuit components are investigated in the following section.

Finally, the fifth assumption relates to how the pNN is constructed from the models of the components. Note that when connected, the individual circuits may behave differently than in isolation. For example, when connecting the activation function circuit to the crossbar as in Figure 2.4a, an additional current passing from the crossbar to the activation function circuit would need to be considered. However, this current is assumed to be negligible due to high input impedances at EGT gates, see [154, Sec. 4.1].

3.2.2 Modelling the nonlinear circuit components

Besides resistance crossbars, the core components of printed NC are the inverter and activation functions circuits. Ideally, their behaviour should be modelled through differentiable, closed-form expressions that can be directly used for gradient-based learning (see Section 2.5). To achieve this, a data-driven approach is taken, where models for the behaviour of the respective circuits are derived from measurement (or simulation) data.

Looking at Figure 2.5b and Figure 2.3b, it can be seen that the measured circuit outputs closely resemble classical sigmoidal shapes. This observation suggests using appropriately parameterized tanh functions to model the behaviour of the respective circuits, see [5]. For this purpose, let $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ be the measurement data, and let $\boldsymbol{\eta}^\star = [\eta_1, \eta_2, \eta_3, \eta_4]^\top$ be a vector of parameters for the model

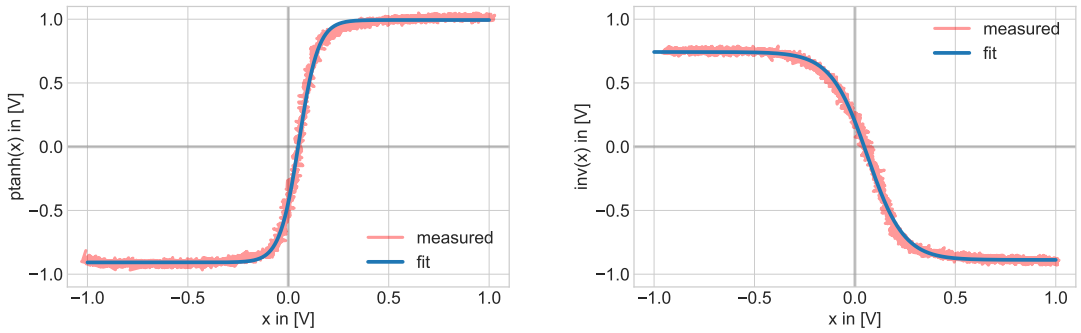
$$\psi_{\boldsymbol{\eta}}(x) = \eta_1 + \eta_2 \cdot \tanh((x - \eta_3) \cdot \eta_4).$$

Then, a suitable parametrization $\boldsymbol{\eta}^\star$ for which $\psi_{\boldsymbol{\eta}}(x)$ closely approximates the data $\mathcal{D}$ may be found as

$$\boldsymbol{\eta}^\star = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} \quad \sum_{n=1}^{N} \big(y_n - \psi_{\boldsymbol{\eta}}(x_n)\big)^2 .$$

To find the solution $\boldsymbol{\eta}^\star$, any optimization algorithm for nonlinear least squares problems [115, Sec. 10.3] can be used. The fitting results for the ptanh model and inv model functions can be seen in Figure 3.1.

Due to the close match between the fitted shape and the measurement data, the respectively parameterized tanh functions are directly used as models for the circuit behaviour in the following. Note that this approach may be simply adapted for other (printed) technologies, as long as the data exhibits a similar, sigmoidal characteristic.



(a) (b)

Figure 3.1: The fits for the ptanh(x) (a) and the inv(x) (b) function based on parameterized tanh-functions.

3.2.3 Printed neural networks

After obtaining the models for the nonlinear circuit components, the equation for a printed neuron can be devised. To accurately reflect the behavior of the crossbar (Section 2.6.1), a concept similar to [5] can be adapted. Here, learning is performed on surrogate parameters from which the value of the respective conductances can be inferred. Through this, no additional treatment is required to satisfy the crossbar constraints ($\sum_i w_i + b \leq 1$) from *Kirchhoff's* law. However, as already discussed in Section 3.1, the approach in [5] was developed for programmable NC and thus considers both, the inverted and noninverted inputs to be present. Since this is undesirable for printed NCs, a selection of only one of the connections should be supported.

To express this selection, learnable parameters called surrogate conductances are introduced. A surrogate conductance θ_i encodes the value of a respective conductance through their absolute value $g_i = |\theta_i|$, while the sign of θ_i expresses if the associated input should be inverted.[2] More specifically, if $\theta_i \geq 0$, x_i is directly used as an input, while in case of $\theta_i < 0$, the inverted input $\text{inv}(x_i)$ is used. Consequently, the weighted sum operation can be expressed as

$$\sum_i w_i \left(x_i \cdot \mathbb{1}_{\{\theta_i \geq 0\}} + \text{inv}(x_i) \cdot \mathbb{1}_{\{\theta_i < 0\}} \right) \quad \text{with} \quad w_i = \frac{|\theta_i|}{\sum_i |\theta_i|},$$

where $\mathbb{1}_{\{\cdot\}}$ denotes an indicator function returning 1 if the respective condition is true, else 0. By additionally applying the ptanh activation function, the complete model for the printed neuron is obtained. See Figure 3.2 for a graph of the output of a single weighted input and the respective printed neuron function. For given surrogate conductances $\boldsymbol{\theta}$, the printed neuron is smooth in the input $\mathbf{x}$. However, with respect to the learnable parameters $\boldsymbol{\theta}$, the function is not smooth at points where $\theta_i = 0$ due to the indicator function. Nevertheless, the function is still continuous at those points as $w_i \to 0$ if $|\theta_i| \to 0$.

From the equation of a printed neuron, layers and eventually networks can be constructed analogous to artificial neural networks (see Section 2.4). The equations for a printed layer are

[2] The surrogate conductance θ_0 encoding g_0 may always be considered positive as an inversion of V_0 (ground) is not necessary.

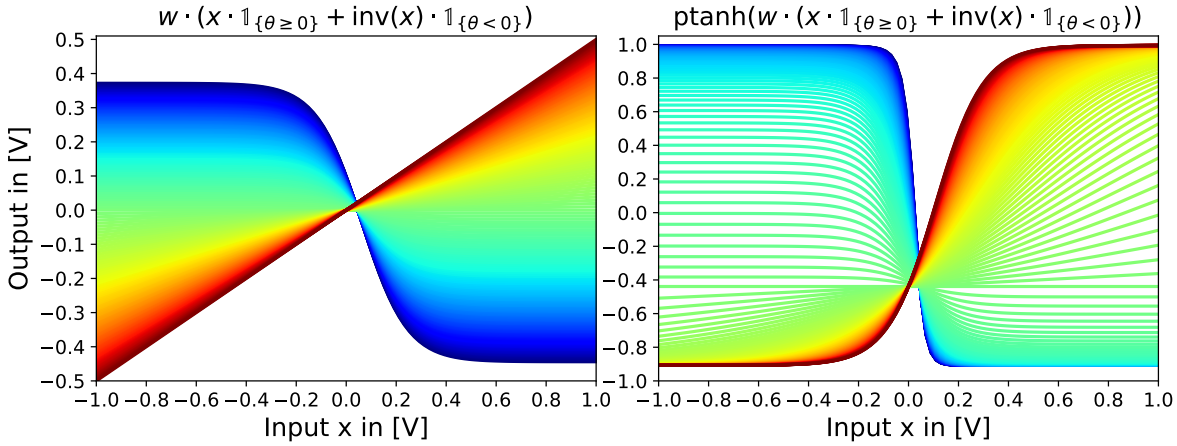


Figure 3.2: The output of the weighted sum operation (left) and the activated weighted sum operation (right) for a printed neuron with one surrogate conductance θ (with fixed $\theta_b = 0$ and $\theta_0 = 1$). The colored lines display the output for different values of $\theta \in [-1, 1]$ indicated through a color range from blue (-1) to red (+1). This leads to the corresponding weight range of $w \in [-0.5, 0.5]$. The crossing point between the lines lies at the root of inv at $x \approx 0.046$. Note that the y-axis scaling of the two plots is different.

then given by

$$\mathrm{ptanh}\left(\mathbf{x}\left(\mathbf{W} \odot \mathbb{1}_{\{\mathbf{S} \geq 0\}}\right) + \mathrm{inv}(\mathbf{x})\left(\mathbf{W} \odot \mathbb{1}_{\{\mathbf{S} < 0\}}\right)\right) \quad \text{with} \quad \mathbf{W} = \boldsymbol{G}\,\mathrm{diag}\left(\boldsymbol{G}^\top \mathbf{1}\right)^{-1},$$

where $\mathbf{S}$ denotes the matrix of surrogate conductances encoding if the respective input is inverted or not. The matrix $\mathbf{G}$, as before, consists of the conductance vectors of each printed neuron (stacked column-wise) and is obtained from $\mathbf{S}$ by taking the element-wise absolute value of the entries. Additionally, the $\mathrm{ptanh}(\cdot)$, $\mathrm{inv}(\cdot)$ and $\mathbb{1}_{\{\cdot\}}$ function should be understood as element-wise operations and $\odot$ denotes and element-wise multiplication. Analogous to an artificial neural network, a pNN can be constructed by composition of printed layers as in (2.1). Note that this layer-wise definition is only accurate if all neurons use the same ptanh activation function. Later, when the activation functions are assumed to vary (see Chapter 5), it may be better to stick to the underlying neuron-level equations. For brevity, the pNN is referred to as $f_{\boldsymbol{\theta}}(\mathbf{x})$ in the following and $\boldsymbol{\theta}$ denotes all learnable parameters, i.e., the surrogate conductances θ_i of all layers.

Using these model equations, the crossbar constraints regarding the weights hold by design. However, the technology constraints regarding the ranges for feasible conductance values $g_i \in [g_{min}, g_{max}] \cup \{0\}$ only hold if the values for the surrogate conductances are chosen appropriately. To find a set of feasible (surrogate) conductance values, the next section introduces a tailored training procedure for pNNs.

3.3 Training printed neural networks

For artificial neural networks, training relates to finding a parameterization for which the network expresses a certain desired behaviour. Usually, the desired behaviour is 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 $\mathbf{x}_n$. Hence, the network approximates the data in some sense while also exhibiting a certain degree of generalisation to correctly predict unseen instances.

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. More specifically, according to the modelling assumptions, the fabrication of the respective NC is only possible if the conductances g_i of the pNN satisfy $g_i \in [g_{min}, g_{max}] \cup \{0\}$. Since the surrogate conductances encode the value of the conductances through their absolute value, $|\theta|$ must lie in $[g_{min}, g_{max}] \cup \{0\}$. Unfortunately, this leads to a disconnected domain of $\theta_i \in [-g_{max}, -g_{min}] \cup \{0\} \cup [g_{min}, g_{max}]$ which complicates the training.

To deal with this, the constraints are addressed in two steps. Initially, the training is performed on a connected superset $\theta_i \in [-g_{max}, g_{max}]$ of the domain through classical gradient-based learning with projections. Solutions in the infeasible region $\theta_i \in [-g_{min}, g_{min}]$ are thereby discouraged through a penalty function. After training, all infeasible values are projected to zero. All steps are discussed in detail in the following alongside an initialisation strategy for training. Furthermore, a loss function is introduced to encourage sufficiently distinguishable outputs. An overview of the main components involved in training pNNs can be seen in Figure 3.3.

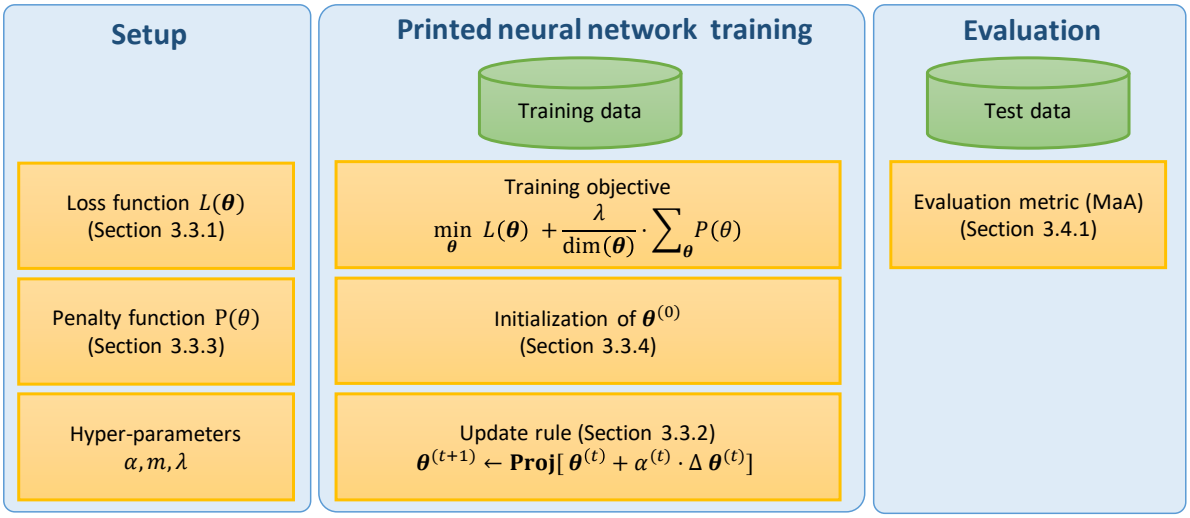


Figure 3.3: An overview for the main components of training printed neural networks.

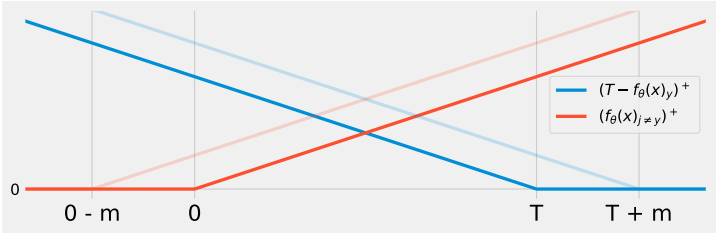


Figure 3.4: An illustration of the components of the loss function for the different outputs $f_{\boldsymbol{\theta}}(\mathbf{x})_y$ and $f_{\boldsymbol{\theta}}(\mathbf{x})_{j\neq y}$. There contribution grows linearly and is bounded below by zero. The loss function returns zero if the correct output $f_{\boldsymbol{\theta}}(\mathbf{x})_y \geq T$ and the other outputs return values $f_{\boldsymbol{\theta}}(\mathbf{x})_{j\neq y} \leq 0$. By choosing a nonzero margin $m > 0$, outputs lying further from 0 and T can be encouraged to achieve an increased signal separation (shaded lines).

3.3.1 The loss function

In neural network training, the loss function guides the learning process and expresses the favourability of a set of parameters $\boldsymbol{\theta}$. As discussed earlier in the modelling assumptions (see Section 3.2.1), a sufficient signal separation should be encouraged to be able to clearly measure and distinguish different outputs. More specifically, the value of the correct output of the network $f_{\boldsymbol{\theta}}(\mathbf{x})_y$ 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 [32] is constructed where

$$L(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} l(\mathbf{x}, y, \boldsymbol{\theta}),$$

with

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

Here, $T \in \mathbb{R}^+$ denotes an implementation-specific measuring threshold, $m \in \mathbb{R}^+$ is a user-defined parameter called margin and $(\cdot)^+ = \max\{0, \cdot\}$. Using this loss function, a positive loss is incurred if the output activation of the neuron associated with the correct class $f_{\boldsymbol{\theta}}(\mathbf{x})_y$ 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 hyper-parameter and can be used to further improve the separation. A conceptual illustration of the components of the loss function can be seen in Figure 3.4.

3.3.2 Gradient-based learning with projections

For a given loss function, a pNN can be trained using the classical, gradient-based learning as described in Section 2.5. The surrogate conductances $\theta_i \in \mathbb{R}$ thereby represent the learnable parameters. However, this does not necessarily result in a set of parameters $\boldsymbol{\theta}$ respecting the constraints $|\theta_i| \in [g_{min}, g_{max}] \cup \{0\}$, which have to hold to ensure the feasibility of the conductance values.

Thus, the training procedure has to be modified, and after each step t, the elements of the iterates $\boldsymbol{\theta}^{(t)}$ are projected on the interval $[-g_{max}, g_{max}]$. More specifically, a parameter update

at iteration t is realized by

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \text{Proj}\left[\boldsymbol{\theta}^{(t)} + \alpha^{(t)}\Delta\boldsymbol{\theta}^{(t)}\right],$$

where $\alpha^{(t)} \in \mathbb{R}^+$ denotes the learning rate in interation t and $\text{Proj}[\boldsymbol{\theta}]$ denotes an element-wise projection of the entries θ of $\boldsymbol{\theta}$, i.e.,

$$\text{Proj}[\theta] = \begin{cases} -g_{max} & \theta < -g_{max} \\ \theta & \theta \in [-g_{max}, g_{max}] \\ g_{max} & \theta > g_{max}. \end{cases}$$

Such operations are also commonly referred to as *clipping*, see [56, Sec. 10.11.1]. A conceptual illustration of the projection can be seen in Figure 3.5.

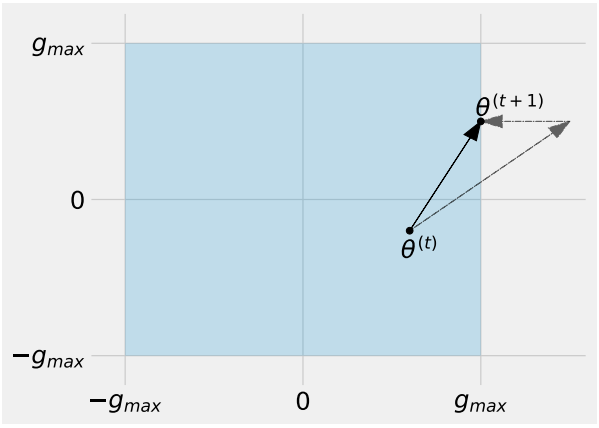


Figure 3.5: A conceptual illustration of an update step with projection to ensure the constraints $\theta_i \in [-g_{max}, g_{max}]$. The dashed arrow pointing outside of the region displays the update $\alpha^{(t)}\Delta\boldsymbol{\theta}^{(t)}$ that is projected back to the closest point in the feasible domain.

Overall, the procedure can be understood as an instance of a gradient projection method [11, Sec. 2.3.1]. The update direction $\Delta\boldsymbol{\theta}^{(t)}$ in each iteration can either be obtained as the negative gradient or through any appropriate optimization algorithm such as Adam [84]. It should be noted, that the loss function from Section 3.3.1 is not differentiable and the pNN model is not smooth due to the use of the indicator function $\mathbb{1}_{\{.\}}$. Hence, the application of gradient-based learning may be seen as an heuristic approach.

Through the projection step, all iterates $\boldsymbol{\theta}^{(t)}$ (most importantly the final one) exhibit values lower or equal to the maximum feasible conductance. However, only conductances in the range of $[g_{min}, g_{max}] \cup \{0\}$ can be fabricated. To guarantee that the values of all surrogate conductances are eventually fabricable, surrogate conductances that still have infeasible values after training, i.e. $|\theta| \in (0, g_{min})$, are projected to 0. After this step, all $|\theta| \in [g_{min}, g_{max}] \cup \{0\}$ and can be mapped to fabricable conductances in a printed NC. Alternatively, infeasible θ could also be mapped to either 0 or g_{min}, depending on which value is closer to $|\theta|$. However, zero-valued surrogate conductances are generally preferred, since they do not need to be printed and therefore decrease the required area, material and production time.

Unfortunately, projecting infeasible conductances to zero may likely also decrease the predictive quality of the pNN. To mitigate an eventual degradation, the learning procedure should be encouraged to avoid the region $(0, g_{min})$. This can be realized through an appropriate penalty function.

3.3.3 A penalty function to encourage feasibility

Penalty functions are commonly used to express (soft) constraints in optimization problems [115, Ch. 17]. The idea behind them is to increase the value of the objective function for infeasible or undesired regions and therefore discourage the optimisation procedure to find solutions there. In theory, a penalty function would be ideal if all infeasible or undesired regions exhibit objective function values higher than the objective function value of the worst feasible point. However, this is often not helpful in practice as the choice of the penalty function also influences the numerical procedure and thereby the success of finding good solutions.

Therefore, the infeasible regions, as well as the optimisation algorithm used, should be considered when designing a penalty function. Ideally, the penalty function not only penalizes objective function values in the infeasible region, but also helps to guide the optimization algorithm back into the feasible region when infeasible iterates are obtained. For gradient-based optimization, this may be realized by a penalty function which has gradients pointing into the infeasible set.[3] Through this, the iterates $\boldsymbol{\theta}^{(t)}$ are pushed away from the infeasible region, as long as the gradient of the loss function does not overwhelm the gradient of the penalty. Since the impact of the latter effect is difficult to assess, penalty functions usually receive an additional scaling factor to tune their contribution to the overall training objective.

In case of the infeasible domain of $|\theta| \in (0, g_{min})$, a penalty function can be constructed out of two hat functions.[4] The hat function are thereby placed in the middle of the infeasible domain with an appropriate width to cover the region $[-g_{min}, g_{min}]$ leading to

$$P(\theta) = \left(1 - \frac{|2\theta - g_{min}|}{g_{min}}\right)^{+} + \left(1 - \frac{|2\theta + g_{min}|}{g_{min}}\right)^{+}. \tag{3.1}$$

The smoothness at the boundary of the feasible domain can additionally increased by squaring the hat functions. An illustration of the resulting penalty functions can be seen in Figure 3.6.

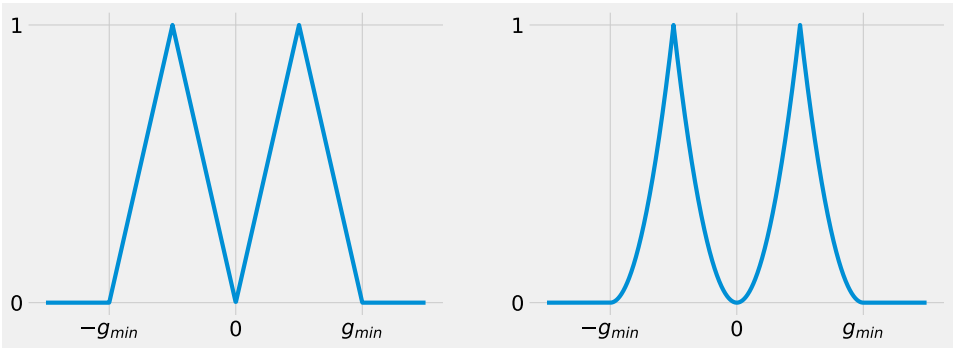


Figure 3.6: Penalty functions for avoiding the region $|\theta| \in (0, g_{min})$. The left displays (3.1), i.e. $P(\theta)$, while the right displays $P(\theta)^2$ for increased smoothness at $-g_{min}$, 0 and g_{min}.

By adding the sum of a penalty function for each surrogate conductance to the loss function, a combined training objective can be constructed, i.e.,

$$\min_{\boldsymbol{\theta}} \quad L(\boldsymbol{\theta}) + \frac{\lambda}{\dim(\boldsymbol{\theta})} \sum_{\forall \theta \in \boldsymbol{\theta}} P(\theta),$$

[3] Note that this is only the case for minimization tasks as the objective function decreases in the direction of the negative gradient. For maximization tasks, the gradient of the objective function should point into the feasible set.

[4] This construction is loosely inspired by penalty methods for weight quantization, see [152].

where $\lambda \in \mathbb{R}^+$ denotes a scaling parameter for the contribution of the penalty term. The length $\dim(\boldsymbol{\theta})$ of $\boldsymbol{\theta}$ is used to normalize the influence of a different number of parameters for different models. Ideally, the contribution of the penalty function should be scaled such that it does not influence the learning too much at the beginning, but encourages feasibility towards the end of the training when the loss function values become low. For this reason, λ should be considered a hyper-parameter that has to be tuned.

To conclude, by optimizing the combined objective under an appropriate choice of λ, the gradient-based learning procedure should avoid iterates $\boldsymbol{\theta}^{(t)}$ with entries in $(0, g_{min})$ while training. If this is successful, projecting infeasible values to zero should not significantly impact the final performance of the pNN.

3.3.4 Parameter initialisation for learning

As discussed in Section 2.5, the initialisation of the parameters can greatly influence the success of gradient-based learning. While initialisation close to optimal parameter vectors may trivially lead to a faster convergence, careful initialisation can also avoid unfavourable learning dynamics [55]. Such effects are common when the objective function contains sigmoidal-shaped components as they have saturation regions where gradients are close to zero. To avoid such regions in the initialisation, the weights are usually initialized at random and rescaled such that the weighted sum does not saturate the activation function at the beginning of the learning procedure [55]. Note that in modern artificial neural networks, such problems may also be addressed through normalisation operations such as *batch normalisation* [75]. However, since implementing eventual shifts from normalisation operations in printed NC would require additional circuitry, it should be avoided. To nevertheless improve the training dynamics, an initialisation strategy for the parameters of pNNs is proposed.

According to the crossbar constraints (see Section 2.6), the weights of a neuron sum up to a value smaller or equal to one. Hence, the weights formed through the surrogate conductances are always scaled down if more inputs are present. Through this, a weight-scaling similar to [55, Eq. (1)] is achieved naturally. The values of the surrogate conductances g_i can therefore simply be initialized uniformly around zero with a constant deviation. Since printed NCs can be expected to contain only a few neurons in each layer (e.g., ten), the mean of the conductances may not reflect the expected value of the distribution well and lie too far from zero. To address this, the values can be centered after initialisation by subtracting the empirical mean. Furthermore, since the surrogate conductance θ_0 (representing g_0) is mainly used for decoupling the weights (see Section 2.6), it should be initially set to the highest possible value $\theta_0 = g_{max}$ in order to allow for maximum decoupling.

Finally, the initialisation of the surrogate conductance for the bias g_b plays an important role due to the characteristics of the ptanh activation function. It can be seen that $\text{ptanh}(0) \approx -0.4$, and $\text{ptanh}(\text{ptanh}(0)) \approx -0.95$ (see Figure 3.7). This means that for the initially small weights around zero, the activations of neurons in subsequent layers will quickly drift into the saturation region of the activation function. However, this region is unfavourable for gradient-based learning, as all inputs produce similar outputs and the calculated gradients may be small (see [56, Sec. 8.4]). Hence, gradient-based learning may progress very slowly.

To mitigate this effect for the initial training steps, the input independent bias b can be set to the root of ptanh initially. Here, the derivative of ptanh is the highest, and the distance to

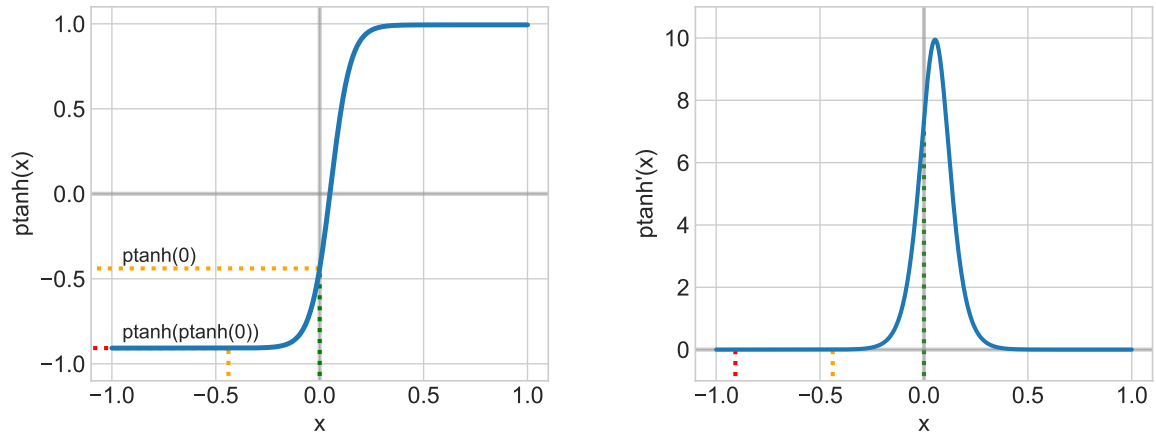


Figure 3.7: The ptanh activation function (left) and its derivative (right). It can be seen that for an activation close to zero, the activations of subsequent layers strongly drift towards the negative region. Here, gradient-based learning may progress slowly since the gradients will be close to zero.

the saturation region is maximized due to the point-symmetry of the tanh. Since the bias b cannot be chosen directly but depends on all conductances of the respective neuron, θ_b must be set accordingly instead. Assuming that the other θ_i have been initialized already, this can be achieved by first reformulating the equation of b as a function of θ_b through

$$\begin{aligned} b = \frac{|\theta_b|}{\sum_i |\theta_i| + |\theta_b|} &\iff |\theta_b| = b \sum_i |\theta_i| + b|\theta_b| \\ &\iff |\theta_b| - b|\theta_b| = b \sum_i |\theta_i| \\ &\iff (1-b) \cdot |\theta_b| = b \sum_i |\theta_i| \\ &\iff |\theta_b| = \frac{b}{1-b} \sum_i |\theta_i|. \end{aligned}$$

Since $b \overset{!}{=} \text{ptanh}^{-1}(0) > 0$, θ_b must be positive. The initialisation of the surrogate conductance is thus given by

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

Through this choice of θ_b in conjunctions with $\theta_0 = g_{max}$ and θ_i initialized small and uniformly around zero, the initial neuron activations should be closely distributed around zero and benefit the training dynamics at the beginning of the training process.

3.4 Experiments

To evaluate the learning performance and expressiveness of the pNN models, they are trained on several small classification tasks from the UCI machine learning repository [42]. In total, nine datasets (eleven classification tasks) are considered. Namely *Acute inflammation* [34], *Breast Cancer Wisconsin*, *Energy efficiency* [144] (two tasks), *Iris* [4, 54], *Balance Scale*, *Seeds* and *Vertebral Column* (two tasks). All tasks are learnable with small models and may be comparable in difficulty to simple processing tasks. Since not all tasks can be solved with 100% accuracy, the results of the learned pNNs are compared to a random guess baseline (most frequent training data class) and a standard (hardware-agnostic) neural network implementation of the same architecture. This network is referred to as reference NN. For the reference NN, a standard tanh activation function is used instead of the ptanh activation function. Note that for the reference NN, as well as for the pNN, activation functions are applied to the output.

