F3 **Faculty of Electrical Engineering**
Department of Circuit Theory

Dissertation thesis

Fully Analog Artificial Neural Network

Ing. Filip Paulů

Supervisor

doc. Dr. Ing. Jiří Hospodka

Prague, July 2023

Declaration

I hereby declare I have written this doctoral thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, July 2023

.....

Ing. Filip Paulů

Acknowledgements

First, I would like to express my gratitude to the Czech Technical University and the Faculty of Electrical Engineering for providing me with the opportunity to pursue my dissertation research.

I would like to express my sincerest thanks to my supervisor doc. Dr. Ing. Jiří Hospodka, for his invaluable guidance, mentorship, and encouragement throughout the entire process. His expertise and dedication helped me to focus on my studies and ensure the success of this project.

I thank my family, and I am grateful for your unwavering support and encouragement during this challenging journey. Your belief in me has been my source of strength and inspiration, for which I am eternally grateful.

Last but not least, I would like to express my appreciation to my fellow students, colleagues, friends, and everyone else who has been a part of this journey, if only by encouraging me or giving me the space for research. Your support and shared knowledge have invaluably helped me overcome challenges and made this experience enriching and enjoyable.

I would also like to thank my English teacher and everyone who has contributed to my understanding and appreciation of the English language. Your collective knowledge, whether shared intentionally or not, has played a crucial role in my ability to write this dissertation. I am truly grateful for the influence you have had on my development as a writer.

Thank you all for your support, encouragement, and contributions that have made this dissertation a reality.

Abstract

This dissertation introduces the concept of Fully Analog Artificial Neural Networks (FAANNs) capable of processing signals directly from electrical sensors. One of the research motives is to improve measurement accuracy with respect to sensor degradation and changes in environmental conditions. The second motive represents the ability to derive non-measured physical quantities dependent on measured ones, which is especially useful when the quantity is difficult to ascertain, for instance, due to a lengthy and costly chemical process.

Therefore, this work introduces a new design of a fully analog learning process based on the backpropagation algorithm using gradient descent, which remains an open problem. It is a recently discussed problem due to the parallelization of computations which thus increases the learning speed. The proposed structure circumvents the von Neumann bottleneck and avoids limitations associated with sampling, synchronization signals, and clock signal control, enabling real-time learning even for high-speed systems.

The last part of the work focuses on the practical application of FAANNs and thus demonstrates their potential. It is a fully analog adaptive high-frequency filter using the proposed neural networks in combination with a filter bank. Subsequent validation and analysis of the properties of this filter using electrical model behavior simulations explores the effectiveness of real-time adaptation.

Keywords: fully analog, neural network, real-time learning, backpropagation, neuromorphic, adaptive filter, analog filter, filter bank, high-speed.

Abstrakt

Tato disertační práce představuje koncept Plně Analogových Umělých Neuronových Sítí (FAANN), které jsou schopny zpracovávat signály přímo z elektrických senzorů. Jedním z motivů výzkumu je zlepšit přesnost měření s ohledem na degradaci senzorů a změny okolních podmínek. Druhý představuje možnost odvození neměřené fyzikální veličiny, která je závislá na těch měřených, což je obzvláště užitečné, když je velikost požadované veličiny obtížně zjistitelná, například kvůli dlouhému a nákladnému chemickému procesu.

Tato práce proto přichází s novým návrhem plně analogového učícího se procesu založeného na algoritmu zpětného šíření pomocí gradientního sestupu, který stále zůstává otevřeným problémem. Jedná se o problém v poslední době velmi diskutovaný z důvodu paralelizace výpočtů a tím zvýšení rychlosti učení. Navrhovaná struktura obchází von Neumannův bottleneck a také se vyhýbá omezením spojeným se vzorkováním, synchronizačními signály a řízením hodinových signálů, což umožňuje učení v reálném čase i v případě velmi rychlých systémů.

V poslední části se práce zaměřuje na praktické využití FAANN a demonstruje tak jejich potenciál. Jedná se o plně analogový adaptivní vysokofrekvenční filtr využívající navrhovaných neuronových sítí v kombinaci s bankou filtrů. Následná validace a analýza vlastností tohoto filtru pomocí simulací elektrického modelu zkoumá efektivitu adaptace v reálném čase.

Klíčová slova: plně analogový, neuronová síť, učení v reálném čase, zpětné šíření, neuromorfní, adaptivní filtr, analogový filtr, banka filtrů, vysokorychlostní.

Contents

List of figures

List of tables

List of source codes

List of acronyms

AANN Analog Artificial Neural Network

AC Alternating Current

ADC Analog-to-Digital Converter

AI Artificial Intelligence

ANN Artificial Neural Network

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

CLI Command Line Interface

CMOS Complementary Metal-Oxide-Semiconductor

CNN Convolutional Neural Network

CPU Central Processing Unit

DAC Digital-to-Analog Converter

DC Direct Current

EEG Electroencephalography

EMG Electromyography

EWC Elastic Weight Consolidation

FAANN Fully Analog Artificial Neural Network

FNN Feedforward Neural Network

FPGA Field-Programmable Gate Array

GEEC Graphic Editor of Electrical Circuits

GPT Generative Pre-trained Transformer

GPU Graphics Processing Unit

GUI Graphical User Interface

IoT Internet of Things

JSON JavaScript Object Notation

LMS Least Mean Squares

LSTM Long Short-Term Memory

MSE Mean Squared Error

NAS Neural Architecture Search

OP Operating Point

PCM Phase-Change Memory

PPG Photoplethysmography

QNN Quantum Neural Network

Q Quality factor

RAM Random Access Memory

RNN Recurrent Neural Network

ReRAM Resistive Random-Access Memory

SGD Stochastic Gradient Descent

SNN Spiking Neural Network

THD Total Harmonic Distortion

TPU Tensor Processing Unit

VDSL Very high-speed Digital Subscriber Line

Chapter 1
Introduction

Artificial Neural Networks (ANNs) have been created based on biological neural networks in the human brain. The concept of artificial neural networks originated in the mid-20th century when scientists such as Warren McCulloch and Walter Pitts investigated the way neural networks work in the brain. In 1943, they published a paper titled "A logical calculus of the ideas immanent in nervous activity", in which they described the essential functions of biological neurons and proposed a model of an artificial neuron [1]. This model became the basis for the development of artificial neural networks.

In the 1950s and 1960s, research on artificial neural networks continued to develop, and the first neural network learning algorithms were developed [2]. However, the limited computing power of computers at the time hampered the development of these networks [3]. In the 1980s, research on artificial neural networks intensified due to advances in parallel computing and the development of new learning algorithms [4]. The first commercial applications of artificial neural networks were also developed at this time, for example, for handwriting recognition and fraud detection [5]. In the 1990s, research on artificial neural networks slowed down as these networks proved to have several limitations, such as slow convergence, susceptibility to overtraining, and inability to explain results [6]. In recent years, research on artificial neural networks has picked up again, as these networks have been shown to have great potential for solving complex tasks such as image recognition [7], natural language [8, 9], and driving autonomous vehicles [10].

Therefore, an artificial neural network is a mathematical model inspired by biological neurons used to solve various machine-learning problems, especially for regression and classification tasks [11, 12]. This mathematical model consists of units called "artificial neurons" that are connected by weighted connections. Each artificial neuron receives input data, which is weighted and passed to an activation function that computes the output of the neuron. The neuron's output is then passed on to other neurons as input, or it is an output of the entire neural network. The learning of this neural network is based on changing the

weights. These are adjusted by various algorithms so that the network output corresponds to the desired output and thus achieves the correct interpretation of the input data [12, 13, 14].

Artificial neural networks are used in many different fields, such as:

- Image processing: classification, segmenting, and recognizing objects in an image [7, 15].
- Speech processing: speech recognition and speech synthesis [16].
- Language translation: machine translation, multilingual translation, contextual understanding, domain-specific translation [9, 17].
- Fraud detection: detecting fraudulent transactions and identifying criminals [18].
- Driving autonomous vehicles: detecting obstacles on the road and plan the best route for travel [10].
- Financial market forecasting: predicting the prices of stocks and other financial instruments [19].
- Weather forecasting: more accurate weather forecasts and warnings of extreme weather conditions [20].
- Robotics: controlling industrial robots and creating robotic prostheses [21].
- Medicine: analysis of medical images, diagnosing diseases, the detection of drug interactions, developing new drugs, or for personalized medicine [22, 23].

Due to their impressive results, ANNs are gaining popularity. They are being used to solve an increasing number of complex problems and are expected to play an increasingly important role in various aspects of our lives [11, 17].

At the same time, more and more different types of ANNs are being developed, differing in their architecture, each with its specific characteristics and applications [11, 20, 24, 25]. Here are some of the most well-known types of artificial neural networks and their focus:

- Perceptron: One of the simplest neural networks used for binary data classification. Its architecture consists of only one neuron [2].
- Feedforward Neural Network (FNN): The most basic and most widely used type of neural network. It is also known as a unidirectional neural network. The model consists of one input layer, one or more hidden layers, and one output layer [26].

- Convolutional Neural Network (CNN): Designed to process image data and have the ability to detect and classify objects in an image. It uses convolutional layers that can detect different image patterns [15, 27, 28, 9].
- Recurrent Neural Network (RNN): Designed to process data sequences such as text or audio. It has the ability to retain state over time and is used, for example, for prediction, text generation, or speech recognition [29, 19, 20].
- Autoencoder: Designed for learning compact representations of input data. It is used, for example, for data dimensionality reduction, image recognition, text translation, or data analysis [30, 31].

Almost all modern ANNs are computed based on the von Neumann architecture, which allows efficient implementation of algorithms for a wide range of problems. In this architecture, the computer's memory is used to store both data and instructions while the Central Processing Unit (CPU) fetches them sequentially. The speed of accessing data and instructions is limited by memory bandwidth and memory bus speed, which leads to inefficiency and performance limitations because the CPU spends a significant amount of time waiting for data and instructions. This phenomenon is known as the "von Neumann bottleneck" [32, 33, 34]. Therefore, in recent years, increasing attention has been paid to developing hardware implementations of ANNs that are able to perform computations faster and more efficiently than conventional CPUs [35, 36, 37].

One approach is the use of Graphics Processing Units (GPUs), which are specialized processors designed for graphics processing, but can also be used for training and testing neural networks. GPUs are capable of parallel computation and have greater computational capacity than CPUs, allowing larger complex networks to be trained and tested. They can achieve speed-ups in the teens, but they are not regarded as cost-effective [27, 28].

Another approach is to use specialized chips developed for computing tensors, Tensor Processing Units (TPUs), which are the cornerstone of ANNs. TPUs feature high-speed and energy efficiency, which can be up to 15 times faster than conventional GPUs and consume up to 30 times less power. The TPU architecture is designed to enable efficient real-time forward processing of large datasets [38, 36]. Similar chips can already be found in today's mobile phones used for image and voice processing. Similarly, specialized Field-Programmable Gate Arrays (FPGAs) are in the pipeline, which are programmable circuits that can be used to create specialized hardware architectures for neural networks [24].

However, as the popularity of ANNs grows, so does the scale and complexity of the problems, leading to an increase in the size of the ANNs needed to solve them [12, 15]. The size of networks is directly related to the computation time and energy consumed, which becomes a limiting factor for their real-time appli-

cations [39]. It is especially problematic for ANN problems involving concept drift, which is a phenomenon that describes a change in the distribution of data over time. Many factors, such as environmental changes, sensor degradation, user preferences change, or new technology developments, can cause concept drift [40, 41, 42]. If the distribution of data changes so much that the machine learning model is no longer able to accurately predict outputs for the new data, the model becomes invalid [43]. These problems occur, for example, in Internet of Things (IoT), real-time image recognition, and antenna signal processing.

Developers are working on creating specialized chips to address these problems. These chips, called Application-Specific Integrated Circuits (ASICs), are designed for specific applications. Neural networks done by ASICs offer much faster and more power-efficient performance compared to FPGAs, but they are more expensive to develop and manufacture [37]. These specialized chips often operate using a continuous signal, further increasing computation speed and reducing power consumption; such are known as Analog Artificial Neural Networks (AANNs) [34, 44]. However, the actual training of the models loaded onto these devices is primarily done using the traditional von Neumann architecture, as mentioned earlier.

The exception is often found in highly specialized neuromorphic computing. The aim is to develop efficient hardware and software systems that can process information, learn, and adapt, much like biological neural systems. Neuromorphic systems frequently employ specialized architectures and components for complex tasks, delivering high efficiency and low power consumption. These systems can consist of both analog and digital elements. They utilize circuits or processors specifically designed to mimic biological neurons and synapses, creating artificial neural networks for data processing, decision-making, and learning [45, 46, 47, 48].

The increasing use of the IoT and the necessity to process signals from more complex sensors like microphones, cameras, or antennas drives the development of edge computing architectures near sensors [49, 50, 51]. The significant computational power in this area motivates the design of hardware capable of analyzing data from nearby sensors without being affected by the von Neumann bottleneck. This is where analog artificial neural networks come into play, as they are not highly specialized and can process data in real-time with low power consumption. Training these networks requires a learning algorithm such as backpropagation using gradient descent [14] to implement real-time training hardware. However, a fully analog training process based on the backpropagation learning algorithm is still not fully developed and is still an open problem [52, 53].

By incorporating an analog learning algorithm, it becomes possible to transition from standalone software or FPGA units to an on-chip analog train-

ing implementation, resulting in a fully hardware-based artificial neural network. While this may decrease the flexibility of the structure, it also provides a significant acceleration in the training process of the artificial neural network [33, 44, 48, 54].

Many on-chip learning process designs for both neuromorphic systems and analog neural networks rely on clock control. This approach enables a more flexible structure but slows down the learning process or makes it impossible to process unsampled signals directly from analog sensors [50, 51]. Embracing a fully analog learning process offers several benefits. First, it reduces the need for data conversion between analog and digital domains, saving time and energy. Second, an analog implementation allows continuous learning directly from analog sensor data, facilitating real-time adaptation of the concept drift in real-time applications. Lastly, fully analog learning leads to fully parallel signal processing, yielding significant speed-up and power savings compared to digital implementations.

For the above reasons, this dissertation presents and verifies a new concept of a Fully Analog Artificial Neural Network (FAANN) that implements a circuit-based solution for training an artificial neural network. The proposed concept is inspired by the backpropagation algorithm and is based on gradient descent. The goal is to develop a hardware ANN that can effectively address the limitations of the von Neumann architecture and avoids synchronization signals, making the concept fully analog and fully parallel. It significantly improves computation speed, power consumption, and real-time usability for tasks such as IoT applications, image processing, and high-frequency signal control.

Chapter 2
Objectives of the dissertation

This dissertation focuses on addressing several objectives to develop and validate a novel concept for processing signals from various electronic sensors.

1. The primary objective is to design a new Fully Analog Artificial Neural Network (FAANN) concept for processing signals from an array of different electronic sensors. This concept should bypass von Neumann's Bottleneck and avoid any synchronization signals, enabling real-time learning capabilities.

2. The next objective is to validate the functionality of the proposed structure by analysis through an electrical behavioral model. It should ensure that the proposed structure will be able to learn and with sufficient speed. Additionally, it will analyze the fundamental behavior of this concept and compare it to traditional ANN approaches.

3. Another objective is to explore the practical applications of the FAANN system by utilizing it to develop an adaptive frequency filter. The adaptive filter should be designed to operate at higher frequencies to show the potential of FAANN.

4. The final objective is to verify the functionality of the designed adaptive filter concept, which is based on FAANN, through additional simulations, particularly considering their configurability and adaptability. These simulations should describe the fundamental advantages and disadvantages.

Chapter 3
State of the art

Nowadays, Artificial Neural Networks (ANNs) have become a crucial part of many diverse fields due to their ability to learn, adapt, and solve complex problems. They are designed to mimic the functioning of the human brain, which consists of interconnected processing nodes, also known as neurons [11].

A significant development in recent years has been the advent of deep learning, a subset of machine learning that utilizes ANNs with many hidden layers [7, 9, 16, 21, 35]. Specifically, Convolutional Neural Networks (CNNs) have triggered image and video processing breakthroughs. These networks use convolutional layers to extract features from images and videos more efficiently than traditional methods [15, 34, 55, 56].

A promising area is the development of capsule networks, a concept introduced as an alternative to CNNs [31]. These networks are capable of processing spatial relationships between objects in an image, potentially surpassing some of the limitations of CNNs. Although still in the experimental stage, they could possibly bring about a revolution in image recognition tasks [57].

On the other hand, Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) have proven to be exceptionally promising for sequence prediction problems, such as natural language processing or time series prediction [8, 58]. These networks have an internal state that allows them to remember previous inputs in the current decision-making process. Thanks to their ability to handle time-dynamic behavior, RNN, and LSTM have become key in sequence-based tasks [29]. Examples of RNN and LSTM usage include weather forecasting and even stock price prediction [19, 20].

A significant milestone in the field of ANNs is the emergence of transformers. Unlike traditional RNNs or CNNs, transformers rely solely on self-attention mechanisms to capture dependencies between different positions in a sequence. This attention mechanism allows transformers to efficiently process long-range dependencies and contextual information without needing sequential processing, making them highly parallelizable and faster to train [59, 60].

Transformers have paved the way for novel architectures such as the Generative

Pre-trained Transformer (GPT) models, widely used for text generation tasks [17]. GPT-4, so far the largest model in the series, has demonstrated impressive language generation capabilities, producing coherent and contextually relevant text across a wide range of prompts. It is currently being explored for various applications, such as its benefits and risks for medical applications [22].

With the arrival of quantum computers, Quantum Neural Networks (QNNs) have emerged as a potential future direction for ANNs. QNNs leverage quantum phenomena to enhance the speed and capacity of traditional neural networks. Although they are only in the initial phase, they could lead to significant progress in this field, but substantial issues persist regarding hardware stability and error correction [25, 61, 62].

ANNs have several advantages that make them suitable for many machine-learning tasks [11]. Some of these include:

- Noise processing: ANNs are highly robust and can handle noise in input data. It makes them suitable for real-world situations where data may not be clean or may contain errors [12, 14].
- Nonlinearity: ANNs are capable of modeling complex nonlinear relationships, which can be critical in many real-world applications where relationships between variables are not straightforward [63].
- Generalization: After training, ANNs can generalize learned examples to new ones. This ability makes them useful for image recognition or natural language processing tasks [7, 8].

Despite remarkable progress, there are still persisting issues in the field of ANNs. Some of these include:

- Lack of transparency or the black box problem: It is often difficult to understand how ANNs make their decisions, as this process can be very complex and opaque. This lack of interpretability can be problematic in situations where it is necessary to understand the decision-making process, such as in healthcare or legal environments [64].
- Overfitting: ANNs, particularly deep neural networks, are prone to overfitting, especially when working with a small dataset. Overfitting means that the network may perform well on training data but poorly on new, unseen data [65].
- Architecture design complexity: ANNs are usually complex architectures, and their design requires considerable effort and expertise. This is currently being addressed by Neural Architecture Search (NAS) algorithms, which automate the design of artificial neural networks and represent a

new development in ANN technology. NAS algorithms attempt to create models that outperform human-designed models, which would significantly accelerate the model development process [66].

- Training time and computational resources: Training ANNs can require substantial amounts of time and computational resources, especially in the case of large datasets or complex architectures. Therefore, they are less suitable for smaller projects or organizations with limited computational resources [32, 43, 59, 60, 67].

Consequently, much recent research is focused on addressing these problems. For example, various techniques such as quantization, compression, and acceleration are being explored in the area of computational resources [35, 68, 69].

Neural network accelerators have recently become a central focus of hardware development that facilitates the computational demands of ANNs. These specialized hardware accelerators, designed to speed up the execution of neural network tasks, have significantly increased the speed and efficiency of training and deploying ANNs [33, 35, 70].

Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs) are two types of neural network accelerators widely used due to their efficient energy use and high-performance capabilities. FPGAs provide a flexible architecture that allows for reprogramming the hardware best to suit the specific needs of a neural network model. On the other hand, ASICs are specifically designed for specific tasks, leading to significantly higher performance compared to more general hardware. However, ASICs lack the flexibility of FPGAs as they cannot be reprogrammed after production [24, 37].

Graphics Processing Units (GPUs) also play an important role in accelerating neural networks, thanks to their highly parallel structure suitable for matrix and vector operations used in ANNs. Graphics processors, originally designed to accelerate computer graphics, have now found widespread use in the field of Artificial Intelligence (AI). At the forefront of this shift are GPUs from companies like NVIDIA, whose CUDA software platform has made it easier for developers to utilize the parallel computing power of GPUs [27, 28, 37].

Another example of a specialized neural network accelerator is the Tensor Processing Unit (TPU). TPUs are Google's custom-built ASICs specifically designed to accelerate machine learning tasks. They have been optimized for TensorFlow, Google's open-source machine learning framework. With their unique architecture and design, TPUs are able to execute large-scale tensor computations efficiently, resulting in significantly faster processing times for tasks such as training and inference in neural networks [35, 36, 38].

Despite the substantial improvements by neural network accelerators, challenges still exist. Active research areas include energy efficiency, computational

power, cost, and the ability to keep up with the rapidly changing environment [40, 43]. Given the pivotal role of neural networks in a wide range of applications, from image recognition to natural language processing, progress in the field of neural network accelerators is expected to continue rapidly in the coming years [35].

Real-time learning in neural networks continues to be a thriving field of research and development, with applications expanding beyond autonomous vehicles, robotic systems, and dynamic recommendation systems. The ability of neural networks to adapt their knowledge on-the-fly, learning from new data as it becomes available, without requiring full retraining, is a cornerstone of modern AI systems [10, 21, 39, 38].

Traditionally, most neural network models are trained on a static dataset and then deployed for inference, with no ability to learn from new data. The learned model is then rendered obsolete and non-valid. However, as the demand for real-time learning capability has grown, so has the suite of methods designed to achieve it [40, 41, 42].

One of the approaches to this is federated learning, where a model is trained across multiple decentralized edge devices, learning from new data as it arrives. This approach not only facilitates real-time learning but also respects the privacy of the data, as the raw data does not need to leave the device. It has proven particularly useful in applications like personalized recommendation systems and healthcare analytics, where data privacy is paramount [71].

Transfer learning is another technique that bolsters real-time learning, where a pre-trained model is used and fine-tuned for a different but related task. This approach can expedite the learning process, especially in real-time scenarios where a model can leverage prior knowledge to make better predictions or decisions. It is especially prevalent in natural language processing tasks and image classification problems [72].

Ensemble learning is another way to enhance real-time learning. In this approach, multiple models or versions of a model contribute to the final prediction. It can increase the robustness and generalizability of the model, making it better-equipped to handle the dynamic nature of real-time data [73].

Continuous learning, also known as lifelong learning or incremental learning, is an approach in machine learning where models are designed to acquire new knowledge, adapt to changing data, and improve their performance over time [74]. It addresses the issue of catastrophic forgetting, which occurs when a neural network learns new tasks and forgets how to perform previously learned tasks. Techniques like Elastic Weight Consolidation (EWC) have shown promise in mitigating this issue, allowing the neural network to learn new tasks in real-time without forgetting the old ones [75].

With the beginning of edge computing and IoT, the need for lightweight

real-time learning models that can operate on devices with limited computational resources is increasing [49, 50, 51]. To this end, various techniques are being explored, such as model quantization, knowledge distillation, and the development of compact neural network architectures like MobileNet [55] and EfficientNet [56]. Another promising direction is the application of TinyML, which specializes in deploying machine learning capabilities on embedded systems, further driving the feasibility of real-time learning in resource-constrained environments [76].

One of the answers to this challenge is the growing research of Analog Artificial Neural Networks (AANNs) and also neuromorphic computing systems, which present a promising trend in the hardware for machine learning. These systems use physical, electrical circuits to mimic the behavior of neurons and synapses, the basic units of biological brains. These circuits, crafted with advanced semiconductor technologies, enable simultaneous data storage and analog computation, potentially significantly improving energy efficiency and speed of machine learning operations [45, 77, 78].

Several key advancements have been made in this field. Notably, neuromorphic computing chips like IBM's TrueNorth [70] and Intel's Loihi [45] have gained recognition. These chips contain Spiking Neural Networks (SNNs), which more closely resemble the activity of biological neurons compared to traditional artificial neural networks. This allows these systems to process data in a highly parallel and energy-efficient manner, enabling more powerful and compact artificial intelligence systems [79].

Another significant advancement in analog neural networks is the emergence of memristors, electronic components which behave as a variable resistance whose value depends on the amount of electric charge flowing through it. Memristors can be used to construct analog weight elements for AANNs, effectively implementing synapse connections in a neuromorphic system. This allows data storage and manipulation in the same place, thereby reducing data movement and improving energy efficiency [44, 46, 50, 53, 77].