For all datasets, the processed versions of the data provided in [53] were used and the features were normalized to a range of $[0, 1]$. Furthermore, train-test-splits with 66% of the data for training and 33% for testing were performed.

3.4.1 The evaluation metric

Generally, classification tasks are most naturally evaluated using the accuracy metric, i.e., the fraction of correctly classified instances. However, due to the assumed measuring limitations, the aspect of signal discrimination also needs to be taken into account. The trained models are therefore evaluated with a more strict version of the accuracy called the Measuring-aware Accuracy (MaA) in the following. Assuming the class variable y is encoded as natural numbers $y \in \mathbb{N}$, the MaA is defined as

$$\mathrm{MaA}(f_{\boldsymbol{\theta}}(\cdot), \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \mathbb{1}_{\{\mathrm{i}==\mathrm{y}\}} \cdot \mathbb{1}_{\{\mathrm{f}_{\boldsymbol{\theta}}(\mathbf{x})_\mathrm{y} \geq \mathrm{T}\}} \cdot \mathbb{1}_{\{\forall \mathrm{j}\neq\mathrm{i} \mid \mathrm{f}_{\boldsymbol{\theta}}(\mathbf{x})_\mathrm{j} \leq 0\}},$$

where $i = \mathrm{argmax}_j\, f_{\boldsymbol{\theta}}(\mathbf{x})_j$ and $\mathcal{D}$ denotes the evaluation data.

The first component, $\mathbb{1}_{\{\mathrm{i}==\mathrm{y}\}}$, expresses if the output that represents the label y returns the highest value and relates to the standard accuracy. Besides the correct classification, the MaA also requires the correct output of the network to exceed the measuring threshold T through $\mathbb{1}_{\{\mathrm{f}_{\boldsymbol{\theta}}(\mathbf{x})_\mathrm{i} \geq \mathrm{T}\}}$, and all other outputs have to return values below zero which is expressed by $\mathbb{1}_{\{\forall \mathrm{j}\neq\mathrm{i} \mid \mathrm{f}_{\boldsymbol{\theta}}(\mathbf{x})_\mathrm{j} \leq 0\}}$. The proposed evaluation metric is thus more strict than the standard accuracy and takes the output signal strength and separation into account. Consequently, the achieved accuracy of a neural network may be higher than its MaA.

Note that the MaA, just like the accuracy, can provide misleading results about the model performance when it comes to strongly imbalanced datasets. For example, on a dataset with two classes and a $9:1$ ratio, a 90 % accuracy can already be achieved by always guessing the majority class. It is thus important to always consider the accuracy (or the MaA) in relation to the accuracy achieved by such simple strategies. Nevertheless, while accuracy values of zero would be extremely uncommon by random chance, MaA values of zero may be frequently observed due to an insufficient signal separation. Beating the accuracy of a random guess baseline is therefore usually simple and expected, while beating such a baseline on the MaA should happen less frequently by random chance.

3.4.2 Hyper-parameter configurations and learning

To find good networks for each task, a grid-search over the initial learning rate α, the penalty coefficient λ and several seeds is performed. All networks are trained for 200 (full-batch) steps and the learning rate is halved every 50 epochs. For all experiments, the same architecture $\#\text{inputs} - 4 - 3 - \#\text{classes}$ neurons with activation functions after each layer (including the output) is used.

Training runs are canceled if the training MaA did not exceed the baseline of the dataset after 50 epochs, or the training MaA did not improve over 20 consecutive updates (see *early stopping* [56, Alg. 7.1]). After training all models, the best model is selected based on the maximum training MaA after projecting the parameters to feasible values. As the penalty function, (3.1) is chosen. The initial learning rate α, penalty coefficient λ and margin m are chosen as one of $\alpha = \{0.01, 0.1, 1\}$, $\lambda = \{0, 0.001, 0.01, 0.1\}$ and $m = \{0.0, 0.1, \cdots, 0.9, 1.0\}$ respectively. Additionally, all experiments are run with 10 different random seeds leading to a total of $3 \times 4 \times 11 \times 10 = 1320$ configurations for each dataset. Since the reference network does not have any infeasible regions and thus has no need for the penalty function of (3.1), λ is used as a parameter for *weight decay* (see [56, Sec. 5.2.2]) instead.

For the pNNs, the surrogate conductances θ_i, are drawn uniformly from $\mathcal{U}[-0.01, 0.01]$ and θ_0 and θ_b are initialized as described in Section 3.3.4. The surrogate conductance θ_0 is fixed to $\theta_0 = g_{max}$ during training to allow for maximum decoupling of the weights throughout the entire training process. The parameters of the reference network are initialized according to the *pytorch* [117] default for linear layers.

For better numerical stability, the values of the surrogate conductances are normalized to a range of $[0.01, 1]$ representing a range of feasible resistances from, e.g., 10MΩ to 100kΩ. Additionally, the measuring threshold is set to $T = 0.1$ relating to 100mV. For the parameter updates (optimizer) Adam [84] is used with the parameter configuration recommended by the authors ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$). All neural networks are implemented using the *pytorch* [117] framework.

3.4.3 Benchmark results

The MaA results for all datasets can be seen in Table 3.1, and the hyper-parameter configurations of the best models can be found in Table 3.2.

The first, main observation is that all pNNs surpass the baseline result. Secondly, almost all pNNs reach an overall comparable performance to the reference networks or even surpass them on some tasks. Furthermore, projecting infeasibly small conductances does only lead to a mild degradation in the MaA [see pNN versus pNN (feasible)]. Especially for printed networks trained with a higher values of $\lambda \geq 0.1$, the projection leaves the results almost unchanged, see *Seeds*, *Tic-Tac-Toe Endgame* and *Mammographic Mass*. The only exception to this is *Energy efficiency (y1)*. However, not all best performing networks chose high penalty terms. This indicates that the penalty term may also hinder the learning process in some cases and not lead to the overall best fabricable (after projecting infeasibly small g to zero) results. The penalty coefficient λ should therefore always be chosen through a grid-search routine. The same is true for the parameter m for which an overall wide range of values was chosen by the best networks (see Table 3.2).

Finally, note that overall better results might be achievable through training for more epochs or choosing the network architecture on a per dataset basis. Also, a more careful hyper-parameter tuning procedure may lead to better results, as some of the best networks chose hyper-parameter settings on the boundary of the provided ranges, such as $\alpha = 1$ and $\lambda = 0$. This is even more prominent for the reference network, where 8 of the 11 networks chose a learning rate of 1. Hence, choosing an extended range for the learning rate may improve the results.

Dataset	Architecture	pNN		pNN (feasible)		reference NN		random guess
	neurons/layer	Train	Test	Train	Test	Train	Test	
Acute Inflammations	6-4-3-2	1	1	1	1	1	1	0.475
Balance Scale	4-4-3-3	0.9354	0.8986	0.9354	0.9034	0.9163	0.8889	0.4396
Breast Cancer Wisconsin	9-4-3-2	0.9765	0.9697	0.9808	0.9697	0.9765	0.9697	0.6667
Energy efficiency (y1)	8-4-3-3	0.8794	0.878	0.8813	0.874	0.8696	0.8583	0.4331
Energy efficiency (y2)	8-4-3-3	0.9047	0.9094	0.9163	0.9094	0.9105	0.9055	0.4646
Iris	4-4-3-3	0.97	0.96	0.98	0.94	0.97	0.98	0.28
Mammographic Mass	5-4-3-2	0.8351	0.8365	0.8351	0.8365	0.818	0.8113	0.5503
Seeds	7-4-3-3	0.95	0.9429	0.95	0.9429	0.9571	0.9571	0.2714
Tic-Tac-Toe Endgame	9-4-3-2	0.9938	0.9716	0.9938	0.9716	0.9922	0.9716	0.6404
Vertebral Column (2 classes)	6-4-3-2	0.8599	0.8738	0.8599	0.8738	0.8551	0.8835	0.6893
Vertebral Column (3 classes)	6-4-3-3	0.7874	0.8447	0.7874	0.835	0.7681	0.7864	0.5146

Table 3.1: The benchmark results of several trained pNNs. For each dataset, a grid-search was performed over different initial learning rates α, seeds, penalty function coefficients λ and margins m. Note that only the projected pNNs could be fabricated.

Models	pNN			reference NN		
Parameters	α	m	λ	α	m	λ
Acute Inflammations	1.00	0.7	0.000	1.0	0.1	0.000
Balance Scale	0.10	0.4	0.010	1.0	0.7	0.000
Breast Cancer Wisconsin	0.10	0.5	0.001	1.0	0.6	0.000
Energy efficiency (y1)	0.10	0.5	0.100	1.0	0.4	0.001
Energy efficiency (y2)	0.01	0.3	0.010	1.0	0.7	0.001
Iris	0.01	0.9	0.001	1.0	0.7	0.010
Mammographic Mass	0.10	0.9	0.100	1.0	1.0	0.001
Seeds	0.01	0.5	0.100	1.0	0.5	0.001
Tic-Tac-Toe Endgame	0.10	0.7	0.100	0.1	0.7	0.000
Vertebral Column (2 classes)	0.10	0.7	0.100	0.1	0.7	0.000
Vertebral Column (3 classes)	0.01	0.1	0.001	0.1	0.9	0.000

Table 3.2: The parameter configurations of the best neural networks.

To conclude, the proposed training procedure allows to find pNNs that achieve similar performances to classical neural networks on a set of small benchmark datasets. Hence, also simple, near-sensor processing tasks, as they may come up in IoT applications, could be performed using printed NCs based on pNNs.

3.5 Conclusion and directions for future work

The unique feature of on-demand fabrication of printed circuits allows to fabricate them at point-of-use. However, designing circuitry in printed electronics is still challenging. To tackle this problem, this chapter proposes the use of printed NCs to obtain on-demand designs of printed circuits through an automatic design approach. For this purpose, pNNs were developed as a model for printed NCs. A pNN model can then be used to learn designs of printed NCs through gradient-based optimisation. After learning, the obtained design can be mapped to the respective hardware primitives of printed NCs and readily printed. Through this procedure, a lean, on-demand design solution for printed circuits is realized. Aside from the benefits of lean design, neuromorphic computing offers other, additional benefits such as native analog computing. Furthermore, the inherent parallelism of the computation may help to mitigate problems of slow processing speeds in printed electronics. The proposed learning algorithm for printed NCs was evaluated by training pNNs on several small benchmark datasets of classification tasks. The trained models achieved comparable performance to hardware-agnostic networks of similar architecture.

Several aspects of the proposed procedure may hold potential for improvements. One of these would be to directly consider the projection of infeasibly small conductance values (to zero) at training time. Through this, the penalty term and the tuning of its associated hyper-parameter could be omitted. In turn, this could encourage more zero-valued conductances in training, which would save production time and material when fabricating the respective printed NC. Unfortunately, gradient-based learning cannot be directly applied in such cases, as there is a whole region for which no useful gradient information could be obtained (the gradient would be zero in the entire region). To address this issue, heuristics like the straight-through estimator [9] could be employed. Here, the network is subjected to the projection in the forward pass, while the backward pass for the parameter update ignores the projection and treats it as an identity. Even though this estimator provides biased information of the gradient, it has been shown to work well in practice, see [146].

In summary, the proposed learning procedure finds a set of conductance values with which the desired functionality can be realized. Regarding automatic circuit design, this could be seen as component sizing with only limited aspects of topology selection, i.e., to connect inverted or noninverted inputs. However, the choice of a suitable neural network architecture for a given task was left to the designer for now. Unfortunately, this choice is seldom straightforward and influences the achievable accuracy and success of training. Additionally, in the case of printed NCs, the network architecture directly influences the required production time, material and area requirements. Hence, finding suitable network architectures is even more important for printed NCs and should be addressed in the future. A promising approach for this may be to adopt techniques from the field of neural architecture search, see [45]. Here, a wide range of techniques have been proposed to find good topologies for classical neural networks and some may only require little modification to be applied to pNNs. Furthermore, other desired properties of the pNNs (and/or the printed NCs) may be additionally used to guide the network architecture search. For example, trying to find the smallest and most energy-efficient topology that achieves a certain accuracy on a given task.

Beyond that, the on-demand fabrication capabilities of inkjet printing could allow even more flexibility in the design of the printed NC. Since all devices in the NC can be fabricated

independently, also the design space of the devices (resistors and transistors) of the activation and inverter function could be included in the training process. This would allow customized and learnable activation and inverter functions for each neuron. For this purpose, the pNN model equations would have to be extended with parameterized models for these circuits, i.e., models of the output given the parameterization of the components (resistors and transistors) and the input.

Aspects such as the power consumption may also be considered, either directly in training by encouraging low conductance values, or in a post-training step. Here, the fact could be leveraged that the weights of the pNNs only depend on the ratio of the conductances. Hence, scaling down all conductances leaves the weights unchanged. For example, the smallest conductances of a neuron could be directly scaled down to the minimum feasible value. This approach would be most effective if all nonzero conductances have similar values. The occurrence of such configurations should thus be encouraged through appropriate penalty terms.

Aside from sizing and topology optimization, layout generation, i.e., placement and routing of the circuit components is also an aspect that should be considered for the on-demand realisation of printed NCs. Since pNNs are based on densely connected artificial neural networks, they display a homogeneous structure and exhibit high connectivity. As the circuits can only be routed on a single plane [127], printing wire cross-overs is necessary and many overlapping connections can be expected. Unfortunately, these wire cross-overs influence the conductances of the connections. While general approaches for routing and placement of inkjet-printed circuits with cross-overs have been proposed, see [127, 128], the homogeneity of the components and the layer-wise connectivity of neural networks could be exploited to develop a simpler, knowledge-based solution for printed NCs. For example, the conductances that can be anticipated from cross-overs could be attributed to the conductances of the crossbar resistors. These may then be sized down appropriately.

Finally, one of the major problems of printed electronics is its high variations [24]. These variations may also strongly influence the accuracy achieved by printed NCs. A classic way to deal with variations is to consider a quantization to somewhat reliably fabricable values (as in digital computing). This however also decreases the expressiveness of the design and usually requires a higher device count to achieve similar functionality. As an alternative, Chapter 5 investigates techniques to improve the robustness of pNNs to variations of the conductances and circuit components. However, first, a framework for modelling variation in printed components such as the activation function and the inverter function is required. The development of such a framework is the topic of the next chapter.

4 A Framework for Data-driven Variation Modelling

One of the major problems of fabricating printed circuits is their large variations [24]. It is therefore imperative to consider these variation when designing printed circuitry. As a first step, variation models for printed devices need to be developed to simulate and validate designs. The common practice for developing such variation models is to collect a set of devices or circuits, for example transistors, and extract their parameters using a device model. Then, the data of the extracted parameters of the different devices is approximated through a normal distribution, see [133]. Even though this is often sufficient for silicon-based technologies, where variations can be attributed to specific effects based on the underlying physics, it may fail to capture aspects of variations in other technologies such as printed electronics [126].

In printed electronics, the fabrication process, and therefore the sources of variations [139], are fundamentally different. The interactions between these unique sources of variation may manifest in complicated correlation structures and non-classical distribution shapes in the device model parameters [126]. Furthermore, models of printed devices, e.g. [100, 125, 58], frequently contain fitting parameters whose relationships to the other device model parameters are often not understood. Their variations can thus only be expressed on a purely empirical, data-driven basis.

To address these challenges, this chapter proposes a flexible modelling framework for developing a variation model based on device measurements. The described framework extends the classical modelling flow, see [133], by a procedure for dealing with fitting parameters through regression models and providing a data-driven approach for the derivation of a variation model. Through this added flexibility, the framework is able to model various distribution shapes and correlation structures that may be observable in emerging technologies such as printed electronics.

The remainder of this chapter is organized as follows: First, a background on classical variation modelling and the additional challenges for modelling variation for printed electronics are discussed. The discussion is followed by a brief review of the core methodological concepts of variation modelling. Then, the proposed framework is introduced. It consists of three steps relating to data transformations, parameter reduction via regression and distribution modelling using Gaussian mixture models. To evaluate the methodology, a variation model of an inkjet-printed transistor is derived. The extracted model is then evaluated by comparing the distribution of key performance parameters to those of fabricated and measured devices. To evaluate the usefulness of the model on the circuit level, measurements of printed physically unclonable function circuits are compared to their simulation results.

The results and methods discussed in this chapter are based on collaborative work that led to [126] and have also partially been reported in [124, Sec. 3.4].

4.1 Preliminaries and background

Modelling and assessing variations is integral for optimizing designs and increasing manufacturing yields. However, simulating the effects of process variation and closely reflecting them in design automation tools is challenging [15, 133]. This is not only true for modern VLSI technologies [88, 89, 122], but even more so for less mature, emerging technologies such as inkjet-printed electronics [24]. In silicon-based technologies, process variations are divided into local and global variations [119]. All variations from lot-to-lot, wafer-to-wafer or chip-to-chip apply to all transistors and are therefore referred to as global variations. The with-in chip variations are referred to as local variations [79, 92]. Many modelling and design approaches therefore directly address these specific characteristics, see [129, 134, 96].

In inkjet-printed electronics, sources of variations cannot be easily divided into local and global variations. This is due to the characteristics of the manufacturing process. Here, all devices are manufactured (printed) individually in multiple additive process steps. In each of these steps, random variations may originate from variations in the ink, substrate or manufacturing tools influencing the dispersion of the ink on the substrate. Additional sources of variations include droplet jetting oddness, satellite drops, wetting and missing droplets (details see [139]). These unique characteristics severely limit the applicability of classical variation modelling approaches that specifically target silicon-based manufacturing characteristics [126].

Another aspect that complicates variation modelling for such emerging technologies is the lack of understanding of the device physics. This often only permits semi-empirical models employing fitting parameters, see [100, 125]. Since the fitting parameters are derived through a deterministic procedure (fitting) given the other parameters, it is often reasonable to assume that variations may be attributed to variations in the other (physical) parameters. However, the concrete relationship may be complicated.

Furthermore, depending on the requirements for their specific envisioned application domain and environment, e.g. IoT [130, 24] or biomedical applications [39], printed electronics may be fabricated using a wide range of different ink compositions and substrates. Hence, methodologies for variation modelling of printed electronics need to be general enough to address various correlation structures induced by the interaction of different materials.

Altogether, these characteristics suggest the use of an empirical variation modelling procedure. The procedure should be flexible enough to address various distribution characteristics and correlation structures among the device parameters. Furthermore, it should be able to adequately deal with fitting parameters present in semi-empirical device models.

4.1.1 Related work on variation modelling

There are two main types of variation models used to assess the robustness and yield of designs, namely corner models and Monte Carlo models [133].

Corner models characterize typical, best-case and worst-case behaviour (corner-cases). These corners may thereby be extracted through, e.g., computing extreme values of for performance parameters or using fixed-sigma variations [22]. Unfortunately, especially fixed corner models are often not very accurate and make overpessimistic predictions. In turn, these inaccuracies, in combination with simplifying assumptions, often lead to unnecessarily pessimistic design-

choices [105]. Nevertheless, due to corner models being quick to evaluate, they are often employed in practice.

On the other hand, Monte Carlo models try to estimate the probability distribution of the device model parameters. These estimated distributions can then be used to directly approximate the distribution of performance metrics or estimate the failure rate of designs. Note that a Monte Carlo model can be used to estimate any performance metric of interest. This makes them more flexible and more general than corner models. In fact, Monte Carlo models can be used to generate corner models by characterising the respective corners through the probability distribution, e.g., their 3-σ behaviour. However, performing a Monte Carlo analysis is generally more involved than using corner models.

A typical approach to the development of a Monte Carlo model is carried out as follows: Initially, a set of device model parameters sensitive to process variation are determined [134]. Then, the empirical distribution of these sensitive parameters is obtained by measuring the current-voltage (or capacitance-voltage) characteristics of several fabricated devices. Based on the measurements, a distribution is fitted for each parameter [133]. Common choices to model the device model parameters are thereby uniform or normal distributions [122, Sec. 1.2], see [112, 134]. However, these distributions may not always reflect the characteristics of the parameters well, and [135] also used the exponential, log-normal and Lorenz (Cauchy) distribution to model parameters extracted from organic thin film transistors. Using different distributions, e.g., normal and log-normal for characteristics at different voltage regions was also suggested [138]. In [129], multivariate normal distributions, along with data transformations such as the logarithm or Box-Cox transformations, were employed. Whether to apply a transformation was decided based on a test for normality.

Since often only a few device model parameters are sensitive to variations [134], efforts were made to automatically reduce the number of parameters needed to express the observed variations through parameter reduction techniques. Here, approaches like principal component analysis [142, 26] or step-wise backward selection techniques [122, Sec. 4.2] have been employed. These methods eliminate parameters that do not contribute significantly to the observed variations.

To address the aforementioned challenges in variation modelling for printed electronics, the use of simple distribution models may not be sufficient [126]. Hence, an overall more flexible modelling procedure is required. The procedure should be able to deal with fitting parameters in semi-physical models and allow to model sufficiently complex distribution shapes.

4.2 A generalized variability modeling framework

In this section, a generalized, three-step variability modelling approach is described. Starting with a dataset of measured current-voltage characteristics of devices or components, device model parameters are extracted, see [125]. Then, in the first step, an appropriate transformation for the model parameters is chosen in order to prevent infeasible values. Furthermore, the data is standardized to achieve unitless quantities. In the second step, the set of device parameters are divided into two disjoint subsets representing the independent parameters (that induce variation) and the dependent parameters (empirical fitting parameters). Variations observed in the dependent parameters are then directly attributed to variations in the independent parameters via a regression model. In the third step, a joint probability density of the independent device parameters is estimated. For a flowchart of the complete methodology see Figure 4.1.

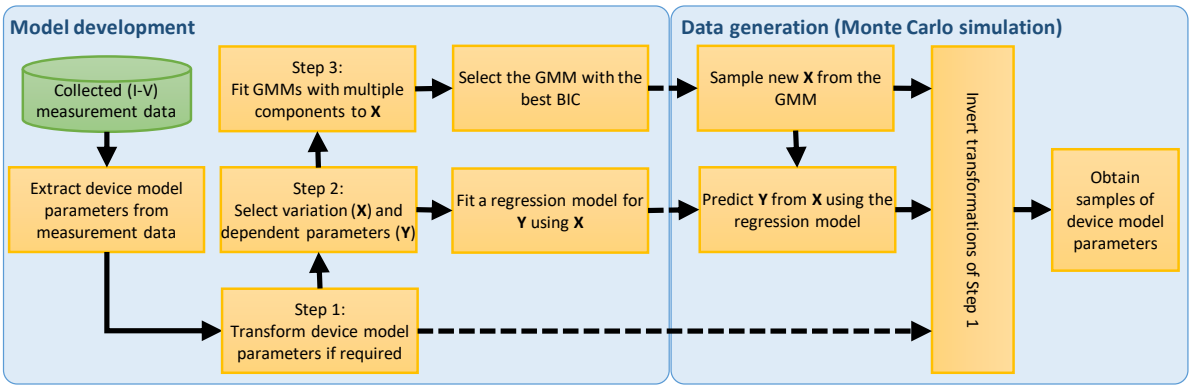


Figure 4.1: The complete flow of model development and Monte Carlo simulation.

After these three steps, the model development process is completed and the model can be used for data generation and simulation. For data generation, samples of the independent parameters can be drawn by sampling from the Gaussian Mixture Model (GMM). The drawn samples can then be used to predict the dependent parameters. The transformations applies in the first step, e.g. standardisation, are inverted to obtain samples from the original distribution. These samples can then be used to approximate various device characteristics such as statistical moments or probabilities for the violation of specifications (e.g., failure rate estimation) through Monte Carlo methods.

4.2.1 Parameter transformations

Many of the device model parameters reflect physical quantities which have a naturally bounded value range. For example, while threshold voltages of transistors can be positive or negative, conductance values can only be nonnegative. To respect such constraints in modelling, one of two approaches can be used.

The first approach would be to model the distribution of each device model parameter by an appropriate distribution that respects its value range, see [135]. Even though the individual choice of distribution can often be motivated by physical knowledge, deriving a joint, nonfactorizing[1] probability distribution can be challenging. However, jointly modelling the parameters

[1] A joint probability density $p(a, b)$ factorizes if the individual random variables are statistically independent, i.e., $p(a, b) = p(a) \cdot p(b)$.

is necessary to express possible correlations between them.

Alternatively, the parameters can be transformed through suitable (invertible) functions before modelling their joint distribution, see [129]. In this case, the applied transformations are inverted after sampling a set of parameters. The restricted inverse images of the transformation should then guarantee valid value ranges. The expressiveness of this approach is limited by the complexity of the applied transformations and the distributions used. In this work, only basic transformations will be employed, while a flexible GMM (see Section 4.2.3) approximates the joint density of the transformed variables.

To give an example of a commonly used transformation, consider a positive valued parameter z. By modelling $\ln(z) \sim \mathcal{N}(\mu, \sigma^2)$ and then inverting the $\ln(\cdot)$ transformation for the sampled values through $\exp(\ln(z))$, only positive samples will be generated. Another useful transformation, that is however not used in this work, is the sigmoid function $(1+\exp(-x))^{-1}$. It can be used to achieve values in the range of $(0, 1)$ or any other bounded interval by shifting and scaling the range appropriately. Hence, choosing appropriate transformations allows to express bounds on the sampled values.

When the probability distribution of multiple physical quantities of different units should be estimated jointly, it is generally beneficial to standardize (or normalize) each individual variable. To standardize a variable z, the (empirical) mean μ_z of the variable is subtracted from z and the result is divided by its (empirical) standard deviation σ_z, i.e.,

$$z_{standardized} = \frac{z - \mu_z}{\sigma_z}.$$

Standardisation removes the unit or scale of a variable and improves the stability[2] of the numerical operations applied in the following. Note that the standardisation can be readily inverted by multiplying the result by the standard deviation σ_z and adding the mean μ_z. Through this, the original scale/unit of the variable can be restored.

4.2.2 Reducing the number of variation parameters through regression

Generally, all device model parameters are required for a complete variation model. However, it is usually assumed that only a smaller number of parameters/effects cause the observed variations [134]. Furthermore, purely empirical fitting parameters in device models should usually not be considered for variation modelling as their values are deterministically calculated given the values of the other parameters (in the fitting procedure).

To limit the parameters responsible for the variation and avoid having to model fitting parameters, the device model parameters are divided into two disjoint subsets. On the one hand, the (independent) variation parameters denoted by $\mathcal{X}$, and on the other hand the dependent parameters $\mathcal{Y}$. The set $\mathcal{X}$ denotes the parameter for which variations should be modelled, while the variations in $\mathcal{Y}$ (dependent parameters) should be explained by variations in $\mathcal{X}$. In other words, only the variables $\mathcal{X}$ are considered to be responsible for the observed variations. The assignment of the device model parameters to these two sets depends on the specific device

[2] Consider two variables that should be modeled in a multivariate normal distribution. If the scales of the variables are vastly different, a high condition number [143, Sec. 12] of the covariance matrix can be expected. This can lead to numerical problems when the matrix has to be inverted, for example, when evaluating the probability density function of a multivariate normal distribution.

and should be left to the user. However, pure fitting parameters would be classical candidates for dependent variables $\mathcal{Y}$.

To obtain the values for the variables $\mathcal{Y}$ given the variables $\mathcal{X}$, a regression model for each parameter in $\mathcal{Y}$ can be fitted. Since the regression model should be expressible in common simulation languages (e.g., Verilog-A or SPICE), a linear regression model, see [13, Ch. 3], can be used for simplicity.

Assuming a given dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ of (possibly transformed) device model parameters with vectors $\mathbf{x}_n$ (from $\mathcal{X}$) of independent device model parameters and a dependent parameter y_n (from $\mathcal{Y}$), linear regression tries to approximate the values of y_n by a linear combination of basis function $\psi_i(\mathbf{x})$ of $\mathbf{x}_n$. Given a set of basis functions $\boldsymbol{\psi}(\mathbf{x}) = [\psi_1(\mathbf{x}), \psi_2(\mathbf{x}), \cdots]^\top$ and a vector of coefficients $\mathbf{w} = [w_1, w_2, \cdots]^\top$, the linear regression model can be expressed by $\boldsymbol{\psi}(\mathbf{x})^\top \mathbf{w}$. Note that even though the function $\psi(\mathbf{x})^\top \mathbf{w}$ is linear in $\mathbf{w}$, it is not necessarily linear in $\mathbf{x}$. To see this, consider the following simple example. Let $\mathbf{x} = [x_1, x_2]^\top$ and let $\boldsymbol{\psi}(\mathbf{x})$ be the function generating all monomial combinations of the arguments $\mathbf{x}$ up to degree two. Then, $\boldsymbol{\psi}(\mathbf{x})$ generates the basis functions $\boldsymbol{\psi}(\mathbf{x}) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^\top$. In this case, the linear regression model is able to express all polynomials of the inputs x_1 and x_2 up to degree two.

To find the optimal vector of coefficients $\mathbf{w}^\star$, the sum of squared errors between the linear regression model $\boldsymbol{\psi}(\mathbf{x}_n)^\top \mathbf{w}$ and the respective y_n is minimized with respect to the parameters $\mathbf{w}$ for all data points $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, i.e.,

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \quad \sum_{n=1}^{N} \Big(\boldsymbol{\psi}(\mathbf{x}_n)^\top \mathbf{w} - y_n \Big)^2 .$$

By using more basis function $\psi_i(\mathbf{x})$, the expressiveness of the model can be increased. However, this also requires more coefficients w_i. Often, not all basis functions provided to the model are required, and using too many basis functions can quickly lead to over-fitting [13, Sec. 1.1]. It is therefore beneficial to select only a relevant subset of the basis functions. This can be achieved through adding an additional penalty term to the optimization problem and solving

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \quad \sum_{n=1}^{N} \Big(\boldsymbol{\psi}(\mathbf{x}_n)^\top \mathbf{w} - y_n \Big)^2 + \lambda \|\mathbf{w}\|_1,$$

where $\lambda \in \mathbb{R}^+$ is a user defined parameter adjusting the strength of the penalty. By penalizing the absolute values of the coefficients, finding sparse solutions $\mathbf{w}^\star$ with zero-valued coefficients for less important basis functions is encouraged. This allows the number of basis functions to be chosen less conservatively, since basis functions that do not contribute significantly to the model quality will obtain zero-valued coefficients. The latter formulation of linear regression is also known as lasso (least absolute shrinkage and selection operator) regression [64, Sec. 3.4.2]. Note that the independent variables $\mathcal{X}$ should be standardized before using lasso regression, otherwise, variables with overall smaller values will naturally require higher coefficients $\mathbf{w}_i$ compared to variables with higher values. This would in turn lead to them getting penalized stronger irrespective of their predictive power for $\mathcal{Y}$.