While significant progress has been made in the area of AANNs and neuromorphic systems, there are still challenges to overcome. One of the main challenges is the analog implementation of learning algorithms. Most current neuromorphic systems use classic digital training algorithms based on gradient descent, which are computationally and memory intensive. Research in this area focuses on developing new algorithms suitable for analog implementations that use the natural physical properties of electronic devices to perform mathematical operations [47, 53, 79].

Analog circuits have been used to implement various learning algorithms, such as backpropagation and Hebbian learning [53, 80]. In backpropagation-based systems, weights are usually stored in digital memory and converted

to analog signals during computation. However, new analog memory devices, such as Phase-Change Memory (PCM) and Resistive Random-Access Memory (ReRAM), have been used to store weights directly in analog form, leading to more efficient learning systems [34, 35].

However, the analog implementation of learning neural networks is not without challenges. Analog devices suffer from issues such as device variability, nonlinearity, and noise, which can lead to inaccurate computations. Moreover, integrating analog devices with digital systems presents substantial challenges. As the technology matures and more robust designs and error correction methods are developed, these obstacles are expected to be overcome, paving the way for the widespread use of analog neural networks in real-world applications. It can cause a shift from the traditional digital-centric AI approach to one that combines digital and analog computation, potentially leading to more powerful, energy-efficient, compact AI systems, and efficient on-device machine learning applications [55, 56, 77, 70].

Despite the advancements achieved thus far, real-time learning of neural networks still faces significant challenges too. Striking a balance between stability and flexibility, computational power and memory constraints, as well as privacy and data security issues, are subjects of ongoing research. Nevertheless, considering the escalating demand for real-time learning capabilities across many applications, substantial progress in this realm is anticipated in the years to come [37, 39, 71].

In conclusion, though still an expeditiously evolving field, analog neural networks, and neuromorphic systems hint at a new generation of AI hardware. Such systems could potentially revolutionize the machine learning domain and provide notable improvements in energy efficiency and speed compared to traditional digital systems. The ongoing research lays a base for more robust and efficient artificial intelligence systems in the future. The rapid progress in this field suggests more exciting developments in the coming years with potential impacts across various sectors [7, 10, 11, 21, 22, 51].

Chapter 4
New structure

This chapter introduces a novel concept for Fully Analog Artificial Neural Networks (FAANNs) [52], specifically designed to meet certain criteria. The network is configured to process various signals from a diverse array of electronic sensors. These include basic sensors that measure temperature, light intensity, and electrical conductivity, as well as more complex sensors such as microphones, cameras, or antennas. Another criterion is to enhance measurement accuracy by accounting for sensor degradation and changing environmental conditions, a phenomenon known as concept drift [40]. Additionally, the FAANN is designed to infer unmeasured quantities dependent on the measured ones. It is an important characteristic when the quantity of interest is costly or time-consuming to measure directly.

These requirements led to the design of FAANNs to be used for real-time learning, even for high-speed applications. The acceleration of the neural network's training process in this research is grounded on several principles. These allow it to effectively bypass the limitations of the von Neumann architecture [32], avoid synchronization signals, clock control, and overcome constraints related to sampling, Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC).

The novel concept of FAANNs introduces a circuit-based solution for training artificial neural networks. This innovative concept is inspired by the backpropagation algorithm and is built on the principles of gradient descent [14], aiming to establish a hardware-based training process. The training process is executed through feedback from an analog electrical circuit, making the entire operation fully analog. This approach also eliminates the need for any clock control that could slow down signal propagation, thereby allowing the neural network to work fully parallel [44, 53, 54].

The newly designed structure aims to maximize versatility, potentially accommodating as many types of neural networks as possible. Therefore, it is designed as a feedforward neural network, serving as the foundation for creating various neural network types, all consisting of identical cells.

4.1 Blocks

When constructing a FAANN, it is essential to include several components, often referred to as "blocks". These blocks help describe the neural network's design process more straightforwardly and clearly.

The electrical implementation of these blocks can be achieved using various analog components, such as operational amplifiers, transistors, and resistors. However, for verifying the functionality of this concept, which is the objective of this dissertation, these blocks are temporarily considered ideal for the sake of simplicity.

In this section, each of these blocks is represented with a schematic symbol, mathematical descriptions, and NetList notations for subsequent simulation.

4.1.1 Multiplier

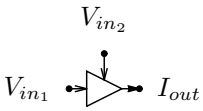

Figure 4.1: Symbol of multiplier.

The multiplier block has the schematic symbol shown in Figure 4.1. It is a function that multiplies two voltages and whose output is a current calculated by

$$I_{out} = K_m \cdot V_{in_1} \cdot V_{in_2}, \tag{4.1}$$

where K_m [A/V^2] is the ratio of output to the product of inputs.

The NetList notation of this block used in simulations is shown in Code 4.1.

```
1 .subckt multiplier in1 in2 out
2 B1 0 out I=0.0005*v(in1)*v(in2)
3 .ends multiplier
```

Code 4.1: NetList notation of multiplier subcircuit.

4.1.2 Activation function

(a) Linear. (b) Nonlinear.

Figure 4.2: Symbols of activation functions.

The blocks representing an activation function have the schematic symbols shown in Figure 4.2. The most commonly used activation function is a sig-

moid [12], which is described by

$$V_{out} = \frac{V_{amp}}{1 + e^{-\frac{I_{in}}{I_{ref}}}}, \tag{4.2}$$

where V_{amp} is a constant determining the maximum possible voltage and I_{ref} is a referential current.

The notations for three distinct activation functions used in simulations, namely the sigmoid, hyperbolic tangent, and linear identity activation functions, are depicted in the NetList language in Code 4.2. Some of these activation functions exhibit nonlinearity in their formulation. This nonlinear behavior can be attributed to two factors. The first relates to the restrictions by voltage constraints, which influence the current traversing the system. Secondly, this nonlinearity has characteristics of the blocks nearer to their real implementation [54].

```
1  .subckt activationFunctionSigmoid in out
2  B1 out 0 V=1/(1+exp(-v(in)*5))
3  .ends activationFunctionSigmoid
4
5  .subckt activationFunctionTanh in out
6  B1 out 0 V=tanh(v(in)*2)
7  .ends activationFunctionTanh
8
9  .subckt activationFunctionIdentity in out
10 B1 out 0 V=5*tanh(v(in)/5)
11 .ends activationFunctionIdentity
```

Code 4.2: NetList notation of activation functions.

4.1.3 Derivative of an activation function

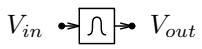

Figure 4.3: Symbol of derivation of activation function.

The block representing the derivative of an activation function has the schematic symbol shown in Figure 4.3. In the case of the sigmoid activation function, the block is described by

$$V_{out} = K_m \frac{e^{\frac{V_{in}}{V_{ref}}}}{(e^{\frac{V_{in}}{V_{ref}}} + 1)^2}, \tag{4.3}$$

where K_m is a proportionality voltage constant and V_{ref} is a referential voltage.

The NetList notation for the three different derivations of the activation function that are used in the simulations are shown in Code 4.3. These are

specifically the derivatives of the nonlinear sigmoid, tangent hyperbolic, and linear identity activation functions.

```
.subckt diffActivationFunctionSigmoid in out
B1 out 0 V=4*exp(v(in)*5)/((exp(v(in)*5)+1)^2)
.ends diffActivationFunctionSigmoid

.subckt diffActivationFunctionTanh in out
B1 out 0 V=2/(cosh(4*v(in))+1)
.ends diffActivationFunctionTanh

.subckt diffActivationFunctionIdentity in out
B1 out 0 V=1
.ends diffActivationFunctionIdentity
```

Code 4.3: NetList notation of derivative of activation functions.

4.1.4 Subtractor

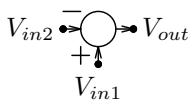

Figure 4.4: Symbol of voltage subtractor.

The subtractor block has the schematic symbol shown in Figure 4.4. It is a function that subtracts two voltages and whose output is calculated by

$$V_{out} = V_{in_1} - V_{in_2}. \tag{4.4}$$

The NetList notation of this block used in simulations is shown in Code 4.4.

```
.subckt subtractor in1 in2 out
B1 out 0 V=v(in1)-v(in2)
.ends subtractor
```

Code 4.4: NetList notation of subtractor.

4.2 Forward propagation structure

The FAANN comprises forward and backward propagation, two crucial components that will be further discussed. Forward propagation involves processing input data through the neural network layers to produce an output. In contrast, backward propagation focuses on adjusting the network weights based on the output errors [12]. This section describes the design and concept implementation of forward propagation in the FAANN.

The focus is on the forward propagation of the formal neuron, which serves as the foundation for other types of neural networks. A graphical representation

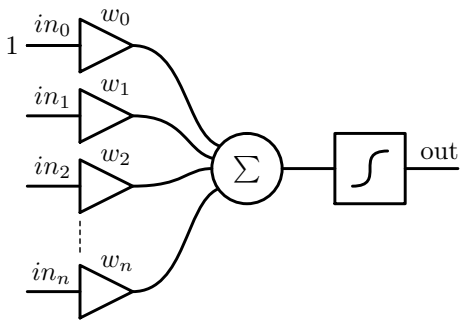

Figure 4.5: Formal neuron.

of this part of the formal neuron is shown in Figure 4.5. In a neural network, forward propagation involves calculating the output of each neuron in the network by applying a set of mathematical operations.

The fundamental equation for the output of a neuron can be expressed as

$$out = S\Big(\sum_{i=0}^{N} w_i in_i\Big), \tag{4.5}$$

where out is the output of the neuron, in_i represents the input values, w_i denotes the corresponding weights, in_0 is the bias term, and S is the activation function. This equation captures the core principle of forward propagation, where input values are multiplied by their respective weights, summed up, and adjusted with a bias term before being passed through the activation function to produce the neuron's output [12, 14].

The forward propagation process starts at the input layer and continues through hidden layers until it reaches the output layer. At each layer, the neurons receive input from the previous layer, perform the above calculations, and pass the output to the next layer. The choice of activation function S is critical in introducing nonlinearity into the network, allowing it to model complex relationships between inputs and outputs [14].

The hardware implementation of forward propagation in neural networks has been a subject of extensive research and development, as it plays a crucial role in the performance of these networks. Numerous studies have explored various approaches to implementing forward propagation in both digital and analog circuits, aiming to optimize speed, energy efficiency, and accuracy [33, 48, 53, 54, 81, 82]. Hardware solutions, such as ASICs, FPGAs, and custom analog circuits, have been employed to execute the complex computations involved in forward propagation. These approaches have made significant strides in reducing the latency and power consumption of neural networks, making them more suitable for real-time applications and near-sensor processing [39, 50]. As a result, hardware implementations of forward propagation continue to advance

the state-of-the-art in neural network technology, enabling new possibilities in machine learning, signal processing, and a wide range of applications [33, 35, 53].

Such a formal neuron in analog form has been implemented numerous times in various designs. For this structure, all its parts will be considered as voltage components. This means that the neuron's inputs, weights, and outputs are represented as voltages. The fundamental equation for the output of a voltage-based formal neuron can be expressed as

$$V_{out} = S\Big(\sum_{i=0}^{N} V_{w_i} V_{in_i}\Big), \tag{4.6}$$

where V_{out} is the output voltage of the neuron, V_{in_i} represents the input voltage values, V_{w_i} denotes the corresponding weight voltages, V_{in_0} is the bias voltage, and S is the activation function. In this configuration, the neuron performs the calculations using voltage levels rather than numeric values.

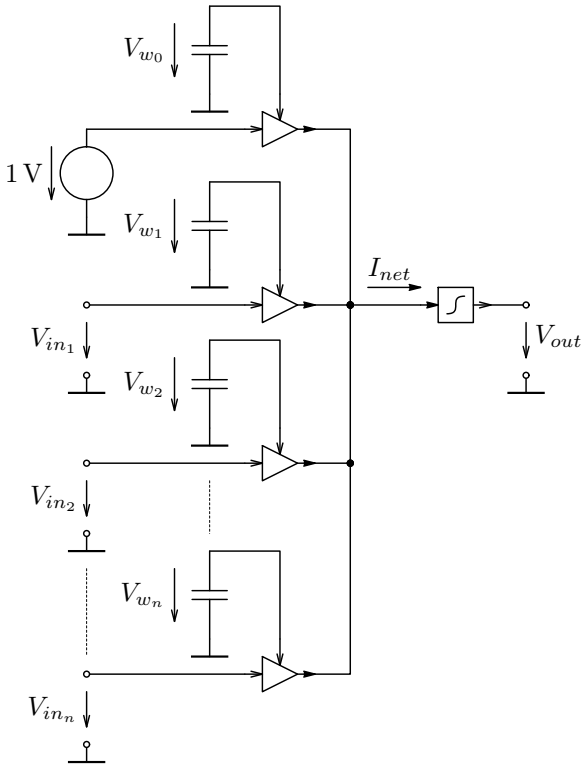

Figure 4.6: Analog implementation of forward propagation of a formal neuron.

The entire behavior of the designed forward propagation of the FAANN is described by Equation (4.6), and the corresponding analog concept design is illustrated in Figure 4.6. The weights V_{w_i} are represented by voltages stored on capacitors, which will be elaborated in the subsequent section. Each input is

connected to a multiplier, where the output is a current determined by Equation (4.1). The sum operation is performed using Kirchhoff's circuit laws at the node preceding the activation function; the resulting current is denoted as I_{net}. Finally, the activation function is applied to the sum.

4.3 Novel backpropagation structure

The most widely used approach to training artificial neural networks involves a combination of backpropagation and either Stochastic Gradient Descent (SGD) or one of its variations [14]. In the field of machine learning, backpropagation stands out as a crucial algorithm, essential for training not only feedforward neural networks but also any differentiable parameterized networks [12].

Backpropagation is an algorithm that computes the gradient of the loss function with respect to the network's weights $\frac{\partial E}{\partial w_k}$ for a single pair of input-output data. This algorithm leverages the chain rule from calculus, a rule used to compute the derivative of the composition of multiple differentiable functions. This method can be used for multiple weight updates by calculating the gradient of the loss function for each example in the training dataset.

The renowned backpropagation algorithm is commonly combined with the stochastic gradient descent method. In this combination, the weight update is defined by the following equation

$$w_{k+1} = w_k - \eta \frac{\partial E}{\partial w_k}, \tag{4.7}$$

where w_k represents a weight at step k (with $k \in \mathbb{N}$, the set of natural numbers), E symbolizes the error, and η denotes the learning rate. This equation defines how the weights of the network should be adjusted to minimize the error between the network's output and the expected output [12].

There are two primary ways of training analog neural networks. The first one employs gradient descent and backpropagation of the error, which is a method also utilized in traditional networks. This process mainly takes place in the digital component of the system and then gets transmitted to the analog part [48]. However, this approach is subject to the von Neumann bottleneck [32].

The second way involves designing systems that replicate the structure and function of the brain, a field sometimes referred to as neuromorphic computing or neuromorphic engineering. This technique deviates from the digital computations performed by conventional artificial neural networks. The training of these networks usually necessitates specialized algorithms based on the physical properties of the used components, and are currently developed only for specific tasks [46].

Regardless, these techniques generally require some form of gradient descent and backpropagation or their approximations, as these are the most effective

methods known for training networks. However, the application of these methods on analog hardware can be demanding, and a significant portion of research in this field is dedicated to discovering effective ways to achieve this.

This work introduces a learning method suited for near-sensor applications and capable of functioning in real-time. The method is rooted in backpropagation with SGD, chosen for reasons previously discussed.

Designing this learning process in a completely analog manner presents a significant challenge due to the stepwise nature of the algorithm. As illustrated in Equation (4.7), k represents a step that needs to be bypassed.

To address this challenge, the process is reconstructed within a continuous domain. It is realized by substituting the step variable k with continuous time t, and integrating the weight w over time. Consequently, the revised formula for weight change appears as

$$w(t) = w(t_0) - \eta \int_{t_0}^{t} \frac{\partial E(\tau)}{\partial w(\tau)} d\tau, \tag{4.8}$$

where $w(t)$ denotes the weight function at continuous time $t \in \mathbb{R}$, and $E(\tau)$ represents the error, which is changing continuously over time too.

4.3.1 Weights implementation

When designing an analog learning circuit, the selection of the suitable component to store and adjust the "weight" value, a key part of the learning process, is crucial.

Memristors and memristor arrays have recently emerged as a promising solution for this. They store the weight value as a resistance level and are particularly appealing due to their similarity to biological synapses. Additionally, their compactness and low power consumption make them an attractive option [77, 83]. However, this technology is still evolving and needs to be more robust for some applications [53].

Another choice is to use capacitors for their ability to store charge. They are a proven and robust technology frequently used in integrated circuits. Their ubiquity within these circuits makes them particularly convenient for concept validation, as their functionality is straightforward to demonstrate.

However, using capacitors in integrated circuits has its drawbacks, mainly their size. Each neuron would require $N + 1$ capacitors as weights, with N being the number of inputs. Larger structures would mean a bigger chip area. The specific implementation depends on technology, but there could be tens of thousands of capacitors on $1\,\mathrm{mm}^2$ of a chip. That is sufficient for a large number of fast near-sensor applications.

The capacitor can be used in several different ways to store a value. In this study, the capacitor's voltage is specifically used as a direct representation of this weight.

For a design where no clock is used to synchronize the process of reading a value and writing a new one, there needs to be some method to ensure that the value changes. It is achieved by connecting a current source to the capacitor to modify its voltage.

In this case the charging of the capacitor is realized by a current source according to the known equation

$$V_w(t) = \frac{1}{C}\int_{t_0}^{t} I(\tau)d\tau + V_w(t_0). \tag{4.9}$$

where V_w is the voltage across the capacitor.

This charging current is to be denoted as $I_{charged}$ and is substituted into the proposed schema of analog forward propagation. The result is the circuit shown in Figure 4.7.

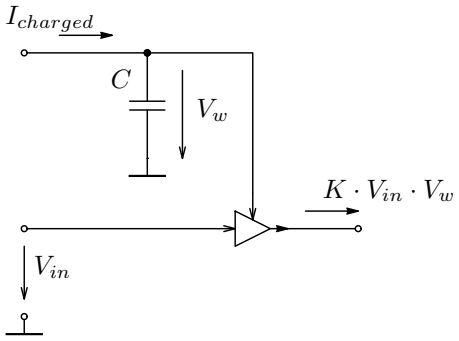

Figure 4.7: Analog implementation of a neural network weight.

If this learning process is inspired on the backpropagation, the capacitor charging current, in this case $I_{charged}$ should be implemented by successively charging the capacitor according to Equation (4.8). This charging current is designed according to

$$I_{charged} = -K_\eta \frac{\partial E}{\partial V_w}, \tag{4.10}$$

where K_η [S] is the conductance coefficient of $I_{charged}$ to partial derivative of the error with respect to weight.

By substituting Equation (4.10) into the general Equation (4.9) for charging a capacitor, the result is

$$V_w(t) = V_w(t_0) - \frac{K_\eta}{C}\int_{t_0}^{t} \frac{\partial E(\tau)}{\partial V_w(\tau)}\, d\tau. \tag{4.11}$$

Comparing this Equation (4.11) with the learning Equation (4.7) for weight change in continuous time, a clear similarity is observed

$$\frac{K_\eta}{C} \equiv \eta, \tag{4.12}$$

where η is the learning rate. Let's define K_η as the learning rate coefficient for FAANN.

4.3.2 Error propagation

The backpropagation process fundamentally relies on solving the formula for weight change, referenced in Equation (4.7). This design is transiting to the Equation (4.11), which requires computing the partial derivative of the error function E with respect to the weight V_w. This process involves the application of Leibniz's chain rule to these networks [12].

To simplify our calculations, we introduce a new variable, *net*, defined as

$$net = \sum_{i=0}^{N} V_{w_i} V_{in_i} \tag{4.13}$$

where the unit of *net* is V^2.

Applying the chain rule twice to the equation for the partial derivative of the error with respect to the weight V_w is the result

$$\frac{\partial E}{\partial V_w} = \frac{\partial E}{\partial V_{out}} \frac{\partial V_{out}}{\partial net} \frac{\partial net}{\partial V_w}. \tag{4.14}$$

Before calculating the first partial derivative for Equation (4.14), defining the error function E is necessary. Often referred to as a loss function or cost function, that is a measure of how much the network's output deviates from what was expected. The primary objective of the neural network is to minimize this error. Choosing an appropriate error function can significantly influence the network's learning rate and stability. In this context, the Mean Squared Error (MSE) is selected as the error function due to its simplicity and ease of implementation in an analog way. The MSE is often favored for regression problems [14].

The MSE is mathematically represented as

$$E = \frac{1}{n} \sum_{i=1}^{n} (V_{out_i} - V_{target_i})^2, \tag{4.15}$$

where E $[\mathrm{V}^2]$ is the error, n is the number of outputs, V_{target_i} is the ideal or expected output and V_{out_i} is the obtained output signal.

To simplify future interpretation, let us define the first partial derivative from Equation (4.14) as

$$V_E = \frac{\partial E}{\partial V_{out}}, \tag{4.16}$$

where V_E [V] is referred to as the voltage error.

The actual calculation of the error voltage V_E is the derivative of the error function E from Equation (4.15) according to one output voltage V_{out}. The result is written as

$$V_E = V_{out} - V_{target}, \tag{4.17}$$

where the result is the difference between the output and target voltage.

The universal solution for the second partial derivative from Equation (4.14) is not unambiguous due to its direct dependency on the activation function, which can differ for each neuron. Specifically, it represents the partial derivative of the activation function with respect to *net*. With a defined activation function, the calculation of this partial derivative becomes feasible. In the context of this electrical design, the function necessitates its creation as an independent sub-circuit. For example, the partial derivative is a constant value in scenarios where the activation function is linear. Concrete examples of derivatives of activation functions as independent sub-circuits are written in Code 4.3 and have the schematic symbol shown in Figure 4.3.

The last partial derivative from Equation (4.14) is solved as

$$\frac{\partial net}{\partial V_w} = V_{in}. \tag{4.18}$$

The entire calculation leads to determine the magnitude of the current that can update the weight V_w. This current can be expressed by substituting into Equation (4.10) as

$$I_{charged} = K_\eta (V_{target} - V_{out}) \frac{\partial V_{out}}{\partial net} V_{in}, \tag{4.19}$$

where all elements are already known.

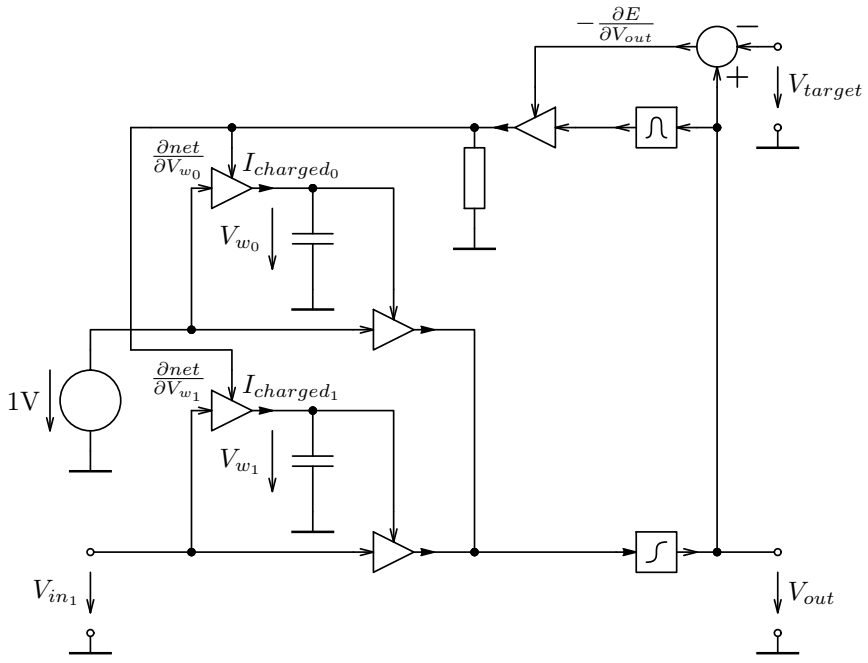

Figure 4.8: Analog implementation of backpropagation for the output layer.

The design of the learning circuit for the output layer can be implemented into the block diagram as seen in Figure 4.8. All parts of Equation (4.19)

are designed in the circuit as known blocks. A more detailed description of individual blocks can be found in Section 4.1.

In the block concept used in this whole work, resistor symbols are used as current-to-voltage converters. This notation is a well-recognized representation of this phenomenon, thus achieving better readability of the block diagrams. However, the resulting implementation could significantly differ.