To find a set of suitable basis functions $\psi(\mathbf{x})$ and a penalty parameter λ, a grid-search[3]

[3] Grid-search refers to an exhaustive search over all combinations of a given set of options. In case of the lasso regression model, this could for example include trying out all combinations of a set of different basis functions $\psi(\mathbf{x})$ and penalty parameter values λ.

procedure in combination with cross-validation [64, Sec. 7.10] can be used. While the original linear regression problem can be solved analytically through the normal equations, solving the lasso problem is not straightforward. Fortunately, software packages, such as [118], offer efficient solvers.

Through regression models, all dependent variables can be expressed as functions of the independent variables $\mathcal{X}$. Hence, all variations in the final model will only be induced by variations in the independent variables $\mathcal{X}$.

4.2.3 Variation modelling using Gaussian mixtures models

Emerging technologies may display various types of correlations between their device parameters. Accurately modelling such effects thus calls for a model that is able to support a wide range of possible correlation structures and distribution shapes [126]. Additionally, to support the integration into simulation frameworks, the implementation of the variation model should require only basic building blocks supported by commonly used simulation languages such as Verilog-A or SPICE.

To achieve this, a GMM [13, Sec. 9.2] is used to represent the joint probability density function of the (transformed) independent variation parameters $\mathcal{X}$. A GMM consists of multiple normal distributions (components) that are combined to a joint distribution in a convex combination. By combining a sufficient number of such components, almost any smooth density can be approximated with arbitrary accuracy [13, p. 111].

Formally, the probability density function of a GMM with K components is given by

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi^{(k)} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)}) \quad \text{with} \quad \sum_{k=1}^{K} \pi^{(k)} = 1 \quad \text{and} \quad \pi^{(k)} \geq 0 \quad \forall k = 1, \cdots, K, \quad (4.1)$$

where $\{\boldsymbol{\mu}^{(k)}\}_{k=1}^{K}$ is the set of all mean vectors, $\{\pi^{(k)}\}_{k=1}^{K}$ is the set of mixing coefficients and $\{\boldsymbol{\Sigma}^{(k)}\}_{k=1}^{K}$ is the set of covariance matrices of the components. It can be readily verified that $p(\mathbf{x})$ is a valid probability distribution as all components $\pi^{(k)}\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)})$ are nonnegative and integrate to $\pi^{(k)}$. Consequently, as the $\pi^{(k)}$ are also nonnegative and sum to one, $p(\mathbf{x})$ integrates to one.

To obtain a GMM that models a desired dataset $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^{N}$ of variables $\mathcal{X}$, an appropriate set of parameters $\boldsymbol{\theta} = \{\pi^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)} \mid k = 1, \cdots, K\}$ needs to be found (estimated).

Maximum likelihood estimation for Gaussian mixture models

A classical way to estimate the parameters of a probability density function is the maximum likelihood approach. In maximum likelihood estimation, the set of distribution parameters $\boldsymbol{\theta}$ is chosen such that the probability of observing the data $\mathcal{D}$ given $\boldsymbol{\theta}$ is maximized. A common assumption is thereby that the data is independent and identically distributed given the parameters $\boldsymbol{\theta}$. Furthermore, instead of maximizing $p(\mathcal{D}; \boldsymbol{\theta})$, $\ln p(\mathcal{D}; \boldsymbol{\theta})$ is maximized for computational

reasons.[4] Thus, the optimal parameters are chosen as

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \quad \sum_n \ln p(\mathbf{x}_n; \boldsymbol{\theta}). \tag{4.2}$$

While an analytical maximum likelihood estimate can be found for some distributions, e.g., the normal distribution, (4.2) cannot be readily solved for a GMM [13, p. 440]. This is because for each data point, only the location, but not its respective assignment to one of the K components is known. Since this information is required in order to estimate $\boldsymbol{\theta}$, it represents a latent variable in the estimation procedure and has to be estimated alongside the other parameters.

One approach to solve maximum likelihood estimation problems with latent variables is to use the expectation maximization algorithm, see [13, Sec. 9.3]. The algorithm consists of two main steps, namely the expectation (E) and the maximization (M) step, that are iteratively repeated until convergence. In the E-step, the latent variable, expressing the assignment of each data point to a corresponding component, is estimated. Based on the estimated assignment, the parameters $\boldsymbol{\theta} = \{\pi^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)} \mid k = 1, \cdots, K\}$ are updated in the M-step.

For GMMs, the E and M step are given by the following equations [13, pp. 438-439]:

- **E-step**: Calculate the responsibilities $c_n^{(k)}$ for each of the data points $\mathbf{x}_n$, $n = 1, \cdots, N$ given the current parameters $\pi^{(k)}$, $\boldsymbol{\mu}^{(k)}$ and $\boldsymbol{\Sigma}^{(k)}$ as

$$c_n^{(k)} \leftarrow \frac{\pi^{(k)} \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)})}{\sum_{k'} \pi^{(k')} \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}^{(k')}, \boldsymbol{\Sigma}^{(k')})}.$$

- **M-step**: Estimate the parameters $\pi^{(k)}$, $\boldsymbol{\mu}^{(k)}$ and $\boldsymbol{\Sigma}^{(k)}$ based on the responsibilities $c_n^{(k)}$ (and $N_k = \sum_n c_n^{(k)}$) as

$$\begin{aligned}
\pi^{(k)} &\leftarrow \frac{N_k}{N} \\
\boldsymbol{\mu}^{(k)} &\leftarrow \frac{1}{N_k} \sum_{n=1}^{N} c_n^{(k)} \mathbf{x}_n \\
\boldsymbol{\Sigma}^{(k)} &\leftarrow \frac{1}{N_k} \sum_{n=1}^{N} c_n^{(k)} (\mathbf{x}_n - \boldsymbol{\mu}^{(k)})(\mathbf{x}_n - \boldsymbol{\mu}^{(k)})^\top.
\end{aligned}$$

Given initial guesses for the parameters $\pi^{(k)}$, $\boldsymbol{\mu}^{(k)}$ and $\boldsymbol{\Sigma}^{(k)}$, iterating these two steps increases the likelihood of the data in each step [13, p. 437]. The procedure can be terminated if the value of the likelihood $p(\mathcal{D}; \boldsymbol{\theta})$ does not change significantly anymore.

For a given number of K components, the distribution of the independent variation parameters $\mathcal{X}$ can be approximated using a Gaussian mixture model, where the parameters are found through the expectation maximisation algorithm. In fact, as already noted before, a GMM with enough components can approximate almost any continuous density with arbitrary accuracy [13, p. 111]. While this certainly allows the approximation of a wide range of possible distribution characteristics, it may also encourage choosing an unnecessarily complex model over simpler ones in favour of a better fit. The problem of choosing the correct/suitable model (complexity) is usually not straightforward and is referred to as model selection in the literature [13, Sec. 1.3].

[4] Note that $p(\mathcal{D}; \boldsymbol{\theta}) = \prod_n p(\mathbf{x}_n; \boldsymbol{\theta})$ by assumption of conditional independence of $\mathbf{x}_n$ given $\boldsymbol{\theta}$. The individual factors $p(\mathbf{x}_n; \boldsymbol{\theta})$ are often smaller than one, hence, the product gets very small which can lead to numerical problems. Additionally, $\ln p(\mathcal{D}; \boldsymbol{\theta})$ may be concave which simplifies the optimisation.

Model selection

Several ways have been proposed to assess the correct model complexity. The first, and often preferred way is to choose the model through a cross-validation procedure, see [13, Sec. 1.3][64, Sec. 7.10]. Here, the data to be modelled is partitioned into subsets of equal size (usually three, five or ten). The model is then estimated on all but one of the subsets (test set) and its quality is recorded. The process is repeated and every subset is selected as a test set once. Finally, the best model is chosen based on its average test set quality (cross-validation performance). Although cross-validation is generally more time-consuming, it offers a principled way to estimate the generalisation error of the model from the training data.

An alternative approach to tackle the model selection problem is through so-called information criteria such as the Akaike Information Criterion (AIC) (see [13, Sec. 1.3]) or the Bayesian Information Criterion (BIC) (see [13, Sec. 4.4.1]). The AIC is given by subtracting the number of model parameters from the maximum log-likelihood, i.e., $\ln p(\mathcal{D};\boldsymbol{\theta}) - \dim(\boldsymbol{\theta})$. Its interpretation is simple. The more parameters a model has (denoted by $\dim(\boldsymbol{\theta})$ here), the better its fit has to be. An alternative to the AIC is the BIC. The BIC is motivated by a Laplace approximation [13, Sec. 4.4] of the (usually hard to compute) marginal likelihood or model evidence in Bayesian model comparison [13, Sec. 3.4]. It expresses how likely the observed data $\mathcal{D}$ was generated by the model. Compared to the AIC, the BIC generally penalizes model complexity more heavily [13, p. 217].

Similar to the cross-validation performance, these information criteria can be used to evaluate and compare configurations of different GMMs, e.g., with a different number of components K. Generally, all selection criteria prefer simple models which capture the essential aspects of the data.

4.2.4 Drawing samples from a Gaussian mixture model

To successfully employ a variation model in circuit simulations, generating samples should be simple and fast. Fortunately, despite their flexibility, GMMs allow for an efficient sampling process. Sampling from a GMM only requires the ability to obtain samples from a standard normal distribution $\mathcal{N}(0,1)$ and a uniform distribution $\mathcal{U}[0,1]$, which should be included in most simulation environments.

As preparation, the interval of $[0,1]$ is divided into K regions with sizes equal to the $\pi^{(k)}$ (note that $\sum_k \pi^{(k)} = 1$). Additionally, let $\boldsymbol{L}^{(k)}$ denote the matrix obtained by the *Cholesky decomposition* [143, Sec. 23] of $\boldsymbol{\Sigma}^{(k)}$, i.e., $\boldsymbol{L}^{(k)}(\boldsymbol{L}^{(k)})^\top = \boldsymbol{\Sigma}^{(k)}$. Finally, note that random vectors $\mathbf{r} \sim \mathcal{N}(0, \boldsymbol{I})$ can be constructed by stacking $\dim(\mathbf{x})$ samples $r_i \sim \mathcal{N}(0,1)$ in a vector $\mathbf{r}$. Based on these preparations, a sample $\mathbf{x}_n$ of a GMM can be obtained through the following two steps:

1. Sample $u_n \sim \mathcal{U}[0,1]$ and select the distribution (component) k based on in which region of $[0,1]$ the sample u_n falls into.
2. Sample $\mathbf{r}_n \sim \mathcal{N}(0, \boldsymbol{I})$ and obtain $\mathbf{x}_n$ as $\mathbf{x}_n = \boldsymbol{\mu}^{(k)} + \boldsymbol{L}^{(k)}\mathbf{r}_n$.

The two-step procedure of sampling is commonly referred to as *ancestral sampling* [13, p. 432]. Naturally, the matrix computations can also be executed entry-wise and do not necessarily require linear algebra software. Through this procedure, samples from a variation model can be generated efficiently.

4.3 A variation model for an inkjet-printed transistor

In this section, the proposed framework is used to develop a Monte Carlo model for an inkjet-printed EGT presented in [100]. To validate the derived model, it is used to predict the distribution of the saturation current I_{sat}, an important performance metric for EGTs. Additionally, simulations for a printed physical unclonable function circuit from [48] are performed and the predicted output voltage is compared to output voltages of fabricated and measured circuits.

To establish familiarity with the measurement data, the EGT model and its parameters are briefly introduced. However, the overall framework is not dependent on this specific device or technology.

4.3.1 The dataset

The dataset used for the experiments consists of current-voltage measurements of 86 printed EGTs with a length of $L = 100\mu m$ and different widths of $W = 200, 400, 600, 800\mu m$. Note that due to the fabrication being a laboratory process, only a small number of devices could be produced and characterized.[5] To account for the different transistor widths, a linear dependence of the current on W/L is assumed and the measured currents are scaled by the channel geometry (L/W) of the respective transistor. After that, the device parameters can be extracted using the device model described in the following.

4.3.2 An EGT device model

To model the current-voltage characteristics of the EGTs, the model of [125], which is based on the EKV model for sub-micrometer MOSFETS [47], is used. Contrary to [125], two of the fitting parameters (f_3 and f_4) are replaced with values more closely reflecting the original EKV model. This leads to the following drain current equation

$$I_{DS} = I_0 \left(\ln\left(1 + \exp\left(\frac{v_p - v_s}{2}\right)^{\gamma}\right) - \ln\left(1 + \exp\left(\frac{v_p - v_d}{2}\right)^{\gamma}\right)\right), \tag{4.3}$$

$$\text{where} \quad I_0 = 2 \cdot n \cdot f_1 \cdot \frac{W}{L} \cdot \phi_t^2, \quad v_p \approx \frac{V_{GS} - V_{th}}{\frac{n}{f_2} \cdot \phi_t}$$

$$\text{and} \quad n = \frac{1}{SS \cdot \phi_t \cdot \ln(10)}.$$

The parameters W and L denote channel width and length respectively, and n denotes the slope factor. Furthermore, v_p, v_d, and v_s represent the channel, drain, and source potentials normalized by the thermal voltage ϕ_t. In total, the model has five parameters, among which the threshold voltage V_{th} and the sub-threshold slope SS are extracted by extrapolation of the linear regions of the measured characteristics curve. Furthermore, the so-called power-law parameter γ is extracted empirically through the Curtice model as described in [100].

The parameters denoted by f_1 and f_2 are fitting parameters, and are determined by minimizing the relative error between the modelled and the measured transistor characteristics (details

[5] The devices were fabricated and characterized by Gabriel C. Marques.

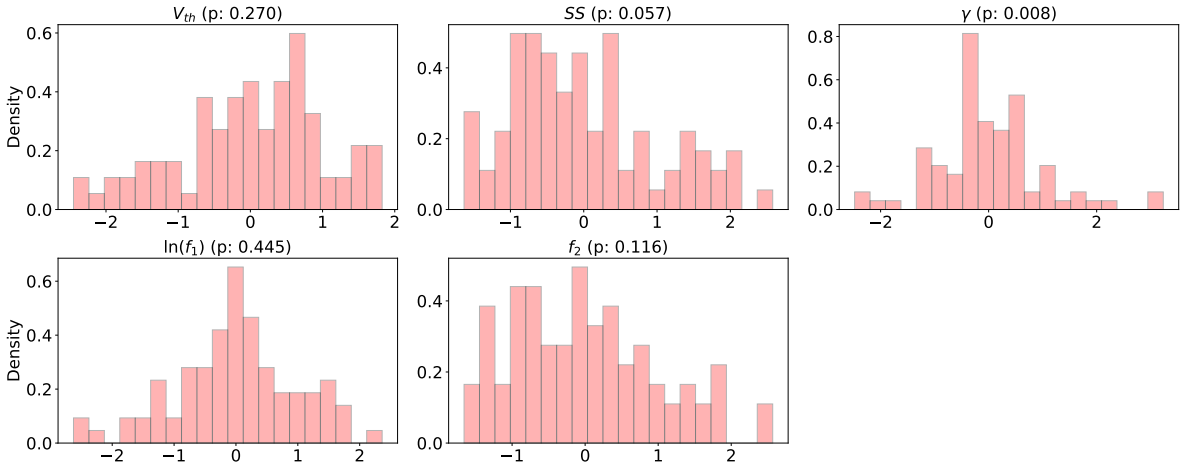


Figure 4.2: The standardized distribution of the device model parameters. The p-value indicates the probability of the data being normally distributed according to the test of D'Agostino and Pearson [36].

see [125]). However, note that f_1 is directly influenced by material properties such as mobility and gate capacitance [126] and cannot have negative values, while f_2 does not directly relate to physical effects.

As the main purpose of the model is to accurately reflect the current at different voltage levels, the quantities of interest in the EGT are the parameter V_{th} (threshold voltage) and the performance metric I_{sat} (saturation current). The threshold voltage V_{th} defines the (On/Off) switching behaviour of the transistor and I_{sat} expresses the strength of the transistor in the form of the current that can flow through the device. While V_{th} can be directly treated as a variation parameter, I_{sat} is a function of all device model parameters and has to be extracted from the resulting model as described in [100].

4.3.3 Applying data transformations

To avoid negative values for f_1, the aforementioned log-transformation (see Section 4.2.1) is used. Then, all parameters are standardized to achieve similar value ranges. The resulting distribution can be seen in Figure 4.2.

To get an idea about the possible complexity of the distribution, a test for normality [36] is performed for each variable (see Figure 4.2). For most parameters, including the transformed parameter $\ln(f_1)$, the assumption of a normal distribution cannot be rejected, while the distribution of the power-law parameter γ can likely not be approximated well with a normal distribution (as $p < 0.01$). Furthermore, the (Pearson) correlation coefficient (see Table 4.1) reveals high correlations between certain parameters, see $\ln(f_1)$ and γ.

To conclude, a suitable model of the variations in the device model parameters has to consider the parameters jointly and needs to be expressive enough to model distribution shapes beyond a simple normal distribution.

	V_{th}	SS	γ	$\ln(f_1)$	f_2
V_{th}	1	0.58	-0.33	0.38	-0.58
SS	·	1	-0.22	0.48	-0.85
γ	·	·	1	-0.71	0.25
$\ln(f_1)$	·	·	·	1	-0.53
f_2	·	·	·	·	1

Table 4.1: The (Pearson) correlation between the transformed device model parameters of the printed EGTs. The high correlation between the variables confirms the assumption that the device model parameters should be modelled jointly.

4.3.4 Parameter selection and reduction

For the parameter reduction step (Section 4.2.2), the two sets of parameters, namely the independent variation parameters $\mathcal{X}$ and the dependent parameters $\mathcal{Y}$, need to be selected. Since the parameters $\{V_{th}, SS, \gamma\}$ directly represent current-voltage-curve characteristics, they are declared independent variation parameters. Furthermore, as f_1 [respectively $\ln(f_1)$] partly represents material properties, it is also declared an independent variation parameter and is jointly modelled with $\{V_{th}, SS, \gamma\}$. This leaves the fitting parameter f_2 as the only dependent variable in the case of the EGT.

From looking at a scatter plot of the variables $\{V_{th}, SS, \gamma, \ln(f_1)\}$ with f_2 (see Figure 4.3), it is evident that a great portion of the variation in f_2 can already be explained through a function of SS. This is also to be expected from the high correlation of f_2 with SS (see Table 4.1).

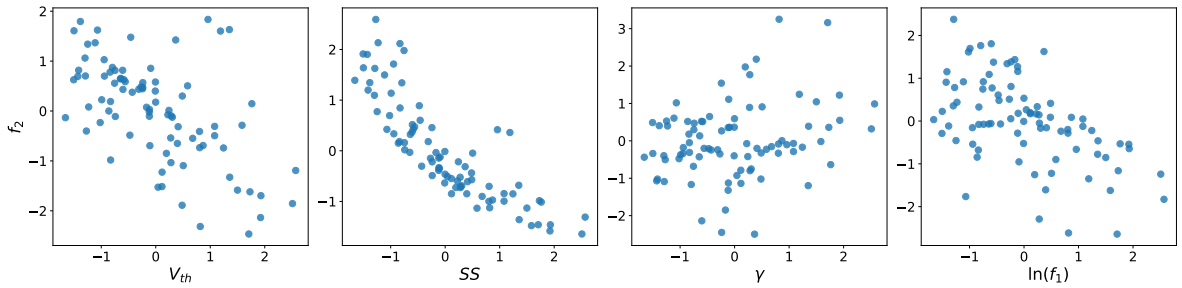


Figure 4.3: The scatter plots of the fitting parameter f_2 with the individual variation parameters. All variables are standardized.

To model f_2 through the other parameters, a lasso regression model as described in Section 4.2.2 is used. Since the dependency in Figure 4.3 seems to be nonlinear, all monomials of the variables up to degree two (15 in total) are provided to the model. The selection of the relevant components is left to the lasso. The penalty term λ is found by employing grid-search and cross-validation. The resulting model uses eight basis functions and shows a mean squared error of 0.128 ($r^2 = 0.73$) on a random held-out test set of 25% of the original data.

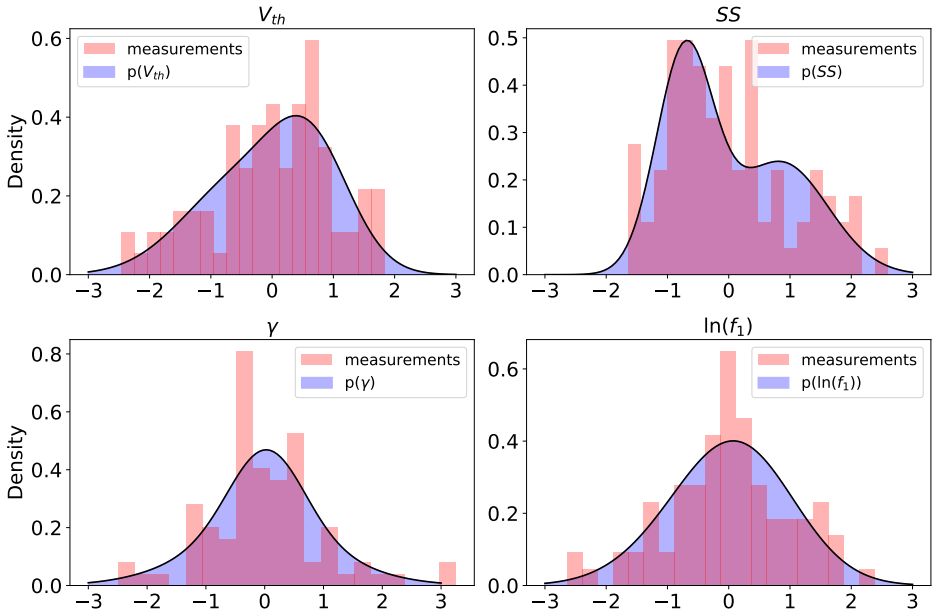


Figure 4.4: The marginal distributions of the variations parameters modeled through a Gaussian mixture model with two components. To display the measurement data of 86 values, 20 histogram-bins were used.

4.3.5 The distribution model

As a last step, the distribution of the (transformed and standardized) variation parameters $\mathcal{X} = \{V_{th}, SS, \gamma, \ln(f_1)\}$ is jointly estimated using a GMM. To select the number of components, the BIC is employed as a model selection criterion and suggests the use of two components for the given dataset.[6] The resulting marginal distributions of the respective parameters can be seen in Figure 4.4. The GMMs can now be used to sample parameter sets of $\mathcal{X}$ as described in Section 4.2.4.

4.3.6 Evaluation

After all steps have been carried out, the Monte Carlo model is complete. To get a first idea of the quality of the model, 1000 samples are drawn from the GMM. Then, the parameter f_2 is predicted using the respective regression model for all samples. After inverting the initial transformations, i.e., standardisation and applying the $\exp(\cdot)$ function to the sample entries of $\ln(f_1)$, the parameters can be plugged in the current equation (4.3). From the resulting curve, the saturation current I_{sat} can be calculated as in [100]. Figure 4.5, shows the resulting distribution of I_{sat} for the sampled devices compared to the I_{sat} distribution of the initial dataset. It can be seen the main characteristics of the curve are closely reflected and no unphysical values are observed. To obtain a quantitative estimate about the difference between the distributions, the test statistic of the (two sample) *Kolmogorov-Smirnov test* [153, p. 245] can be used. It compares distributions of two samples based on the maximum difference between their empirical cumulative distribution functions. For the samples from the I_{sat} distributions, it gives a value of 0.1039 (p-value of 0.3127).

[6] From one to five components were evaluated with giving BIC values of 912, 904, 928, 967 and 1008 respectively.

To validate the accuracy of the model on the circuit level, a simulation for a printed physical unclonable function circuit proposed in [48] (see Figure 4.6a) is set up. The circuit has two outputs (Out and $\overline{Out}$) for which variations can be observed. For circuit simulation, the cadence virtuoso design environment in combination with the design kit from [102] is used.

The histogram of 1000 ($\times 2$ Outputs) simulations is displayed alongside 36 (18×2) output voltage measurements of the respective circuit.[7] By visual comparison, the simulation captures the main characteristics of the distribution (see Figure 4.6b).

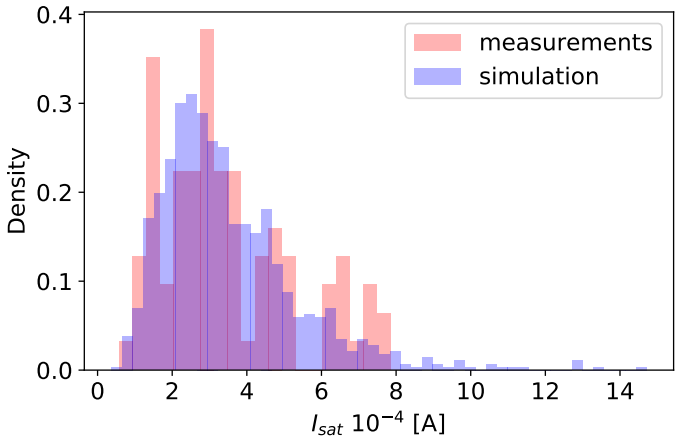
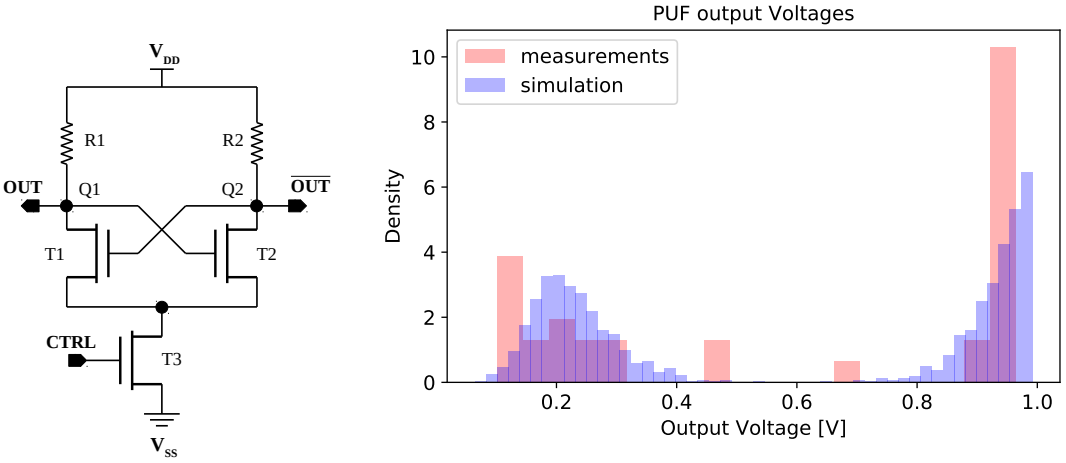


Figure 4.5: The distribution of the saturation current I_{sat} based on 1000 sampled sets of parameters. To display the measurement data of 86 values, 20 histogram-bins were used.



(a) (b)

Figure 4.6: Figure (a) shows the schematic of the physical unclonable function circuit from [48] and was adopted from [126]. Figure (b) shows the comparison between the Monte Carlo simulation and measured output responses of the circuit for 36 (18×2) measurements of output voltages. The circuit simulation was performed by Farhan Rasheed and similar evaluation has been reported in [124, Sec. 3.5.3].

[7] Note that due the fabrication being a laboratory process, only a small number of samples could be obtained.

4.4 Conclusion and directions for future work

In emerging technologies such as inkjet-printed electronics, devices and circuits suffer from high variations. These variations originate from the unique features of the production process and can lead to various distribution shapes and correlation structures among the device model parameters. Additionally, the common presence of fitting parameters in these semi-empirical models has to be considered when modelling variations.

To address these challenges, a general, data-driven framework for developing a variation model based on a given device model was proposed. It specifically addresses empirical fitting parameters often present in device models of novel technologies. The fitting parameters are thereby approximated through a regression model and do not need to be considered when modelling variations. Additionally, the framework is able to model various distribution shapes and correlation structures in the physical parameters by approximating their distribution through a GMM. The number of components of the GMM is thereby automatically selected through the BIC. The proposed approach is very flexible and subsumes classical variation modelling flows. The framework was used to develop a variation model of a printed EGT and the distribution of simulated saturation currents closely matched that of measured devices. To further test the model, Monte Carlo simulations for a printed physical unclonable function circuit were performed. Here, the simulation results displayed a close proximity to actually measured output voltages of fabricated circuits.

Even though the framework is purely data-driven and therefore technology independent, it still relies on a specified device model. However, developing such a device model can be challenging and often requires substantial expert knowledge. A logical next step would be to combine the framework with a data-driven modelling procedure that directly extracts a device model from measurement data. Unfortunately, the development of a data-driven model compatible with the proposed framework is not straightforward. This is due to implicit requirements on the device model that have not been explicitly stated yet. First, physically plausible behaviour, such as monotonicity of the current with respect to the voltage, needs to be satisfiable through relatively simple conditions. For example, a positivity constraint on the parameter f_1 in the EGT model guarantees an increasing current for increasing input voltages. Due to the simplicity of this condition, it can be easily enforced though transformations of the respective parameter. Secondly, the model needs to be identifiable (see [131]), in other words, there must exist no two sets of model parameters yielding the same result. Both of these conditions are common in physics-based models, as candidate parameters for variation modellings, such as V_{th}, represent well-defined effects. However, many data-driven modelling approaches, like approximations of the current-voltage characteristics with neural networks (see [57]), will not be compatible with the framework since they lack the aforementioned identifiability.[8]

To conclude, the presented framework allows to derive variation models based on given device models. Due to the high flexibility in modelling the joint distribution of the parameters, it is especially suitable for modelling variation in emerging technologies such as printed electronics. Furthermore, the generality of the approach allows designers to quickly generate new variation models for new devices or adapted technologies, e.g., devices printed on new substrates or with novel ink compositions.

[8] A neural network is generally not identifiable as the labeling of the neurons is arbitrary. For a brief introduction to neural networks see Section 2.4.

5 Variation-aware Training for Printed Neural Networks

In Chapter 3, pNNs were introduced as a model to design (train) printed NCs. Unfortunately, as seen in the last chapter, one aspect that makes the design of printed circuits challenging are the high variations in printed structures [24, 139, 126]. These variations influence the electrical properties of printed devices and thus the circuit outputs. If not considered, the printed NCs may display much lower accuracies than initially expected. To address this problem, this chapter introduces a variation-aware training method for pNNs. As a first step, variation models are developed for the core components of printed NC, namely the resistance crossbar and the inverter and activation function circuits. The latter models are thereby developed using the techniques described in the last chapter. Based on these variation models, the training problem is modified to minimize the expected loss under variation. The training procedure is adapted accordingly. The main components and steps of the modified procedure can be seen in Figure 5.1.

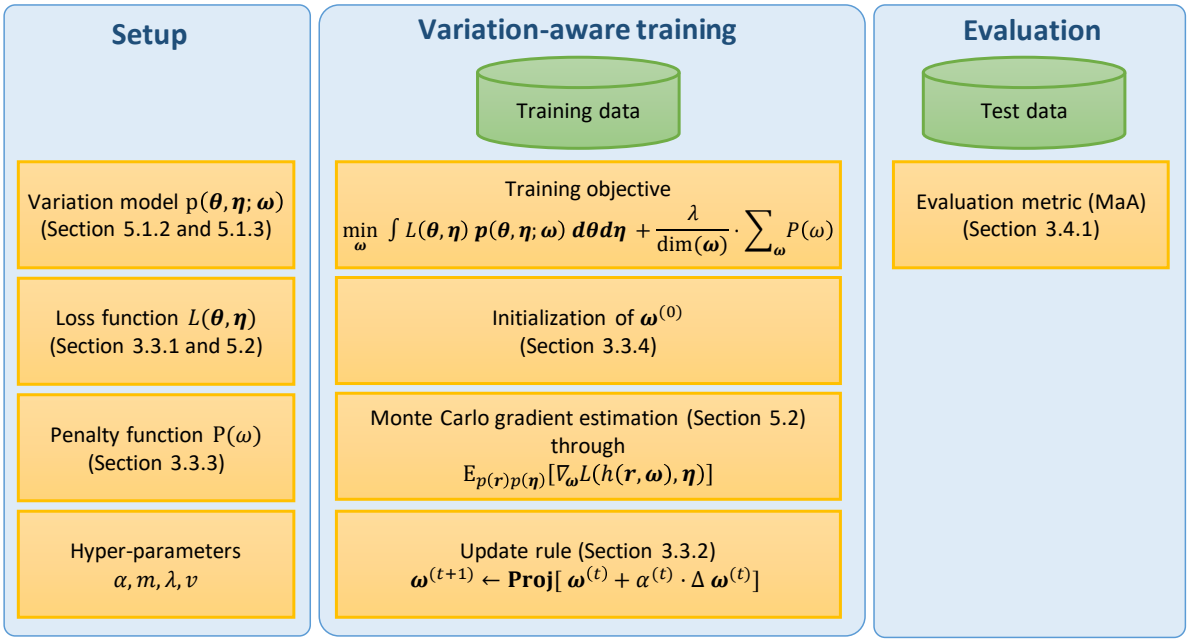


Figure 5.1: The main steps of the variation-aware training procedure for printed neural networks.

The rest of the chapter is outlined as follows. First, related work on training methods for dealing with variations and faults in the context of NCs is reviewed. Then, the variation models for the core components of printed NCs are developed, followed by a derivation of the variation-aware training procedure. The effectiveness of the approach is then demonstrated in various experiments. The chapter closes with a discussion of the results and possible directions for future work.

5.1 Preliminaries and background

To give context to the proposed approach in this chapter, the following section reviews related work on increasing the tolerance of hardware implementations of neural networks and NCs to variations and faults. Then, variation models for the core components of printed NC (see Section 2.6), namely the resistance crossbars and the inverter and activation function circuits, are developed. The model for the conductance of printed resistors is based on theoretical considerations, and the models for the circuit components are developed using the data-driven modelling approach described in Chapter 4.

5.1.1 Related work

Various approaches have been proposed to increase the robustness of neural networks and NCs against different kinds of variations and faults. The main body of research focuses on perturbation of the neural network parameters through either additive or multiplicative variation/noise, see [91, 17, 137, 10, 43, 44, 93], as well as digital stuck-at-0/1 faults, see [91, 17, 163, 158]. More recently, also timing faults have been considered [37]. The application of such parameter perturbations was not always directly motivated by achieving robustness with respect to the simulated variations, but also with respect to other, undesired effects. Most prominently, various kinds of integration and fabrication errors [43, 44], as well as value perturbations through round-offs [91].

Mitigating the impact of such variations is thereby addressed through either off-device or on-device methods. On-device methods directly optimize the model parameters based on the measured responses of a circuit [98, 93]. Through this, the true device characteristics and variation are directly observed and can be dealt with accordingly. On the other side, off-device methods derive the model parameters based on models of the circuit behaviour [94, 93]. The obtained parameters, i.e., conductance values of crossbars, can then be mapped to the respective hardware. Additionally, the devices can be tested before performing the mapping. Through this, mapping sensitive weights to devices that exhibit high variations (see [93]) may be prevented, and weights of highly varying devices may be mapped to zero, see [78]. The sensitivity of a weight can thereby be assessed through its gradient with respect to the loss. Since on-device approaches can leverage the specific characteristics of the given circuit and its devices, they provide more accurate solutions than off-device methods. In turn, off-device training is generally faster since it does not depend on the time-intensive feedback loop or assessment of the individual circuit characteristics [93].

However, as already discussed in Section 3.1, printed NCs only permit off-device training, as on-device training is not applicable due to the limited adjustment capabilities of printed resistors after fabrication.[1] Also, methods based on pretesting cannot be applied as the variation happens in the mapping/fabrication step and cannot be assessed beforehand. For these reasons, the following focuses solely on model-based off-device training methods.

Several different off-device approaches for reducing the effects of variations have been proposed. One branch of these methods focuses on fault tolerance via adding regularisation terms to the loss function, e.g. [18, 44, 10, 93, 91], to encourage the finding of "flat minima" [70]. Here, the motivation is that in flat regions of the loss function, the loss of the network should be less

[1] Details on the possibilities of tuning the printed resistances after fabrication are discussed in Chapter 6.

sensitive to small perturbations of the parameters. Hence, this should lead to improved robustness with respect to parameter variations. Furthermore, some works try to achieve "maximally fault-tolerance" through the formulation of a min-max optimization problem against adversarial node-deletion [114, 38]. This idea is similar to approaches aiming at unifying the magnitude of the weights (see [137]) to equalize their importance. The notion of enforcing (small) and similar weight magnitudes can also be seen in [17, 18] and is closely linked to the classical method for regularizing neural networks through weight decay, see [56, Sec. 5.2.2].

Aside from penalty terms, a common strategy to robustify neural networks is the injection of faults and parameter variations in training, see [17, 18, 111, 43, 37]. Although their motivation is slightly different, these approaches are similar to noise injection methods such as [103, 3, 60, 1, 140], which aim to improve the training speed or generalisation of neural networks [69]. This relationship between generalisation and fault and variation tolerance was also noted and discussed in [111, 21].

Besides off-device approaches, which encourage the neural networks to exhibit robustness by themselves, several works [29, 46, 27, 17, 18, 40, 120] try to identify important neurons which are then split or replicated to generate explicit redundancies. In order to leave the network output unchanged, the outputs for the replicated neurons are scaled accordingly [44]. Through these explicit redundancies, certain variations are expected to average out and the joint sensitivity of the respective nodes is lower.

To conclude, to robustify pNNs against variations, some concepts from the related work may be adapted. First of all, methods utilizing node replications for redundancy could theoretically be employed, however, they are unfavourable for printed NCs as this would directly increase the size of the circuit.

Weight-decay-like penalty function techniques, such as [18], may present a promising direction to investigate. Out of all the discussed approaches, employing penalty functions is rather simple and cheap from the perspective of training. However, an adaptation is not straightforward as it may encourage solutions conflicting with the technology constraints. More specifically, the coupling of the weights needs to be taken into account and, e.g., simply encouraging overall smaller conductances may lead to infeasible solutions. Finally, it is not clear how component variations, such as variations of the activation function, could be considered through penalty terms.

Hence, especially methods considering variation in training can be seen as promising due to their generality. With respect to the types of variations, stuck-at-faults should be largely irrelevant for the type of printed NCs under consideration. In the context of printed electronics, stuck-at-faults would relate to missing prints that could simply be repeated. More relevant on the other hand would be variations that the components, i.e., the crossbar resistors, activation function, and inverter circuits, may experience. To this end, variation models for these components are developed in the following. The developed models can then be used to consider the respective variations in training.

5.1.2 A simple variation model for printed resistors

The resistor crossbars perform the weighted sum operation in the printed NC. Their desired conductance values are realized by printing certain geometries with droplets of conductive ink. Based on the geometry and material properties, different conductances may be obtained (see Section 2.6). Assuming that the sizes of the droplets forming the conductive structure exhibit a constant, independent variation, the total variation depends on the number of droplets printed. Naturally, larger structures exhibit higher conductivity[2] and also require more droplets. This leads to a relative conductance variation with

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

where $\overline{g}$ denotes the target conductance that should be printed and r is a random variable with the probability density function $p(r)$. By choosing different target conductances $\overline{g}$, the distribution of the observed conductances g changes. Most notably, trying to print higher conductance values leads to more variation.[3] Alternatively, this variation model could also be motivated through variations of the material-related properties influencing the conductivity as in [156].

Depending on the choice of $p(r)$, different types of relative variation can be expressed, e.g.,

$$\begin{aligned} p(r) &= \mathcal{N}(\mu, \sigma) \quad &\Rightarrow \quad g &\sim \mathcal{N}(\mu + \overline{g}, \sigma \cdot \overline{g}) \\ p(r) &= \mathcal{U}(a, b) \quad &\Rightarrow \quad g &\sim \mathcal{U}((1 + a) \cdot \overline{g}, (1 + b) \cdot \overline{g}). \end{aligned}$$

However, note that not all choices $p(r)$ lead to a valid model for conductance variations. For example, choosing $p(r) = \mathcal{N}(0, \sigma)$ may yield samples of r for which g is negative. To avoid such implausible values, a symmetrical uniform distribution $p(r) = \mathcal{U}[-\epsilon, \epsilon]$ with $\epsilon \in [0, 1]$ is selected. Hence,

$$p(g\,;\, \overline{g}, \epsilon) = \mathcal{U}[(1 - \epsilon) \cdot \overline{g}, (1 + \epsilon) \cdot \overline{g}]. \tag{5.1}$$

Here, ϵ denotes the maximum variation that is observed and will be referred to as variation level in the following. As an example, consider $\epsilon = 0.1$. In this case, $g \sim \mathcal{U}[0.9\,\overline{g}, 1.1\,\overline{g}]$, so the observed conductance values vary up to 10% from the intended target conductance $\overline{g}$.

Depending on the perspective taken, the choice of a uniform distribution may be seen as either a weak or a strong assumption. On the one hand, the uniform distribution has heavy tails, where extreme values are just as likely as realisations of the target value $\overline{g}$. On the other hand, the maximum variation is always bounded through ϵ and therefore offers little "surprise" with respect to outliers. Nevertheless, since the focus in the following will mainly be on expected behaviour rather than on rare events and outliers, this should not be a problem. Additionally, the choice of the uniform distribution offers a convenient way to avoid implausible values in the form of negative conductances.

If the variation for a whole set (vector) of fabricated conductances $\boldsymbol{g}$ is assumed independent, the joint density of $\boldsymbol{g}$ is given by

$$p(\boldsymbol{g}\,;\, \overline{\boldsymbol{g}}, \epsilon) = \prod_i p(g_i\,;\, \overline{g}_i, \epsilon). \tag{5.2}$$

[2] Assuming a constant channel length.

[3] This form is also often used to model multiplicative weights variation for neural networks, see [43, 93].

Note that the assumption of independence can surely be questioned. For example, a nozzle defect would influence a whole range of subsequently printed resistors. Hence, their variation would not be independent but tied together through the occurrence of this event. However, assuming a healthy production process, the independence assumption should be justified.

5.1.3 Variation models for the activation and inverter circuits

Aside from variations in the conductances of the crossbar resistors, printed NCs may also experience variations in the circuits for the activation and the inverter function. These variations are induced by variations of the components (transistors and resistors). Since a closed-form variation model for these components cannot be easily derived, a data-driven approach may be taken.

For this, a dataset of multiple ptanh and inv functions is required. Ideally, this dataset would come from measurements of fabricated circuits. However, this would be costly and time consuming since, as for now, the fabrication of inkjet-printed devices is only a laboratory process. Therefore, a circuit simulation is used to generate the output data. The simulations were carried out for multiple variations levels of $\epsilon \in \{0, 5, 10, 15, 20, 25, 30\}$ %. Here, voltage fluctuations were assumed to be standard normal distributed. The transistor (EGT) variations were modelled through the variation model developed in [126], and (5.2) was used for conductance variations (see also [156]). In total, 1000 simulations were performed for each variation level ϵ.[4]

The datasets of each simulated ptanh and inv circuit can then be modelled through an appropriately parameterized tanh function as described earlier in Section 3.2.2. Contrary to before, the parameter η_4 in

$$\psi_{\boldsymbol{\eta}}(x) = \eta_1 + \eta_2 \cdot \tanh((x - \eta_3) \cdot \eta_4)$$

should be restricted to $\eta_4 > 0$ to guarantee identifiability. Otherwise, due to the point symmetry of $\tanh(x)$, there could be two sets of parameters $\boldsymbol{\eta}$ that lead to the exact same shape of $\psi_{\boldsymbol{\eta}}(x)$.[5] This is undesirable, as it would require the variation model to express a strict dependence between the signs of η_2 and η_4. To preserve the orientation of the functions irrespective of the variation, the parameters η_2 and η_4 are transformed accordingly. Hence, η_4 is modelled through $\ln(\eta_4)$. Applying the inverse transformations $\exp(\cdot)$ to samples then guarantees positivity (details see Section 4.2.1). Similarly, since η_2 is responsible for the orientation of $\psi_{\boldsymbol{\eta}}(x)$, it is modelled as $\ln(\eta_2)$ for ptanh and $\ln(-\eta_2)$ for inv.

In the last step, the datasets of the transformed parameters are modelled through a GMM as described in Section 4.2.3. The number of components for each GMM is selected based on the highest average log-likelihood achieved in a 5-fold cross-validation. Through the fitted GMMs, samples from the distributions of $p(\boldsymbol{\eta}; \epsilon)$ can be obtained by sampling from the respective GMM and inverting the transformations applied to η_2 and η_4. See Figure 5.2 for sampled inv and ptanh functions of different variation levels ϵ.

[4] All circuit simulations were set up and carried out by Dennis D. Weller.
[5] As an example, consider $\boldsymbol{\eta} = [0, 1, 0, -1]$ and $\boldsymbol{\eta} = [0, -1, 0, 1]$.

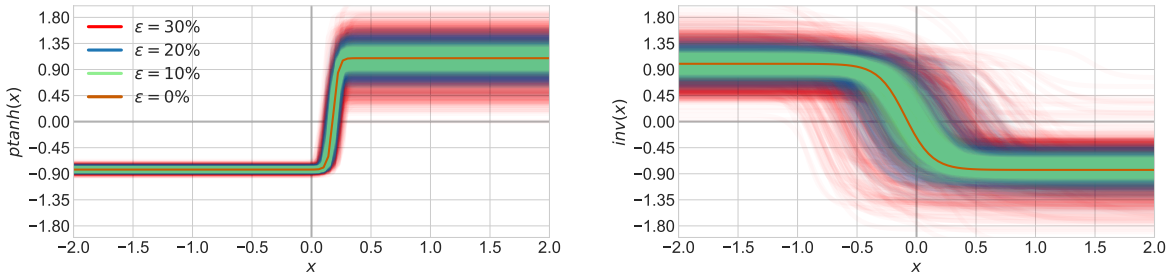


Figure 5.2: The modelled distributions of the ptanh and inv functions for different variation levels ϵ.

5.2 Variation-aware training of printed neural networks

In Chapter 3, the training of pNNs was described as minimizing the loss function $L(\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$ of the network. After training, the achieved parameter vector $\boldsymbol{\theta}$ was assumed to be fabricable as long as it satisfied the technology constraints, i.e., $g_i = |\theta_i| \in [g_{min}, g_{max}] \cup \{0\}$. However, if the fabrication of the crossbar resistors is subject to variations, the desired g_i may not be realized exactly. To account for this uncertainty, the optimisation problem for training is modified. Training with the modified objective is referred to as variation-aware training in the following. For simplicity, the modification of the training objective is first introduced only for conductance variations of the crossbar resistors and then extended to consider variations of the activation and inverter functions. Finally, the general concept of variation-aware training is reviewed.

5.2.1 Variation-aware training under conductance variation

To find configurations of conductances that is robust against variations, the respective variations should be directly considered in the training of the pNN. Due to the simple relationship between the conductances g and the surrogate conductances θ, a variation model of $|\theta|$ can be readily derived by considering that the sign of θ is unaffected by variations (it encodes connectivity). Hence, given the variation model of (5.1) for conductances, the variation of surrogate conductances θ may be modelled as

$$p(\theta; \omega, \epsilon) = \mathcal{U}[(1-\epsilon)\cdot\omega, (1+\epsilon)\cdot\omega],$$

where ω denotes the target surrogate conductance (i.e., $|\omega| = \overline{g}$)[6]. It is easy to see that $p(|\theta|; |\omega|, \epsilon) = p(g; \overline{g}, \epsilon)$, hence, choosing this variation model for θ induces the correct variation for the conductances g. Also, note that the variation cannot lead to sign flips, as $\text{sign}(\theta) = \text{sign}(\omega)$. The joint density of the vector of surrogate conductances $\boldsymbol{\theta}$ can be derived similarly and is given by

$$p(\boldsymbol{\theta}; \boldsymbol{\omega}, \epsilon) = \prod_i p(\theta_i; \omega_i, \epsilon),$$

where $\boldsymbol{\omega}$ summarizes the ω_i. To keep the notation brief in the following, the dependence on the variation level ϵ is mostly omitted.

[6]The choice of ω instead of $\overline{\theta}$ is to improve the readability. Note that $\boldsymbol{\omega}$ could generally be seen as parameters of the distribution $\boldsymbol{\theta}$ that are learnable, and can be influenced.

When training pNNs before, the parameters $\boldsymbol{\theta}$ could be chosen directly. In contrast, the surrogate conductances are now random variables with a distribution $p(\boldsymbol{\theta};\boldsymbol{\omega})$. Hence, what sets of network parameters $\boldsymbol{\theta}$ are realized depends on $\boldsymbol{\omega}$, which represents the new decision variables. Furthermore, since $\boldsymbol{\theta}$ is random, the loss $L(\boldsymbol{\theta})$ is also random. Naturally, the new training objective would be to minimize the expected loss, i.e., the average loss of the pNNs under variation of $\boldsymbol{\theta}$. This results in a stochastic optimisation problem and can be formalized by

$$\min_{\boldsymbol{\omega}} \quad \mathbb{E}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})] = \int L(\boldsymbol{\theta})p(\boldsymbol{\theta};\boldsymbol{\omega})\mathrm{d}\boldsymbol{\theta}. \tag{5.3}$$

Here, the optimization variables are the target surrogate conductances $\boldsymbol{\omega}$ that determine the distribution of the observed $\boldsymbol{\theta}$ and $L(\boldsymbol{\theta})$. Unfortunately, $L(\boldsymbol{\theta})$ is nonlinear and there is generally no closed-form expression for $\mathbb{E}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})]$. Hence, it can only be evaluated approximately through, e.g., the Monte Carlo method [108]. Using Monte Carlo estimation, an estimate of the expected value in (5.3) can be obtained for a given $\boldsymbol{\omega}$ as

$$\mathbb{E}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})] \approx \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{\theta}_n) \quad \text{with} \quad \boldsymbol{\theta}_n \sim p(\boldsymbol{\theta};\boldsymbol{\omega}).$$

This estimate is unbiased, and, assuming that the variance of the estimate is bounded, converges (almost surely) to the expected value as $N \to \infty$ [56, Sec. 17.1.2]. Furthermore, according to the *central limit theorem* (see [153, Sec. 5.4]), the variance of the estimate is given by $N^{-1} \cdot \mathbb{V}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})]$ [56, Sec. 17.1.2]. However, since the variance of $L(\boldsymbol{\theta})$ is hard to assess, the main insight is that the estimate tends to get more accurate with more samples (higher N).

To employ gradient-based learning for (5.3), an estimate of the gradient of the objective function with respect to $\boldsymbol{\omega}$ is required. For this, techniques from the literature of Monte Carlo gradient estimation can be used, see [109]. One way to estimate the gradient is thereby the so-called pathwise gradient estimator[7]. It is applicable if samples from the distribution $p(\boldsymbol{\theta};\boldsymbol{\omega})$ can be obtained by transforming samples from a base distribution $p(\mathbf{r})$ to samples of $p(\boldsymbol{\theta};\boldsymbol{\omega})$ through a differentiable deterministic transformation $h(\boldsymbol{\omega},\mathbf{r})$. The transformation $h(\boldsymbol{\omega},\mathbf{r})$ is thereby referred to as a sampling path [109, Sec. 5.1].

Formally, this can be expressed by

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta};\boldsymbol{\omega}) \quad \Longleftrightarrow \quad \boldsymbol{\theta} = h(\boldsymbol{\omega},\mathbf{r}) \quad \text{with} \quad \mathbf{r} \sim p(\mathbf{r}).$$

With the knowledge of the transformation $h(\boldsymbol{\omega},\mathbf{r})$, the *law of the unconscious statistician* (see [153, Theorem 3.6]) can be used to rewrite the expected value $\mathbb{E}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})]$ as $\mathbb{E}_{p(\mathbf{r})}[L(h(\boldsymbol{\omega},\mathbf{r}))]$. In case of the variation model from Section 5.1.2, the transformation $h(\boldsymbol{\omega},\mathbf{r})$ can be readily derived as

$$\boldsymbol{\theta} = h(\boldsymbol{\omega},\mathbf{r}) = \mathrm{diag}(\mathbf{1}+\mathbf{r})\boldsymbol{\omega} \quad \text{with} \quad \mathbf{r} = [r_1, r_2, \cdots]^\top \quad \text{and} \quad r_i \sim \mathcal{U}[-\epsilon,\epsilon]. \tag{5.4}$$

This allows to explicitly attribute the variation to the random vector $\mathbf{r}$, and the function $h(\boldsymbol{\omega},\mathbf{r})$ transforms samples from $p(\mathbf{r})$ to samples from $p(\boldsymbol{\theta};\boldsymbol{\omega})$. Thus, according to [109, Sec. 5.2], the

[7] The pathwise gradient estimator is also known as "reparameterization trick" [85] in machine learning.

expression of the gradient can be rewritten as

$$\begin{aligned}\nabla_{\boldsymbol{\omega}}\mathbb{E}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})] &= \nabla_{\boldsymbol{\omega}} \int L(\boldsymbol{\theta})p(\boldsymbol{\theta};\boldsymbol{\omega})\mathrm{d}\boldsymbol{\theta} \\ &= \nabla_{\boldsymbol{\omega}} \int L\big(h(\boldsymbol{\omega},\mathbf{r})\big)p(\mathbf{r})\mathrm{d}\mathbf{r} \\ &= \int \nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r})\big)p(\mathbf{r})\mathrm{d}\mathbf{r} \\ &= \mathbb{E}_{p(\mathbf{r})}[\nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r})\big)]\end{aligned}$$

Note that the integral in the second line does not depend on the parameters $\boldsymbol{\omega}$. Using this transformation, the gradient of the objective function in (5.3) can be estimated by

$$\nabla_{\boldsymbol{\omega}}\mathbb{E}_{p(\boldsymbol{\theta};\boldsymbol{\omega})}[L(\boldsymbol{\theta})] = \mathbb{E}_{p(\mathbf{r})}[\nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r})\big)] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r}_n)\big) \quad \text{with} \quad \mathbf{r}_n \sim p(\mathbf{r}).$$

Through the pathwise gradient estimator, an estimate of the gradient can be obtained each time a gradient is required by the optimization algorithm. However, as the loss $L(\boldsymbol{\theta})$ is not differentiable everywhere, using these gradient estimates for learning pNNs should be seen as a heuristic approach.

Since $\boldsymbol{\theta}$ is subject to variation, the set of optimization parameters is $\boldsymbol{\omega}$. It represents the target (surrogate) conductance values and determines the distribution of $\boldsymbol{\theta}$. The parameters $\boldsymbol{\omega}$ need to obey the same constraints as the surrogate conductances $\boldsymbol{\theta}$ before, hence, the learning rules from Section 3.3 should be used for training.

It should be noted that the construction of the gradient estimator through the pathwise derivative did not make explicit use of the fact that a uniform distribution was chosen in the variation model for the conductances. Choosing for example a normal distribution would have only influenced the selection of the base distribution $p(\mathbf{r})$. On the other hand, the assumption of relative noise was directly used to derive the sampling path $h(\boldsymbol{\omega},\mathbf{r})$. Thus, if other variation models should be used, the applicability of the pathwise gradient estimator has to be reassessed.

5.2.2 Variation-aware training with activation and inverter variations

Besides conductance variations, the variation-aware training procedure can also be extended to consider variations of the circuit components. For this, the variation models $p(\boldsymbol{\eta})$ developed in Section 5.1.3 are used. Since the variation in $\boldsymbol{\eta}$ influences the output of the pNN, the loss becomes also a function of $\boldsymbol{\eta}$. Thus, the new training objective becomes

$$\min_{\boldsymbol{\omega}} \quad \mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta};\boldsymbol{\omega})}[L(\boldsymbol{\theta},\boldsymbol{\eta})] = \int L(\boldsymbol{\theta},\boldsymbol{\eta})p(\boldsymbol{\theta},\boldsymbol{\eta};\boldsymbol{\omega})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{\eta}. \tag{5.5}$$

The distributions of the surrogate conductances $\boldsymbol{\theta}$ and the circuit model parameters $\boldsymbol{\eta}$ can be assumed to be independent, as they would be fabricated independently. Furthermore, since the circuit models do not contain learnable parameters that may be adjusted during training, $p(\boldsymbol{\theta},\boldsymbol{\eta};\boldsymbol{\omega}) = p(\boldsymbol{\theta};\boldsymbol{\omega})p(\boldsymbol{\eta})$. The estimation of the objective in (5.5) is therefore simple, as long as samples from $p(\boldsymbol{\eta})$ can be drawn efficiently. Fortunately, this is the case when $p(\boldsymbol{\eta})$ is a GMM (see Section 4.2.4). Hence, given $\boldsymbol{\omega}$, the objective in (5.5) can be evaluated using Monte Carlo

estimation, where

$$\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta};\boldsymbol{\omega})}[L(\boldsymbol{\theta},\boldsymbol{\eta})] \approx \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{\theta}_n,\boldsymbol{\eta}_n) \quad \text{with} \quad \boldsymbol{\theta}_n \sim p(\boldsymbol{\theta};\boldsymbol{\omega}),\ \boldsymbol{\eta}_n \sim p(\boldsymbol{\eta}).$$

Estimates for the gradient can also be obtained in a similar manner by using the sampling path of (5.4). The parameters $\boldsymbol{\eta}$ do not complicate the estimation of the gradient as their distribution does not depend on $\boldsymbol{\omega}$. This allows the reformulation of

$$\begin{aligned}
\nabla_{\boldsymbol{\omega}}\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{\eta};\boldsymbol{\omega})}[L(\boldsymbol{\theta},\boldsymbol{\eta})] &= \nabla_{\boldsymbol{\omega}} \int L(\boldsymbol{\theta},\boldsymbol{\eta})p(\boldsymbol{\theta},\boldsymbol{\eta};\boldsymbol{\omega})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{\eta} \\
&= \nabla_{\boldsymbol{\omega}} \int L(\boldsymbol{\theta},\boldsymbol{\eta})p(\boldsymbol{\theta};\boldsymbol{\omega})p(\boldsymbol{\eta})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{\eta} \\
&= \nabla_{\boldsymbol{\omega}} \int L\big(h(\boldsymbol{\omega},\mathbf{r}),\boldsymbol{\eta}\big)p(\mathbf{r})p(\boldsymbol{\eta})\mathrm{d}\mathbf{r}\mathrm{d}\boldsymbol{\eta} \\
&= \int \nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r}),\boldsymbol{\eta}\big)p(\mathbf{r})p(\boldsymbol{\eta})\mathrm{d}\mathbf{r}\mathrm{d}\boldsymbol{\eta} \\
&= \mathbb{E}_{p(\mathbf{r})p(\boldsymbol{\eta})}[\nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r}),\boldsymbol{\eta}\big)].
\end{aligned}$$

Hence, the gradients can be estimated through samples of $p(\mathbf{r})$ and $p(\boldsymbol{\eta})$ as

$$\mathbb{E}_{p(\mathbf{r})p(\boldsymbol{\eta})}[\nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r}),\boldsymbol{\eta}\big)] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\omega}} L\big(h(\boldsymbol{\omega},\mathbf{r}_n),\boldsymbol{\eta}_n\big) \quad \text{with} \quad \mathbf{r}_n \sim p(\mathbf{r}),\ \boldsymbol{\eta}_n \sim p(\boldsymbol{\eta}).$$

As before, the estimated gradients can now be used in the learning procedure to (heuristically) minimize the expected loss under the given variation.

5.2.3 Variation-aware training in general

Variation-aware training, as introduced here, mostly boils down to the use of Monte Carlo gradient estimation (see [109]) to minimize the expected loss function under variations of the circuit components. With respect to the treatment of these variations, two different cases can be distinguished. Namely, variations that depend on the values of the learnable parameters and variations that do not.

If the variations do not depend on the learnable parameters, such as $p(\boldsymbol{\eta})$ in Section 5.2.2, an estimate for the gradient can be obtained by averaging evaluations of the gradient of the loss functions with sampled parameters from the distribution. If the variations depend on the values of the learnable parameters, like in the case of the surrogate conductances with $p(\boldsymbol{\theta},\boldsymbol{\omega})$, more care needs to be taken. Here, the choice of the parameters influences the values sampled. In this case, it is necessary to decouple the sampling process from the gradient computation. This is done by finding a sampling path that allows to transform samples from a base distribution, such as $p(\mathbf{r})$, to samples from the desired distribution through differentiable transformations. The transformation can be seen as a part of the model with its parameterization being learned, and gradient estimation becomes similar to the first case. In other words, an estimate for the gradient can be obtained by averaging evaluations of the gradient of the loss functions with transformed samples from the base distribution.