4.3.3 Error propagation between layers

The backpropagation of each neuron in the hidden layers does not directly depend on the error function. Instead, the backpropagation of each neuron in the hidden layers is influenced by the errors of all neurons that are connected to its output [12].

Thus, the computation of a neuron's error voltage in the hidden layers is dependent on calculation of the sum of the errors of neurons in the layer after according to their input. This relationship is mathematically represented as

$$V_E = \sum_n \frac{\partial E}{\partial V_{in_n}}, \tag{4.20}$$

where n represents a neuron connected to the output of the neuron whose error voltage is being calculated.

Again, the chain rule is used twice to calculate this formula. The modified equation then becomes

$$V_E = \sum_n \frac{\partial E}{\partial V_{out_n}} \frac{\partial V_{out_n}}{\partial net_n} \frac{\partial net_n}{\partial V_{out}}. \tag{4.21}$$

The first two partial derivatives in the sum are solved in Section 4.3.2. The last term in the equation is resolved as

$$\frac{\partial net_n}{\partial V_{out}} = V_{w_n}, \tag{4.22}$$

where V_{w_n} is the voltage on the weight of neuron n.

In summary, the expression can be simplified to

$$V_E = \sum_n V_{E_n} \frac{\partial V_{out_n}}{\partial net_n} V_{w_n}. \tag{4.23}$$

The design example of a learning circuit with a hidden layer is shown in Figure 4.9. In this figure, the layers are marked in blue borders, while green borders highlight the output layers. The evolution of this design is based on Equation (4.23), whose implication is shown in red in the figure.

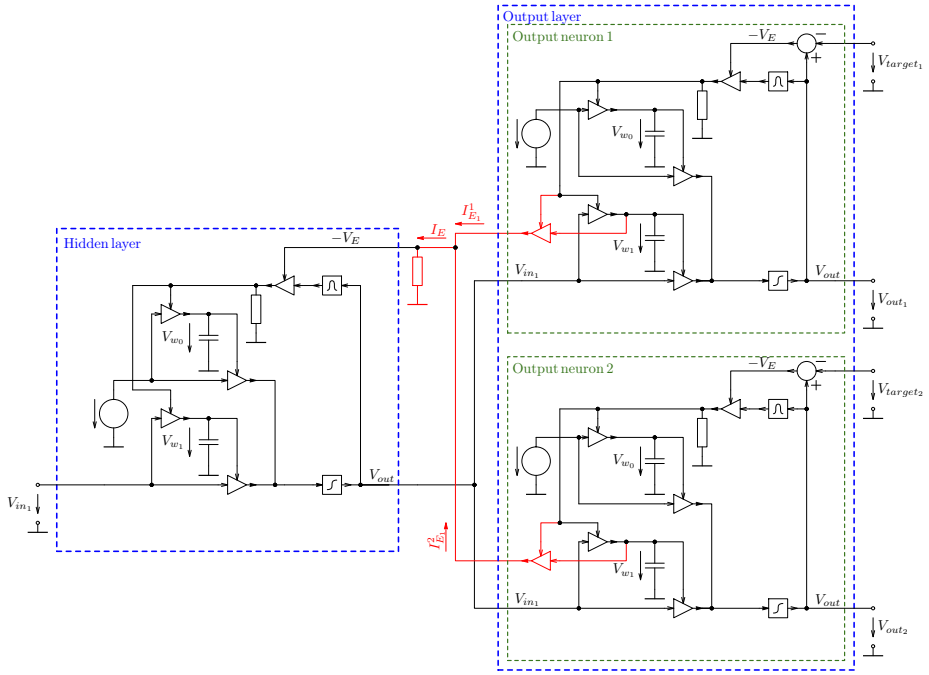

Figure 4.9: Analog implementation of backpropagation between layers.

Each computation in the summation process is executed using a single multiplier. The following equation determines this multiplier's output

$$I_{E_i} = -G_E V_E \frac{\partial V_{out}}{\partial net} V_{w_i}, \tag{4.24}$$

where i is the neuron input number, and G_E [S] is the electrical conductivity constant.

The actual summation is again performed according to Kirchhoff's circuit laws at the node according to

$$I_E = \sum_n I_{E_i}^n. \tag{4.25}$$

where for FNN, i is the output number of the counted neuron, which is the same as the input number according to Equation (4.24), and n is the neuron number at the output of the counted neuron.

4.4 Circuit design

The final step of creating this new structure is the formulation of a circuit capable of representing a fully analog neuron. This circuit includes both forward and backward propagation, irrespective of whether it is part of the input layer or any other layer within the neural network. Furthermore, the circuit design should remain as simple as possible. It is also necessary for the circuit to allow for adjustments in the learning rate without requiring structural modifications.

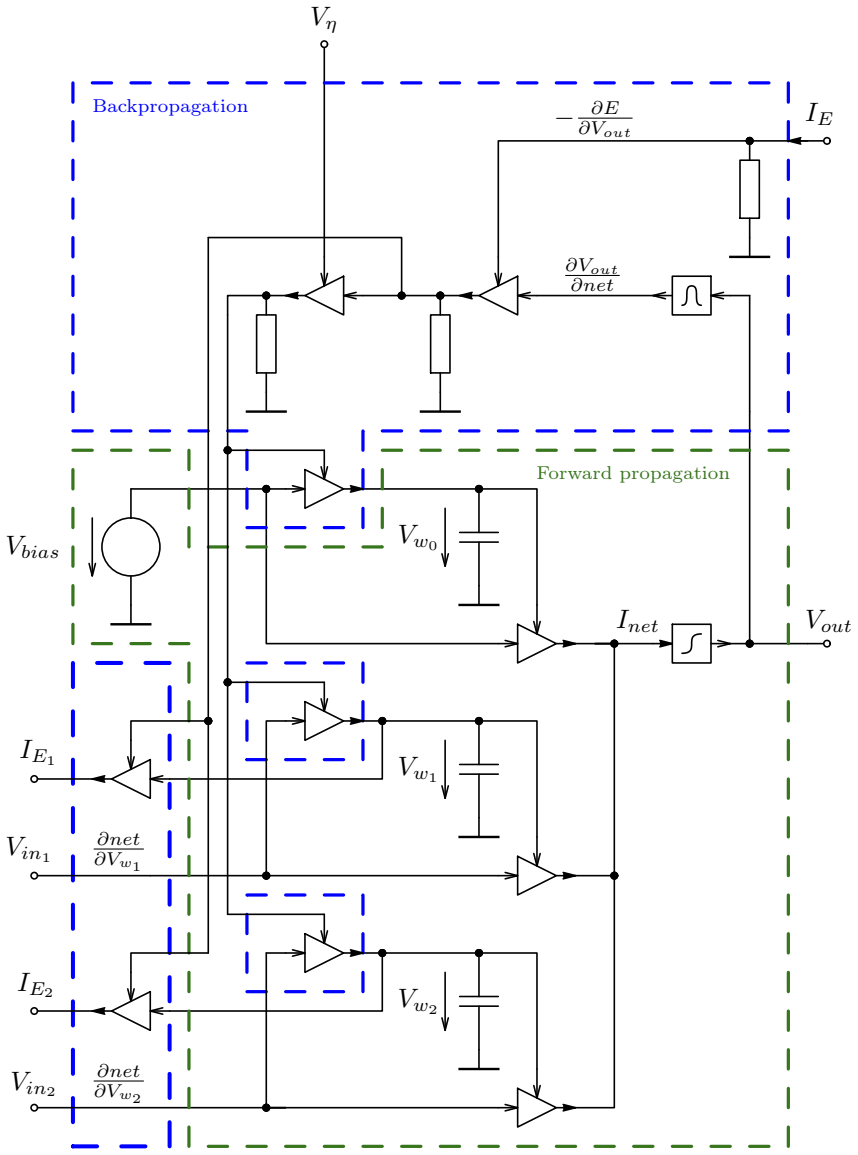

Figure 4.10: A fully analog neuron's block implementation including a learning circuit with two inputs.

These requirements were carefully considered in the design of the fully analog neuron with learning capability, as depicted in Figure 4.10. A green boundary delineates components required for forward propagation, while those required for backward propagation are highlighted with a blue boundary. The figure also illustrates the voltages corresponding to the equations mentioned earlier.

A new voltage, V_η, is introduced, which serves as an alternative to the learning rate found in digital networks. This voltage directly affects the coefficient K_η from Equation (4.11). Therefore, if $V_\eta = 0, V$, the network does not learn. Conversely, if $V_\eta > 0\,V$, the weights are adjusted based on the magnitude of the error and proportionate to the size of this voltage.

Additionally, the current outputs I_{E_N} are present, serving to propagate the error back to the preceding layers. This proposed circuit can be utilized as a neuron with any number of inputs. It necessitates adding more sections of the subcircuit with input voltage V_{in} and current output I_E, each associated with a single capacitor and three multipliers.

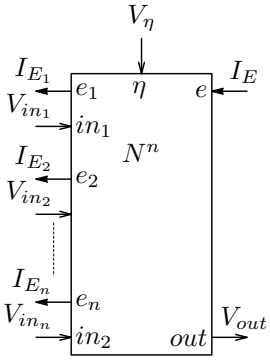

Figure 4.11: Symbol of an analog neuron with n inputs.

For further simplification, the entire circuit of the fully analog neuron will be represented by the symbol shown in Figure 4.11. This subcircuit is tagged by the letter N throughout this work, and the superscript defines the number of inputs.

Figure 4.12 demonstrates how these blocks are interconnected. It represents a feedforward fully analog neural network comprised of two inputs, four neurons in the first hidden layer, two in the subsequent hidden layer, and a single output. It incorporates two input signals represented as voltage sources V_{in_1} and V_{in_2}.

The voltage signal V_{target} is set to the values expected at the output. Additionally, the voltage V_η determines the learning rate, which remains consistent for all neurons within the network. The output currents I_E at the first hidden layer are not used, as the error does not propagate further. Subsequent chapters provide examples demonstrating the use of this network.

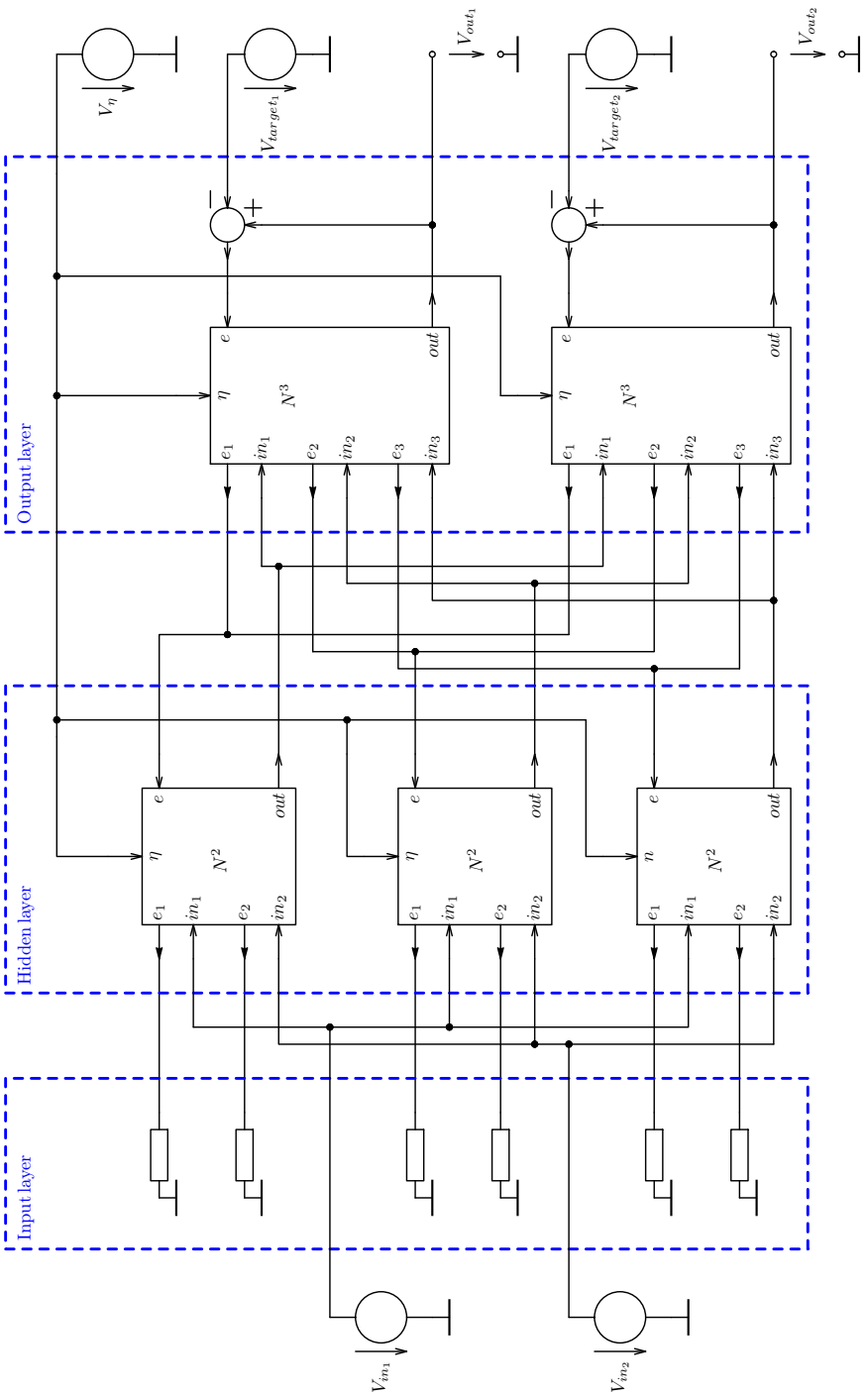

Figure 4.12: The block implementation of FAANN with two inputs, three neurons in the hidden layer, and two outputs.

Chapter 5
Verification of functionality

This chapter aims to fulfill this dissertation's second objective, which involves validating the proposed structure's functionality via an electrical behavioral model. The focus is on validating that the designed neural network concept is able to learn effectively.

Rather than focusing on validation directly on the hardware or transistor-level simulations, this process takes a higher-level behavioral approach. This strategy involves circuit simulation, where the neural network is defined using blocks defined by mathematical expressions. That simplifies the process and provides a broader view and better understanding of the overall behavior of the network [84].

Simulations focus mainly on ensuring the proper functioning of the network's learning process. As part of this, the influence of hyperparameters on the learning process is examined. A comparative analysis with traditional artificial neural network approaches is performed over the simulation results.

5.1 Circuit simulation equipment

The initial design of all circuits and their simulations for this dissertation were executed using the Graphic Editor of Electrical Circuits (GEEC) program [85]. This program uses ngspice as its core for simulation. GEEC is used extensively throughout the forward propagation chapter and partially in the backpropagation section. The outputs were not only the results of simulations that could be saved directly in the program but also all circuits and diagrams in this dissertation [86].

However, the complexity of circuits grew due to several factors. These included the need to almost completely redesign the neural network with every change of certain parameters and the need to read the weight and output values and reassign them back to the circuit for the next epoch. This continued with the challenging preparation of visualizing the results from each measurement. The preparation of voltage sources as input data for the neural network also

became complicated, as it needed to be done separately for each dataset and each epoch once it was discovered that shuffling elements improved the neural network's learning. With these complications, preparing circuits for simulation took significant time, and the process became prone to errors.

These and many other problems led to the creation of an independent FAANN simulation library to automate all processes as much as possible. This library is named "faann-simulator".

5.1.1 Structure of the faann-simulator

The faann-simulator program is written in TypeScript, with its computational part running on node.js [87]. TypeScript was chosen for its simplicity and clarity in defining the neural network's structure. Thanks to its statically-typed language and intelliSense, using the library becomes remarkably straightforward, even without an in-depth documentation study [88]. The ngspice is used as the core for simulation, launched by the spawn system method. Ngspice is an open-source spice simulator for electric and electronic circuits. It is a mixed-level/mixed-signal circuit simulator. Its code is based on three open-source software packages: Spice3f5, Cider1b1, and Xspice [89].

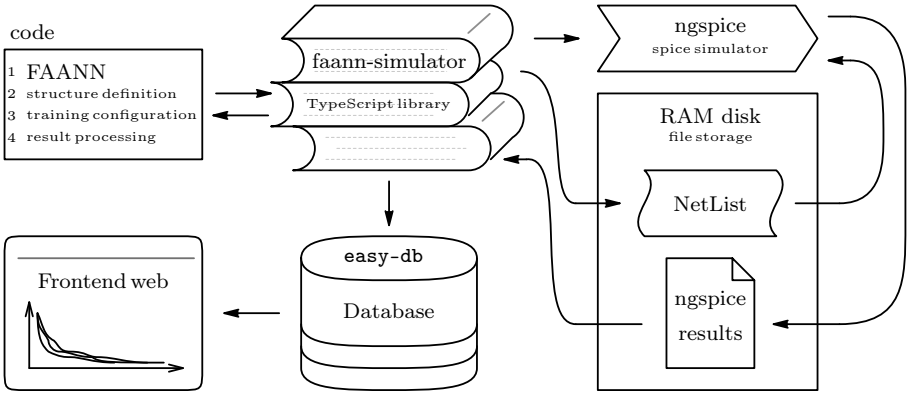

Figure 5.1: Structure of the faann-simulator library.

The ngspice program is fed with a file containing the NetList, which encompasses a complete circuit description, including the simulation and output parameters specification. Following this, ngspice creates a file with the simulation results. The faann-simulator library parses and subsequently processes this file. The processed data is then stored in the easy-db database, which suits this case. The primary reason is that the data is instantly available and can be further processed in JavaScript Object Notation (JSON) files. Additionally, the dynamism of the database allows the storage and modification of data in various formats without the need to rewrite the entire program, which is beneficial for development and testing. The database also offers the same

Application Programming Interface (API) for the node.js environment and the web browser environment, which is utilized to select and visualize the simulation results. This combination of technologies enables the use of the library on any operating system. This entire structure is illustrated in Figure 5.1.

5.1.1.1 Application programming interface

The library faann-simulator is designed to maximize user-friendliness and simplifies the process of testing various neural network configurations. Code 5.1 provides an example of basic usage of the library for training a FAANN.

The neural network structure and path to the spice program are defined during the initialization. This process necessitates two parameters since the library facilitates not just background simulations but also the display of simulation progress in the native ngspice graphical interface. For Windows systems, ngspice offers two executables, one for the Command Line Interface (CLI) and another for the Graphical User Interface (GUI). Thus, separate paths need to be defined for each. To display the simulation progress, only replace the `train` method with the `trainPlot` method, which accepts the same parameters.

```
1  import FAANN from "faann-simulator";
2
3  const faann = new FAANN({
4      spiceCLI: "ngspice",
5      spiceGUI: "ngspice",
6      structure: [2, 3, 1],
7  });
8
9  const result = await faann.train({
10     testName: "xor-example",
11     durationTime: 0.25,
12     learningRate: 0.5,
13     epochs: 200,
14     datasetName: "xor",
15     trainingSet: [
16         { input: [0, 0], output: [0] },
17         { input: [0, 1], output: [1] },
18         { input: [1, 0], output: [1] },
19         { input: [1, 1], output: [0] },
20     ],
21     // optional parameters
22     shuffleSet: true,
23     saveDetail: false,
24     saveWeightsAfterPredict: true,
25     showProgress: "full",
26     transitionTime: 0.0025,
27 });
```

Code 5.1: A simple example of how to use the faann-simulator library.

The `train` method is asynchronous and returns an object containing the simulation results. The parameters for this method are as follows:

`testName` Type: string. Test designation. This parameter is useful for subsequent result analysis.

`datasetName` Type: string. Dataset designation. This parameter is also helpful for subsequent result analysis.

`shuffleSet` Optional. Type: boolean. Default: false. Shuffle the dataset for each epoch. This feature is developed only for comparison purposes with traditional ANN. The network is primarily designed for time series, where maintaining data order is crucial.

`trainingSet` Type: Data. Training data.

`durationTime` Type: number [ps]. Time duration for one data sample.

`transitionTime` Optional. Type: number [ps]. Default: `durationTime` / 1000. The transition time from one data value to another that is part of the `durationTime`. If this parameter is set close to the `transitionTime` value, the waveform behaves less jumpy and closer to the natural behavior of the received signals from the sensors.

`learningRate` Type: number [V]. The rate of change of weight in response to error. The voltage supplied to V_η during the learning process.

`epochs` Type: number [-]. The number of training repetitions on the `trainingSet`.

`saveDetail` Optional. Type: boolean. Default: false. Save voltage progress during the training. These data are storage-intensive, so it is recommended to save them only when a close observation of training progress is required.

`saveWeightsAfterPredict` Optional. Type: boolean. Default: false. Save the voltage degradation on the capacitors after the measurement of statistics, which is performed for each epoch. If the value is true, the parallelization of calculations for reading and setting the weights is disabled in this measurement process.

`showProgress` Optional. Type: false | "full" | "line" | "line-predicts" | "line-errors" | "line-weight". Default: "line". The amount of information displayed during training in the console.

The structure definition of the network in the library is designed to be as transparent and configurable as possible. Therefore, its most straightforward and shortest definition is simply an array of numbers. Each number represents the count of neurons in a particular layer. The first number is the count of

neurons in the input layer, the following numbers in the array are counts of neurons in hidden layers, and the last is the count in the output layer. In this number-based definition, the default activation function, i.e., the sigmoid function, is set.

If a different activation function is needed, an object-based definition is used. It allows for the specification of a unique activation function for each layer. The count of neurons in a layer is again indicated by a number under the `count` key, while the activation function type is specified under the `activationFunction` key. This library implements three different activation functions: sigmoid, tanh, and identity.

Furthermore, if different activation functions are required within one layer, it is possible to use an array of objects where each object represents a single neuron and its activation function. Examples of all three methods of inputting the structure are provided in Code 5.2.

```
const structureShort = [2, 3, 1];

const structureMedium = [
    1,
    { count: 3, activationFunction: "tanh" },
    { count: 1, activationFunction: "identity" },
];

const structureLong = [
    1,
    [
        { activationFunction: "sigmoid" },
        { activationFunction: "sigmoid" },
        { activationFunction: "tanh" },
    ],
    { count: 1, activationFunction: "identity" },
];
```

Code 5.2: Example of neural network structure definition for faann-simulator.

For more advanced usage of this library, it is possible to read and set the weights of individual neurons. The same is true when reading the entire network structure. This functionality is beneficial not only for displaying ongoing values but also for saving and loading the network at specific phases of training. Code 5.3 demonstrates how to work with this functionality.

Information about the network structure and its weights are also stored directly in the database for each epoch separately, enabling their display in the web interface. Hence, during the training, it is possible to download the network weights and subsequently load them into another network instance. This instance can then continue the training from a certain point using a different approach.

```
const structure = faann.getStructure();
const weights = faann.getWeights();

const newFaann = new FAANN({
    ...spiceConfig,
    structure,
    weights,
});

const newWeights = newFaann.getWeights();
faann.setWeights(weights);
```

Code 5.3: Demonstration of manipulating a faann instance and its weights with the faann-simulator for possible saving and loading of the model.

5.1.1.2 Training process

The entire training process of the library unfolds in two phases. The first phase is the training itself. Initially, voltage sources of the inputs and target outputs are set according to the training data using the `pulse` parameter, similar to what is used in Code 5.4. These data are randomly shuffled or not, based on the `shuffleSet` parameter. Subsequently, a complete netlist is created containing the set sources and the neurons as subcircuits.

These neurons are connected according to the network structure defined during initialization. The neurons' weights are randomly generated or loaded from previously saved values. The netlist also includes detailed transient analysis information and a list of output voltages for subsequent analysis. Then, the simulation is executed in ngspice, and upon completion, the results are saved next to the input netlist. These files are deleted after parsing and processing.

Considering the high number of files handled during training, some of which can be large, a Random Access Memory (RAM) disk is used to speed up the entire process and save the hard disk lifetime. In this step, the trained weights of the network are obtained from the simulation results and directly set in the neural network instance.

The second phase is the evaluation of the trained network. In this stage, the voltage sources of the inputs are set to one specific input sample, and the voltage V_η is set to 0 V. After measuring all output voltages, the network error is calculated, and all measured parameters, including the weights, are saved in the database. This entire process is repeated for each epoch. Figure 5.2 depicts a simplified training process.