Finally, it should be noted that also other approaches for gradient estimation besides the pathwise gradient estimator exist. These may be applicable if a sampling path cannot be derived. A review of different gradient estimators and their properties can be found in [109].

5.3 Experiments

The following presents the evaluation of the variation-aware training approach. For ease of terminology, inv and tanh are simply referred to as "circuit models" in the following. More specifically, "measured inv/ptanh" refers to the models of the circuits extracted from measurements (as in Section 5.1.3), while "simulated inv/ptanh" refers to the respective functions extracted from data of circuit simulations. Additionally, the name of the dataset will often be used to refer to the network trained on the data, e.g., "the results on iris" refers to the results of the best pNN trained on the iris dataset.

The evaluation is split into several parts. First, only variations in the conductances of the crossbar resistors are considered through $p(\boldsymbol{g};\epsilon)$. The experiments are thereby repeated twice, once for the measured circuit models (as used for the results in Chapter 3) and once based on the simulated circuit models (for $\epsilon = 0$ %). These cases are treated separately due to the strong mismatch between both models (see Figure 5.3). Additionally, performing experiments for different curve shapes can help to assess if the curve characteristic influences the robustness of pNNs.

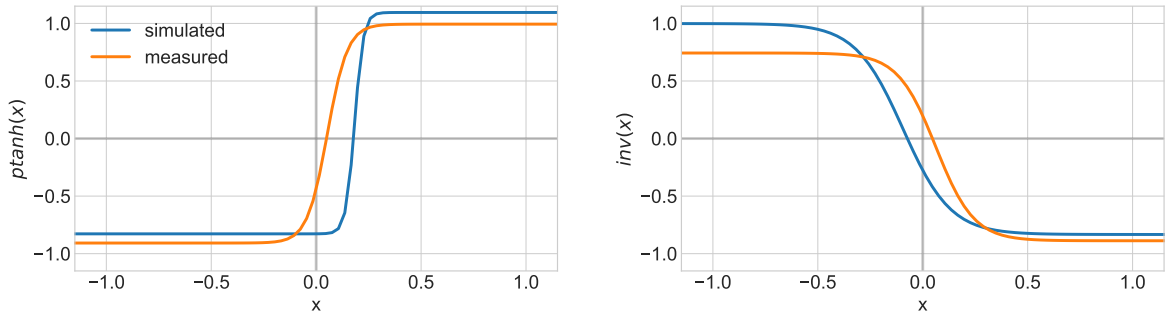


Figure 5.3: The circuit model functions for ptanh and inv extracted from data of measurements and circuit simulations. For the ptanh activation function, the simulated circuit displays a steeper behavior and is shifted further away from zero. The inv models display a similar slope, but the roots have different signs.

After the experiments for conductance variation, both, conductance and circuit model variation is considered jointly. For this, only networks trained with the simulated circuit models can be used, and the variations are obtained through the models of $p(\boldsymbol{\eta};\epsilon)$ developed in Section 5.1.3. The variation level for all experiments is chosen as $\epsilon \in \{0, 5, 10, 15, 20, 25, 30\}\%$ for the *pendigits* dataset and $\epsilon \in \{10, 20, 30\}\%$ for the other datasets. To distinguish between the variation used for training and testing, the test variation level is denoted by ϵ for $p(\mathbf{r};\epsilon)$. The variation level used for training is denoted by v for $p(\mathbf{r};v)$. If the concrete variation is unknown, v could be seen as a hyper-parameter, else, setting $v = \epsilon$ would be a natural choice. An overview of the performed experiments can be seen in the following table.

	$p(\boldsymbol{g};\epsilon)$ (Section 5.1.2)	$p(\boldsymbol{\eta};\epsilon)$ (Section 5.1.3)	hyper-parameters
measured ptanh/inv (Section 5.3.2)	✓	X (no dataset)	(Table 3.2)
simulated ptanh/inv (Section 5.3.3)	✓	✓	(Table 5.3)

To obtain baseline results and select the hyper-parameters, each experiment is preceded by

a grid-search. From the grid-search results, the networks achieving the best training MaA is selected (as in Section 3.4). Since no variations are considered in the grid-search, the selected hyper-parameters might not be optimal for the networks trained with variation. Thus, better results may be achievable by tuning the hyper-parameters separately for each variation level.

The estimated gradients for learning are obtained as described in Section 5.2 using $N = 100$ samples of $p(\mathbf{r}; v)$ with the respective given training variation v. Contrary to the experiments in Section 3.4, the surrogate conductances θ_0 for g_0 are updated in training now (before g_0 was fixed to g_{max} to allow for maximum decoupling). Finally, all mean and standard deviations reported in the following are also estimated using 100 samples.

For the evaluation, datasets from the UCI database [42] are used. In addition to the datasets already employed in Section 3.4, the *pendigits* (see Section 5.3.1) dataset is used for evaluation and visualisation.

5.3.1 The pendigits dataset

To present a more challenging task, the *pendigits* (pen-based recognition of handwritten digits) dataset from the UCI database [42] is used in the following evaluation. The dataset consists of approximately 11000 data points with 16 input features from which one of 10 different classes should be predicted. The distribution of the classes is well balanced. The input values range from $[0, 100]$ and are rescaled to $[0, 1]$ in the following. The dataset is already split into test and train data and consists of 7494 training and 3498 testing samples. Since the dataset requires the search for a suitable architecture of the network, the training data is split into training 66% and validation 33% data which is used for finding the hyper-parameters and stopping the training preemptively to prevent over-fitting.

Architecture selection for pendigits

Since the task is more challenging, the architecture of the network is selected via grid-search similar to Section 3.4. The best model is thereby selected based on the validation error. The details of architecture and hyper-parameter search are as follows.

From preliminary experiments, a two hidden-layer architecture was chosen with 15 to 20 neurons in the first and 10 to 15 neurons in the second hidden-layer. As in the previous experiments, the values of $\lambda = \{0, 0.1, 0.01, 0.001\}$, a learning rate $\alpha = \{1, 0.1, 0.01\}$ and margins $m = \{0, 0.1, \cdots, 0.9, 1.0\}$ are explored. Each network is trained for up to 2000 epochs with the Adam [84] optimizer in standard configuration ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$) using full-batch updates. The learning rate is halved every 100 epochs. To save training time and improve generalisation, the learning procedure is stopped preemptively if the network does not surpass the baseline MaA of about 0.1 after 100 epochs, or if the validation MaA does not improve over 20 steps (see *early stopping* [56, Alg. 7.1]). Finally, as in the previous experiments, the measuring threshold is set to $T = 0.1$ and the feasible conductance range is set to $g \in [0.01, 1]$. Each configuration is run with a single seed only, to reduce the required time of the experiment. In total, this leads to $(5 \times 5) \times 3 \times 4 \times 11 = 3300$ (neurons $\times$ neurons $\times \alpha \times \lambda \times m$) configurations. Only networks with feasible parameters are considered, and the evaluation is done using the MaA (see Section 3.4.1).

5.3.2 Experiments using measured circuit models

The networks considered in the following use the (nominal) models of inv and ptanh extracted from measurement data (see Section 3.2.2). As stated earlier, this configuration only allows to evaluate variation in the conductances of the crossbar resistors.

The best models for each dataset are found via the described grid-search procedure. For *pendigits*, the best model uses a configuration of $\lambda = 0.001$, $\alpha = 0.01$ and $m = 0.5$, with 19 and 14 neurons in the first and second hidden layer. After the projection of infeasible surrogate conductances to zero, the network achieved a train, validation and test MaA of 0.9586, 0.9483 and 0.9085 respectively. For the other datasets, the grid-search results of Chapter 3 can be used. The results and parameter configurations can be found in Table 3.1 and Table 3.2 respectively.

Conductance variation

The results for applying conductance variation on *pendigits* can be seen in Figure 5.4. Clearly, the higher the variation, the stronger the observed degradation in MaA. Furthermore, the standard deviation of the MaA results increases with the level of variation ϵ.

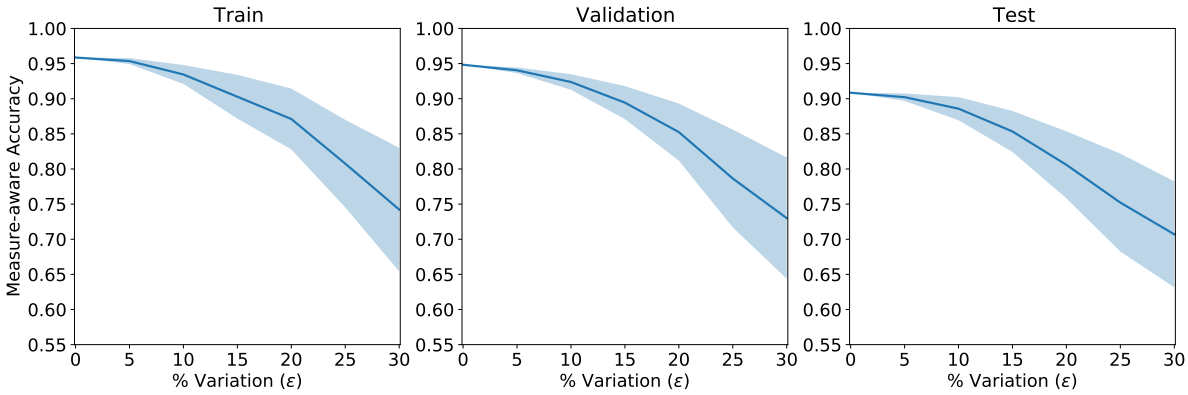


Figure 5.4: The MaA degradation of the best model for the *pendigits* dataset over various levels of variation ϵ. The solid line displays the mean MaA, and the shaded area represents the standard deviation for 100 sampled sets of surrogate conductances.

When the same network is retrained with different levels of training variation v, a higher robustness to the test variation ϵ can be observed (see Figure 5.5). Additionally, networks trained with higher v also tend to exhibit lower MaAs when evaluated under lower ϵ. Most notably $\epsilon = 0$. This observation suggests that there might be a trade-off between the robustness of the solution and high accuracy under $\epsilon = 0$. However, this does not need to be the case and would depend on the loss surface.

Overall, the results suggest that training with higher v leads to solutions that are more robust and tend to degrade less from the $\epsilon = 0\%$ performance. This can also be observed in Figure 5.6, where the relative degradation with respect to the MaA at $\epsilon = 0$ is displayed. Furthermore, when training with higher levels of variation v, the standard deviation is generally lower (see Figure 5.7), indicating that the found parameters $\boldsymbol{\omega}$ tend to not only produce better results under variation on average, but also lead to more stable results overall.

To summarize, on *pendigits*, training with variation may lead to a lower MaA at $\epsilon = 0\%$,

but is able to produce better results for higher levels of test variation ϵ. Additionally, networks trained with higher v tend to exhibit a lower standard deviation in the achieved MaAs.

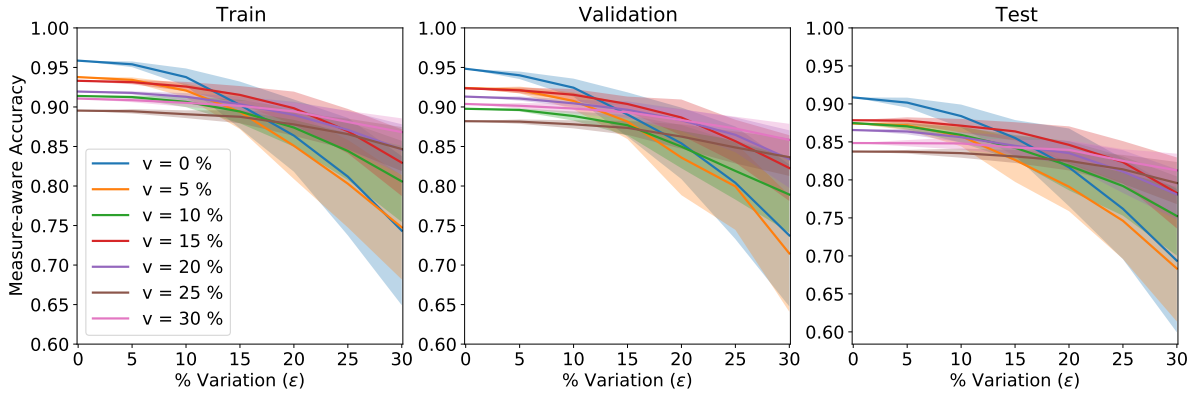


Figure 5.5: Mean and standard deviation of the MaA trained with different levels of conductance variation v.

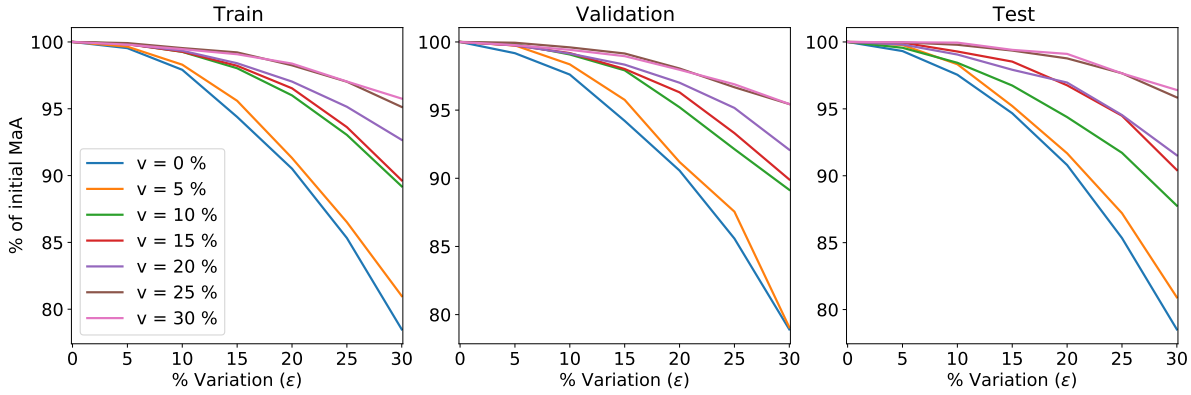


Figure 5.6: The relative degradation of the MaA trained with different levels of conductance variation v. The results are reported with respect to the initial value at $\epsilon = 0\%$.

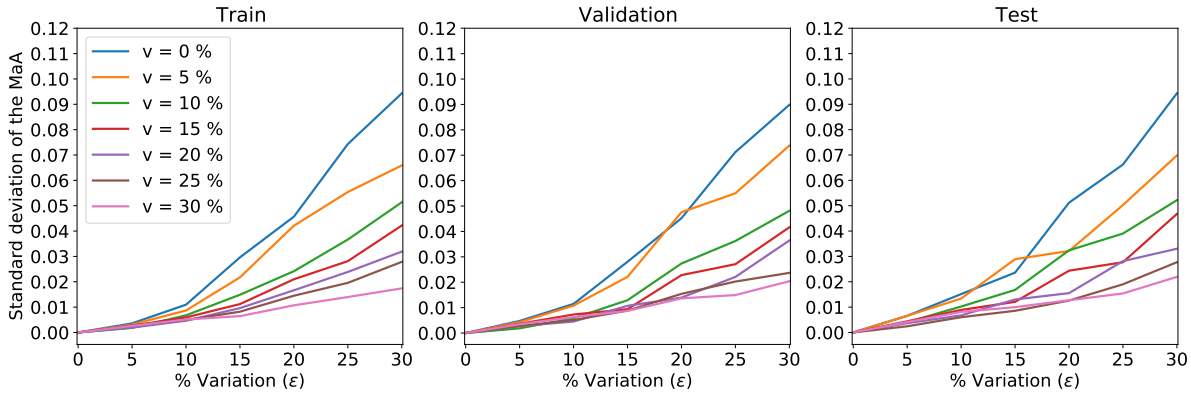


Figure 5.7: The standard deviation of the MaA trained with different levels of conductance variation v.

To further test these hypotheses, variation-aware training is applied to the datasets from Section 3.4. Each dataset is evaluated and tested for the variation level it is trained on, i.e. $v = \epsilon$. Hence, it is assumed that the true variation level is known. The results are compared

with the respective results of the optimal network trained without variation. For this, each network (trained with and without variation) is retrained 10 times for the respective variation level ϵ using different seeds. The networks achieving the highest average training MaA for 100 sampled sets of parameters are reported in Table 5.1.

		Trained with variation ($v = \epsilon$)			Trained without variation		
Dataset	0 %	10 %	20 %	30 %	10 %	20 %	30 %
Acute Inflammations	1.00	1.00 ± 0.00	0.94 ± 0.08	0.91 ± 0.09	0.99 ± 0.04	0.93 ± 0.10	0.85 ± 0.15
Balance Scale	0.90	0.87 ± 0.01	0.86 ± 0.02	0.84 ± 0.03	0.88 ± 0.03	0.85 ± 0.04	0.82 ± 0.05
Breast Cancer Wisconsin	0.97	0.97 ± 0.00	0.97 ± 0.01	0.97 ± 0.01	0.98 ± 0.01	0.97 ± 0.01	0.96 ± 0.02
Energy efficiency (y1)	0.87	0.83 ± 0.02	0.81 ± 0.03	0.79 ± 0.04	0.86 ± 0.02	0.80 ± 0.09	0.73 ± 0.07
Energy efficiency (y2)	0.91	0.88 ± 0.03	0.84 ± 0.03	0.82 ± 0.02	0.88 ± 0.04	0.81 ± 0.05	0.77 ± 0.07
Iris	0.94	0.94 ± 0.04	0.88 ± 0.09	0.86 ± 0.11	0.90 ± 0.07	0.83 ± 0.12	0.74 ± 0.14
Mammographic Mass	0.84	0.80 ± 0.01	0.78 ± 0.03	0.77 ± 0.05	0.82 ± 0.03	0.76 ± 0.05	0.72 ± 0.11
Seeds	0.94	0.93 ± 0.03	0.91 ± 0.05	0.86 ± 0.07	0.92 ± 0.04	0.88 ± 0.07	0.82 ± 0.10
Tic-Tac-Toe Endgame	0.97	0.86 ± 0.02	0.80 ± 0.01	0.76 ± 0.03	0.90 ± 0.07	0.76 ± 0.04	0.70 ± 0.06
Vertebral Column (2 classes)	0.87	0.83 ± 0.03	0.80 ± 0.04	0.75 ± 0.03	0.83 ± 0.04	0.78 ± 0.07	0.74 ± 0.09
Vertebral Column (3 classes)	0.83	0.71 ± 0.03	0.61 ± 0.10	0.51 ± 0.14	0.66 ± 0.11	0.55 ± 0.18	0.45 ± 0.21

Table 5.1: The test mean and standard deviation of the MaA results under conductance variation. For each configuration, the training was performed with 10 different seeds and the mean and standard deviation of the test MaA of the network achieving the best average training MaA over 100 samples is reported.

In line with the observations on *pendigits*, the average MaA degrades for higher levels of variation. For lower variation levels of 10%, training under variation leads to worse results, see *Tic-Tac-Toe Endgame*. However, for $\epsilon = 20\%$, training with variation produces similar or better results. For $\epsilon = 30\%$, networks trained with variation achieve strictly better average MaAs. Furthermore, irrespective of the achieved MaA, the results for training with variation display lower standard deviations across all results compared to their counterparts trained without variation. This further supports the hypothesis that more robust solutions are found through variation-aware training.

5.3.3 Experiments using simulated circuit models

The networks considered in the following use the circuit models extracted from data of circuit simulations ("simulated circuit models") as developed in Section 5.1.3. For the baseline, and when only conductance variation is considered, the $\epsilon = 0\%$ version of both functions is used. Since these adapted circuit models may influence the training dynamics, the grid-search procedure for finding the hyper-parameters is repeated. The results can be seen in Table 5.2 and the set of selected parameters is reported in Table 5.3.

Overall, the networks achieve a similar performance to the reference networks (same as before) and clearly surpass the baseline of a random guess. Additionally, their achieved results are within about 3 % of the previously achieved results of the pNNs using the measured circuit models (see Table 3.1). It is notable that almost all best performing networks choose a learning rate of $\alpha = 0.01$ on the low end of the search grid (see Table 5.3). Hence, exploring lower learning rates may lead to improved results for pNNs using the simulated circuit models. One reason for the preference towards these lower learning rates may be the increased steepness and thus higher sensitivity of the activation function (see Figure 5.3).

Dataset	Architecture	pNN		pNN (feasible)		reference NN		random guess
	neurons/layer	Train	Test	Train	Test	Train	Test	
Acute Inflammations	6-4-3-2	1	1	1	1	1	1	0.475
Balance Scale	4-4-3-3	0.9258	0.913	0.9258	0.913	0.9163	0.8889	0.4396
Breast Cancer Wisconsin	9-4-3-2	0.9786	0.974	0.9786	0.974	0.9765	0.9697	0.6667
Energy efficiency (y1)	8-4-3-3	0.8541	0.8622	0.8658	0.8543	0.8696	0.8583	0.4331
Energy efficiency (y2)	8-4-3-3	0.9105	0.9055	0.9163	0.9016	0.9105	0.9055	0.4646
Iris	4-4-3-3	0.99	0.96	0.98	0.96	0.97	0.98	0.28
Mammographic Mass	5-4-3-2	0.818	0.8019	0.818	0.8019	0.818	0.8113	0.5503
Seeds	7-4-3-3	0.9643	0.9714	0.9643	0.9714	0.9571	0.9571	0.2714
Tic-Tac-Toe Endgame	9-4-3-2	0.9922	0.9748	0.9922	0.9748	0.9922	0.9716	0.6404
Vertebral Column (2 classes)	6-4-3-2	0.8502	0.8544	0.8357	0.8738	0.8551	0.8835	0.6893
Vertebral Column (3 classes)	6-4-3-3	0.8164	0.8155	0.8213	0.8155	0.7681	0.7864	0.5146

Table 5.2: The training results (test MaA) of the best pNNs when using the simulated circuit models. The results of the reference network and the random guess baseline are identical to the results in Table 3.1.

Models	pNN			reference NN		
Parameters	α	m	λ	α	m	λ
Acute Inflammations	0.10	0.4	0.01	1.0	0.1	0.000
Balance Scale	0.01	0.7	0.01	1.0	0.7	0.000
Breast Cancer Wisconsin	0.01	0.3	0.01	1.0	0.6	0.000
Energy efficiency (y1)	0.01	0.1	0.10	1.0	0.4	0.001
Energy efficiency (y2)	0.01	0.7	0.10	1.0	0.7	0.001
Iris	0.01	0.5	0.01	1.0	0.7	0.010
Mammographic Mass	0.01	0.9	0.10	1.0	1.0	0.001
Seeds	0.01	1.0	0.10	1.0	0.5	0.001
Tic-Tac-Toe Endgame	0.01	0.6	0.10	0.1	0.7	0.000
Vertebral Column (2 classes)	0.01	0.6	0.00	0.1	0.7	0.000
Vertebral Column (3 classes)	0.01	1.0	0.10	0.1	0.9	0.000

Table 5.3: The parameter configurations of the best pNNs using the simulated circuit models.

Consequently, also the grid-search from Section 5.3.1 is repeated to find new hyper-parameters for *pendigits*. Here, training, validation and test MaAs of 0.8789, 0.8763, 0.833 are achieved respectively under a configuration of $\alpha = 0.01$, $\lambda = 0.001$ and $m = 0.2$ with 15 and 13 neurons in the hidden layers. Hence, the best network uses the same configuration for α and λ as for the measured circuit models, but has fewer neurons in both hidden layers (previously 19 and 14). Additionally, the training margin m is lower with 0.2 versus 0.5. With the achieved results, the best found network reaches an about 7% points lower MaA than the best previously found network with the measured circuit models. Note that this has nothing to do with either one reflecting the reality better (they are both evaluated purely computational), but solely with the suitability of their model for training. However, as always, better results may be achievable by an extended hyper-parameter search.

The worse performance of the simulated circuit models may for example be explained by the higher steepness of the ptanh function. Through this, the learning algorithm obtains mostly small gradients which could hinder the learning. Additionally, its root is further from zero requiring a higher bias for compensation (see Section 3.3.4). Since the weights and the bias

must sum up to one, the increased bias may leave less freedom for the values of the weights. Uncovering the exact mechanism would however require a detailed set of experiments that is beyond what is intended in the chapter.

Conductance variation

As expected, subjecting the network of *pendigits* to conductance variations clearly leads to a degradation in the average MaA (see Figure 5.8). However, compared to the previous results for the measured circuit models (Figure 5.4), the performance of the network decreases more drastically under variation.[8] This effect may be attributed to either the use of the simulated circuit models, or the selected set of hyper-parameters/architecture.

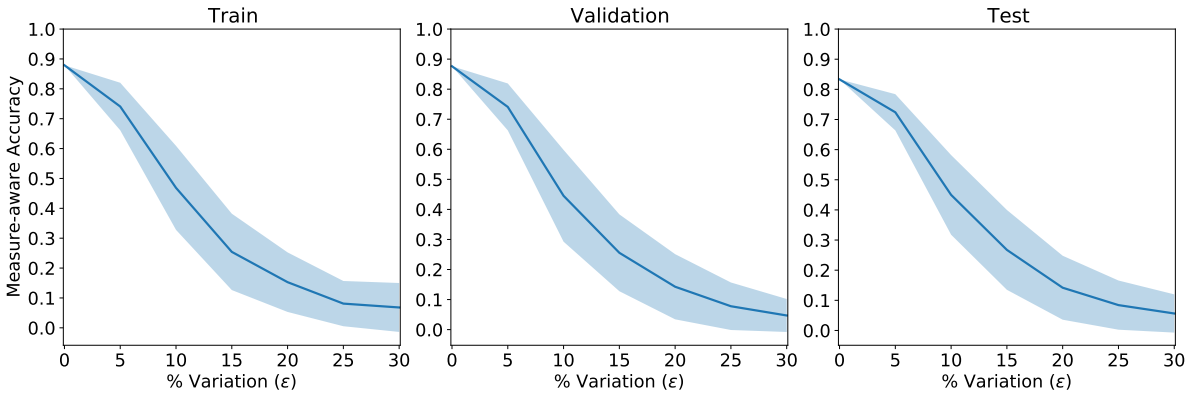


Figure 5.8: The MaA degradation of the best model for *pendigits* under various levels of conductance variation when the simulated circuit models are used. The solid line displays the mean MaA, and the shaded area represents the standard deviation for 100 sampled sets of the parameters.

While the effect of the circuit models on the robustness is hard to assess, assumptions can be made about the influence of the hyper-parameters. For example, the choice of the margin parameter m could have a robustifying effect and was chosen as 0.2 here compared to 0.5 before. For higher m, the network is encouraged to place the outputs further away from the misclassification thresholds T and 0. Thus, small perturbations of the output induced by variations may not lead to a misclassification immediately. However, training with higher m might in turn also lead to solutions which exhibit a higher sensitivity with respect to parameter perturbations. Such an increased sensitivity could then nullify eventual gains in robustness.

With respect to variation-aware training, the results for training prove less stable than under the measured circuit models. The training for each variation level v (including $v = 0$) is thus repeated with 10 different seeds. For each v, the network achieving the best average validation MaA at $\epsilon = 0$ is displayed in Figure 5.9.

Although the conductance variation has a severe effect on the initial network ($v = 0\%$), training with variations greatly improves the results for higher levels of ϵ. Most notably, some networks trained with variation even achieve similar average MaAs at $\epsilon = 0$ to the network trained without variation. However, increasing v did not always lead to better results. This is

[8] The severity of this difference is revealed by noticing that the y-axis of the new results is displayed for an MaA range from 0 to 1, while before, the minimum displayed value was 0.6.

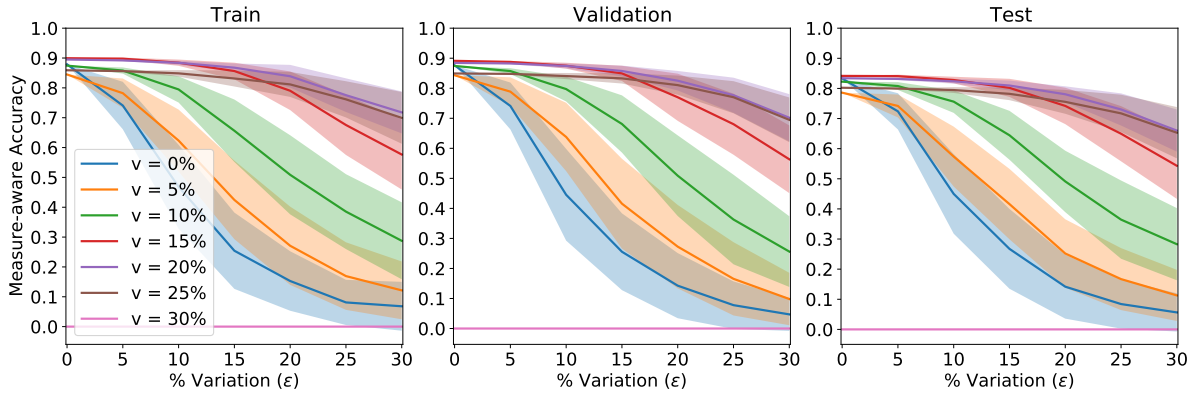


Figure 5.9: Mean and standard deviation of MaA trained with different levels of conductance variation. All networks use the simulated circuit models.

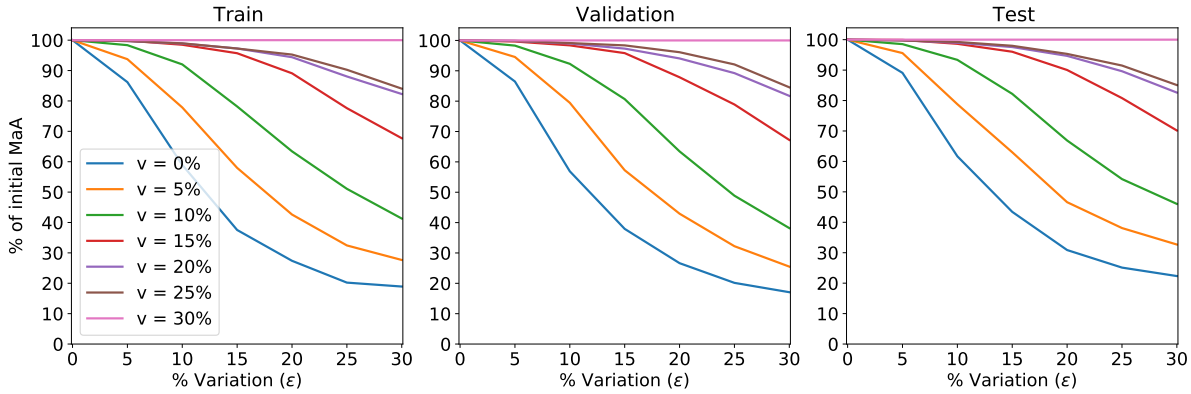


Figure 5.10: Degradation of the mean MaA trained with different levels of conductance variation. The results are reported with respect to the initial value for $\epsilon = 0\%$ variation. All networks use the simulated circuit models.