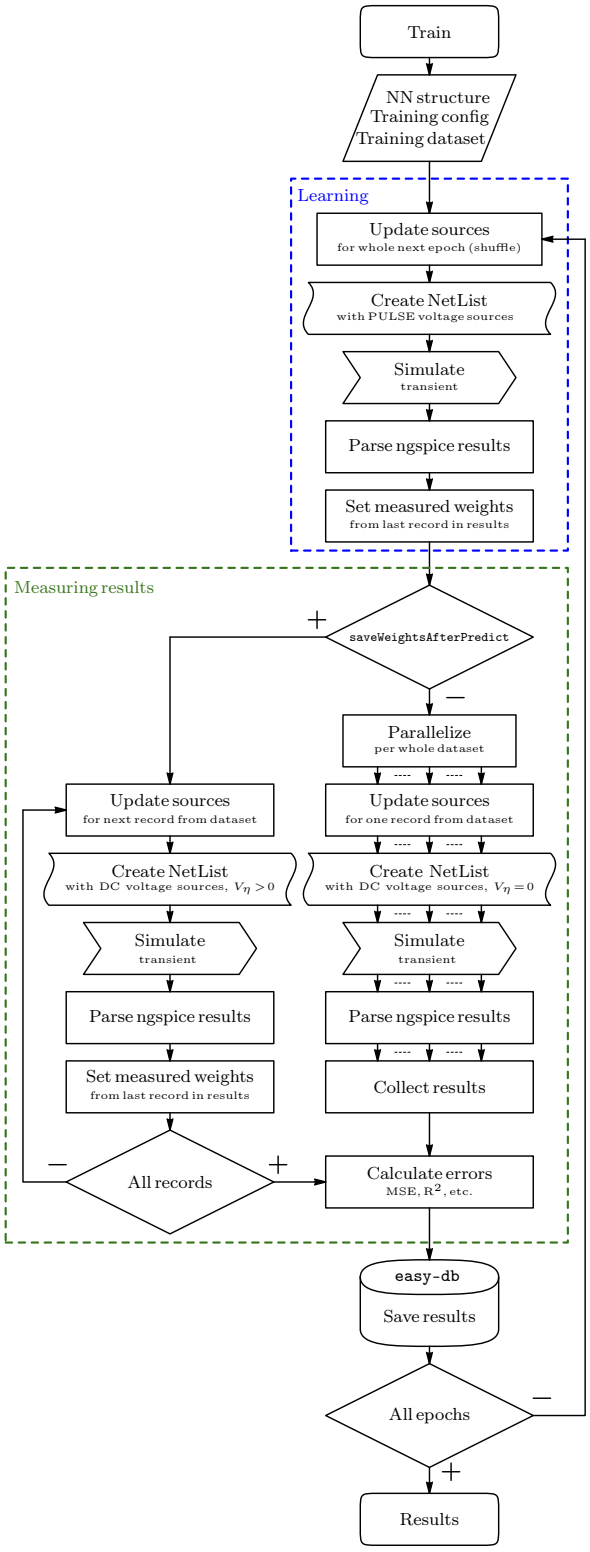

Figure 5.2: The learning process diagram of the faann-simulator library.

5.1.2 Outputs

The library offers several outputs. Firstly, there is the trained neural network itself and the result object, which contains the training and evaluation results from Code 5.1. These results can be immediately processed and used for subsequent processes in the program. Another output is the database, which can be accessed from multiple programs to display training results, aggregate them, and perform evaluations.

The library also includes a graphical interface for displaying and evaluating results. This interface provides real-time monitoring of the currently trained network, its structure, error progression, and other parameters. Additionally, the interface allows users to view aggregated results from all networks based on parameters such as `testName`, and make comparisons. These results can be exported as data that can be interpreted in other programs, like LaTeX. An example of this can be seen in Figure 5.3.

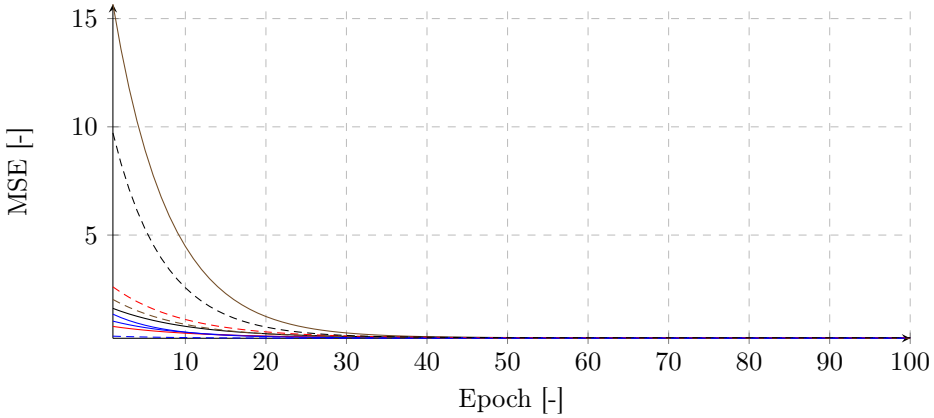

Figure 5.3: The learning curve of the FAANN model exported from the faann-simulator.

5.2 Forward propagation

This section presents an exploration of the forward propagation capabilities of the FAANN proposal. The focus is to ascertain that FAANN's forward propagation is working satisfactorily.

A simulation is carried out, drawing parallels between the proposed FAANN and the traditional ANN. The configuration of both networks comprised two inputs, a single hidden layer with two neurons, and one output, as seen in Figure 5.4. All activation functions are defined as a sigmoid.

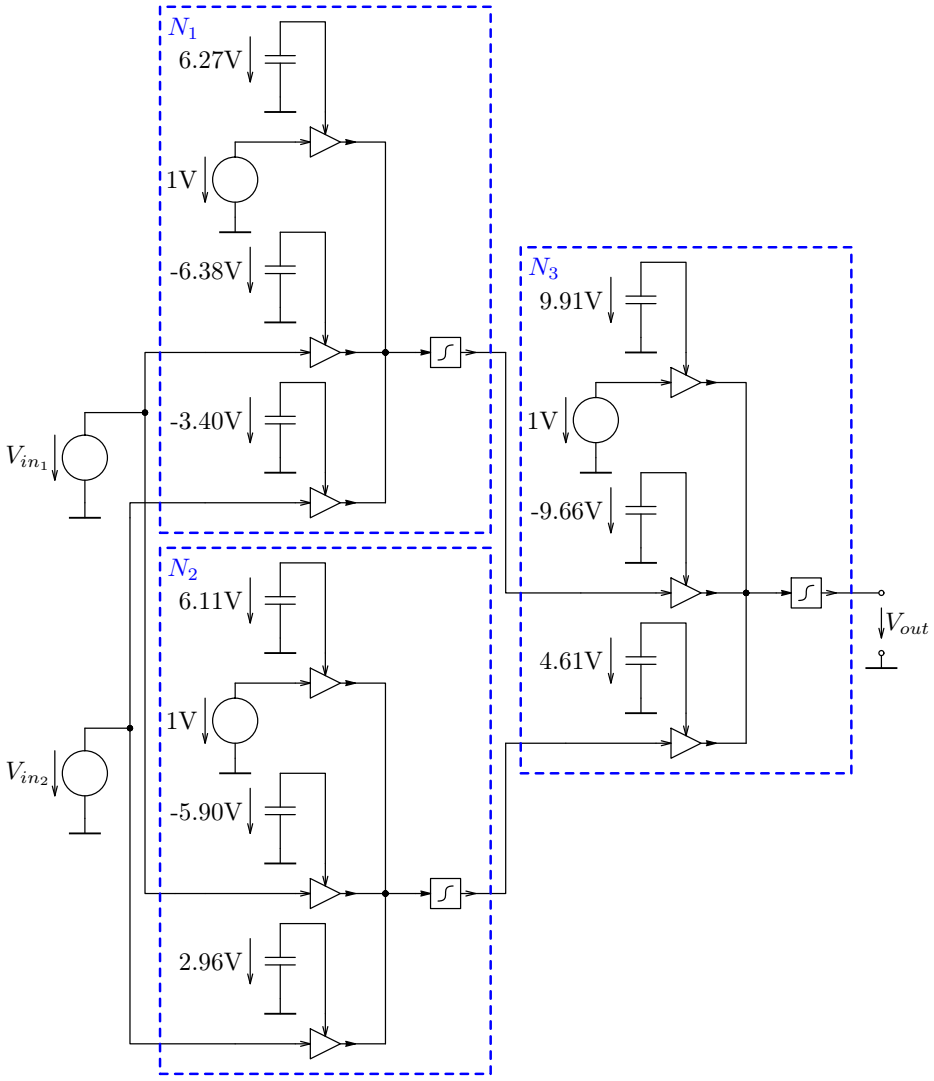

Figure 5.4: Schematic of the forward propagation of FAANN in GEEC to demonstrate the solution of the XOR problem.

Input 1	Input 2	Expected output	ANN output	FAANN output [V]
0	0	0	0.014	0.01400
0	1	1	0.984	0.98426
1	0	1	0.987	0.98699
1	1	0	0.012	0.27115

Table 5.1: Comparison of ANN and FAANN outputs for XOR simulation with the same set weights.

The conventional ANN was trained to utilize the data presented in Table 5.1. The table also documents the output from the ANN. For better comparability of results, this network was not fully trained.

	N_1	N_2	N_3
Bias weight	6.27	6.11	9.91
First input weight	-6.38	-5.90	-9.66
Second input weight	-3.40	2.96	4.61

Table 5.2: Neural network weights used to solve the XOR problem.

Table 5.2 captures the weights of the trained neural networks. In the case of FAANN, the weights are set to the same values as voltages of capacitors V_w, as seen again in Figure 5.4. Subsequently, an Operating Point (OP) analysis was executed for each combination of input voltages. The outputs from this analysis are also recorded in Table 5.1, under the FAANN output column.

The ANN and FAANN did not yield identical results despite identical network structures. This discrepancy can be attributed to the inherent differences in the element structure. However, these differences are part of the learning process. It is therefore expected that if the learning process is carried out directly at FAANN, its output will be correct.

A parametric Direct Current (DC) analysis was performed to further investigate the differences in FAANN network behavior. Input two is set to 1 V, and input one is set to values from 0 V to 1 V. The result is shown in Figure 5.5.

Despite the observed differences, this study validates the functionality of the FAANN model and its suitability for further exploration.

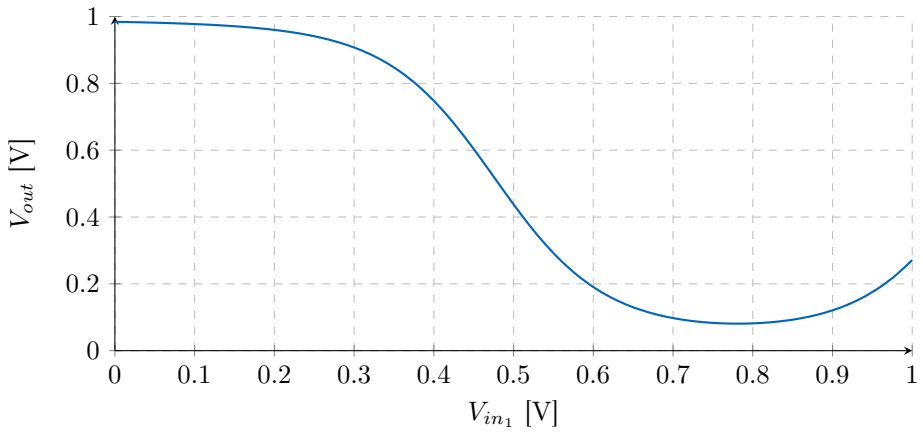

Figure 5.5: DC analysis of FAANN for XOR simulation with $V_{in_2} = 1V$.

5.3 Learning process

This chapter explores the learning process of the FAANN model. The functionality of this model is validated through simulations, providing insights into the fundamental mechanisms that drive the model's learning capabilities. The initiation of this exploration with the simplest learning scenarios facilitates a thorough understanding of the core principles underlying the model's learning process.

To validate the FAANN's learning process, the well-known XOR problem is used as a benchmark. This problem, renowned for its non-linear separability characteristic, is an excellent metric for evaluating the model's problem-solving aptitude and capacity to learn and adapt [53, 54, 90]. Subsequently, the impact of individual parameters on the learning process is examined. The investigation aims to elucidate how parameter alterations can influence learning outcomes.

In the final stages, an investigation is conducted into the potential effect of specific electrical parasitic properties on the learning process. This investigation aims to determine whether these properties can disrupt or invalidate the learning process.

5.3.1 Weights update analysis

The initial simulation is conducted to ensure the proper convergence of weights and the output voltage. This particular simulation involved two neurons in the hidden layer and a single neuron in the output for the purpose of simplifying the analysis. Subsequently, two output neuron excursions are monitored to investigate correct functionality. This circuit is created, and the transient numeric analysis is executed using the GEEC platform.

Figure 5.6 presents the outcome of this preliminary simulation. The evolution of the weights appears to align with expectations, evidencing a pattern that suggests a correct and functional update mechanism. Observations indicate that both weights adapt dynamically, aiming to match the output voltage with the predetermined target voltage, V_{target}. This adaptive behavior is a crucial factor for the successful operation of our concept.

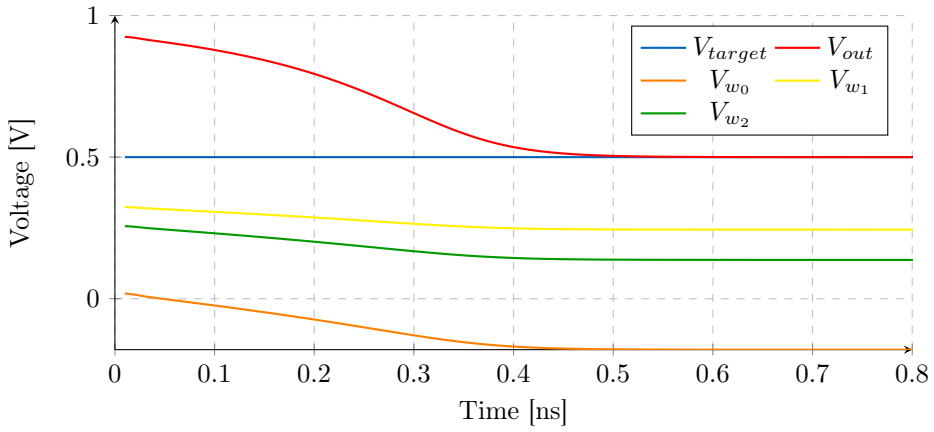

Figure 5.6: Weights monitoring in one step of FAANN learning.

5.3.2 Learning from dataset

The primary design goal of the FAANN is to function with analog sensors, not necessarily to learn from datasets. Still, comprehending its operation and the advantages it provides is crucial when comparing this model with traditional neural networks.

V_{in_1} [V]	V_{in_2} [V]	V_{target} [V]
0.2	0.6	0.9
0.8	0.4	0.1

Table 5.3: Dataset for simulation of FAANN with two inputs.

This section's second simulation exhibits the complete learning process of the FAANN, illustrated through transient analysis. The learning dataset, consisting of two rows, is represented as voltage values and integrated as voltage sources. This dataset, given in Table 5.3, employs the `pulse` notation for inputs and target values.

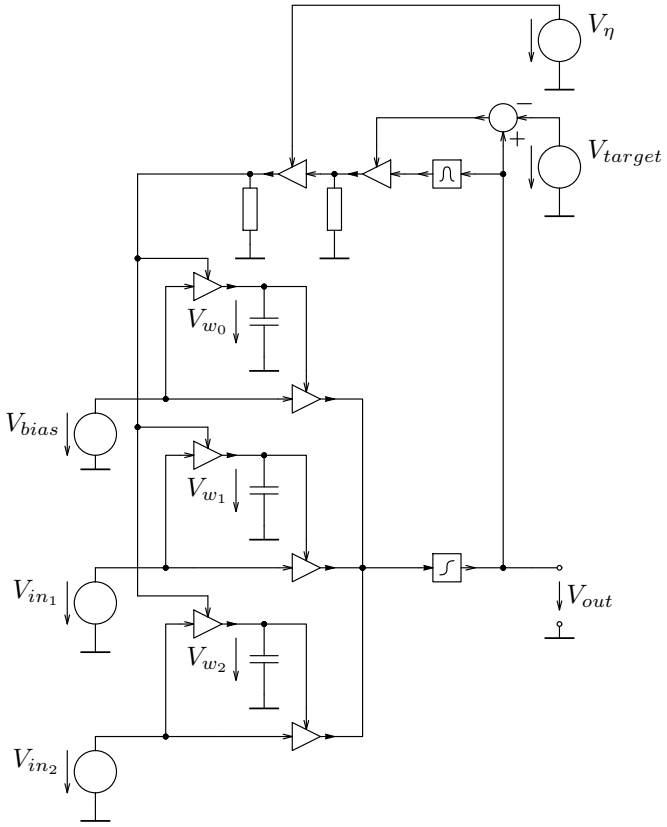

Figure 5.7: Schematic circuit of a single analog neuron with two inputs for learning a dataset using simulation in GEEC.

The network structure for this simulation involves a single neuron with two inputs, the schematic of which is provided in Figure 5.7. Each row in the dataset changes every 200 ps, and the variable V_η toggles between 0 V and 0.2 V during the learning process at intervals of 400 ps. This arrangement allows for a detailed examination of each teaching stage and the corresponding learned values.

```
1 Vbias in0 0 dc 1
2 Vin1 in1 0 dc 0 pulse 0.2 0.8 0 2p 2p 200p 400p
3 Vin2 in2 0 dc 0 pulse 0.6 0.4 0 2p 2p 200p 400p
4 Veta eta 0 dc 0 pulse 0 0.2 0 2p 2p 400p 800p
5 Vtarget target 0 dc 0 pulse 0.9 0.1 0 2p 2p 200p 400p
```

Code 5.4: Voltage sources definition for circuit in Figure 5.7.

The results are displayed in Figure 5.8. One of the observations from this simulation is the FAANN's fast learning ability in this particular setup. Notably, by completing 20 epochs, the output voltage V_{out} closely aligns with the target voltage V_{target}.

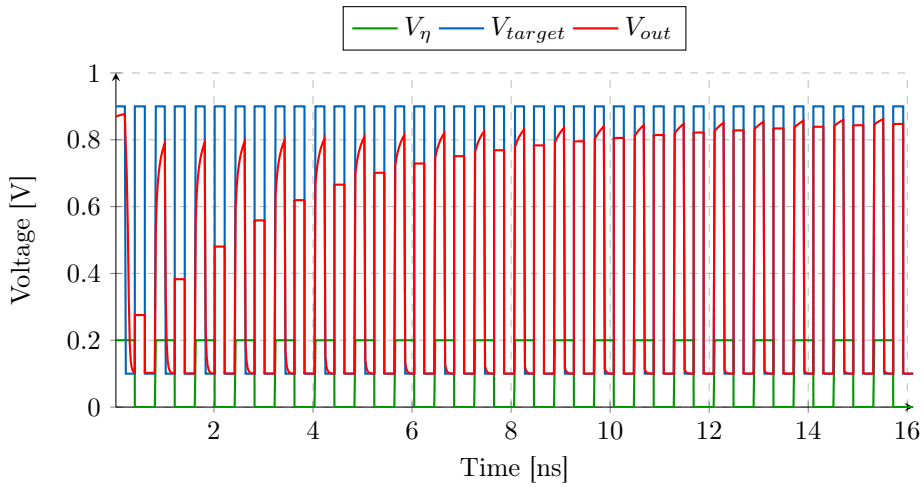

Figure 5.8: Transient simulation of the FAANN learning process with a dataset containing two records.

5.3.3 Learning multilayer structure

This simulation aims to validate the learning process between layers in FAANN. The network utilizes an analog implementation featuring a single hidden layer with two neurons and one neuron in an output layer. The dataset used for this purpose is identical to the one previously used, as shown in Table 5.3. All other experimental setup remains the same as that in the preceding simulation.

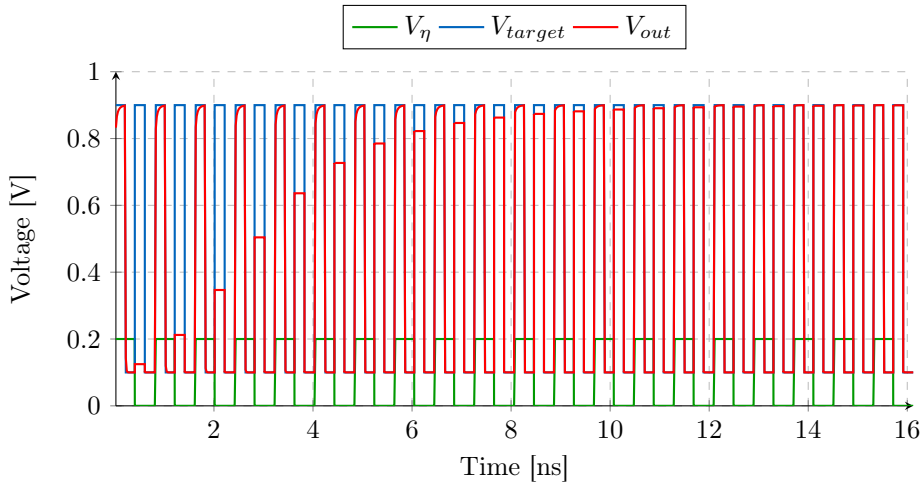

Figure 5.9: Transient simulation of the learning process of a FAANN structure with a hidden layer.

The simulation results are presented in Figure 5.9. The key observation is that the multilayer configuration learns faster than the single-layer structure. The neural network demonstrates a learned state as early as epoch 12. This outcome aligns with expectations, as the increased capacity for information representation contributes to the accelerated learning process.

5.3.4 XOR problem verification

The XOR problem shown in Table 5.4 is often used as a test case for neural networks because it's a simple problem that requires nonlinear decision boundaries, which in turn requires a certain level of complexity from the model. It cannot be solved using a single linear classifier or perceptron because the classes (1's and 0's) are not linearly separable.

Furthermore, the XOR problem was historically significant in developing neural networks. In the late 1960s, the perceptron model, the simplest form of a neural network, was shown to be unable to solve the XOR problem. It led to a significant decrease in interest and funding for neural network research. It wasn't until the development of the multi-layer perceptron and backpropagation in the 1980s that it was shown that neural networks could solve the XOR problem, leading to a resurgence in neural network research.

Thus, the XOR problem is an excellent baseline test case for neural networks for a lot of research and has particular historical significance. It is used in both older studies of learning algorithms [54] and more modern studies that look at the use of memristors [53] or SNNs [90].

In this particular study, the XOR problem is used to validate the capabilities of the FAANN.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 5.4: Training dataset of the XOR problem.

A configuration for the FAANN model comprised of two input neurons, eight neurons in the hidden layer, and one output neuron for this subsection. This architecture was chosen to be sufficiently complex to exhibit learning deficiencies and effectively solve the XOR problem.

A significant part of the simulation setup is the initialization of the network weights. To account for the randomness in weight initialization and its potential impact on network performance, ten different simulation are run, each with a

unique set of initial random weights.

The key performance metric for these analyses and for FAANN is the MSE, which is measured and calculated at the end of each training epoch. The MSE provides a quantitative measure of the network's performance, indicating the difference between the network's predictions and the actual values. Lower MSE values correspond to better network performance and model accuracy.

The simulation results are depicted in Figures 5.10, 5.11, and 5.12. Each figure represents the performance of the FAANN for a different learning rate η, with each colored line within a figure representing a different set of initial network weights.

Learning rate η plays a significant role in determining the network's learning efficiency. It can be manipulated by altering the value of V_η or adjusting the time allocated for training on a single dataset row.

Below are the results of these simulations and analyses, providing an understanding of the FAANN's performance in handling the XOR problem under various learning rate settings.

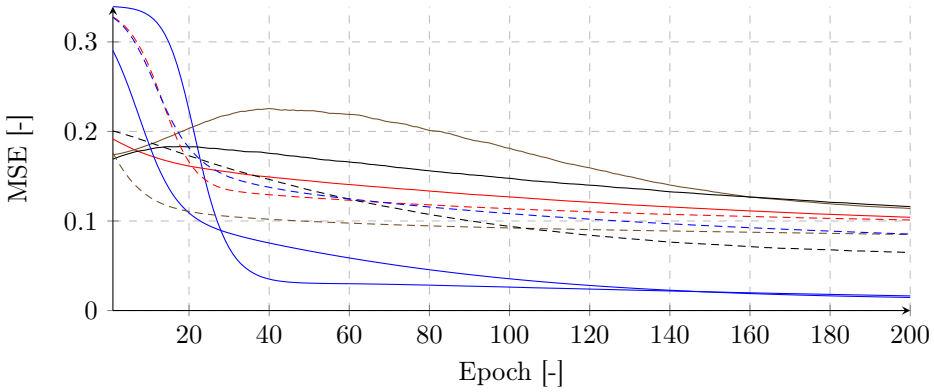

Figure 5.10: The learning curve of the FAANN model with a small analog learning rate (0.1 V) for the XOR problem.

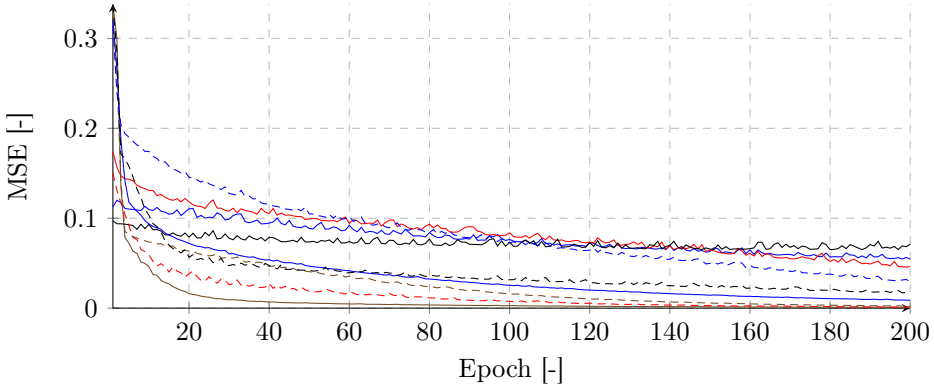

Figure 5.11: The learning curve of the FAANN model with an approximately optimal analog learning rate (0.5 V) for the XOR problem.

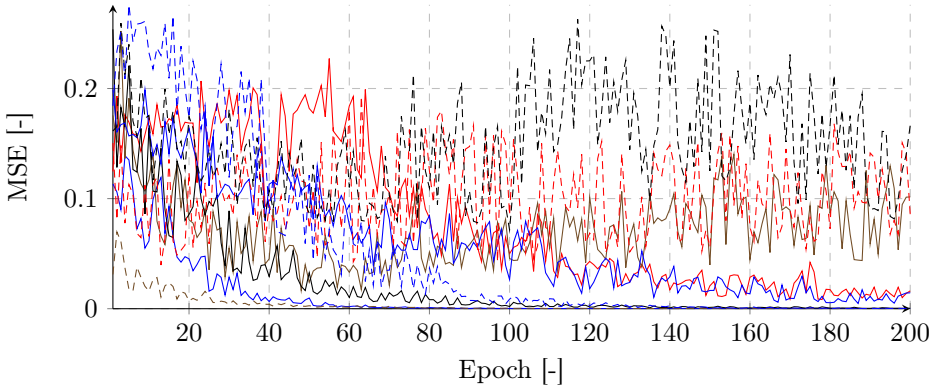

Figure 5.12: The learning curve of the FAANN model with a large analog learning rate (0.9 V) for the XOR problem.

5.3.5 Dependence on parasitic properties

The premise of this study assumes that the neural network can adapt and learn despite the inherent parasitic properties. Nevertheless, the tolerance of this adaptation has its limits, and this subsection aims to explore these constraints.

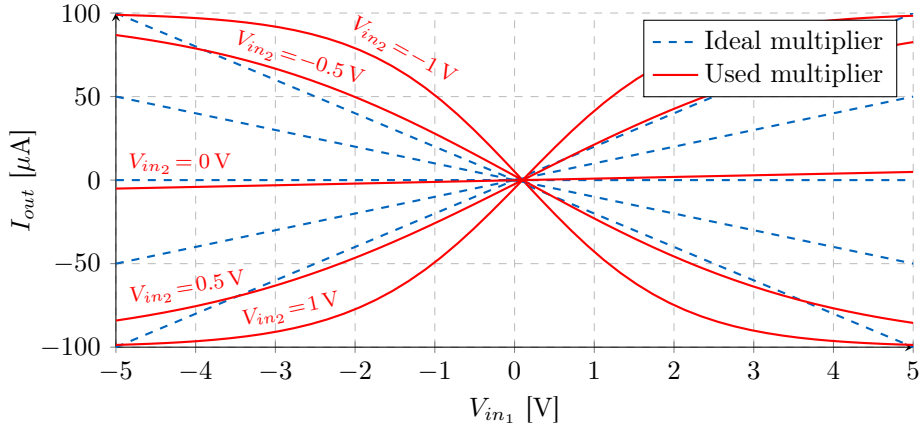

Figure 5.13: Demonstration of the inaccuracies of the multiplier block used with parasitic properties included.

The circuit element most impacted by parasitic properties is the multiplier block. Its role in the network makes it sensitive to inaccuracies, which can significantly affect overall network performance. These inaccuracies are modeled as

$$I_{out} = K_I \cdot tanh(K_m \cdot V_{in_1} \cdot (V_{in_2} + V_{off})), \tag{5.1}$$

where K_I is a constant that determines the maximum current flowing through the circuit at the maximum allowable input voltages. A hyperbolic tangent function distorts the multiplication itself, and a voltage offset is added at the second input. The effect of these inaccuracies is reflected in the multiplier block's output current I_{out}. Used configuration inaccuracies of the multiplier block are shown in Figure 5.13.

The simulation results showcasing the impact of these inaccuracies is illustrated in Figure 5.14. For comparison, Figure 5.11 shows the results of simulations with the same configuration only without these parasitic properties of the multiplier. Although these properties impact network performance, it is critical to note that it does not fundamentally impede the network's functionality.

While the current analysis focused on the multiplier block, similar simulations can examine the network's resilience to noise, the effect of Complementary Metal-Oxide-Semiconductor (CMOS) capacitance non-linearity, and other circuit-level properties. However, these simulations can only partly represent the actual behavior at the block design level, as the real impact largely depends on the

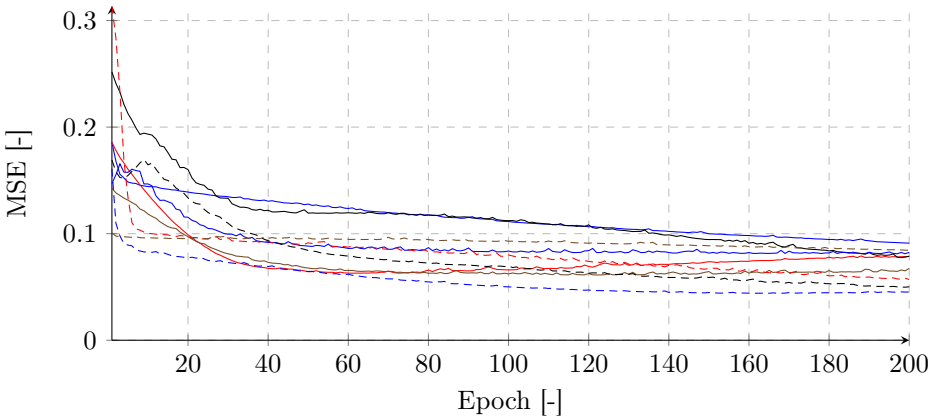

Figure 5.14: The learning curve of the FAANN model with a modified multiplier block incorporating parasitic properties and with an approximately optimal analog learning rate (0.5 V) for the XOR problem.

specific implementation used.

In conclusion, while parasitic properties introduce certain challenges and inaccuracies, the designed neural network exhibits a commendable degree of tolerance, maintaining its core functionality.

5.4 Comparison with classical ANN

The development of FAANNs is driven by a different set of challenges than those addressed by traditional ANNs. Specifically, FAANNs are designed to address the real-time processing of analog signals, a task inherently different from the data processing intended for classical ANNs. It is essential to understand that these two types of networks are used for disparate problem domains; thus, direct performance comparisons are generally inappropriate and potentially misleading.

However, for the purpose of clarifying the functional mechanisms and characteristics of FAANNs, this research adopts a comparative approach. This analysis is not aimed at evaluating which model is better but rather at using established knowledge about ANNs to explain FAANN behavior. If similarities in learning processes are found, FAANNs may exhibit behavioral patterns that mirror those of classical ANNs.

Moreover, this comparative view provides an opportunity to indirectly assess the FAANN concept's learning speed. This comparison makes it possible to understand whether the FAANN system is able to learn in real-time, which is its essential feature.

5.4.1 Learning properties

In this subsection, the focus is on comparing the learning properties of FAANNs and classical ANNs. As demonstrated in Sections 5.3.2 and 5.3.3, FAANNs exhibit faster learning rates as the network structure becomes more complex, a property shared with classical ANNs.

Further investigation in Section 5.3.4 revealed how FAANNs respond to changes in the learning rate. This reaction mirrors the behavior of classical ANNs, furthering the similarities between the two types of networks. Upon further adjustments to the learning rate, it was observed that FAANNs have the potential to settle into local minima. This behavior, illustrated in Figure 5.15, is typically not considered advantageous. However, it is another characteristic that FAANNs share with classical ANNs.

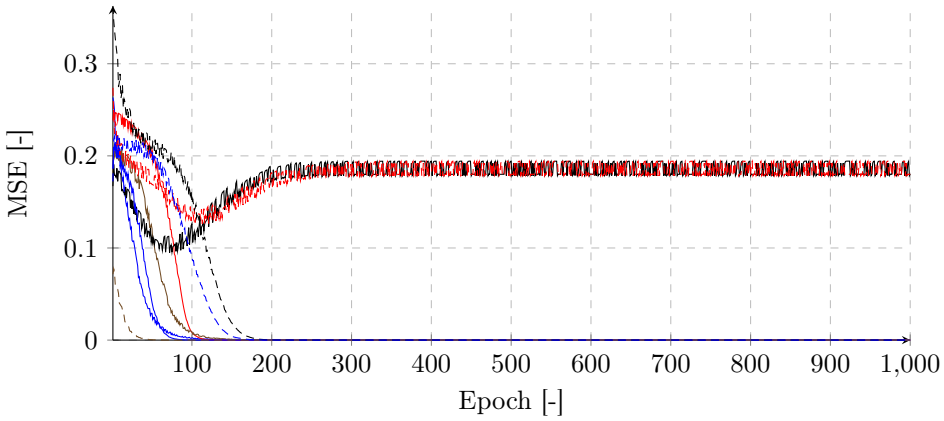

Figure 5.15: The learning curve of the FAANN model with a local minimum for the XOR problem.

The simulations show that FAANNs have properties similar to those of classical ANNs. It suggests that data scientists can deploy this neural network similarly to how they would use traditional ANNs. In essence, the operational familiarity and comparable learning properties of FAANNs could lower the barrier to their adoption in the data science community.

5.4.2 Learning speed

A primary advantage of using a FAANN instead of a classical neural network is its faster training speed. The simulations in this subsection demonstrate a comparative analysis between the proposed FAANN and a traditional neural network constructed using TensorFlow, an end-to-end open-source platform for machine learning. All computations for ANN are executed on various hardware, namely CPU, GPU, and TPU, all of which are run via Google Colab [36, 37]. Training and comparisons of the neural networks are made on identical datasets, with configurations that most closely mimic the learning flow of FAANN as described in this dissertation.

The constructed circuit contains blocks defined on a formula level that expected the result in hardware implementation to be slower. The most time-intensive process in this concept is the charging of capacitors. Therefore, these capacitors' values are set close to the possible final hardware implementation. That means all capacitors feature a capacitance of 0.1 pF, with a maximum charged current of 100 μA per capacitor. These values would be adjusted in response to the specific implementation to ensure that parasitic properties do not disrupt the charging process. Notably, the four-quadrant multiplier block in this design is the most sensitive to parasitic properties, which can affect speed. However, it can reach up to 40 GHz in some implementations [91].

Four neural network structures are developed and presented in Table 5.5. Each structure is realized in both analog and classical forms. The outcomes are displayed in Table 5.6.

In FAANNs, the training time depends only on the size of the dataset and the training time of one row of the dataset. In contrast, in a classical neural network, the training duration is more due to the network structure, not exclusively by the dataset size. Implementing online learning (also known as incremental learning or sequential learning) provides only a single row from the dataset learned at each stage [74]. This factor contributes to the slower computation speed on the GPU, optimized for parallel computing, compared to the CPU [36]. As a point of interest, the simulation training process duration of FAANN on a single i7 processor is for test 1 approximately 8 seconds, while for test 4, approximately 2 hours.

The results show that potential real-time speed is around several orders of magnitude faster as compared to the known implementations.

	Test 1	**Test 2**	**Test 3**	**Test 4**
Size of dataset	2	10	100	500
Number of hidden layers	1	2	3	4
Number of neurons	3	5	16	21
Number of epochs	1000	1000	1000	1000

Table 5.5: Size of the dataset and structure of neural networks for speed comparison.

	CPU	GPU	TPU	**FAANN**
Test 1	$2.017 \pm 0.020\,\mathrm{s}$	$4.872 \pm 0.264\,\mathrm{s}$	$2.017 \pm 0.035\,\mathrm{s}$	**2 ns**
Test 2	$6.373 \pm 0.097\,\mathrm{s}$	$13.599 \pm 0.314\,\mathrm{s}$	$6.284 \pm 0.190\,\mathrm{s}$	**10 ns**
Test 3	$54.678 \pm 1.215\,\mathrm{s}$	$120.152 \pm 1.034\,\mathrm{s}$	$53.910 \pm 0.958\,\mathrm{s}$	**100 ns**
Test 4	$274.644 \pm 2.281\,\mathrm{s}$	$624.357 \pm 6.211\,\mathrm{s}$	$272.727 \pm 2.159\,\mathrm{s}$	**500 ns**

Table 5.6: Comparison of time spent on training neural networks in 2021.

Chapter 6
Adaptive frequency filter

In this chapter, one of the many applications of FAANN that use its potential is presented and explored. Specifically, the focus is on the adaptive frequency filter [92, 93].

Filters are an essential building block for signal processing in almost every modern electronic system. When used in environments where conditions change over time or are simply poorly controlled, there is a need to change filter parameters to ensure proper system functionality. It is not only in these situations that adaptive filters are an attractive option [42, 94, 95].

Adaptive filters are used in a large number of applications. For example, for biological signals such as Photoplethysmography (PPG), Electromyography (EMG), and Electroencephalography (EEG), especially in real-time adaption [42, 96, 97, 98]. Furthermore, they are also used for lower frequencies in acoustics, including echo cancellation, noise control, array processing, and acoustic communication filtering [99]. For higher frequencies, they are used, for example, in antenna arrays, to improve the reception quality of multiple signals in radio engineering systems [100].

Nowadays, most adaptive filters have been implemented using digital circuits [96, 98, 101, 102, 103]. These are often preferred due to their simplicity and efficiency; however, they are primarily suited for lower-frequency applications. When it comes to adapting both digital and analog filter types, algorithms such as Least Mean Squares (LMS) or Heuristic are predominantly used [94]. These adaptive algorithms inherently operate in digital form, resulting in computational power limitations for digital processing at higher frequencies.

Specific implementations have been developed to address these challenges using FPGA technology [102, 103]. Switched filters, such as switched capacitors or filter-based adaptive fuzzy finite-time control for switched nonlinear systems, are examples of technologies that filter analog signals. Despite their analog nature, the adaptation mechanism in these filters is often executed by algorithms, or their switching speed ultimately imposes limitations for high-frequency applications [104].

All the adaptive filters mentioned above face limitations stemming from factors such as sampling, ADCs, DACs, and the clock speed of computing machines. Furthermore, filters capable of operating at high frequencies do not offer real-time adaptation, further limiting their applicability in dynamic situations. These issues can be particularly problematic in systems with restricted computational resources for digital signal processing, such as satellite systems, making them less adaptable to dynamic operational requirements [102]. In contrast, planar analog implementations of filters demonstrate greater consistency at higher speeds and lower power consumption, presenting a compelling alternative [42, 94, 95, 105].

Due to the limitations of existing adaptive filters, this chapter introduces an innovative concept of a fully analog adaptive filter based on a FAANN [52]. The innovative aspect of this approach lies in its analog nature that does not use any clock control, which enables both high-frequency operation and real-time adaptation capability, thus overcoming the limitations of traditional adaptive filter technologies [92]. In addition, the adaptive mechanism of this new concept allows for the adaptation of a wide range of filter types, including high-order filters, offering unprecedented versatility in the field of analog adaptive filtering [93].

6.1 Proposed structure

In discrete-time filters, the fundamental element is the delay [94]. Neural networks apply the same element to process sequences or signals, such as RNN or LSTM. However, an element acting as a signal delay suitable for these purposes does not exist in analog circuits. Several solutions address this issue, such as switching capacitors, but these are not fast enough for high-frequencies [104].

In the scope of this research, several attempts were made to construct analog versions of recurrent neural networks for higher frequencies. Neurons in RNN are characterized by having an additional input that accepts the previous output. For instance, an integrator was used for the delay, or the signal was delayed using a low-pass filter or a simple phase shift. The attempts included delaying feedback on individual neurons as well as on entire networks. The outcomes of these trials were unsatisfactory, as most of them either strongly diverged or failed to learn within the required time range. The development of an analog version of RNN and LSTM continues.

However, several alternative solutions were proposed for adaptive frequency filters. This dissertation presents one of them based on a filter bank [98, 105]. In this concept, an input signal is fed into the filter bank. The outputs from each filter bank are then supplied to the existing FAANN structure as inputs. Figure 6.1 shows an example of such a circuit.

This structure is trained in the same way as the FAANN, using the reference signal V_{target}. This means that to train the filter it is unnecessary to apply the frequency characteristics or the mathematical description. Having a reference signal in the time domain is sufficient, and the frequency domain is learned from this signal. That is one of the reasons why this concept expands the possibilities and ease of use and makes it a solution for a wide range of applications.

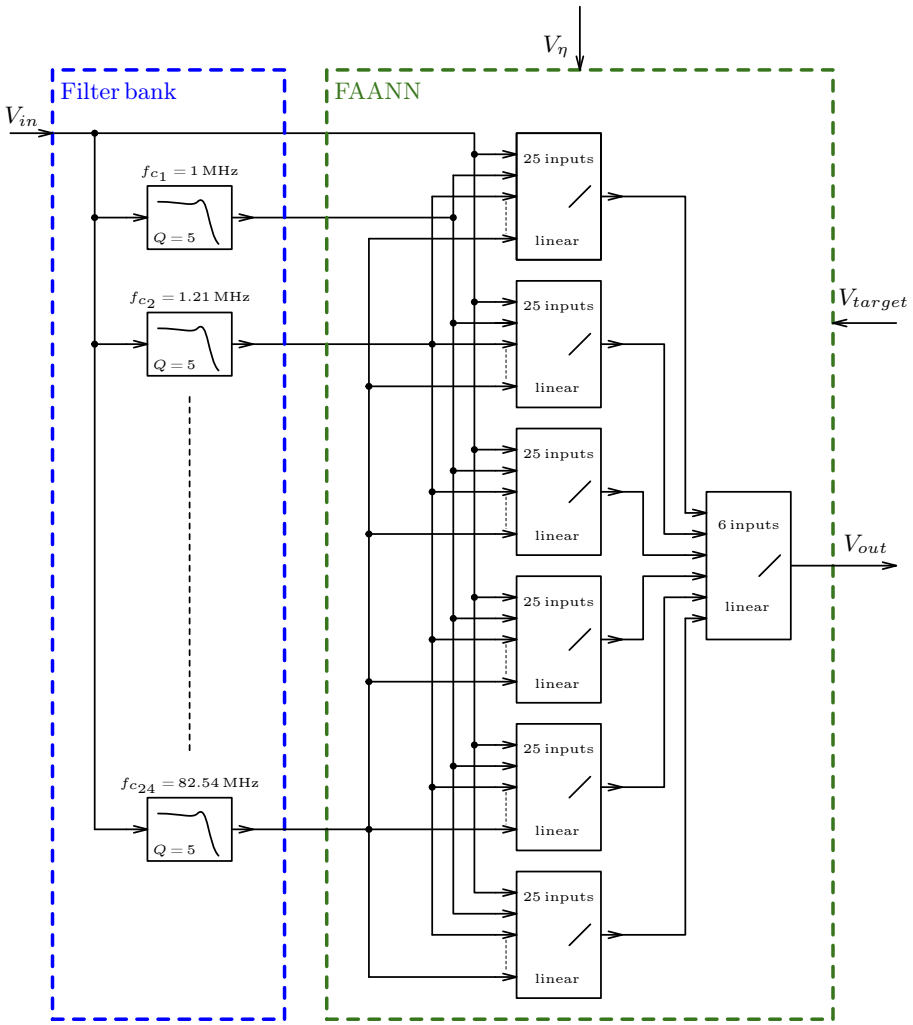

Figure 6.1: The structure of a proposed simple generic, fully analog adaptive filter for frequencies from 1 MHz to 100 MHz.

This structure can have multiple input and output signals depending on the application. An example would be an antenna array where each input is connected to a signal from one antenna, which can be routed directly or through its filter bank, and the output is a signal processed by all antennas. Alternatively, the output could consist of two separate signals.

This concept of adaptive frequency filter structure introduces an entirely new set of hyperparameters that need to be configured. It starts with the already-known neural network structure used here as an adaptive mechanism. It continues with the selection of the correct filter type, the number of filters, their order, and frequency. These aspects are explored in the following sections.

6.2 faann-simulator for adaptive frequency filters

Section 5.1.1 elaborates on the structure and application of the faann-simulator library, primarily for processing digital data. It provides the capacity to compare it with traditional ANN. However, the primary objective of this library is not this comparison but the simulation of FAANN structures, particularly for analog signals. In the context of this research and future development, it is anticipated that not only the network structures but also the signals, both input and output, are subject to change. The goal, therefore, is to develop a library capable of simulating all these changes while maintaining a user-friendly interface. Moreover, it is designed to be simple and conducive to further research. Therefore, this section details how this library can be applied to simulate analog signals, especially for adaptive frequency filters.

The overarching goal of this chapter is to demonstrate that this adaptive frequency filter concept can adjust to filters of higher frequencies and orders. It should be noted that the faann-simulator employs ngspice, which can handle a variety of analyses. However, given that the FAANN structure is highly nonlinear, using Alternating Current (AC) analysis is not feasible, which would have been beneficial for filters. Due to this limitation, all simulations conducted are of the transient type. This section presents a comprehensive overview of the process and the outcomes of these transient-type simulations.

6.2.1 Input filters

The faann-simulator library offers the ability to define filter banks for the input signal, as depicted in Figure 5.1, within the input layer definition. Here, a familiar coding approach is utilized, as shown in Code 6.2, where the third style for structure definition using an array of objects is used when creating an instance of the neural network.

For each input, it is possible to define a separate filter bank using an object with the type `input` and the property `outputs`, representing an array containing individual filters' definitions. An example of such a structure is presented in Code 6.1, which includes only one input signal directly connected to the network and utilizes five additional RC low-pass filters. This network also consists of a hidden layer with six neurons and an output layer with a single neuron.

```
const faann = new FAANN({ ...config, structure: [
   [{ type: "input", outputs: [
       { filter:"no" },
       { filter:"lowPass", capacity: 10, resistance: 15 },
       { filter:"lowPass", capacity: 10, resistance: 159 },
       { filter:"lowPass", capacity: 10, resistance: 1592 },
       { filter:"lowPass", capacity: 10, resistance: 15920 },
       { filter:"lowPass", capacity: 10, resistance: 159200 },
   ] }],
   { count: 6, activationFunction: "identity" },
   { count: 1, activationFunction: "identity" },
]});
```

Code 6.1: An example of filter definition in the FAANN input layer in the faann-simulator library.

Code 6.2 illustrates different ways of defining filters, with each line representing a different approach. The first line states that the input signal is directly connected to the network without filtering. The second line defines an RC low-pass filter with a capacitance of 10 pF, and a resistance of 1592 Ω for the input signal. The third line defines an RC high-pass filter with the same parameters as in the previous case. The fourth line introduces an RLC low-pass filter, where the inductance is 22 μH.

```
{ filter: "no" }
{ filter: "lowPass", capacity: 10, resistance: 1592 }
{ filter: "highPass", capacity: 10, resistance: 1592 }
{ filter: "lowPassRLC", resistance: 3000, inductance:
    22000000, capacity: 10 }
{ filter: "laplace", s_xfer: "num_coeff=[3947841760435743.5]
    den_coeff=[1 12566370.614359174 3947841760435743.5] int_ic
    =[0 0]" }
{ filter: "laplace", s_xfer: get2OrderFilter(10000000, 5) }
```

Code 6.2: List filter definition types in the FAANN input layer in the faann-simulator library.

Furthermore, ngspice allows the definition of elements using transfer functions in the S-Domain Laplace format, using the `s_xfer` parameter [89]. The fifth line in Code 6.2 demonstrates the definition of a second-order low-pass filter using a transfer function in Laplace format. The final line defines the same filter as the previous case but utilizes the auxiliary function `get2OrderFilter`, whose implementation is shown in Code 6.3.