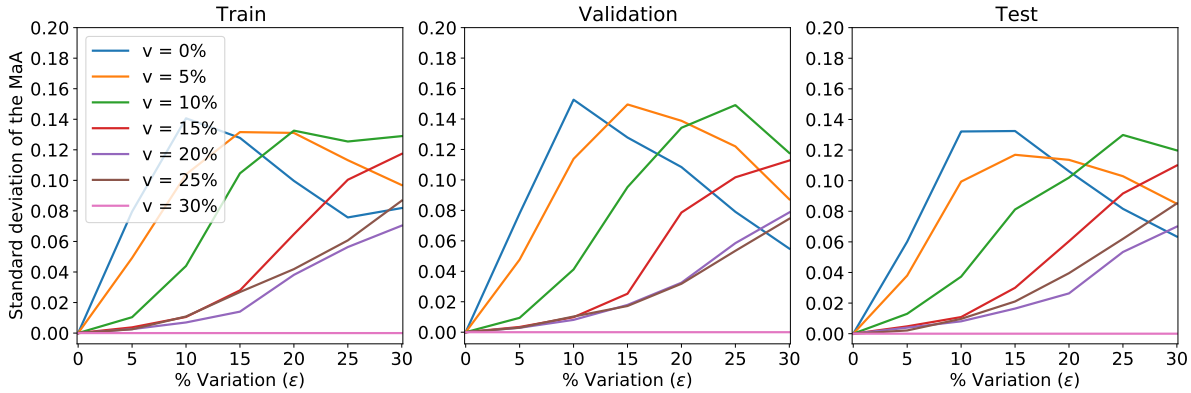


Figure 5.11: Standard deviation of the MaA trained with different levels of conductance variation. Note that the standard deviations of networks with low average MaA values are likely decreased due to the MaA being bounded at zero. All networks use the simulated circuit models.

evident by the fact that no network with an acceptable accuracy could be found for $v = 30\%$ over 10 different seeds. On the other hand, consistent with previous observations, the relative

		Trained with variation ($v = \epsilon$)			Trained without variation		
Dataset	0 %	10 %	20 %	30 %	10 %	20 %	30 %
Acute Inflammations	1.00	0.82 ± 0.00	0.85 ± 0.07	0.81 ± 0.03	0.98 ± 0.05	0.91 ± 0.15	0.74 ± 0.19
Balance Scale	0.91	0.84 ± 0.03	0.81 ± 0.04	0.77 ± 0.05	0.78 ± 0.08	0.66 ± 0.12	0.56 ± 0.14
Breast Cancer Wisconsin	0.97	0.97 ± 0.00	0.97 ± 0.01	0.97 ± 0.01	0.96 ± 0.02	0.83 ± 0.22	0.69 ± 0.27
Energy efficiency (y1)	0.85	0.71 ± 0.12	0.67 ± 0.18	0.72 ± 0.14	0.34 ± 0.23	0.23 ± 0.21	0.16 ± 0.18
Energy efficiency (y2)	0.90	0.85 ± 0.04	0.82 ± 0.02	0.82 ± 0.02	0.78 ± 0.11	0.62 ± 0.15	0.49 ± 0.20
Iris	0.96	0.92 ± 0.08	0.88 ± 0.10	0.81 ± 0.13	0.83 ± 0.15	0.60 ± 0.21	0.44 ± 0.24
Mammographic Mass	0.80	0.70 ± 0.10	0.55 ± 0.00	0.53 ± 0.09	0.61 ± 0.11	0.54 ± 0.10	0.52 ± 0.08
Seeds	0.97	0.91 ± 0.04	0.89 ± 0.05	0.83 ± 0.08	0.87 ± 0.08	0.70 ± 0.13	0.55 ± 0.18
Tic-Tac-Toe Endgame	0.97	0.91 ± 0.03	0.79 ± 0.04	0.73 ± 0.04	0.82 ± 0.15	0.66 ± 0.18	0.55 ± 0.19
Vertebral Column (2 classes)	0.87	0.77 ± 0.03	0.71 ± 0.05	0.68 ± 0.07	0.74 ± 0.07	0.66 ± 0.10	0.61 ± 0.14
Vertebral Column (3 classes)	0.82	0.68 ± 0.02	0.38 ± 0.07	0.36 ± 0.11	0.62 ± 0.12	0.54 ± 0.16	0.49 ± 0.16

Table 5.4: The test mean and standard deviation of the MaA results for variation-aware training. Each entry displays the average test performance of the respective training configuration initialized with 10 different seeds. The mean and the standard deviation are calculated using 100 samples. All networks use the simulated circuit models for the activation and the inverter function.

MaA degradation is lower for networks trained with higher v (see Figure 5.10).[9] Finally, the standard deviation of the achieved MaA also (initially) decreases with the employed training variation v (see Figure 5.11). In contrast to before (Figure 5.7), the standard deviation does not increase monotonically with ϵ. A likely reason for this is that the MaA, as well as the accuracy, can only take on values between zero and one. This possibly leads to smaller standard deviations for poorly performing networks since they cannot achieve an accuracy below zero. The same effect would obviously also occur for high performing configurations with a mean MaA close to one.

Similar effects can also be observed for the other datasets (see Table 5.4). The results are generated, as earlier, by retraining each model for each variation level and reporting the $v = \epsilon$ results. For both, the networks trained with and without variation, the training is repeated 10 times (for 10 seeds), and the test MaA of the networks achieving the best average training MaA for the given ϵ is reported.

It can be seen that employing variation-aware training almost universally leads to better results for all reported variation levels. Most notably, for $\epsilon = 30\%$, the improvement is more than 20% points in average MaA on about half of the datasets. However, *Vertebral Column (3 classes)* (for $\epsilon \geq 20\%$) and *Acute Inflammations* (for $\epsilon < 30\%$) form an exception. Since this is inconsistent with the previous observation, it is likely either due to the use of the simulated circuit models, the hyper-parameters, or unfortunate training dynamics.

When comparing the overall results for the simulated circuit models to the previous results from the measured circuit models (Table 5.1), a clear pattern emerges. Training with the circuit models extracted from the simulation data almost universally leads to a worse average MaA under conductance variation. This is true for both, the networks trained with and without variation. This suggests that the shape of the simulated circuit models may generally be less suitable for training. Hence, circuits (or circuit models) exhibiting similar characteristics, e.g., steepness and strongly shifted root, should be avoided if possible. Nevertheless, variation-aware

[9] This does not mean anything for the bad performing networks at $v = 30\%$ though.

training manages to greatly improve the average MaA under variation in both cases.

Conductance and circuit model variation

Finally, the networks are subjected to both, conductance and circuit model variation. The circuit model variations are thereby expressed through variations of the circuit model parameters $\boldsymbol{\eta}$ for different variation levels ϵ (details see Section 5.1.3). In contrast to before, every neuron now has its own activation and inverter function depending on the sampled parameters $\boldsymbol{\eta}$. Consequently, for variation-aware training, the gradient is also estimated based on both, variation of the conductances, as well as variations of the parameters $\boldsymbol{\eta}$ (details see Section 5.2.2).

When applying the full variation to *pendigits*, the MaA already drops to around 0.4 at $\epsilon = 5\%$ (see Figure 5.12). Unsurprisingly, this is substantially lower than when only conductance variation was applied (with about 0.7 at $\epsilon = 5\%$).

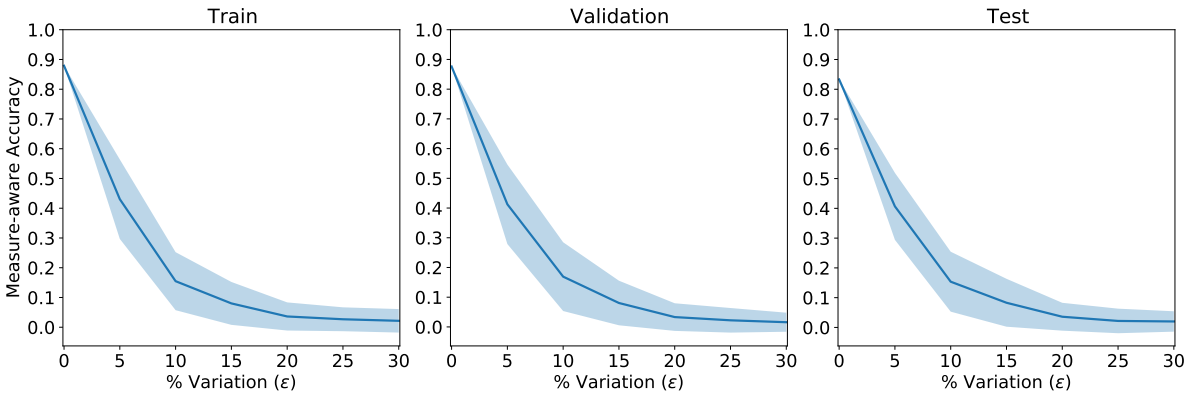


Figure 5.12: The MaA results on *pendigits* under various levels of variation ϵ. The variation is applied to the conductances of the crossbar resistors and the circuit model parameters $\boldsymbol{\eta}$ of inv and ptanh. The solid line displays the mean MaA, and the shaded area represents the standard deviation for 100 sampled sets of parameters.

For variation-aware training on *pendigits*, the training for each variation level v (including $v = 0$) is repeated with 10 different seeds. As before, the networks achieving the best average validation MaA at $\epsilon = 0$ are displayed (see Figure 5.13). Evidently, also under the joint variation of conductances and the circuit models, variation-aware training manages to drastically improve the average MaA. The improvement is especially notable in the range of $\epsilon = 5\%$ to $\epsilon = 15\%$. However, the achieved MaA are generally much lower compared to when only conductance variation was applied. For $v \geq 25\%$, none of the found networks displays an acceptable accuracy. Hence, as already observed before, choosing v too high can lead to bad results. Finally, consistent with the previous observations, networks trained with higher v exhibit less relative degradation in MaA (see Figure 5.14) and display a lower standard deviation in their results (see Figure 5.15). Note again that the unintuitive decrease in standard deviation with respect to higher ϵ is likely due to the limited value range of the MaA.

To conclude the experiments, the procedure is applied to the other datasets. As before, all networks are trained 10 times using different seeds and the best performing networks are reported. The results can be seen in Table 5.5. Unsurprisingly, and in line with the observations on *pendigits*, the combined variation almost universally leads to worse results for both training with and without variation. The only exception to this is *Acute Inflammations*, for which

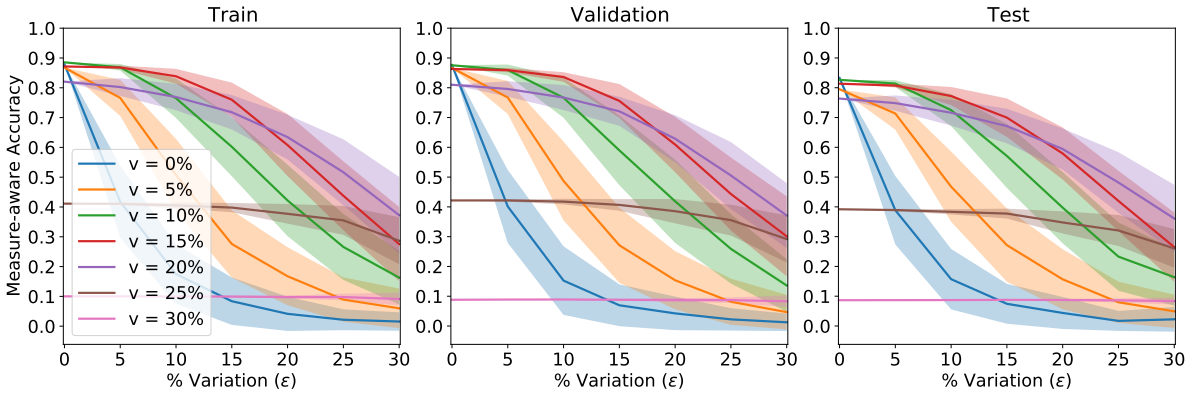


Figure 5.13: Mean and standard deviation of the train, validation and test MaA trained with different levels of variation v.

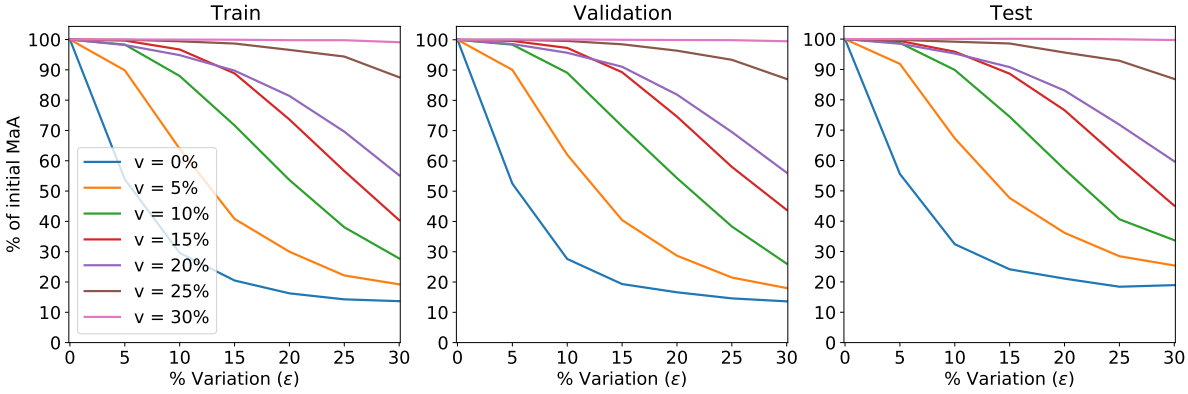


Figure 5.14: Degradation of the MaA trained with different levels of variation v. The results are reported with respect to the initial value for $\epsilon = 0\%$ variation.

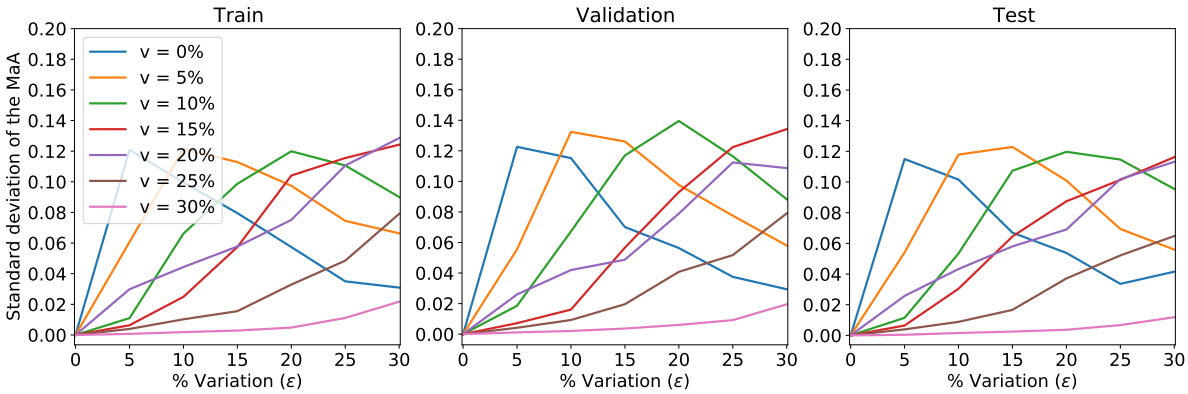


Figure 5.15: Standard deviation of the MaA trained with different levels of variation v.

training under the combined variation performs better than under conductance variation only. This is likely due to unfortunate training dynamics in the latter case. Consistent with previous results, networks trained with variation almost always outperform networks trained without variation for all variation levels $\epsilon \geq 10\%$. The exceptions are *Mammographic Mass*, where both perform comparable, and *Vertebral Column (3 classes)*, where variation-aware training yields worse results for $\epsilon = 15\%$ and $\epsilon = 20\%$.

		Trained with Variation ($v = \epsilon$)			Trained without variation		
Dataset	0 %	10 %	20 %	30 %	10 %	20 %	30 %
Acute Inflammations	1.00	1.00 ± 0.02	0.98 ± 0.05	0.88 ± 0.13	0.85 ± 0.10	0.75 ± 0.20	0.64 ± 0.24
Balance Scale	0.91	0.81 ± 0.04	0.75 ± 0.09	0.63 ± 0.13	0.71 ± 0.11	0.57 ± 0.15	0.41 ± 0.20
Breast Cancer Wisconsin	0.97	0.97 ± 0.01	0.95 ± 0.03	0.90 ± 0.15	0.92 ± 0.11	0.64 ± 0.28	0.46 ± 0.29
Energy efficiency (y1)	0.85	0.66 ± 0.16	0.70 ± 0.14	0.63 ± 0.18	0.28 ± 0.20	0.19 ± 0.19	0.12 ± 0.17
Energy efficiency (y2)	0.90	0.82 ± 0.04	0.54 ± 0.12	0.50 ± 0.13	0.72 ± 0.12	0.47 ± 0.15	0.31 ± 0.23
Iris	0.96	0.91 ± 0.08	0.56 ± 0.13	0.47 ± 0.17	0.68 ± 0.19	0.42 ± 0.19	0.28 ± 0.20
Mammographic Mass	0.80	0.55 ± 0.00	0.49 ± 0.17	0.42 ± 0.24	0.55 ± 0.10	0.51 ± 0.12	0.41 ± 0.20
Seeds	0.97	0.88 ± 0.06	0.58 ± 0.07	0.53 ± 0.12	0.72 ± 0.14	0.47 ± 0.15	0.34 ± 0.16
Tic-Tac-Toe Endgame	0.97	0.81 ± 0.03	0.71 ± 0.07	0.63 ± 0.09	0.73 ± 0.19	0.54 ± 0.19	0.47 ± 0.21
Vertebral Column (2 classes)	0.87	0.72 ± 0.06	0.68 ± 0.07	0.66 ± 0.13	0.69 ± 0.09	0.58 ± 0.15	0.49 ± 0.24
Vertebral Column (3 classes)	0.82	0.59 ± 0.07	0.28 ± 0.19	0.06 ± 0.15	0.56 ± 0.18	0.46 ± 0.16	0.37 ± 0.21

Table 5.5: The test mean and standard deviation of the MaA results for variation-aware training for conductance and circuit model parameter variations of ptanh and inv. Each entry displays the average test performance of the respective training configuration initialized with 10 different seeds. The mean and the standard deviation are calculated through 100 sets of sampled parameters.

5.3.4 Summary of experimental observations

To conclude, variation-aware training almost always increased the accuracy (MaA) under simulated conductance variations. Additionally, the achieved results generally exhibit a lower standard deviation. This was found independent of using the circuit models for the ptanh and inv function derived from measurement data or from data extracted through circuit simulations. In the case of the latter, the achieved MaAs were generally worse compared to networks trained with the circuit models extracted from measurement data. For some datasets, e.g., *pendigits*, this effect could also be partly explained through a lower training margin m, which might also lead to more robust results. However, the effect was almost universally observed on all datasets irrespective of the values of m. Hence, with respect to the design of activation and inverter function circuits, circuits displaying characteristics similar to the simulated circuit models should be avoided.

While finding ideal shapes would require a detailed analysis, it may be conjectured that very steep functions with roots far from zero are unfavourable. This could be because activation functions with steep slopes have larger regions with small gradients that lead to slow learning. On the other hand, if the root of the activation function is far from zero in a pNN, a large bias is required to keep the activation around zero where good gradient signals can be obtained. However, due to the coupling of the weights and the bias, a large bias also limits the magnitude of the weights and makes them less flexible overall.

By construction, training with a specified variation level v should lead to the best results under $\epsilon = v$. However, this is seldom the case in the experiments. For example, $v = 15\%$ provides better results at $\epsilon = 10\%$ than $v = 10\%$ in Figure 5.13. Such observations could be explained by various factors. First of all, the quality of the achievable results depends on the loss surface and the training dynamics. For example, the solution (parameters) for $v = 15\%$ and $v = 20\%$ may have the same location and could be found by either value of v. However, one of the choices may lead to a better training process, e.g., provide better (estimated) gradients. Hence, the optimisation is guided more effectively and thus achieves better training results.

The possibly high variance of the estimated gradients may also be a reason for training runs to fail completely, e.g., for $v \geq 20\%$ in Figure 5.13. A natural way to combat this would be to use more samples for the estimation of the gradients. However, this also increases the training time. Finally, it should be noted that since neural network training problems are generally nonconvex, finding good, let alone optimal solutions cannot be guaranteed. Additionally, the insufficient smoothness of the pNN model may hinder the progress of gradient-based learning approaches.

5.4 Conclusion and directions for future work

This chapter introduced a variation-aware training procedure for pNNs. As a first step, variation models of the conductances of the crossbar resistors and the activation and inverter functions were developed. The model for the conductance variation is based on relative, uniform variation, while the variation models of the circuit components were developed through a data-driven approach using data generated by circuit simulations. Based on these variation models, the training procedure for pNNs was modified to minimize the expected loss under the modelled variations (variation-aware training). To find a solution to this stochastic optimisation problem, the gradients were estimated using Monte Carlo gradient estimation. With these gradients, the techniques described in Chapter 3 can be employed for learning.

The method was evaluated experimentally by training and testing pNNs on a variety of datasets. Each network was thereby trained and evaluated under several levels of variation. From the experiments, it can be concluded that training with this objective often leads to an increased expected accuracy (MaA) under variation. This is especially prominent when considering multiple sources of variation and higher variation levels. Training pNNs this way should therefore lead to printed NCs that display better results after fabrication. It was also observed that the choice of the circuit (models) for the activation and inverter function influences the sensitivity of the networks. Hence, such effects should be considered when designing new circuit components in the future.

Besides the proposed variation models, the general methodology may also be adapted for more complex, physically motivated or data-driven variation models. For this, two main cases should be distinguished. In short, if the variation, i.e., distribution of the varying quantity, can be influenced by learnable parameters or not. If the choice of the learnable parameters does not influence the variation, the derivation of gradient estimates is straightforward and simply relates to averaging evaluations of the gradient for sampled sets of parameters. This was the case of the component variation models for inv and ptanh. However, if the learnable parameters influence the distribution, e.g., in the case of the relative variation of the conductances, the applicability of the pathwise gradient has to be re-evaluated and an appropriate sampling path needs to be found. Depending on the variation model, also other types of gradient estimators may be required. Finally, note that also further sources of variations, such as input variations, 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.

To improve the robustness of the found networks, it may not be required to train with the exact variation model. Trying to minimize the expected loss under any variation may generally

help to find "flat" regions in the loss surface that are robust against perturbations. However, in this work, no experiments were carried out to support this hypothesis.

As a future direction of work, closed-form variation models for the circuit components based on their conductances and transistor sizes could be developed. The transistor sizes and conductances could then be considered as learnable parameters similar to the conductances of the crossbar resistors. To efficiently learn those parameters, the models should ideally be developed such that they qualify for the use of the pathwise gradient estimator. Through this, the network would be able to learn shapes of the respective activation and inverter functions that not only benefit the learning dynamics, but also the robustness of the pNNs and should thus lead to more robust NCs. Note that fabricating activation and inverter functions individually for each neuron should not be a problem given the on-demand printing capabilities of inkjet printing.

To speed up the learning process, the number of samples used for gradient estimation could be adjusted dynamically. For example, in the early stages of learning, gradients estimates based on a lower number of samples may be sufficient to decrease the loss. Then, towards the end of learning, the number of samples could be increased to achieve lower variance estimates and improve the convergence. Alternatively, in each gradient estimation step, samples could be drawn until no significant changes in the estimate are observed over a certain number of samples. Note however that the samples would not be independent in this case.

Aside from variations, other aspects that degrade the performance of printed NCs may already be anticipated in training. One of these could be accuracy degradation due to aging of the components. In a similar manner to variation models, aging models of the parameters could be developed. These models could then be used to pose the training problem of pNNs as minimizing the loss over its lifetime. This could for example be formulated as $\int L\left(\boldsymbol{\theta}(t)\right) \mathrm{d}t$, where $\boldsymbol{\theta}(t)$ is given by an aging model. The integral could thereby be approximated via simple uniform discretization, Monte Carlo methods or any other approach that uses differentiable computations.

Even though the proposed approach of variation-aware training could strongly improve the average performance, not all pNNs displayed acceptable results. In these cases, the feature of additive manufacturing could be employed to partially reprint crossbar resistors of the printed NC and create the intended behaviour. One such approach is described in the next chapter.

6 Post-fabrication Tuning for Printed Neuromorphic Circuits

Through the variation-aware training technique proposed in the last chapter, the average accuracy of the pNNs under variation could be increased notably. Unfortunately, not all fabricated printed NCs may reach the desired quality. To improve the performance of these NCs and increase the fabrication yield, this chapter proposes a post-fabrication tuning procedure. The approach utilizes the unique feature of additive manufacturing to adjust the conductances of the crossbar resistors. For this, inkjet printing can be used to selectively add additional material on top of previously fabricated devices. Leveraging this capability, the procedure works as follows.

Based on an assessment of the actually fabricated conductances, the method aims to improve the accuracy of a pNN model for the printed NC. This is achieved by successively restoring the initially desired weights of the neurons. For the restoration of each neuron, the post-fabrication capabilities are limited by physical and technological constraints which have to be respected when choosing new conductance values to fabricate. For example, as printing additional material increases the conductive path, the value of the new conductances can only be larger than the already fabricated ones. To reduce the material consumption and save time, the approach aims to find the minimal adjustment required to restore the desired weights. The method is evaluated on several pNNs and datasets from the previous chapters.

The rest of this chapter is organized as follows. First, the preliminaries for the method are introduced. These relate primarily to the assessment of the conductance values of printed resistors and the capabilities of adapting them after fabrication. The introduction is followed by a discussion of modelling assumptions. After the preliminaries, the post-fabrication tuning methodology is outlined. The methodology is then evaluated, and the chapter closes with a discussion and possible directions for future work.

The results in this chapter are based on [67], which is accepted for publication by the time of writing this thesis.

6.1 Preliminaries and background

The following introduces preliminary concepts required for the post-fabrication procedure. The main points are thereby the assessment of printed resistors and a discussion of the possibilities to modify their conductance via additive manufacturing. For the procedure, only variations in the conductances of the crossbar resistors are considered. The activation and inverter function components are assumed to exhibit no variation (details see Section 6.1.3).

6.1.1 Assessment of printed resistors

As a first step, an assessment of the circuit characteristics of the fabricated NC is required. In particular, the conductances of the fabricated resistors must be obtained. This can be achieved by measuring all devices. Since this can be time consuming, only a certain subset of resistors may be measured. The subset could thereby be found by investigating resistors that display high variations in their geometries. Since printed electronic components usually have sizes >10µm, variations in the geometry could already be found via optical inspection through a commercial microscope. See Figure 6.1, for an example of printed resistors of different quality.

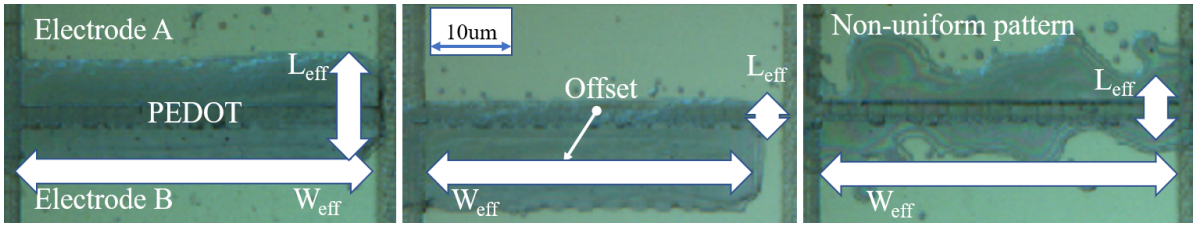


Figure 6.1: Printed (PEDOT) resistors of different quality. The device on the left is well printed, the middle resistor is misaligned, and the printing on the right shows a nonuniform pattern. Depending on the quality of the printing, the target width and length of the conductance is not achieved, influencing the conductive properties of the device. Image taken by Dennis D. Weller and adopted with permission.

Alternatively, if the printer is equipped with a camera (see [74]), the optimal inspection could be carried out while printing. The assessment of the nonuniformity of the printed geometries may be aided by appropriate computer-vision methods for detecting printed components (see [49]). The resistors found could then be measured after fabrication, while the other resistors can be assumed to have the intended values.

6.1.2 Modifying printed resistors

After the conductance values of the printed resistors have been assessed, they can be modified by selectively printing additional material on top of them. Such techniques have already been employed for tuning resistances in printed random number generators [50, 155]. Here, the amount of material that is added, i.e., the number of layers and their width, determines the final conductance value (see Figure 6.2).

Since adding material increases the size of the conductive channel, the conductance of a given device can only be increased through this procedure. This constraint, along with restrictions on the minimal size of printable structures, has to be respected when developing methods for

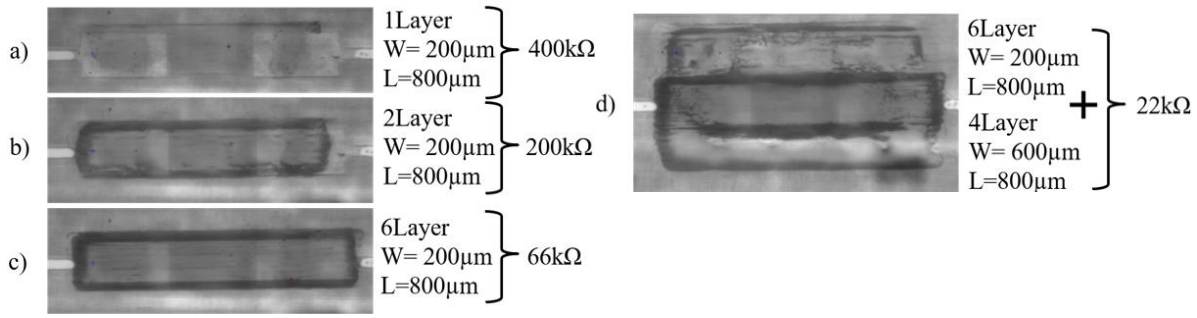


Figure 6.2: Printed (PEDOT) resistors of different geometries and conductance values. Through different geometries and number of printed layers, different conductance values can be achieved. Generally, the bigger the conductive channel, i.e., the more material is printed, the higher the conductivity of the device. Figure a) and b) and c) represent conductances of different values. Figure d) shows two combined printed geometries of different width and numbers of layers. The figure was adopted from [67].

post-fabrication tuning in the following.