Overall, the faann-simulator library provides a flexible and versatile approach for defining input filters, allowing researchers to adapt the simulation to various filter configurations and explore different filtering techniques within the FAANN framework.

```
function get2OrderFilter(fc: number, q: number) {
    const om = 2 * Math.PI * fc;
    return `num_coeff=[${om * om}] den_coeff=[1 ${om / q} ${om
        * om}] int_ic=[0 0]`;
}
```

Code 6.3: A function converts a second-order low-pass filter's cutoff frequency and quality factor to ngspice notation.

6.2.2 Analog training

Training analog neural networks constitutes a distinct discipline compared to digital network training. This difference is apparent considering that all inputs and outputs are analog and therefore are continuous in nature. Instead of using a dataset, it necessitates a description of continuous signals.

To cater to this specific requirement, the faann-simulator library introduces a definition of input voltage sources. Moreover, it is also necessary to define a V_{target}. Here, the library provides a universal solution: users can define any target circuit that accepts input signals and generates the target output voltage. In this way, any circuit that FAANN attempts to learn can be defined. In the case of this study, which focuses on the adaptive frequency filter, the reference filter is defined by this target circuit. Figure 6.2 shows an example of a reference filter connection.

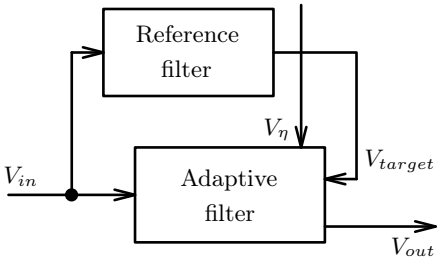

Figure 6.2: Block diagrams used in adaptive filter learning simulations.

The faann-simulator library introduces two new methods enabling work with analog signals. The first, `trainAnalogue`, allows for the training of analog circuits. The second, `predictAnalogue`, facilitates predicting the analog output of the target circuit while the target circuit is not present. Examples of the use of these methods, along with a list of all parameters, can be found in Code 6.4.

```
const analogueResult = await faann.trainAnalogue({
    testName: "adaptive-filter-example",
    learningRate: 0.5,
    trainingTime: 100000,
    stepTime: 100,
    inputSources: [
        { type: "sin", amplitude: 0.5, periodTime: 10000 }
    ],
    targetCircuit: (inputs: string[], targets: string[]) => `
        R1 ${inputs[0]} ${targets[0]} 1592
        C1 ${targets[0]} 0 10p
    `,
    // optional parameters
    fourier: "10MEG",
    saveDetail: false,
    plotNodes: ["v(in1)", "v(out1)", "v(target1)-v(out1)"],
    verbose: 2,
});

const analogueResult = await aann.predictAnalogue({
    testName: "adaptive-filter-example",
    durationTime: 1000,
    startTime: 100000,
    stepTime: 100,
    inputSources: [
        { type: "sin", amplitude: 0.5, periodTime: 10000 }
    ],
});
```

Code 6.4: Demonstration of FAANN training and prediction using analog signals with the faann-simulator library.

Both methods are asynchronous and return an object containing the simulation results. The parameters for these methods are as follows:

`testName` Type: string. This parameter is used for test designation, which is beneficial for subsequent result analysis.

`learningRate` Type: number [V]. This refers to the rate of change of weight in response to error. It is the voltage supplied to V_η during the learning process.

`trainingTime` Type: number [ps]. This refers to the total time of the training process.

`stepTime` Type: number [ps]. This is the time interval between individual steps of the simulation output.

`startTime` Type: number [ps]. This is only for the `predictAnalogue` method. It refers to the time from which the output prediction begins. It is used

to avoid simulating the entire circuit during the time when the circuit is settling.

`inputSources` Type: InputSource. This is the definition of input sources. More information can be found in Section 6.2.2.1.

`targetCircuit` Type: function. This is the definition of the target circuit. More information can be found in Section 6.2.2.2.

`fourier` Optional. Type: string [Hz]. If filled in, the library adds the .fourier command to the netlist, which allows the acquisition of both the circuit's frequency characteristics and Total Harmonic Distortion (THD) analysis from the already performed transient analysis. The value is the fundamental frequency of the Fourier transformation.

`plotNodes` Optional. Type: string[]. If filled in, the specified voltage nodes will be plotted from the simulation results using the native ngspice library.

`saveDetail` Optional. Type: boolean. Default: false. This saves the progress of the voltage during the training. These data are storage-intensive, so it is recommended to save them only when a close observation of training progress is required.

`showProgress` Optional. Type: 0 | 1 | 2. Default: 2. This parameter determines the level of progress information displayed during the training process. The value 0 displays nothing, 1 displays a progress bar, and 2 displays initial statistics along with a progress bar.

This approach opens up the vast potential of using the faann-simulator library in diverse applications that deal with analog signals.

6.2.2.1 Input sources

In the faann-simulator library, input signals can be defined as voltage sources in three ways, all illustrated in the code examples provided in Code 6.5.

```
1 { type: "rectangular", v0: -1, v1: 1, periodTime: 100,
      transitionTime: 1 }
2 { type: "sin", amplitude: 1, periodTime: 100 }
3 { type: "random", distribution: "Uniform", durationTime: 10,
      params: [] }
```

Code 6.5: List of input source definition types for training with the faann-simulator library.

The first way to define an input signal is as a square wave signal, which fluctuates between two voltage states, V_0 and V_1, over a defined period referred

to as `periodTime`. The second way involves defining the input signal as a simple sinusoidal wave.

The third way to define an input signal is as a random signal generated from a statistical distribution. Possible states for this distribution are "Uniform", "Gaussian", "Exponential", or "Poisson". Each type of distribution will generate a distinct set of signals, thus allowing a wide variety of signal behaviors to be modeled [89].

6.2.2.2 Designating a target circuit

The `targetCircuit` parameter is a function that takes two parameters inputs and targets and returns NetList.

The `inputs` parameter refers to an array of input nodes, while `targets` is an array of target nodes. This function provides a versatile framework where the user can define any circuit that the subsequently designed adaptive filter will attempt to learn and emulate.

An example of defining a target circuit is shown in Code 6.6. This example demonstrates how to establish a transfer function for a fourth-order Chebyshev low-pass filter with a cut-off frequency of 100 MHz. Unless otherwise noted, this target circuit is used as the reference filter in the simulations in this chapter.

```
1 targetCircuit: (inputs: string[], targets: string[]) => `
2 A1 ${inputs[0]} ${targets[0]} chebfilter
3 .model chebfilter s_xfer(num_coeff=[2.511957e+30] den_coeff=[1
      53880210 4649291000000000 1.342868e+23 2.818462e+30]
    int_ic=[0 0 0 0])
4 `,
```

Code 6.6: An example of a target circuit defined as a transfer function for a fourth-order low-pass Chebyshev filter with a faann-simulator library.

6.2.3 Adaptive frequency filter training

The sizes of simulated neural networks and filter banks can be quite large and, more importantly, variable, especially in the conceptual phase of development. Creating these structures and reading frequency characteristics from transient analyses in faann-simulator would be very challenging. Therefore, the program is updated as shown in the flowchart in Figure 6.3. Here it is essential to mention that in the learning phase of the adaptive filter, a source with a random waveform close to white noise is used as the input signal. This ensures all frequencies are represented in the input at a similar rate during learning. The target signal is then obtained using the reference filter according to Figure 6.2. In the measurement phase, V_η is then set to 0, and a sinusoidal source of the particular frequency is used as the input signal.

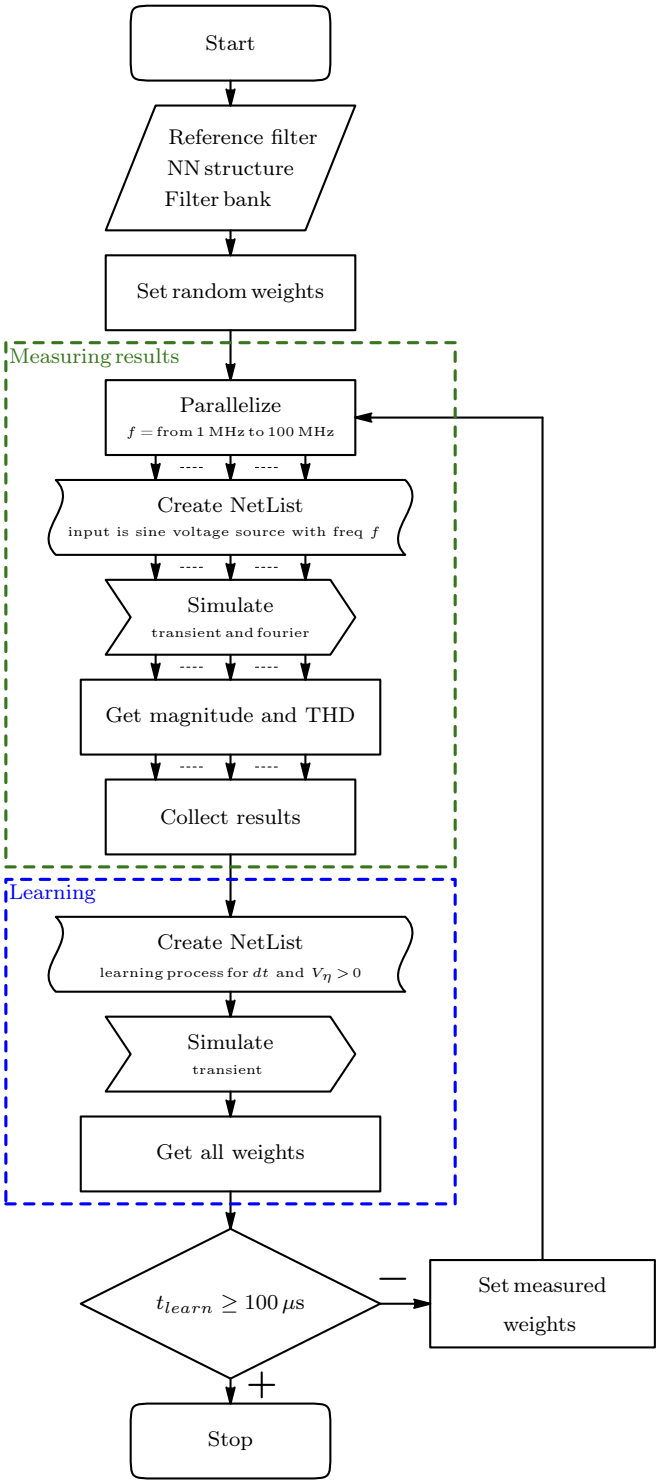

Figure 6.3: Algorithm flow diagram for the simulation of a fully analog adaptive filter.

6.3 Validation of the adaptive filter concept

This section intends to verify the developed concept of analog frequency adaptive filters, mirroring the FAANN validation process by simulations. Initially, the focus is on affirming the principles of adaption and filtration within the time domain. This primary step has proved crucial in fine-tuning the proper functioning and simulation of the proposed filter. The examination also delves into how the FAANN is capable of processing signals from various types of filters. Furthermore, the appropriate neuronal network structures for this task and their parameters are explored. Finally, the ability of the filter to adapt to higher-order filter types is verified. This thorough analysis substantiates the proposed concept's effectiveness and resilience, underscoring its potential for practical application.

6.3.1 Learning progression over time

In adaptive filter simulations, the FAANN's learning is done through a teacher-assisted learning principle that utilizes analog feedback based on backpropagation. The desired output from the neural network after adaptation is that the learning signal is fed into V_{target}. All the learned attributes, including filtering of specific frequencies or phase shifts, are unknown to this filter, and these features are learned solely from the time domain waveform. It enables adaptation to a filter with an unknown transfer function simply by feeding a weighted signal waveform into V_{target}.

In this phase of validation, a positive voltage is applied to V_{η}, resulting in the weights within the neural network changing by the voltage differences between V_{out} and V_{target}, as represented in Equation (4.8). To safeguard the learning process from potential influence by the chosen input signal and to enable the filter to learn at the desired frequencies, the input signal V_{in} is selected as random noise, believed to contain all frequencies at the same rate. The generation of this noise through a voltage source is made possible by ngspice, as indicated in the netlist entry shown in Code 6.7.

```
1 Vin1 in1 0 dc 0 trrandom (1 10n)
```

Code 6.7: NetList notation for random noise as a voltage source for ngspice.

The voltage outputs are visualized in Figure 6.4 and subsequently after 98 μs in Figure 6.5, where V_{out} and the reference V_{target} are displayed using transient analysis. This visual analysis shows that the filter's adaptation process is already underway within the 1 μs. By the conclusion of the 100 μs learning time, as displayed in Figure 6.5, the difference between V_{out} and V_{target} has been minimized. This remarkably rapid adaptation within a short span further underscores the effectiveness of the proposed concept, suggesting its strong potential

for real-time applications.

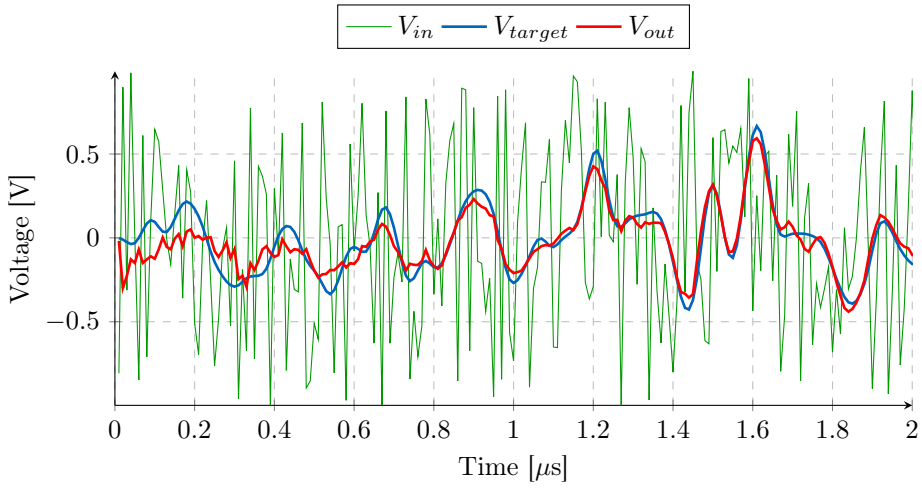

Figure 6.4: Demonstration of the beginning of continuous real-time FAANN filter learning using transient analysis with the input signal V_{in} as random noise.

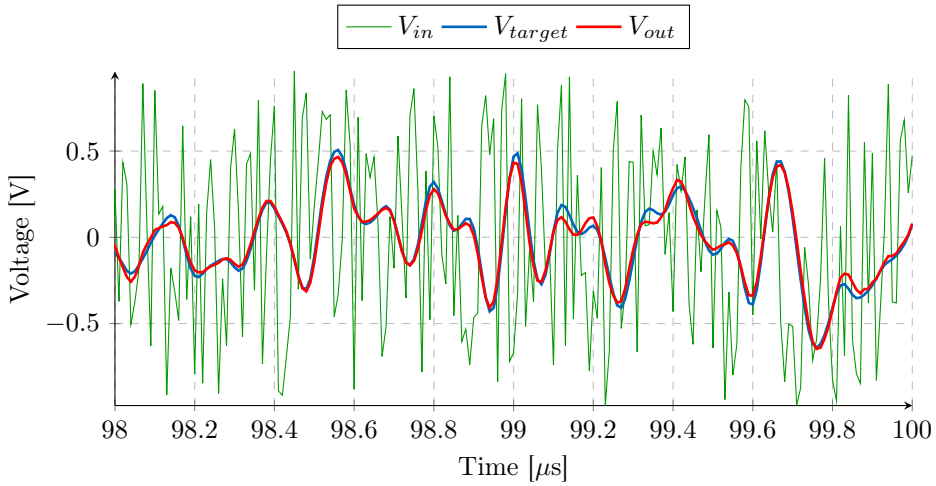

Figure 6.5: Demonstration of continuous real-time FAANN filter learning after 98 μs using transient analysis with the input signal V_{in} as random noise.

6.3.2 Frequency characteristics

For a first verification that the proposed adaptive filter can also adapt to the frequency response over the time domain, only first-order filters as reference filters are simulated in this subsection. Additionally, the filter bank contains the same filter as the reference, among others.

First, a low-pass reference filter with cutoff frequency $f_c = 10\,\text{MHz}$ is tested. The filter bank contains five different filters. The first filter passes the input signal, and the others are the low-pass filters with cutoff frequencies of 100 kHz, 1 MHz, 10 MHz, 100 MHz and 1 GHz. After a training time of 100 μs, the resulting frequency response is shown in Figure 6.6.

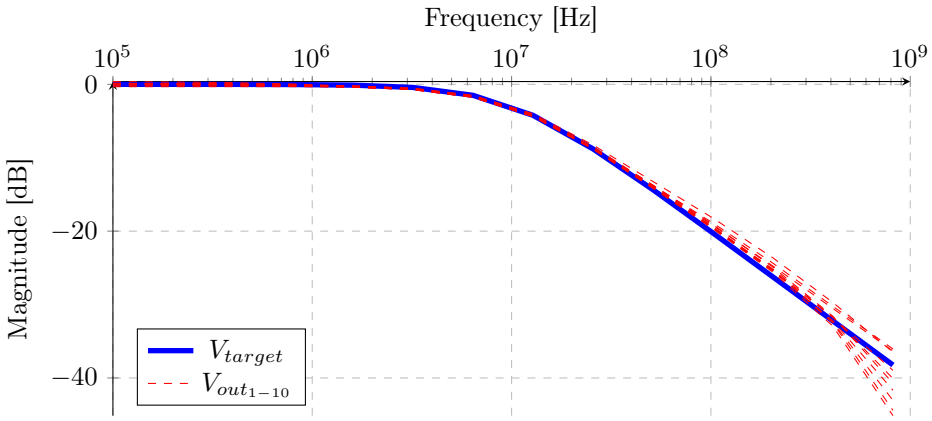

Figure 6.6: Comparison of the amplitude characteristics of the adapted filter and the reference low-pass filter for ten different learning processes.

Since the learning process occurs in the time domain, verifying that adaptation occurs similarly for low and high frequencies is necessary. For this reason, the same test is performed with the high-pass filter with cutoff frequency $f_c = 10\,\text{MHz}$. The filter bank contains the same configuration but now with high-pass filters. After a training time of 100 μs, the resulting frequency response is shown in Figure 6.7.

All types of simulations were run ten times with different initial excursions in FAANN. All twenty simulations performed show that the proposed adaptive filter can indeed adapt to the frequency domain, even for both upper and lower passbands.

6.3.3 Filter error measurement

One of the most exciting features of all adaptive algorithms is their learning process in learning epochs, or in this case, in real-time. For this purpose, at each stage of the frequency characteristic measurement, the error from the reference

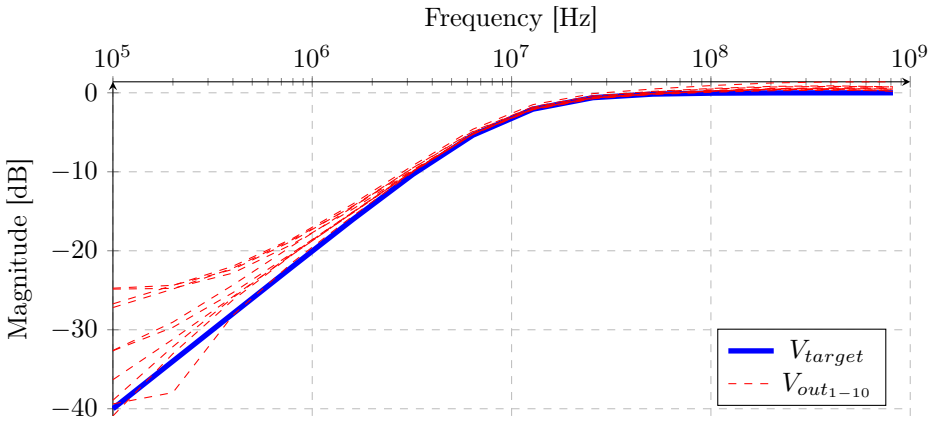

Figure 6.7: Comparison of the amplitude characteristics of the adapted filter and the reference high-pass filter for ten different learning processes.

amplitude is measured at each frequency separately, and the MSE of frequency amplitudes is calculated from them according to

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{V^i_{out_{pp}}}{2} - \frac{V^i_{target_{pp}}}{2}\right)^2, \tag{6.1}$$

where n is the number of measured frequency and $V_{out_{pp}}$ and $V_{target_{pp}}$ are the change between highest value and lowest value in nodes. However, this calculation is only possible under the assumption of a minimum output signal distortion; this is addressed in Section 6.3.4. The new simulation of learning simulations like in Figure 6.4 were run only for 10 μs. After this learning process, $V_{out_{pp}}$ and $V_{target_{pp}}$ were measured for each frequency separately. The learning simulations are run after that for the next 10 μs, and the output voltages are repetitively measured. This process repeats until 100 μs. The results are plotted in Figure 6.8 to show how well the filter is learned as a function of time. The graph shows the results for ten filters learned from scratch with the same reference filter.

6.3.4 Total harmonic distortion

A frequency filter is a linear system, while a neural network, being a highly nonlinear system, therefore exhibits considerable potential for distortion. THD must be evaluated and eliminated for this reason. The degree of distortion depends on factors such as the activation functions employed in the neural network and the level of adaptation that can be achieved to minimize the distortion.

The learned adaptive filter to a first-order low-pass filter is selected for simplicity. Subsequently, 0 V is applied to V_η, and frequencies from 100 kHz to 1 GHz are incrementally introduced to the input V_{in}. For each frequency, a

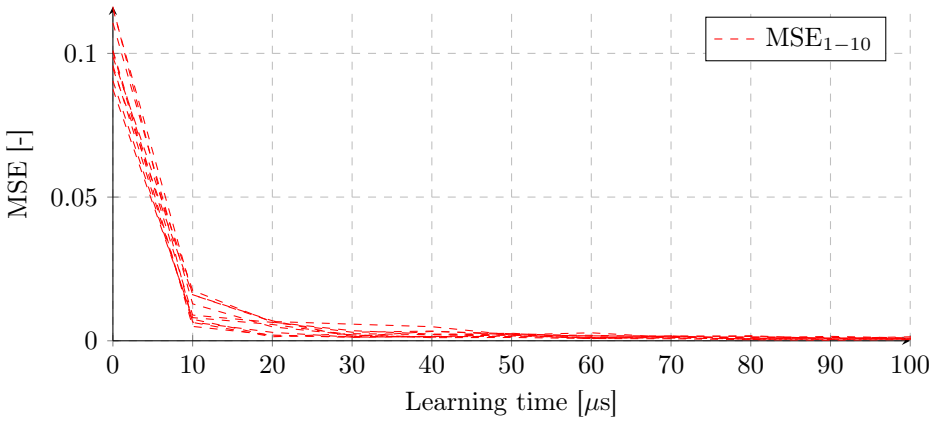

Figure 6.8: The learning curve for an adaptive frequency filter using MSE for frequency filter error.

transient analysis is conducted with THD measurement for the frequency under examination. The THD is calculated directly in ngspice using Fourier analysis, as shown in the NetList in Code 6.8, of the transient analysis output where the result includes the THD value [89].

```
1 fourier 10MEG v(out1)
```

Code 6.8: NetList notation for Fourier analysis with THD output for ngspice.

The outcome of such THD analysis for each frequency independently is displayed in Figure 6.9. The graph presents the results for ten filters, each trained from scratch with the same low-pass reference filter with a cutoff frequency of 10 MHz. The simulations reveal that for filters that have already been trained for only 100 μs, the THD remains below 5 % for all frequencies. Notably, a marginally lower distortion is observable for lower frequencies.

When utilizing a neural network for adaptive filtering, it is crucial to select appropriate parameters, one of which is the choice of the activation function. Thus far, the FAANN has been implemented with three activation functions: "linear", "tanh", and "sigmoid". Consequently, three distinct neural networks have been created, each featuring a different activation function in their hidden layers. Because of the significant difference in THD between the functions, the subsequent simulation used a complex reference filter whose frequency response is shown in Figure 6.18.

The THD is calculated from each measured frequency and depicted in the graph shown in Figure 6.10. Five adaptations are performed for each neural network, starting with different random weights. The graph reveals that, in terms of signal distortion, the linear activation function is the most suitable for filtering applications.