6.1.3 Modelling assumptions

In the following, the assumptions for the post-fabrication procedure are outlined. First of all, it is assumed that only the crossbar resistors exhibit variation. The other circuit components, like the activation and inverter function, exhibit no variation. This could for example be justified in a split-manufacturing scenario, where the activation and inverter circuits were prefabricated through an industrial replication printing process with comparably little variations. Then, only the crossbar resistors are left to be fabricated to configure the circuit on-demand via inkjet printing.

The conductances $\boldsymbol{g}$ are denoted by $\boldsymbol{g}^{old}$ for the initially fabricated conductances and $\boldsymbol{g}^{new}$ denotes the targeted conductances after employing the post-fabrication tuning procedure. As before, individual conductances in the set/vector $\boldsymbol{g}$ are referred to as g_i.

1. The conductances g^{old} of the printed resistor crossbars can be accurately assessed.
2. A limited range of conductance $g_i \in [g_{min}, g_{max}]$ can be realized.
3. Reprinting increases the conductance of the resistor, hence $g_i^{new} \geq g_i^{old}$.
4. Reprinted resistors exhibit a lower variation $\epsilon_{reprint}$.

The first assumption is essential, as without an accurate assessment of the characteristics of the fabricated device, the results of the procedure may be erroneous.

The second assumption relates to the technology constraints and the feasible conductance ranges. It was also already considered when training pNNs (see Section 3.2.1). The value of the minimum conductance g_{min} is thereby determined by the printing resolution and the smallest feasible conductive pattern. On the other hand, the maximum value g_{max} is realized through the maximal width of the conductive channel printed with the maximum number of layers.

The third assumption states that the conductance can only be increased through reprinting. As printing on top of an old resistor creates an additional conductive path, the overall

conductivity is increased.[1]

From the second and third assumption follows that the adjustment made when reprinting may not be lower than a minimal value T, hence $g_i^{new} \geq g_i^{old} + T$. The adjustment distance T represents a technology constraint relating to the smallest printable conductive pattern of the printer used for tuning. If the same printer is used for fabrication and tuning, $T = g_{min}$ should be a reasonable choice.

The fourth assumption directly relates to the expected improvement achievable through reprinting. The variation introduced in reprinting should be lower than the initial fabrication variations, i.e. $\epsilon_{reprint} < \epsilon$, and may be realized through slower printing speeds.

Based on these assumptions, the next section describes a procedure to find a new, feasible set of conductances $\boldsymbol{g}^{new}$, which aims to restore the initially intended weights of a printed neuron.

6.2 Post-fabrication tuning for printed neural networks

The following outlines the post-fabrication tuning procedure. For the procedure, it is assumed that a pNN model was learned and mapped to a respective printed NC. After fabrication of the NC, it did not reach satisfying results due to variations in the fabrication process. This then triggers the post-fabrication procedure, where, as a first step, all conductances are assessed through measurements.[2] The pNN model is then updated with the actually measured conductances and a maximum number of k neurons is chosen for reprinting.

The procedure then successively identifies candidate neurons to reprint. For these neurons, adjustments of their resistances are calculated which allow for a restoration of their initially intended weights. The neurons are thereby chosen greedily based on the expected accuracy improvement for reprinting. After either k neurons have been identified for reprinting or no improvement is expected from additional reprinting steps, the adjustments for the selected neurons can be inkjet-printed. A flowchart for the full post-fabrication procedure can be seen in Figure 6.3.

The core of the post-fabrication procedure, i.e., the method for finding and restoring the intended behavior of a neuron, is described in the next subsection.

6.2.1 Restoring the weights of a printed neuron

For the training of the pNNs, the (surrogate) conductances are used as learnable parameters (see Chapter 3.3). They are preferred over directly finding a set of weights $\mathbf{w}$ as they allow a simpler treatment of the circuit and technology constraints. However, the weights $\mathbf{w}$, which are formed as a function of the conductances $\boldsymbol{g}$ through

$$w_i(\boldsymbol{g}) = \frac{g_i}{\sum_j g_j},$$

still determine the behaviour of a neuron. As long as the conductances form the correct weights, the pNN, and hence the printed NC, yield the correct output.

[1] Both resistors can be seen as connected in parallel, and their total conductances is given by the sum of the individual conductances.

[2] Alternatively, only the conductances displaying a mismatch in their geometric shape may be measured. For the other conductances, the intended values of the pNN model may be considered.

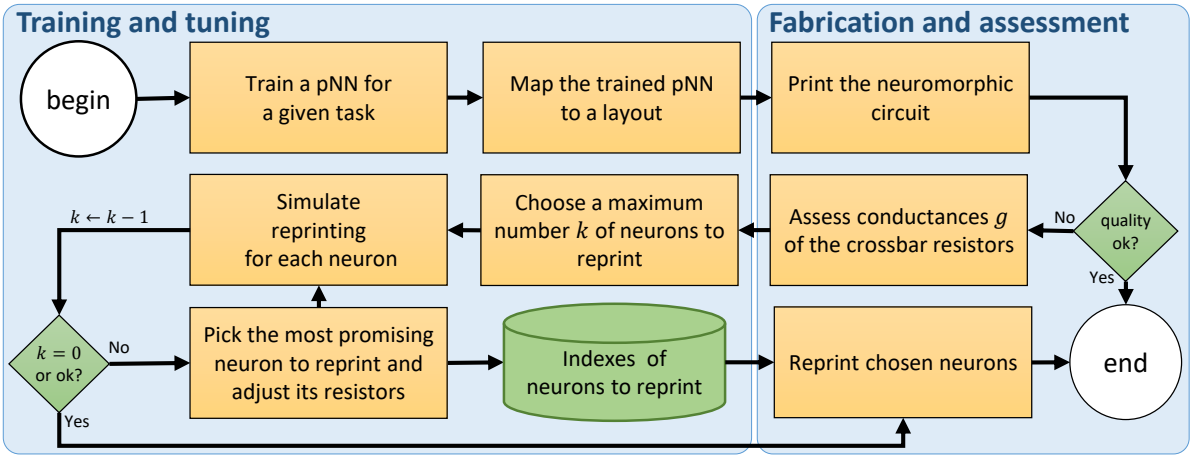


Figure 6.3: A flowchart of the full post-fabrication procedure. The left part is done in software and based on a printed neural network model. The right part relates to measurement and fabrication of the printed NC. The figure was adopted from [67] with slight modifications.

Note that, while the conductances $\boldsymbol{g}$ uniquely determine the weights $\mathbf{w}(\boldsymbol{g})$, the reverse is not true. In fact, the weights $\mathbf{w}(\boldsymbol{g})$ are scale-invariant with respect to the conductances $\boldsymbol{g}$, as for any scaling factor $s \in \mathbb{R}$,

$$w_i(s \cdot \boldsymbol{g}) = \frac{(s \cdot g_i)}{\sum_j (s \cdot g_j)} = \frac{\cancel{s} \cdot g_i}{\cancel{s} \cdot \sum_j g_j} = \frac{g_i}{\sum_j g_j} = w_i(\boldsymbol{g}). \tag{6.1}$$

In other words, there is a whole (half-)space of possible vectors $\boldsymbol{g}$ yielding the same weights $\mathbf{w}(\boldsymbol{g})$. This key observation forms the basis of the post-fabrication procedure.

In the following, for a given neuron, let $\boldsymbol{g}^\star$ be the conductances obtained from the training algorithm, and let $\boldsymbol{g}^{old}$ denote the actual conductance of the printed resistor after fabrication. Due to (6.1), $\mathbf{w}(\boldsymbol{g}^\star) = \mathbf{w}(s \cdot \boldsymbol{g}^\star)$. Thus, by finding a scaling factor s for which $\boldsymbol{g}^{new} = s \cdot \boldsymbol{g}^\star$ is feasible, the same, originally intended weights $\mathbf{w}(\boldsymbol{g}^\star)$ are realized. Additionally, the newly obtained conductances $\boldsymbol{g}^{new}$ suffer from less variation due to a slower, more careful fabrication. More specifically, as $\epsilon_{reprint} < \epsilon$ according to the assumptions.

In principle, any scaling factor s can be chosen for which the specified constraints hold, i.e., all new conductances $g_i^{new} \neq 0$ must lie in the feasible domain and have at least distance T from the initially fabricated g_i^{old}. Formally, this can be expressed through a system of inequalities

$$\begin{aligned} \forall i \text{ with } g_i^{old} \neq 0: \quad & s \cdot g_i^\star \geq T + g_i^{old} \\ & s \cdot g_i^\star \leq g_{max} \end{aligned}$$

of the single variable s. All conductances g_i with $g_i^{old} = 0$ do not require an adjustment and directly imply $g_i^{new} = 0$. Note that depending on the respective $\boldsymbol{g}^{old}$ and g_{max}, such an s may not exist.

To save time and material, the optimal scaling factor $s^\star$ should be chosen such that an overall

minimal adjustment for all conductances $\boldsymbol{g}$ is required. This can be formalized by

$$\begin{aligned} s^\star = \quad & \underset{s \in \mathbb{R}^+}{\operatorname{argmin}} \quad \sum_i |s \cdot g_i^\star - g_i^{old}| \\ & \text{s.t} \\ & \forall i \text{ with } g_i^{old} \neq 0 : \\ & s \cdot g_i^\star \geq g_i^{old} + T \\ & s \cdot g_i^\star \leq g_{max}. \end{aligned} \tag{6.2}$$

The formulation represents a one-dimensional optimization problem with two times as many constraints as there are nonzero conductances in the neuron. A visual sketch of the problem can be seen in Figure 6.4.

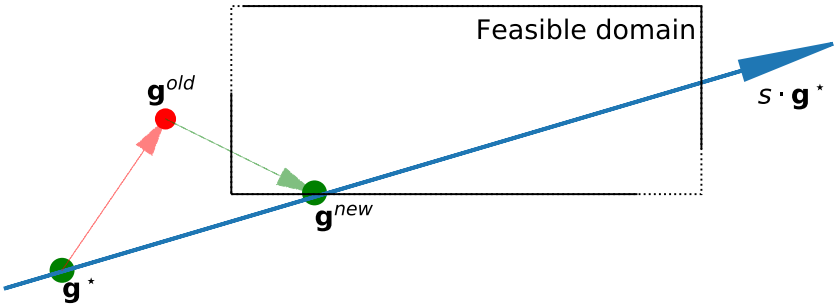


Figure 6.4: A sketch of the post-fabrication tuning problem for a single neuron. The domain is defined by the constraints. Here, $\boldsymbol{g}^\star$ denotes the intended conductances which are perturbed through variation to $\boldsymbol{g}^{old}$. By solving (6.2), a new set of feasible $\boldsymbol{g}^{new} = s^\star \cdot \boldsymbol{g}^\star$ can be found.

To find a solution for (6.2), the problem is first reformulated. For this, note that the original constraints can be summarized by

$$\begin{aligned} \forall g_i^{old} \neq 0 : \quad s \cdot g_i^\star \geq T + g_i^{old} \iff s \geq \max_i \left\{ \frac{T + g_i^{old}}{g_i^\star} \right\} \\ s \cdot g_i^\star \leq g_{max} \iff s \leq \min_i \left\{ \frac{g_{max}}{g_i^\star} \right\}, \end{aligned}$$

which provides an upper and a lower bound for s. The lower bound and the upper bound will be referred to as $\underline{s}$ and $\overline{s}$ and in the following. Furthermore, by noting that the terms of the sum are nonnegative according to the constraints, i.e., $\forall i$ with $g_i^{old} \neq 0 \ : \ (s \cdot g_i^\star - g_i^{old}) \geq T \geq 0$, the problem can be stated equivalently as

$$\begin{aligned} s^\star = \quad & \underset{s \in \mathbb{R}}{\operatorname{argmin}} \quad \sum_i (s \cdot g_i^\star - g_i^{old}) \\ & \text{s.t.} \quad \underline{s} \leq s \leq \overline{s}, \end{aligned} \tag{6.3}$$

where $\underline{s}$ and $\overline{s}$ can be computed as seen above. Since (6.3) is a linear optimization problem on a compact domain, it always has a solution if the feasible set is not empty, i.e., if $\underline{s} \leq \overline{s}$. Additionally, the solution must lie on the boundary of the feasible set. The solution for the problem is therefore given as either $\underline{s}$ or $\overline{s}$ depending on which of the two leads to the lower value of the objective function.

After obtaining $s^\star$, the initially fabricated conductances $\boldsymbol{g}^{old}$ of the neuron can be adjusted to $\boldsymbol{g}^{new} = s^\star \cdot \boldsymbol{g}^\star$. Then, according to (6.1), $w(\boldsymbol{g}^{new}) = w(\boldsymbol{g}^\star)$. In the following, the process of finding $\boldsymbol{g}^{new}$ is simply referred to as reprinting.

6.2.2 Post-fabrication tuning for printed neural networks

Through the reprinting procedure presented in the last section, the weights of single neurons can be restored. Trivially, all network conductances could be reprinted to benefit from the smaller variation. However, since reprinting is time consuming, only a limited number of neurons, say k, should be chosen for reprinting. This leads to the question of finding the best k neurons to reprint.

For a given neuron, the expected accuracy (or MaA) improvement for reprinting can be estimated by evaluating the pNN with the given weights $\mathbf{w}(\boldsymbol{g}^{old})$ and the intended weights $\mathbf{w}(\boldsymbol{g}^{\star})$. Based on this estimate, promising neurons to reprint can be identified. However, for N neurons and k planned reprints, there are $\binom{N}{k}$ different configurations to evaluate. This may already be prohibitive for small networks and moderate values of k. As an example, consider the pNN for *pendigits* with 43 neurons. Here, finding the best $k = 10$ neurons would already result in more than 10^9 possible combinations which would have to be evaluated.

To reduce the required runtime, a greedy, sequential strategy can be pursued. Here, in each of the k steps, the expected accuracy (or MaA) improvement is evaluated for all neurons and one neuron is chosen for reprinting. The greedy strategy therefore only requires $\sum_0^{k-1} N - k$ evaluations of the reprinting procedure. Finally, note that also less than k steps can be performed if no improvement can be expected by reprinting an additional neuron.

6.3 Experiments

The proposed post-fabrication procedure is now evaluated on the datasets described earlier in this work. Since printing the respective NC devices would be too expensive and time consuming, the MaAs of reprinted pNNs are used for evaluation.

6.3.1 Experimental setup

All experiments are performed twice, once using the component models extracted from measurements, and once using the component model extracted from simulation data (details see Section 5.3). The networks to reprint are taken from the grid-search procedures in Section 3.4 and Section 5.3. Note that these networks are trained without variation. The circuit models, i.e., the inv and ptanh, are assumed to have no variations. The variation of the crossbar resistors is simulated according to the model from Section 5.1.2, for variation levels of $\epsilon \in \{0, 5, 10, 15, 20, 25, 30\}\%$ for *pendigits* and $\epsilon \in \{0, 10, 20, 30\}\%$ for the other datasets. All reported results are calculated using 100 simulated fabrication and post-fabrication cycles for the respective pNNs.

The post-fabrication procedure requires an extended range of printable conductance values to increase conductances that were set to g_{max} in training. As discussed before, the extended range may be realized by printing multiple stacked layers of conductive material. For the experiments, a conductance range of $1 : 600$, numerically relating to $g_{min} = 0.01$ and $g_{max} = 6$ is selected.[3] Note that the conductance range used for training was $1 : 100$, relating to $g_{min} = 0.01$ and $g_{max} = 1$.

[3] This range was suggested by Dennis D. Weller in a personal conversation.

When reprinting the conductances of given neuron, the conductances g^{old} of the old crossbar resistors are given. Hence, only the size of the adjustment $\boldsymbol{g}^{new} - \boldsymbol{g}^{old} \geq T$ is considered to vary with the post-fabrication variation level of $\epsilon_{reprint} = 5\%$. The minimal adjustment distance T is set to g_{min}. This choice can be justified by the assumption that g_{min} represents the conductance of the smallest printable geometry and therefore also limits the resolution of the adjustment.

Finally, in each of the k reprinting steps, the neuron to reprint is selected based on the best train (validation for *pendigits*) MaA achieved after simulating one reprinting step (without variation) for each neuron. All reported tables and figures in the following show average test data performance results for 100 simulated fabrications[4], followed by the post-fabrication tuning processes.

6.3.2 Experiments with measured circuit models

First, the post-fabrication procedure is applied to the networks from Table 3.1. The fabrication of the initial network is simulated 100 times under various levels of variation. Post-fabrication tuning is then applied to each network and different percentages of the neurons of the network are reprinted. The results for *pendigits* are displayed in Figure 6.5.

Evidently, the more neurons are reprinted, the better the achieved MaA. Moreover, the higher the variation, the more neurons need to be reprinted to recover MaA levels close to the initial accuracy. However, the effectiveness of reprinting saturates when reprinting more than 30% of the neurons ($\approx$ 12 of 42). This is likely due to the post-fabrication variation $\epsilon_{reprint}$, which does not permit to recover the $\epsilon = 0\%$ result. Reprinting may even decrease the results slightly since the estimated improvement does not consider the post-fabrication variation.

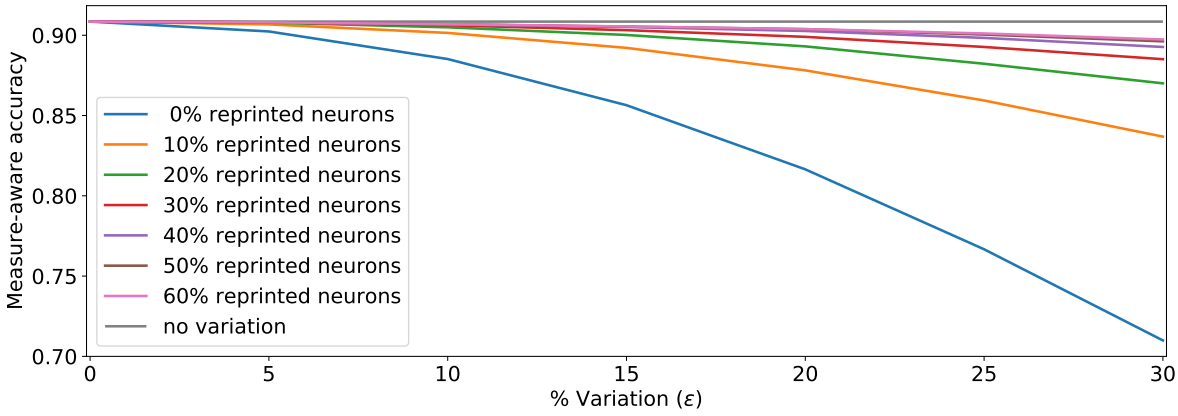


Figure 6.5: The post-fabrication procedure applied to the *pendigits* dataset under different variation levels ϵ (x-axis). The network uses the ptanh and inv function extracted from measured data. The line denotes the mean average test MaA for the respective configuration for 100 samples.

To further investigate this effect, the procedure is applied to networks of the smaller datasets. For each dataset/network, the test set results for different percentages of reprinted neurons are reported. The results for each network are normalized with respect to their MaAs at $\epsilon = 0\%$ and can be found in Figure 6.6.

[4]Applying ϵ variation to the conductances of all crossbar resistors.

As for *pendigits*, reprinting about $20-30\%$ of the neurons allows to restore $90-95\%$ of the initial $\epsilon = 0\%$ MaA for most datasets. Two exceptions are *Vertebral Column* with 3 classes and *Tic-Tac-Toe Endgame*. Note that these networks have already been comparably sensitive to conductance variation earlier (see Table 5.1).

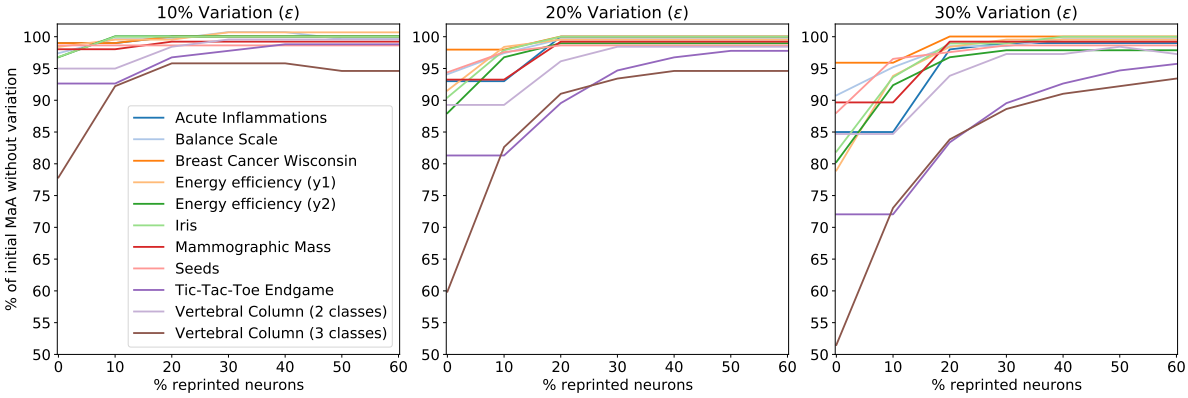


Figure 6.6: The post-fabrication procedure applied to the networks of Table 3.1 for different variation levels ϵ. The x-axis displays the % of the total neurons that are reprinted (k). The y-axis denotes the % of the $\epsilon = 0\%$ MaA that could be recovered.

6.3.3 Experiments with simulated circuit models

The experiments are now repeated for the best pNNs using the simulated circuit component models. In the previous experiments, the networks trained with these circuit models were observed to be generally less robust to variation. Hence, it can be expected that a higher percentage of neurons needs to be reprinted to restore a comparable performance. This is also directly confirmed by the experimental results (see Figure 6.7 and 6.8).

For *pendigits*, about 40% of the neurons (compared to 20% under the measured circuit models) need to be reprinted to recover about 90% of the $\epsilon = 0\%$ MaA. Additionally, due to the generally worse performance of the networks under variation, the initial network does not only start off worse, but is likely also more impaired by the 5% post-fabrication variation.

For the other datasets, the observed effects are similar. For most datasets and variation levels, about $80-90\%$ of the $\epsilon = 0\%$ MaA can be achieved after reprinting $20-30\%$ of the neurons. An exception to this is *Energy efficiency (y1)*. However, the network for this dataset has already been shown to be extremely sensitive to variation in Table 5.4. Thus, since post-fabrication still introduces variation, even reprinting a high percentage of neurons may not always permit achieving satisfying results.

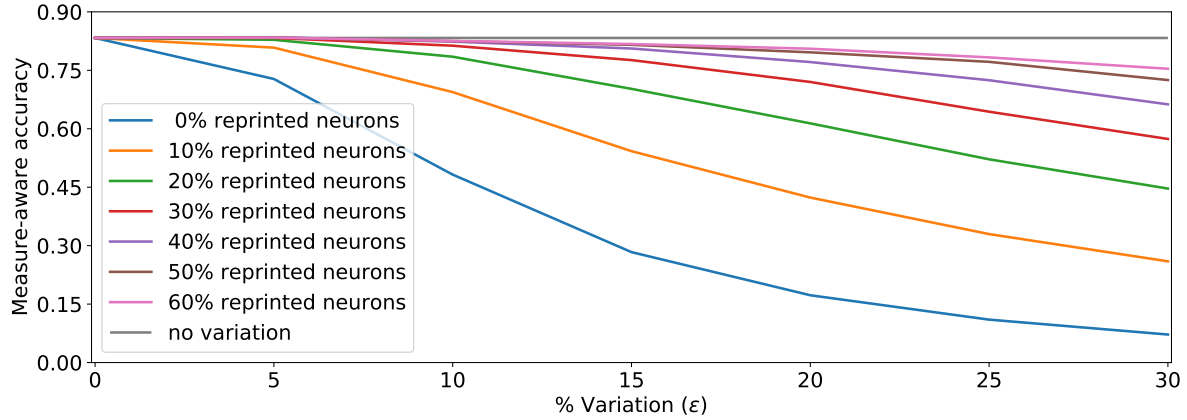


Figure 6.7: The post-fabrication procedure applied to the *pendigits* dataset under different variation levels ϵ (x-axis). The network uses the circuit models extracted from simulated data. The line denotes the mean average test MaA for the respective configuration for 100 samples.

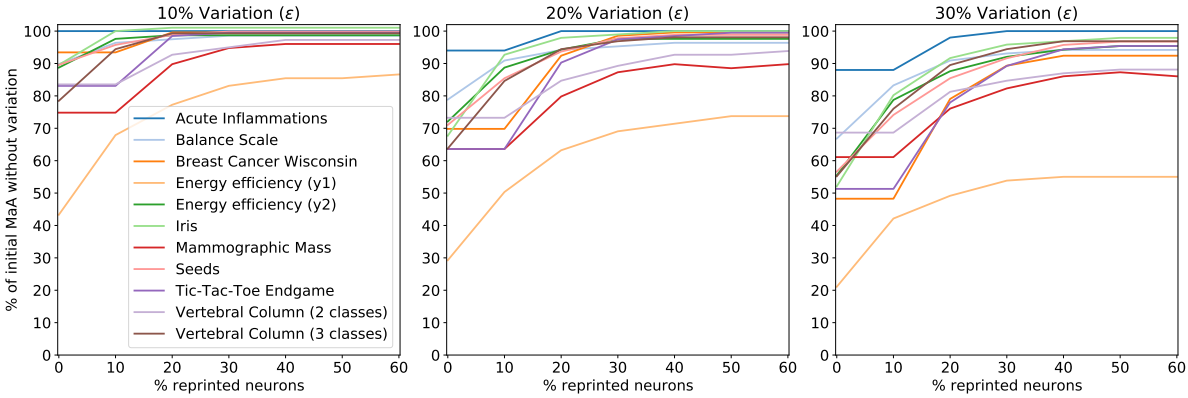


Figure 6.8: The post-fabrication procedure applied to the networks of Table 5.2 for different variation levels ϵ. The x-axis displays the % of the total neurons that are reprinted (k). The y-axis denotes the % of the $\epsilon = 0\%$ MaA that could be recovered.

6.4 Conclusion and directions for future work

In this chapter, a post-fabrication tuning procedure for increasing the performance of fabricated printed NCs was proposed. The approach constructs a pNN model of the NC based on an assessment of the circuit components and proposes a set of neurons to reprint. The reprinting configuration for a neuron is chosen by solving an optimization problem. The method was evaluated using multiple datasets and various parameter variation levels. Overall, even under simulated variation of up to 30%, close to $80-90\%$ of the expected accuracy could be achieved by reprinting $20-30\%$ of the neurons for most datasets.

Contrary to variation-aware training proposed in Chapter 5, the post-fabrication procedure does not require an explicit assumption about the variations. Hence, it can deal with any perturbation of the conductance values. For example, the procedure could also be used to address degradations such as aging. However, the assessment of the printed components through measurements and the following reprinting step require substantially more effort than simply using a modified training procedure.

Finally, reprinting $20-30\%$ of the neurons of a network can mean greatly varying costs depending on which neurons are chosen. Reprinting a neuron always requires the reprinting of all its conductances. Thus, the proposed procedure can be costly for neurons with many weights. This is a problem to be considered for future work. Ideally, a method should be able to find a fixed-size subset of network conductances that should be reprinted. Although this could be formulated as an optimization problem, finding a good solution would likely be challenging due to its discrete nature and the technology constraints. One possible solution may be to choose the conductances to reprint based on some heuristic beforehand and then optimize the conductance values through classical optimisation procedures. Another approach could be to explicitly enforce sparse solutions in the problem formulation. Consequently, many possible approaches may be developed to recover the performances of printed NC in post-fabrication steps.

7 Summary and Outlook

Printed electronics is an emerging technology with enormous potential for IoT applications. One of its prospects is to enable the custom fabrication of smart devices through inkjet-printing. However, the design of these printed components poses many challenges, e.g., through their comparably high variations. This work proposed the adoption of neuromorphic computing for printed circuitry to address some of these design challenges.

As a first topic, aspects of training printed neuromorphic circuits were described. Training thereby represents an automatic circuit design solution by an optimisation routine. Through these approaches, a wide range of near sensor processing tasks in the context of the IoT could be addressed. This may be especially interesting for custom smart devices in the future.

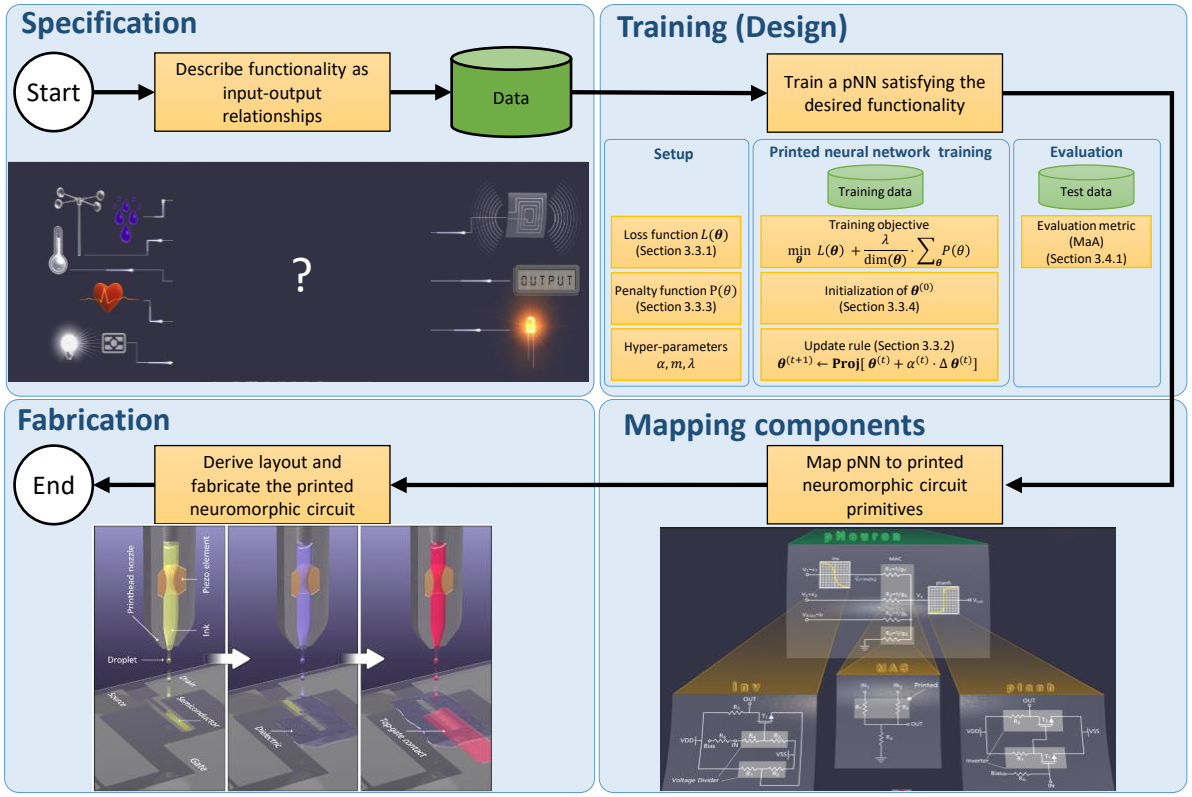


Figure 7.1: The proposed procedure allows for an on-demand design and fabrication of printed circuits given a specification of a desired functionality. The on-demand design is realized through training a pNN model according to the specification. The derived designs can be readily fabricated through the on-demand fabrication capabilities of inkjet-printed electronics. This figure is partly composed of figures from [156].