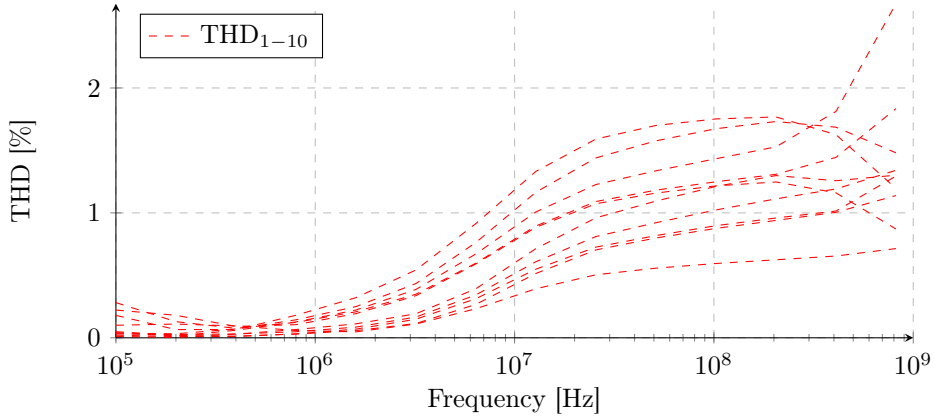

Figure 6.9: Signal distortion of the learned first-order low-pass filter for ten different learning processes.

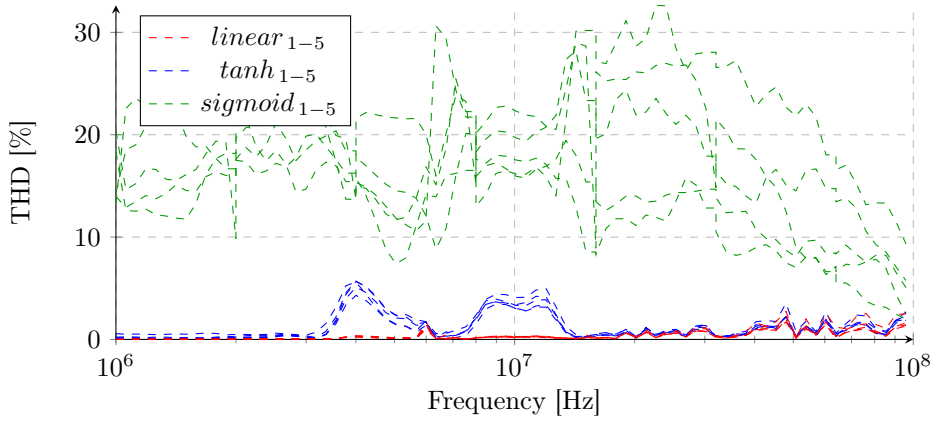

Figure 6.10: Measured THD of the adapted filter with different activation functions.

Subsequent simulations demonstrated that the THD with the linear activation function rapidly falls below 5% for the pass-band frequencies of the tested filters. The THD continues to decrease as the learning process progresses.

6.4 Exploration of adaptive properties

In the preceding section, it was demonstrated that the proposed adaptive frequency filter is capable of adaptation. This section investigates the boundaries of this adaptive filter and its properties. The impacts of setting various hyperparameters on the overall adaptation quality are explored through simulations. As part of this investigation, different parameters will be systematically chosen to construct a generic adaptive filter capable of universal adaptation from 1 MHz

to 100 MHz. These parameters will be adjusted in a manner that will allow for observation of the effects on the ultimate quality of adaptation in subsequent simulations.

6.4.1 Dependence on Filter bank

The filter bank is the initial part of the investigation into the adaptive filter. An examination will be conducted to ascertain how the final quality of adaptation is influenced by the types of filters, the number of filters, their order, and their inaccuracies within the bank. Due to the non-linear propagation of signals in neural networks, it is hypothesized that the effects of filters in the bank could be intensified or possibly transformed. Simulations and analyses are carried out to substantiate these hypotheses. Understanding the dependency of adaptation quality on these filter bank parameters is vital for comprehending the behavior of the adaptive frequency filter. It lays the foundation for its optimization in diverse application settings.

6.4.1.1 Filter type

The first test demonstrates the dependence of the direction of the bank pass relative to the reference. The filter bank in this simulation is the same as in the low-pass simulation above; that is, one without filtering and then five low-pass filters with different cutoff frequencies of 100 kHz, 1 MHz, 10 MHz, 100 MHz and 1 GHz. However, the reference filter is set as a first-order high-pass filter with a cutoff frequency of $f_c = 10\,\text{MHz}$. The result of this simulation is shown in Figure 6.11.

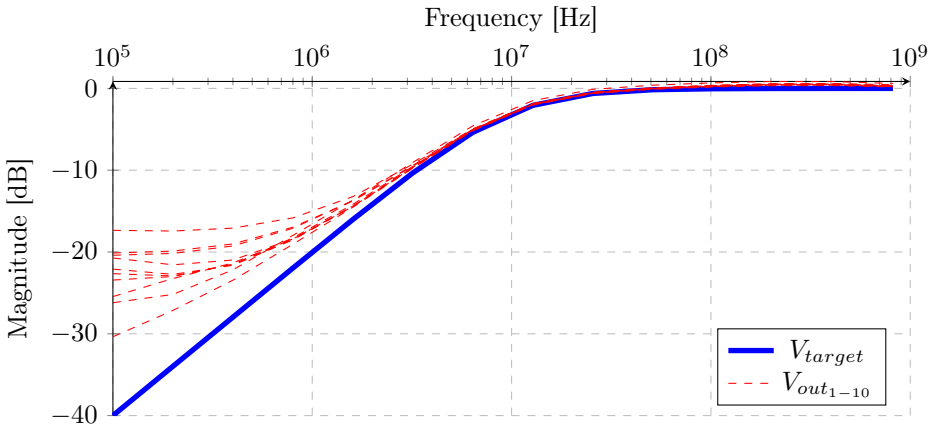

Figure 6.11: Comparison of the amplitude characteristics of the adapted filter as a high-pass filter with only low-pass filters in a filter bank for ten different learning processes.

The graph shows that this adaptive filter can learn high-pass behavior even if only a low-pass filter exists in the filter bank. Other simulations not presented here have shown that the same process works with inverted passbands. Another positive feature is that, after learning the filters constructed this way, the THD is still below 5 % for all measured frequencies. When creating a filter bank, it is possible to use only one type of filter that fits the application.

6.4.1.2 Cutoff frequencies

The next test determines whether the adaptive filter can adapt the cutoff frequencies of the filters in the bank. For this purpose, the reference filter is set to a first-order low-pass filter with a cutoff frequency of $f_c = 10\,\text{MHz}$. However, the filter bank is reduced to only three filters. The first filter passes the input signal, and the others are low-pass filters with cutoff frequencies of 1 MHz and 100 MHz. The frequency response of this simulation can be seen in Figure 6.12.

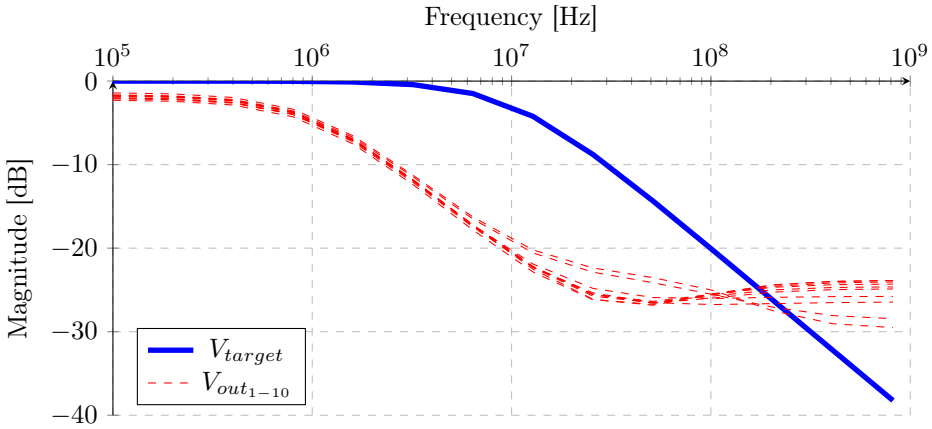

Figure 6.12: Comparison of the amplitude characteristics of the adapted filter ($f_c = 100\,\text{MHz}$) with a not fitting filter in the bank (1 MHz and 100 MHz) for ten different learning processes.

Naturally, the next question is how this concept behaves with a bank of filters set entirely out of range. For this situation, the cutoff frequency of the reference filter is set to $f_c = 100\,\text{MHz}$. The filter bank is again reduced to three filters. The first filter passes the input signal, and the others have their cutoff frequency set to 1 MHz and 10 MHz. After 100 μs of learning, the frequency characteristics are shown in Figure 6.13.

The results of both these tests show that if the filter bank is inconsistent with the reference, the concept cannot fully adapt. However, even so, the filter tends to achieve the best possible fit in both the time and frequency domains. Thus, if the filter bank contains a sufficient number of filters in the specified

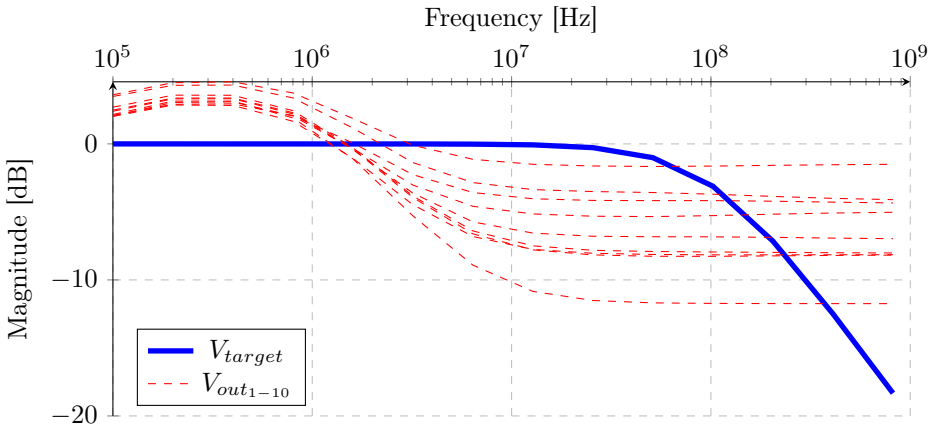

Figure 6.13: Comparison of the amplitude characteristics of the adapted filter ($f_c = 100\,\text{MHz}$) with filters in the bank out of range (1 MHz and 10 MHz) for ten different learning processes.

range, the adaptive filter can come as close as possible to any needed filter.

6.4.1.3 High-order filters

Another fascinating feature of the adaptive filter, which is being investigated here, is its ability to adapt to high-order filters. The assumption that this concept could adapt to high-order filters if similar-order filters are present in the bank would not come as a surprise. However, as indicated in Section 6.4.1.1, thanks to FAANN, this concept can generate filters of one type from filters of another type. Here arises a possibility that the adaptive filter can also adapt to higher-order filters, exceeding those present in the filter bank.

However, it remains to be seen if the lower-order filters in the bank can generate a high-order filter. To investigate this, first-order and second-order filters with varying Quality factor (Q) are chosen for simulations, the results of which are shown in Figure 6.14. From the figure, it can be seen that the first-order filter and filters with $Q \leq 0.5$ are unable to adapt to the higher-order filter. In contrast, 2nd order filters with $Q > 0.5$ possess this capability.

Figure 6.14 also shows that when Q is small, the amplitude's rate of decline in the frequency domain is more gradual. Conversely, if Q is too high, the rapid decreasing tendency prevents a flat pattern in the pass-band region, which may be mitigated through various measures, such as adjusting the number of filters in the bank. These findings emphasize the importance of carefully selecting filter parameters to optimize the filter's performance and adaptability of adaption.

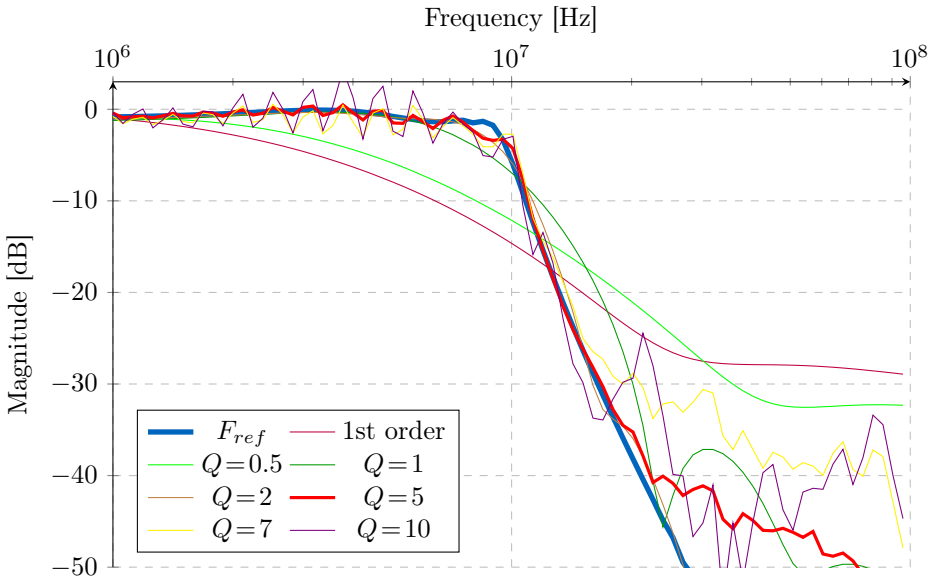

Figure 6.14: Frequency characteristics of adapted filters with different filter types in the bank. Filters with defined Q are 2nd order.

6.4.1.4 Number of filters in the bank

Another critical parameter is the number of filters in the bank itself. Thus, banks of varying numbers of second-order filters with $Q = 5$ are created so that they are logarithmically uniformly distributed between 1 MHz and 100 MHz. The results of these simulations are shown in Figure 6.15. Again, the results show that if there are fewer filters, it is more challenging to adapt between the cutoff frequencies of the filters in the bank. Conversely, the more filters there are in the bank, the more likely and easier it is for the filter to adapt.

A filter bank containing second-order filters with $Q = 5$ will be used in all other simulations in this chapter. The number of these filters will be 24 to show the shortcomings caused by an insufficient filter bank. The cutoff frequencies of these filters are: 1 MHz, 1.21 MHz, 1.47 MHz, 1.78 MHz, 2.15 MHz, 2.61 MHz, 3.16 MHz, 3.83 MHz, 4.64 MHz, 5.62 MHz, 6.81 MHz, 8.25 MHz, 10 MHz, 12.12 MHz, 14.68 MHz, 17.78 MHz, 21.54 MHz, 26.1 MHz, 31.62 MHz, 38.31 MHz, 46.42 MHz, 56.23 MHz, 68.13 MHz and 82.54 MHz.

6.4.1.5 Adaptation sensitivity to filter inaccuracies

As has been demonstrated, one of the significant drawbacks of filter banks is that they should contain a larger number of filters and, secondly, that the filters they contain should be very accurate. This second drawback can be mitigated with the described adaptive filter concept because the adaption does not assume

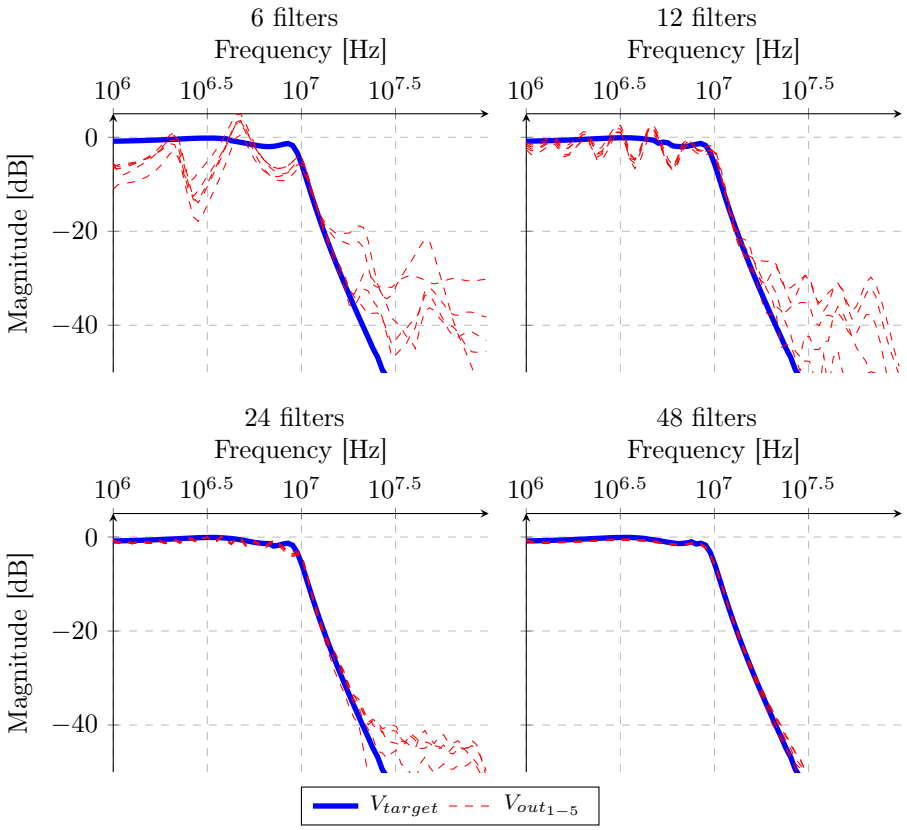

Figure 6.15: Frequency characteristics of adapted filters with a different number of filters in the bank.

specific filter values but uses what is available. In the following simulations, each filter bank is set with random variation of filter parameters, namely $Q \pm 20\%$ and cutoff frequency $f_c \pm 10\%$. The resulting frequency waveforms in Figure 6.16 show that even with such parameter variance, the result is not much worse. It allows the filter bank to assemble itself from higher tolerance components and thus simplify the manufacturing process.

6.4.2 Dependence on neural network structure

Another integral aspect of the proposed adaptive filter to explore is the structure of the FAANN and its influence on adaptation outcomes. This subsection presents the filter adaptation with various neural network structures and their corresponding effects on the results of adaptation.

Initially, a simulation is carried out with a neural network consisting of only one output neuron, the outcomes of which are illustrated in Figure 6.17 at the top of the figure. Subsequently, structures with one and two hidden layers are established, each layer containing 3, 6, or 12 neurons. Once more, for each

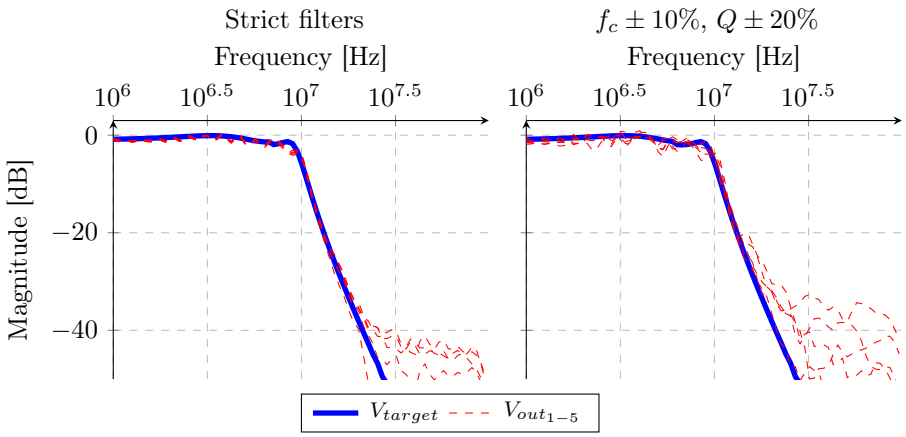

Figure 6.16: Comparison of frequency characteristics of adapted filters with precisely defined filters in the bank versus filters with deviations $f_c \pm 10\%$ and $Q \pm 20\%$ in the bank.

structure, five adaptation processes are carried out.

In order to be able to compare the changes in the adaptive filter attributes, a reference 4th-order Chebyshev filter with a cutoff frequency of 10 MHz is used throughout the chapter unless otherwise noted.

From the frequency characteristics in Figure 6.17 at the top, it can be seen that one neuron already handles the adaption in a decent way. A better choice of other neural network parameters or more extended learning could solve the visible minor attenuation in the pass-band part of the filter. The graph in Figure 6.17 shows that the smaller neural network structure performs better with the current adaptive filter training parameters. One of these parameters is the learning length, set to 100 μs for all simulations in this paper, to allow comparison of the other parameters. However, tests that are not included in this work show that larger neural network structures have a greater potential to adapt to the desired reference filter but require a longer learning time.

For the purpose of this chapter, a structure with one hidden layer of six neurons with linear activation functions was chosen, which is used in all other simulations in this chapter.

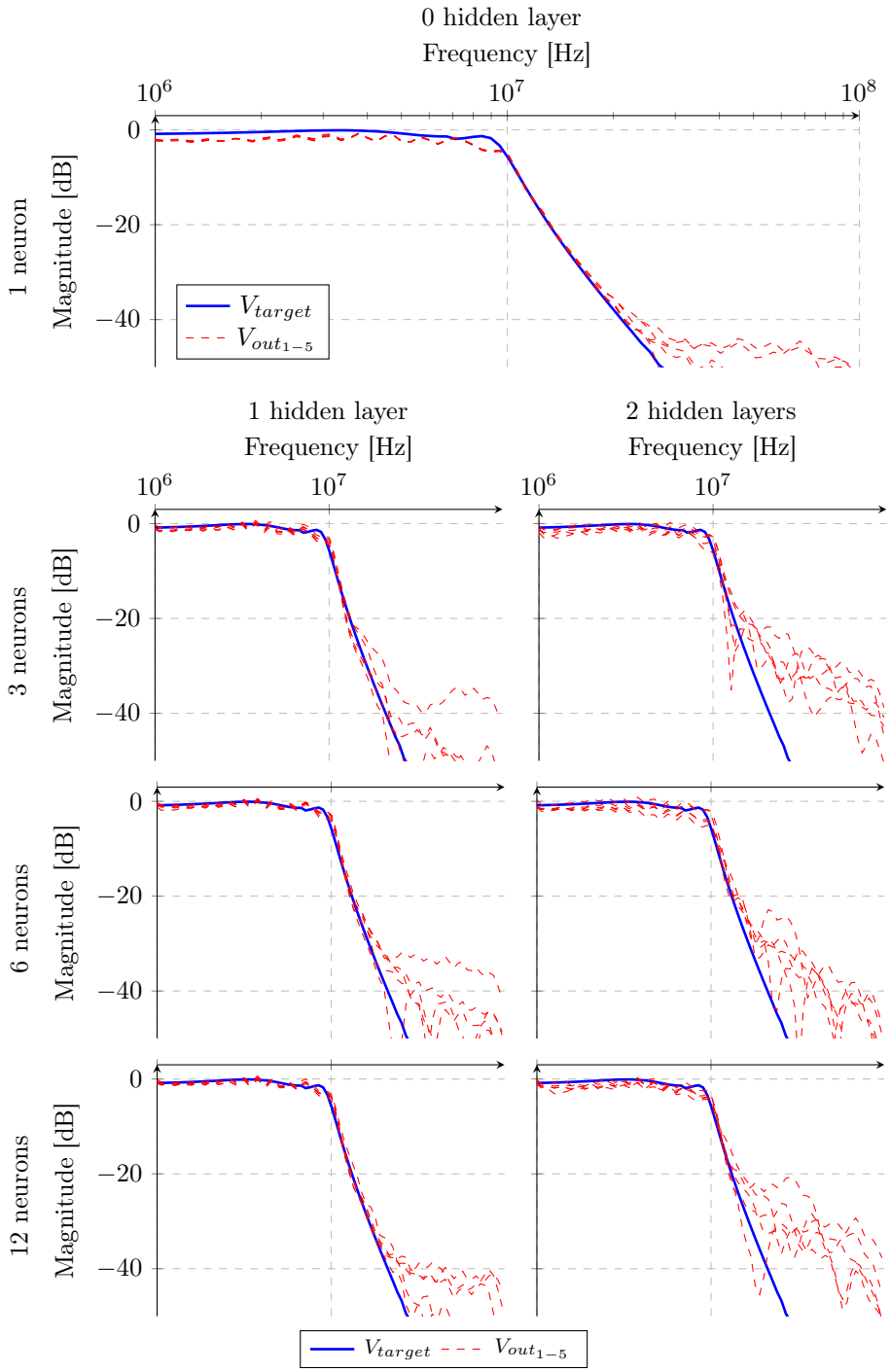

Figure 6.17: Frequency characteristics of adapted filters for different FAANN structures.