The resulting design flow of training, mapping and fabrication can be seen in flowchart of Figure 7.1. The procedure can be summarized as follows: From an initial idea, the desired functionality is described in the form of input-output relationships. These relationships are then

converted into training data for a pNN model. Through the training procedure (Section 3.3 or Section 5), a pNN expressing the specified functionality is found. The circuit can then be realized by mapping the learned pNN to the respective hardware primitives of printed NCs (Section 2.6). In the last step, the printed NC can be fabricated using inkjet printing.

To address the comparably high variations of printed components in designs, a general, data-driven variation modelling framework was proposed. The approach supports a fast development of variation models based on nominal semi-empirical models. The parameters of such models often exhibit complicated correlation structures and distribution shapes. To provide adequate flexibility, the distributions are approximated through Gaussian mixture models with an adaptively chosen complexity. Hence, the framework automatically resorts to the classical choice of simple uncorrelated Gaussian distributions for each parameter if adequate. The framework was also used to develop a variation model for a printed transistor (EGT), that has been used in multiple works, e.g. [50, 156]. A drawback of the framework is that it still requires a given nominal model with identifiable parameters, which limits its applicability. These limitations suggest further research toward fully data-driven approaches that allow the development of variation models directly from measurement data.

To deal with the high variation of components in printed NCs, two approaches were described in this work. The first relates to a variation-aware training procedure for pNN models. For this, variation models for the core components of the printed NC, namely the crossbar resistors, the activation function and the inverter circuit, were developed. The variation models were then used to modify the training objective of pNNs to account for this variation. Finally, the training problem was solved using Monte Carlo gradient estimates. The second approach relates to a post-fabrication tuning procedure for printed NCs. Here, the unique feature of additive manufacturing is employed to modify inaccurately printed resistors after their fabrication. For this, the conductances of the crossbar resistors of each neuron are assessed. Then, a minimal adjustment for the conductances is calculated for which the intended weights would be restored.

For now, all methods were evaluated using the proposed equations, and variations were considered based on the proposed variation models. However, to thoroughly evaluate the proposed concepts and ideas, real circuits should be fabricated and measured. Also, the method may be evaluated on data of several, real use-cases for the design of smart devices, possibly incorporating printed sensors. Unfortunately, the immaturity of the technology and production processes does not permit extensive experiments yet. For now, the feasibility of small printed NCs, as discussed in this work, was only demonstrated for a few components of a printed NC, see [157, 156].

In this work, the perspective taken on the fabrication of printed NCs focuses mainly on inkjet-printed circuitry that allows for full customization. While this aligns with the vision of personalized fabrication [130], there may be high redundancies in the designs for NCs for certain tasks. Hence, some designs could be reused with slight adaptations. Here, use-cases for split manufacturing may arise, allowing for the combination of multiple printing techniques. Commonly required parts, such as activation function and inverter circuits, or even entire sub-networks of printed NC, could be fabricated in batches through high volume replication printing techniques. A user can then inkjet-print the missing components on-demand to configure and complete the circuit. The configuration could thereby be found through the techniques described in this work, provided that a differentiable model of the pre-printed circuitry is available. Finding good candidates for pre-printable sub-networks could be a future research task.

A promising direction for this could be to investigate techniques from multi-task-learning [19].

In general, the training of printed pNNs can be seen as an optimisation-based design approach for printed NCs. The space of realizable circuits is thereby given through the functionality that the pNN (and consequently the printed NC) can approximate. Conceptually, this offers many benefits. For example, desirable properties like robustness to variations, as seen in Chapter 5, can be addressed by modifying the optimisation objective. Further aspects, such as robustness to aging effects, may be integrated in a similar fashion. For example, the training goal could be to optimize for a high accuracy over the expected lifetime under consideration of an aging model of the components. Furthermore, approximation trade-offs between circuit size or power consumption could be considered by also searching for suitable pNN architectures. For this, techniques from neural architecture search [45] could be adopted.

Especially in the light of growing computational resources, optimisation-based design approaches may hold great potential. However, as for now, their adoption is still limited [80]. Nevertheless, to deliver on the promise of custom fabrication and enable true, on-demand realisation of printed devices, employing such design solutions is likely inevitable. Hopefully, the ideas described in this work will inspire more research in this direction.

Bibliography

[1] S. U. Ahmed, M. Shahjahan, and K. Murase. Injecting chaos in feedforward neural networks. *Neural Processing Letters*, 34(1):87–100, 2011.

[2] A. Almusallam, R. N. Torah, D. Zhu, M. J. Tudor, and S. P. Beeby. Screen-printed piezoelectric shoe-insole energy harvester using an improved flexible PZT-polymer composites. *Journal of Physics: Conference Series*, 476:012108, 2013.

[3] G. An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, 1996.

[4] E. Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509, 1936.

[5] M. Ansari, A. Fayyazi, A. Banagozar, M. A. Maleki, M. Kamal, A. Afzali-Kusha, and M. Pedram. Phax: Physical characteristics aware ex-situ training framework for inverter-based memristive neuromorphic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(8):1602–1613, 2018.

[6] M. Ansari, A. Fayyazi, M. Kamal, A. Afzali-Kusha, and M. Pedram. Octan: An on-chip training algorithm for memristive neuromorphic circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(12):4687–4698, 2019.

[7] M. Barros, J. Guilherme, and N. Horta. Analog circuits optimization based on evolutionary computation techniques. *Integration*, 43(1):136–155, 2010.

[8] M. Beigl, H.-W. Gellersen, and A. Schmidt. Mediacups: Experience with design and use of computer-augmented everyday artefacts. *Computer Networks*, 35(4):401–409, Mar. 2001.

[9] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. `https://arxiv.org/abs/1308.3432`, 2013.

[10] J. L. Bernier, J. Ortega, I. Rojas, E. Ros, and A. Prieto. Obtaining fault tolerant multilayer perceptrons using an explicit regularization. *Neural Processing Letters*, 12(2):107–113, 2000.

[11] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, Massachusetts, 2 edition, 1999.

[12] R. Bhattacharya, M. M. de Kok, and J. Zhou. Rechargeable electronic textile battery. *Applied Physics Letters*, 95(22):223305, 2009.

[13] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[14] N. Bleier, M. H. Mubarik, F. Rasheed, J. Aghassi-Hagmann, M. B. Tahoori, and R. Kumar. Printed microprocessors. In *2020 ACM/IEEE 47th Annual International Symposium*

on Computer Architecture (ISCA), pages 213–226, 2020.

[15] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[16] S. K. Bose, J. Acharya, and A. Basu. Is my neural network neuromorphic? taxonomy, recent trends and future directions in neuromorphic engineering. `https://arxiv.org/abs/2002.11945`, 2020.

[17] C.-T. Chin, K. Mehrotra, C. K. Mohan, and S. Rankat. Training techniques to obtain fault-tolerant neural networks. In *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, pages 360–369, 1994.

[18] C.-T. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka. Modifying training algorithms for improved fault tolerance. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 333–338, 1994.

[19] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[20] J. A. Cavallo, M. C. Strumia, and C. G. Gomez. Preparation of a milk spoilage indicator adsorbed to a modified polypropylene film as an attempt to build a smart packaging. *Journal of Food Engineering*, 136:48 – 55, 2014.

[21] P. Chandra and Y. Singh. Fault tolerance of feedforward artificial neural networks- a framework of study. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 1, pages 489–494 vol.1, 2003.

[22] F. Y. Chang. Generation of 3-sigma circuit models and its applicaton to statistical worst-case analysis of integrated circuit designs. In *1977 11th Asilomar Conference on Circuits, Systems and Computers, 1977. Conference Record.*, pages 29–34, 1977.

[23] J. Chang, T. Ge, and E. Sanchez-Sinencio. Challenges of printed electronics on flexible substrates. In *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 582–585, 2012.

[24] J. S. Chang, A. F. Facchetti, and R. Reuss. A circuits and systems perspective of organic/printed electronics: Review, challenges, and contemporary and emerging design approaches. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 7(1):7–26, 2017.

[25] H. Chaouchi. *Introduction to the internet of things.* John Wiley & Sons, 2013.

[26] B. Cheng, N. Moezi, D. Dideban, G. Roy, S. Roy, and A. Asenov. Benchmarking the accuracy of pca generated statistical compact model parameters against physical device simulation and directly extracted statistical parameters. In *2009 International Conference on Simulation of Semiconductor Processes and Devices*, pages 1–4, 2009.

[27] C.-T. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka. Robustness of feedforward neural networks. In *IEEE International Conference on Neural Networks*, volume 2, pages 783–788, 1993.

[28] A. Chortos, J. Liu, and Z. Bao. Pursuing prosthetic electronic skin. *Nature Materials*, 15(9):937–950, 2016.

[29] L.-C. Chu and B. W. Wah. Fault tolerant neural networks with hybrid redundancy. In *1990 IJCNN International Joint Conference on Neural Networks*, volume 2, pages

639–649, 1990.

[30] M. W. Cohen, M. Aga, and T. Weinberg. Genetic algorithm software system for analog circuit design. *Procedia CIRP*, 36:17–22, 2015.

[31] C. M. Costa, R. Gonçalves, and S. Lanceros-Méndez. Recent advances and future challenges in printed batteries. *Energy Storage Materials*, 28:216 – 234, 2020.

[32] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[33] Z. Cui. *Printed electronics: Materials, technologies and applications.* John Wiley & Sons, 2016.

[34] J. Czerniak and H. Zarzycki. Application of rough sets in the presumptive diagnosis of urinary system diseases. In *Artificial Intelligence and Security in Computing Systems*, pages 41–51. Kluwer Academic Publishers, 2003.

[35] W. Daems, G. Gielen, and W. Sansen. An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits. In *Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)*, pages 431–436, 2002.

[36] R. D'Agostino and E. S. Pearson. Tests for departure from normality. empirical results for the distributions of b_2 and $\sqrt{b_1}$. *Biometrika*, 60(3):613–622, 1973.

[37] J. Deng, Y. Rang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu. Retraining-based timing error mitigation for hardware neural networks. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 593–596, 2015.

[38] D. Deodhare, M. Vidyasagar, and S. Sathiya Keethi. Synthesis of fault-tolerant feedforward neural networks using minimax optimization. *IEEE Transactions on Neural Networks*, 9(5):891–900, 1998.

[39] P. Descent, R. Izquierdo, and C. Fayomi. Printing of temperature and humidity sensors on flexible substrates for biomedical applications. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2018.

[40] F. M. Dias and A. Antunes. Fault tolerance improvement through architecture change in artificial neural networks. In *Advances in Computation and Intelligence*, pages 248–257. Springer, 2008.

[41] Z. Du, D. D. Ben-Dayan Rubin, Y. Chen, L. Hel, T. Chen, L. Zhang, C. Wu, and O. Temam. Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 494–507, 2015.

[42] D. Dua and et al. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2017.

[43] P. J. Edwards and A. F. Murray. Fault tolerance via weight noise in analog vlsi implementations of mlps-a case study with epsilon. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(9):1255–1262, 1998.

[44] P. J. Edwards and A. F. Murray. Toward optimally distributed computation. *Neural Computation*, 10(4):987–1005, 1998.

[45] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. `https://arxiv.org/abs/1808.05377`, 2019.

[46] M. D. Emmerson and R. I. Damper. Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. *IEEE Transactions on Neural Networks*, 4(5):788–793, 1993.

[47] C. C. Enz, F. Krummenacher, and E. A. Vittoz. An analytical mos transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications. *Analog Integrated Circuits and Signal Processing*, 8(1):83–114, 1995.

[48] A. T. Erozan, M. S. Golanbari, R. Bishnoi, J. Aghassi-Hagmann, and M. B. Tahoori. Design and evaluation of physical unclonable function for inorganic printed electronics. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 419–424, 2018.

[49] A. T. Erozan, M. Hefenbrock, M. Beigl, J. Aghassi-Hagmann, and M. B. Tahoori. Reverse engineering of printed electronics circuits: From imaging to netlist extraction. *IEEE Transactions on Information Forensics and Security*, 15:475–486, 2020.

[50] A. T. Erozan, G. Y. Wang, R. Bishnoi, J. Aghassi-Hagmann, and M. B. Tahoori. A compact low-voltage true random number generator based on inkjet printing technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(6):1485–1495, 2020.

[51] A. Fayyazi, M. Ansari, M. Kamal, A. Afzali-Kusha, and M. Pedram. An ultra low-power memristive neuromorphic circuit for internet of things smart sensors. *IEEE Internet of Things Journal*, 5(2):1011–1022, 2018.

[52] A. Fayyazi, S. Kundu, S. Nazarian, P. A. Beerel, and M. Pedram. Sprram: A predefined sparsity based memristive neuromorphic circuit for low power application, 2018.

[53] M. Fernández-Delgado and et al. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[54] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[55] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256. PMLR, 2010.

[56] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[57] R. H. Griffin, D. E. Root, J. Xu, A. Dadvand, T. Chu, and Y. Tao. Artificial neural network modelling and simulation of organic field effect transistors and circuits. In *2019 IEEE International Flexible Electronics Technology Conference (IFETC)*, pages 1–5, 2019.

[58] R. H. Griffin, D. Shleifman, A. Dadvand, N. Graddage, T. Chu, and Y. Tao. Improved circuit model fitting of inkjet-printed otfts and a proposal for standardized parameter reporting. *IEEE Transactions on Electron Devices*, 65(6):2485–2491, 2018.

[59] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović. Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[60] N. C. Hammadi and H. Ito. Improving the performance of feedforward neural networks by noise injection into hidden neurons. *Journal of Intelligent and Robotic Systems*, 21(2):103–115, 1998.

[61] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[62] J. P. Harvey, M. I. Elmasry, and B. Leung. Staic: An interactive framework for synthesizing cmos and bicmos analog circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(11):1402–1417, 1992.

[63] R. Hasan and T. M. Taha. Enabling back propagation training of memristor crossbar neuromorphic processors. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 21–28, 2014.

[64] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: Data mining, inference and prediction.* Springer, 2 edition, 2009.

[65] B. He, s. Zhang, F. Yang, C. Yan, D. Zhou, and X. Zeng. An efficient bayesian optimization approach for analog circuit synthesis via sparse gaussian process modeling. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 67–72, 2020.

[66] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[67] M. Hefenbrock, D. D. Weller, J. Aghassi-Hagmann, M. Beigl, and M. B. Tahoori. In-situ tuning of printed neural networks for variation tolerance. In *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, page to appear, 2022.

[68] M. d. Hershenson, S. P. Boyd, and T. H. Lee. Optimal design of a cmos op-amp via geometric programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(1):1–21, 2001.

[69] K. Ho, C.-S. Leung, and J. Sum. On weight-noise-injection training. In M. Köppen, N. Kasabov, and G. Coghill, editors, *Advances in Neuro-Information Processing*, pages 919–926. Springer, 2009.

[70] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[71] M. Hu, H. Li, Q. Wu, and G. S. Rose. Hardware realization of bsb recall function using memristor crossbar arrays. In *Proceedings of the 49th Annual Design Automation Conference*, pages 498–503. ACM, 2012.

[72] J. Huang, S. Zhang, C. Tao, F. Yang, C. Yan, D. Zhou, and X. Zeng. Bayesian optimization approach for analog circuit design using multi-task gaussian process. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[73] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[74] F. D. Inc. Dimatix materials printer dmp-2850 data sheet. `https://www.fujifilm.ca/shared/bin/dimatix_materials_printer_dmp-2850.pdf`. accessed: 21.11.2020.

[75] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 448–456. PMLR, 2015.

[76] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. Shi. Iprail-intellectual property reuse-based analog ic layout automation. *Integration*, 36(4):237–262, 2003. Analog and mixed-signal IC design and design methodologies.

[77] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[78] S. Jin, S. Pei, and Y. Wang. On improving fault tolerance of memristor crossbar based neural network designs by target sparsifying. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 91–96, 2020.

[79] M. Kanno, A. Shibuya, M. Matsumura, K. Tamura, H. Tsuno, S. Mori, Y. Fukuzaki, T. Gocho, H. Ansai, and N. Nagashima. Empirical characteristics and extraction of overall variations for 65-nm mosfets and beyond. In *2007 IEEE Symposium on VLSI Technology*, pages 88–89, 2007.

[80] G. Kendall. Is evolutionary computation evolving fast enough? *IEEE Computational Intelligence Magazine*, 13(2):42–51, 2018.

[81] Y. Khan, A. Thielens, S. Muin, J. Ting, c. Baumbauer, and A. C. Arias. A new frontier of printed electronics: Flexible hybrid electronics. *Advanced Materials*, 32(15):1905279, 2020.

[82] Y. Kim, A. Chortos, W. Xu, Y. Liu, J. Y. Oh, D. Son, J. Kang, A. M. Foudeh, C. Zhu, Y. Lee, et al. A bioinspired flexible organic artificial afferent nerve. *Science*, 360(6392):998–1003, 2018.

[83] Y. J. Kim, S.-E. Chun, J. Whitacre, and C. J. Bettinger. Self-deployable current sources fabricated from edible materials. *Journal of Materials Chemistry B*, 1:3781–3788, 2013.

[84] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[85] D. P. Kingma and M. Welling. Auto-encoding variational bayes. `https://arxiv.org/abs/1312.6114`, 2014.

[86] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In *Artificial Intelligence in Design '96*, pages 151–170. Springer, 1996.

[87] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[88] K. Kuhn. Variability in nanoscale cmos technology. *Science China Information Sciences*,

54(5):936, 2011.

[89] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai. Process technology variation. *IEEE Transactions on Electron Devices*, 58(8):2197–2208, 2011.

[90] Y. Lee, J. Y. Oh, W. Xu, O. Kim, T. R. Kim, J. Kang, Y. Kim, D. Son, J. B. H. Tok, M. J. Park, et al. Stretchable organic optoelectronic sensorimotor synapse. *Science advances*, 4(11):eaat7387, 2018.

[91] C. Leung, W. Y. Wan, and R. Feng. A regularizer approach for rbf networks under the concurrent weight failure situation. *IEEE Transactions on Neural Networks and Learning Systems*, 28(6):1360–1372, 2017.

[92] C. H. Lin, M. V. Dunga, D. D. Lu, A. M. Niknejad, and C. Hu. Performance-aware corner model for design for manufacturing. *IEEE Transactions on Electron Devices*, 56(4):595–600, 2009.

[93] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang. Vortex: Variation-aware training for memristor x-bar. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[94] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell. Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 63–70, 2014.

[95] L. Liu, Y. Feng, and W. Wu. Recent progress in printed flexible solid-state supercapacitors for portable and wearable energy storage. *Journal of Power Sources*, 410-411:69 – 77, 2019.

[96] X. Liu, A. M. Gough, and J. Li. Semiconductor corner lot generation robust to process variation: Modeling and analysis. *IISE Transactions*, 50(2):126–139, 2018.

[97] W. Lyu, P. Xue, F. Yang, C. Yan, Z. Hong, X. Zeng, and D. Zhou. An efficient bayesian optimization approach for automated optimization of analog circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(6):1954–1967, 2018.

[98] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose. Bsb training scheme implementation on memristor-based circuit. In *2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 80–87, 2013.

[99] H. Marien, M. Steyaert, and P. Heremans. *Analog organic electronics: Building blocks for organic smart sensor systems on foil*. Springer, 2012.

[100] G. C. Marques, S. K. Garlapati, D. Chatterjee, S. Dehm, S. Dasgupta, J. Aghassi, and M. B. Tahoori. Electrolyte-gated fets based on oxide semiconductors: Fabrication and modeling. *IEEE Transactions on Electron Devices*, 64(1):279–285, 2017.

[101] G. C. Marques, F. Rasheed, J. Aghassi-Hagmann, and M. B. Tahoori. From silicon to printed electronics: A coherent modeling and design flow approach based on printed electrolyte gated fets. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, ASPDAC '18, page 658–663. IEEE Press, 2018.

[102] G. C. Marques, F. Rasheed, J. Aghassi-Hagmann, and M. B. Tahoori. From silicon

to printed electronics: A coherent modeling and design flow approach based on printed electrolyte gated fets. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 658–663, 2018.

[103] K. Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

[104] P. Maulik and L. Carley. Automating analog circuit design using constrained optimization techniques. In *1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pages 390–393, 1991.

[105] C. C. McAndrew, I. Lim, B. Braswell, and D. Garrity. Corner models: Inaccurate at best, and it only gets worst .... In *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*, pages 1–4, 2013.

[106] W. McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

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

[108] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[109] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020.

[110] M. H. Mubarik, D. D. Weller, N. Bleier, M. Tomei, J. Aghassi-Hagmann, M. B. Tahoori, and R. Kumar. Printed machine learning classifiers. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 73–87, 2020.

[111] A. F. Murray and P. J. Edwards. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks*, 5(5):792–802, 1994.

[112] K. Myny, P. van Lieshout, J. Genoe, W. Dehaene, and P. Heremans. Accounting for variability in the design of circuits with organic thin-film transistors. *Organic Electronics*, 15(4):937 – 942, 2014.

[113] S. Mühl and B. Beyer. Bio-organic electronics—overview and prospects for the future. *Electronics*, 3(3):444–461, 2014.

[114] C. Neti, M. H. Schneider, and E. D. Young. Maximally fault tolerant neural networks. *IEEE Transactions on Neural Networks*, 3(1):14–23, 1992.

[115] J. Nocedal and S. Wright. *Numerical optimization.* Springer, 2 edition, 2006.

[116] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley. Synthesis of high-performance analog circuits in astrx/oblx. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(3):273–294, 1996.

[117] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[118] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[119] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. Matching properties of mos transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1439, Oct 1989.

[120] D. S. Phatak and I. Koren. Complete and partial fault tolerance of feedforward neural nets. *IEEE Transactions on Neural Networks*, 6(2):446–456, 1995.

[121] M. Pietsch, S. Schlisske, M. Held, N. Strobel, A. Wieczorek, and G. Hernandez-Sosa. Biodegradable inkjet-printed electrochromic display for sustainable short-lifecycle electronics. *Journal of Materials Chemistry C*, 8:16716–16724, 2020.

[122] K. Qian. *Variability modeling and statistical parameter extraction for CMOS devices*. PhD thesis, EECS Department, University of California, Berkeley, 2015.

[123] E. Ragonese, M. Fattori, and E. Cantatore. Printed organic electronics on flexible foil: Circuit design and emerging applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(1):42–48, 2021.

[124] F. Rasheed. *Compact modeling and physical design automation of inkjet-printed electronics technology*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2020. 43.22.03; LK 01.

[125] F. Rasheed, M. S. Golanbari, G. C. Marques, M. B. Tahoori, and J. Aghassi-Hagmann. A smooth ekv-based dc model for accurate simulation of printed transistors and their process variations. *IEEE Transactions on Electron Devices*, 65(2):667–673, 2018.

[126] F. Rasheed, M. Hefenbrock, M. Beigl, M. B. Tahoori, and J. Aghassi-Hagmann. Variability modeling for printed inorganic electrolyte-gated transistors and circuits. *IEEE Transactions on Electron Devices*, 66(1):146–152, 2019.

[127] F. Rasheed, M. Hefenbrock, R. Bishnoi, M. BeigI, J. Aghassi-Hagmann, and M. B. Tahoori. Predictive modeling and design automation of inorganic printed electronics. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 30–35, 2019.

[128] F. Rasheed, M. Hefenbrock, R. Bishnoi, M. Beigl, J. Aghassi-Hagmann, and M. B. Tahoori. Crossover-aware placement and routing for inkjet printed circuits. *Journal on Emerging Technologies in Computing Systems*, 16(2), 2020.

[129] S. Reda and S. R. Nassif. Accurate spatial estimation and decomposition techniques for variability characterization. *IEEE Transactions on Semiconductor Manufacturing*, 23(3):345–357, 2010.

[130] P. Rosa, A. Câmara, and C. Gouveia. The potential of printed electronics and personal fabrication in driving the internet of things. *Open Journal of Internet Of Things (OJIOT)*, 1(1):16–36, 2015.

[131] T. J. Rothenberg. Identification in parametric models. *Econometrica*, 39(3):577–591, 1971.

[132] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-

propagating errors. *Nature*, 323(6088):533–536, 1986.

[133] S. K. Saha. Modeling process variability in scaled cmos technology. *IEEE Design & Test of Computers*, 27(2):8–16, 2010.

[134] S. K. Saha. Compact mosfet modeling for process variability-aware vlsi circuit design. *IEEE Access*, 2:104–115, 2014.

[135] M. A. Sankhare, M. Guerin, E. Bergeret, P. Pannier, and R. Coppard. Full-printed otft modeling: Impacts of process variation. In *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 1–3, 2014.

[136] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic. Autockt: Deep reinforcement learning of analog circuit designs. *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[137] D. Simon. Distributed fault tolerance in optimal interpolative nets. *IEEE Transactions on Neural Networks*, 12(6):1348–1357, 2001.

[138] M. Slimani, F. Silveira, and P. Matherat. Variability modeling in near-threshold cmos digital circuits. *Microelectronics Journal*, 46(12, Part A):1313 – 1324, 2015.

[139] E. Sowade, E. Ramon, K. Y. Mitra, C. Martínez-Domingo, M. Pedró, J. Pallarès, F. Loffredo, F. Villani, H. L. Gomes, L. Terés, et al. All-inkjet-printed thin-film transistors: Manufacturing process reliability by root cause analysis. *Scientific Reports*, 6:33490, 2016.

[140] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[141] G. Stehr, M. Pronath, F. Schenkel, H. Graeb, and K. Antreich. Initial sizing of analog integrated circuits by centering within topology-given implicit specifications. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, pages 241–246, 2003.

[142] K. Takeuchi and M. Hane. Statistical compact model parameter extraction by direct fitting to variations. *IEEE Transactions on Electron Devices*, 55(6):1487–1493, 2008.

[143] L. N. Trefethen and D. Bau. *Numerical linear algebra*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 1997.

[144] A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

[145] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram. Interstice: Inverter-based memristive neural networks discretization for function approximation applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(7):1578–1588, 2020.

[146] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[147] M. Vanderroost, P. Ragaert, F. Devlieghere, and B. De Meulenaer. Intelligent food packaging: The next generation. *Trends in Food Science & Technology*, 39(1):47–62, 2014.

[148] Z. Vasicek and L. Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation*, 19(3):432–444, 2015.

[149] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[150] H. Wang, J. Yang, H.-S. Lee, and S. Han. Learning to design circuits. `https://arxiv.org/abs/1812.02734`, 2020.

[151] Y. Wang, C. Chung, C. Lin, and C. Lin. The study of the electrocardiography monitoring for the elderly based on smart clothes. In *2018 Eighth International Conference on Information Science and Technology (ICIST)*, pages 478–482, 2018.

[152] Y. Wang, W. Wen, L. Song, and H. H. Li. Classification accuracy improvement for neuromorphic computing systems with one-level precision synapses. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 776–781, 2017.

[153] L. Wasserman. *All of statistics*. Springer, 2004.

[154] D. Weller. *Digital and analog computing paradigms in printed electronics*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2021. 43.22.03; LK 01.

[155] D. D. Weller, N. Bleier, M. Hefenbrock, J. Aghassi-Hagmann, M. Beigl, R. Kumar, and M. B. Tahoori. Printed stochastic computing neural networks. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 914–919, 2021.

[156] D. D. Weller, M. Hefenbrock, M. Beigl, J. Aghassi-Hagmann, and M. B. Tahoori. Realization and training of an inverter-based printed neuromorphic computing system. *Scientific Reports*, 11(1), 2021.

[157] D. D. Weller, M. Hefenbrock, M. B. Tahoori, J. Aghassi-Hagmann, and M. Beigl. Programmable neuromorphic circuit based on printed electrolyte-gated transistors. In *Proceedings of the Asia South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[158] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang. Fault-tolerant training with on-line fault detection for rram-based neural computing systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[159] C. Yakopcic, R. Hasan, T. M. Taha, M. R. McLean, and D. Palmer. Efficacy of memristive crossbars for neuromorphic processors. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 15–20, 2014.

[160] K.-E. Yang, C.-Y. Tsai, H.-H. Shen, C.-F. Chiang, F.-M. Tsai, C.-A. Wang, Y. Ting, C.-S. Yeh, and C.-T. Lai. Trust-region method with deep reinforcement learning in analog design space exploration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1225–1230, 2021.

[161] L. Zhang, U. Kleine, and Y. Jiang. An automated design tool for analog layouts. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(8):881–894, 2006.

[162] S. Zhang, F. Yang, C. Yan, D. Zhou, and X. Zeng. An efficient batch constrained bayesian optimization approach for analog circuit synthesis via multi-objective acquisition ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021.

[163] Z.-H. Zhou and S.-F. Chen. Evolving fault-tolerant neural networks. *Neural Computing & Applications*, 11(3-4):156–160, 2003.