6.5 Adaption efficiency

In this chapter, a simpler general adaption filter was developed for adaption in the range of 1 MHz to 100 MHz using a filter bank containing 24 second-order filters with $Q = 5$, six hidden layer neurons and one output neuron. One particular filter was designed and subjected to the adaption test for low-pass and high-pass 4th-order Chebyshev-type filters with cutoff frequencies of 3 MHz, 7 MHz, 15 MHz, and 40 MHz. Subsequently, the 8th order Chebychev-type band-pass and band-stop signals were also tested with a range of 2.5 MHz - 5.5 MHz, 6 MHz - 9 MHz, 13 MHz - 21 MHz, and 32 MHz - 56 MHz.

Each of these adaptions was run five times with different initial weights. These simulations' results can be seen in Figure 6.19 and Figure 6.20. At the same time, all adaption runs lasted only 100 μs of simulated time. Despite this short learning time, the results are very promising.

6.5.1 Load test

The last simulation in this dissertation is a load test of this adaptive filter. For this purpose, a dual band-pass filter used, for example, in Very high-speed Digital Subscriber Line (VDSL) modems, was chosen [106]. The attempt to adapt to this more complex filter can be seen as red lines in Figure 6.18. The THD of the filter learned in this way can also be seen as red lines in Figure 6.10. The plot shows that an adaption filter set up in this way is unsuitable for such a challenging task. However, just a tiny change in the filter bank to 48 second-order filters and the result can be seen in the same Figure 6.18 as the green line.

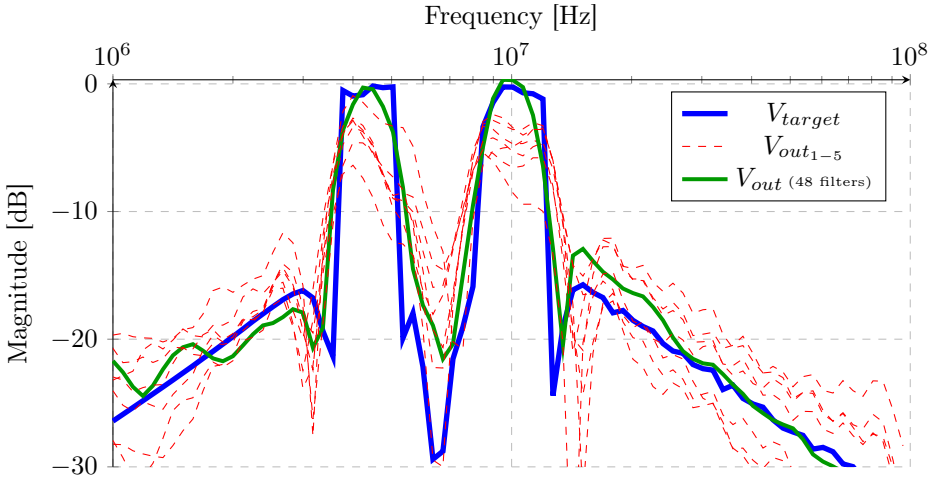

Figure 6.18: Frequency characteristics of the adapted filter to dual band-pass filter.

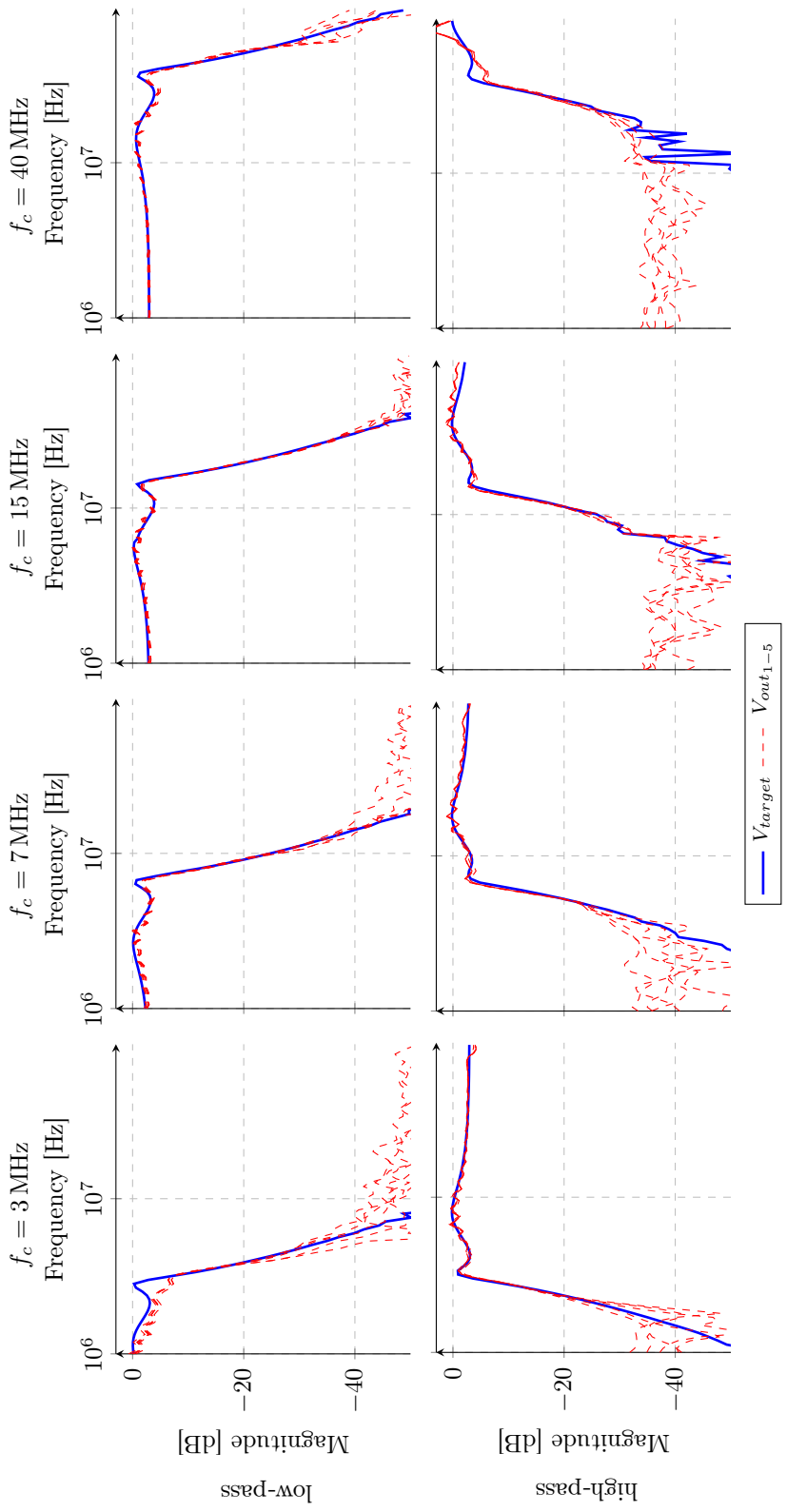

Figure 6.19: Adaption of one particular adaption filter to low-pass and high-pass 4th order Chebyshev-type filters.

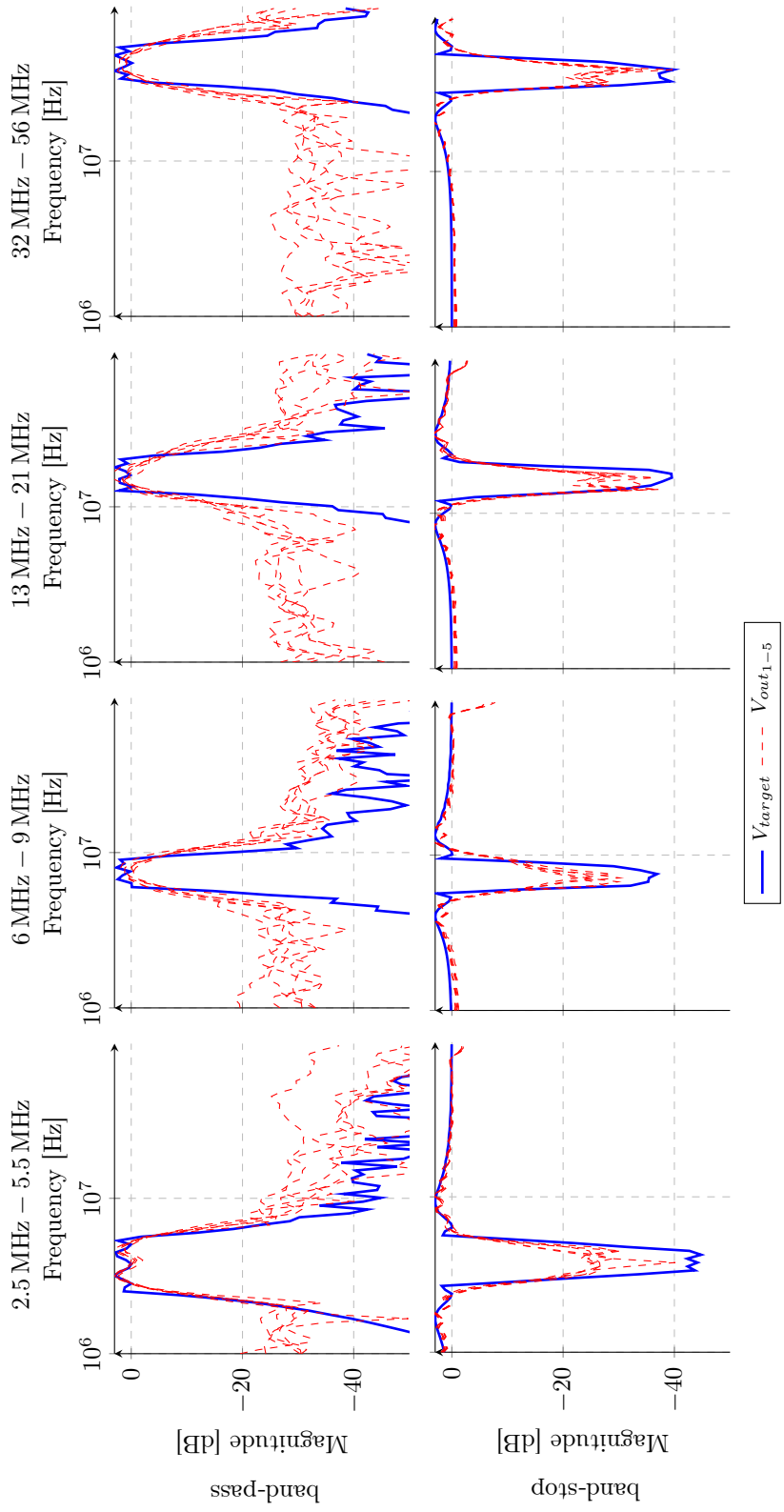

Figure 6.20: Adaption of one particular adaption filter to the 8th order Chebychev-type band-pass and band-stop.

Chapter 7
Conclusion

This dissertation primarily focuses on the development of a Fully Analog Artificial Neural Network (FAANN) designed to process signals directly from an extensive array of electronic sensors. Signal processing aims to enhance measurement precision while considering concept drift and the possibility of providing not directly measured properties of the signal system.

Nowadays, neural networks are popular and are used for a wide range of applications in different fields. Therefore, various neural network computation accelerators are currently being developed. Increasingly, these accelerators are being implemented in analog form, mainly for computation parallelization and power consumption. It is also true for implementing neural network learning processes, which currently often face problems associated with sampling, ADC, DACs, synchronization signals, and clock control when working real-time. These factors together make real-time learning a significant challenge for high-speed systems.

Based on the analysis, a new concept of FAANN was proposed. The main innovation is the fully analog learning process derived from the backpropagation algorithm using gradient descent. This strategy addresses the problems of analog implementation and effectively circumvents the von Neumann bottleneck, and allows real-time learning. At the same time, the concept is carefully formulated to exclude synchronization signals, thereby becoming fully analog and, therefore, fully parallel. It offers significant improvements in signal processing speed and anticipated energy consumption, demonstrating the immense potential for applications such as Internet of Things (IoT) and high-speed applications, image processing, and high-frequency signal control.

The subsequent four sections of this paper discuss the contributions of this work, with each section dedicated to a specific objective of this dissertation.

7.1 The new design

Chapter 4 is devoted to the design of a new fully analog neural network concept where the emphasis is on avoiding the von Neumann bottleneck and not utilizing any synchronization signals or any clock control. This approach enabled a fully analog and, thus, fully parallel computational design for neural networks to be developed, which is a crucial feature of this system.

The design's most significant innovation lies in transposing the weight learning process into the analog world inspired by the backpropagation algorithm founded on the gradient descent method. This transposition was achieved through the process of charging capacitors with current. Unlike traditional methods that employ discrete steps, the approach allows for the continuous charging of capacitors, thus enabling weight changes.

This concept is mathematically developed to intermediate results, which serve as the foundation for implementation using well-known electronic blocks.

The result is an analog neuron that not only processes signals but also contains learning circuits. It makes connecting several such neurons into various types of neural networks possible, enabling diverse applications.

As part of the design, a demonstration is also prepared on how these neurons can be integrated into a Feedforward Neural Network (FNN) with hidden layers and an implemented Mean Squared Error (MSE) analog function as a loss function.

7.2 Functionality validation

Chapter 5 is dedicated to validating and exploring the proposed concept. It utilizes established graphical electrical circuit simulators GEEC and a specially designed faann-simulator library for analog neural networks. This library uses ngspice as a core for calculations and simulations.

Thanks to the electrical behavior simulation models in Section 5.3.4, it is demonstrated that the proposed design possesses the ability to learn. The well-known XOR problem was used to validate this learning process. This problem is recognized in machine learning due to its non-linear separability and serves as an excellent metric for assessing a model's ability to solve problems and adapt.

Simulations also show that the new design is partially resilient to the parasitic properties of the used blocks. Compared with classical neural networks, designed neural networks demonstrate similarity in behavior. It is a very positive result, suggesting that this neural network could be deployed similarly to the traditional artificial neural networks, thereby lowering the barrier to their acceptance within the data science community.

Section 5.4.2 gives the learning speed comparison results, showing that the

designed concept could offer an order-of-magnitude acceleration compared to classic ANNs. It suggests the possibility of deploying the design in real-time, even for large networks, which is a promising prospect.

7.3 Adaptive frequency filter

Chapter 6 is devoted to the practical application of FAANN, showing its full potential through implementing a fully analog adaptive frequency filter. However, it is important to underline that this specific example represents only one of the many possible applications that could effectively employ FAANN.

The adaptive frequency filter is designed to facilitate adaptation at high frequencies, for higher orders, and various filter types, all in real-time adaptation. A pivotal element of this design is the filter bank, which is directly connected to the neural network's input layer. Thanks to this configuration, the proposed neural network takes care of the adaptation or, more accurately, the learning process.

The learning of such a proposed adaptive frequency filter consists of learning in the time domain. It implies that the training of the filter does not require the use of frequency characteristics or complex mathematical descriptions. Instead, only a reference signal is needed for the filter to learn, from which the frequency domain gains knowledge. This concept opens up new possibilities, simplifies usage, and makes the adaptive frequency filter a universal solution for a wide range of applications.

Figure 6.1 shows one of the possible configurations of the adaptive frequency filter used in all simulations mentioned in the following section.

7.4 Adaptive filter properties exploration

Simulations in Section 6.3 have verified the basic functionality of this concept. It is shown that the adaptive filter converges to the reference filter depending on the filter bank and other hyperparameters. At the same time, in the presented simulations, the nonlinearities decrease to units of % THD during adaptation.

In Section 6.4, the properties of the most critical parameters of the neural network used, as well as the filter banks, are described through simulations:

`Neural network activation function.` For frequency filtering purposes, linear activation functions are most suitable. It is mainly due to the smaller output signal distortion, as shown in Figure 6.10.

`Neural network structure.` Figure 6.17 shows that simpler structures can adapt more efficiently in the same learning time. On the other hand, larger structures can adapt better after a longer learning time.

`Filter type dependence.` As demonstrated in Section 6.4.1.1, due to the non-linearity of the neural network, it is possible to adapt to the high-pass filter with the low-pass filters. Simulations in Figures 6.19 and 6.20 demonstrated that only low-pass filters in the bank are necessary to adapt to filters types such as low-pass, high-pass, band-passes, and stop-bands.

`High-order adaptive filters.` Section 6.4.1.3 explores the possibilities of adapting to a higher-order frequency filter. The simulations plotted in Figure 6.14 show that the proposed adaptive frequency filter is able to adapt to higher-order filters required to use at least second-order filters with quality factor $Q > 0.5$ in the bank.

`Number of filters in the bank.` Figure 6.15 shows that the adaptive filter learns more effectively with a higher number of filters in the bank. Twenty-four filters seem to be adequate for an adaptation frequency range of two decades.

Based on the mentioned findings, a simple general adaptive filter is proposed to adapt the frequency band from 1 MHz to 100 MHz. This filter consists of only six neurons in one hidden layer, one output neuron, and 24 filters, which consist of only second-order low-pass filters in the bank. All simulations with this simple adaptive filter configuration have shown that it can be adapted in 100 μs to various Chebyshev's 4th-order low-pass and high-pass and 8th-order band-passes and stop-bands as is seen in Figures 6.19 and 6.20. The same filter is tested to adapt to the more complex dual-band filter in common use, and the result is satisfactory when the number of filters in the bank is increased to 48, as shown in Figure 6.18.

7.5 Summary

In this research, a new concept of fully analog artificial neural networks for signal processing is successfully designed with learning capabilities that do not rely on any clock or synchronization signal. The functionality of this system is verified, and it exhibited similar behavior to traditional artificial neural networks. Based on this concept, an adaptive frequency filter is constructed, demonstrating excellent characteristics. It shows the potential of this technology to address the challenges of neural networks for real-time learning and processing signals directly from sensors.

Chapter 8
Bibliography

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[2] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[3] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, p. 480, 1969.

[4] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[5] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The state of the art in online handwriting recognition," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 8, pp. 787–808, 1990.

[6] K. A. Smith, "Neural networks for combinatorial optimization: a review of more than a decade of research," *Informs journal on Computing*, vol. 11, no. 1, pp. 15–34, 1999.

[7] Y. Li, "Research and application of deep learning in image recognition," in *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*, pp. 994–999, IEEE, 2022.

[8] C. Dong, Y. Li, H. Gong, M. Chen, J. Li, Y. Shen, and M. Yang, "A survey of natural language generation," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–38, 2022.

[9] R. H. Abiyev, M. Arslan, and J. B. Idoko, "Sign language translation using deep convolutional neural networks," *KSII Transactions on Internet & Information Systems*, vol. 14, no. 2, 2020.

[10] D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal, N. Kumar, G. P. Joshi, and W. Cho, "A review on autonomous vehicles: Progress, methods and challenges," *Electronics*, vol. 11, no. 14, p. 2162, 2022.

[11] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[13] S. B. Maind, P. Wankar, *et al.*, "Research paper on basic of artificial neural network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96–100, 2014.

[14] O. K. Ernst, "Stochastic gradient descent learning and the backpropagation algorithm," *University of California, San Diego, La Jolla, CA, Tech. Rep*, 2014.

[15] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[16] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19143–19165, 2019.

[17] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[18] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 315–324, 2020.

[19] A. Moghar and M. Hamiche, "Stock market prediction using lstm recurrent neural network," *Procedia Computer Science*, vol. 170, pp. 1168–1173, 2020.

[20] Z. Karevan and J. A. Suykens, "Transductive lstm for time-series prediction: An application to weather forecasting," *Neural Networks*, vol. 125, pp. 1–9, 2020.

[21] H. Su, W. Qi, C. Yang, J. Sandoval, G. Ferrigno, and E. De Momi, "Deep neural network approach in robot tool dynamics identification for bilateral teleoperation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2943–2949, 2020.

[22] P. Lee, S. Bubeck, and J. Petro, "Benefits, limits, and risks of gpt-4 as an ai chatbot for medicine," *New England Journal of Medicine*, vol. 388, no. 13, pp. 1233–1239, 2023.

[23] D. Bzdok and J. P. Ioannidis, "Exploration, inference, and prediction in neuroscience and biomedicine," *Trends in neurosciences*, vol. 42, no. 4, pp. 251–262, 2019.

[24] A. Muthuramalingam, S. Himavathi, and E. Srinivasan, "Neural network implementation using fpga: issues and application," *International Journal of Electrical and Computer Engineering*, vol. 2, no. 12, pp. 2802–2808, 2008.

[25] Y. Kwak, W. J. Yun, S. Jung, and J. Kim, "Quantum neural networks: Concepts, applications, and challenges," in *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 413–416, IEEE, 2021.

[26] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[27] X. Li, G. Zhang, H. H. Huang, Z. Wang, and W. Zheng, "Performance analysis of gpu-based convolutional neural networks," in *2016 45th International conference on parallel processing (ICPP)*, pp. 67–76, IEEE, 2016.

[28] U. Chavan and D. Kulkarni, "Performance analysis of parallel and scalable gpu based convolutional neural network," in *Computing in Engineering and Technology: Proceedings of ICCET 2019*, pp. 467–478, Springer, 2020.

[29] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[30] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 2020.

[31] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*, pp. 44–51, Springer, 2011.

[32] X. Zou, S. Xu, X. Chen, L. Yan, and Y. Han, "Breaking the von neumann bottleneck: architecture-level processing-in-memory technology," *Science China Information Sciences*, vol. 64, no. 6, p. 160404, 2021.

[33] P. Narayanan, A. Fumarola, L. L. Sanches, K. Hosokawa, S. C. Lewis, R. M. Shelby, and G. W. Burr, "Toward on-chip acceleration of the back-propagation algorithm using nonvolatile memory," *IBM Journal of Research and Development*, vol. 61, no. 4/5, pp. 11–1, 2017.

[34] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[35] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[36] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking tpu, gpu, and cpu platforms for deep learning," *arXiv preprint arXiv:1907.10701*, 2019.

[37] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 77–84, IEEE, 2016.

[38] J. Sengupta, R. Kubendran, E. Neftci, and A. Andreou, "High-speed, real-time, spike-based object tracking and path prediction on google edge tpu," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 134–135, IEEE, 2020.

[39] G.-B. Huang, Q.-Y. Zhu, and C. K. Siew, "Real-time learning capability of neural networks," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 863–878, 2006.

[40] A. S. Iwashita and J. P. Papa, "An overview on concept drift learning," *IEEE access*, vol. 7, pp. 1532–1547, 2018.

[41] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[42] R. K. Pandey and P. C.-P. Chao, "An adaptive analog front end for a flexible ppg sensor patch with self-determined motion related dc drift removal," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.

[43] Y. Sun, K. Tang, Z. Zhu, and X. Yao, "Concept drift adaptation by exploiting historical knowledge," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 10, pp. 4822–4832, 2018.

[44] E. Rosenthal, S. Greshnikov, D. Soudry, and S. Kvatinsky, "A fully analog memristor-based neural network with online gradient training," in *2016 IEEE international symposium on circuits and systems (ISCAS)*, pp. 1394–1397, IEEE, 2016.

[45] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[46] A. Pantazi, S. Woźniak, T. Tuma, and E. Eleftheriou, "All-memristive neuromorphic computing with level-tuned neurons," *Nanotechnology*, vol. 27, no. 35, p. 355205, 2016.

[47] F. Zenke and E. O. Neftci, "Brain-inspired learning on neuromorphic substrates," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 935–950, 2021.

[48] A. F. Murray, D. Del Corso, and L. Tarassenko, "Pulse-stream vlsi neural networks mixing analog and digital techniques," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 193–204, 1991.

[49] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[50] O. Krestinskaya and A. P. James, "Binary weighted memristive analog deep neural network for near-sensor edge processing," in *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*, pp. 1–4, IEEE, 2018.

[51] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[52] F. Paulů and J. Hospodka, "Design of fully analogue artificial neural network with learning based on backpropagation," *Radioengineering*, vol. 30, no. 2, pp. 357–369, 2021.

[53] O. Krestinskaya, K. N. Salama, and A. P. James, "Learning in memristive neural network architectures using analog backpropagation circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 719–732, 2018.

[54] T. Morie and Y. Amemiya, "An all-analog expandable neural network lsi with on-chip backpropagation learning," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 9, pp. 1086–1093, 1994.

[55] D. Sinha and M. El-Sharkawy, "Thin mobilenet: An enhanced mobilenet architecture," in *2019 IEEE 10th annual ubiquitous computing, electronics & mobile communication conference (UEMCON)*, pp. 0280–0285, IEEE, 2019.

[56] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[57] M. K. Patrick, A. F. Adekoya, A. A. Mighty, and B. Y. Edward, "Capsule networks–a survey," *Journal of King Saud University-computer and information sciences*, vol. 34, no. 1, pp. 1295–1310, 2022.

[58] L. Yao and Y. Guan, "An improved lstm structure for natural language processing," in *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, pp. 565–569, IEEE, 2018.

[59] D. Neimark, O. Bar, M. Zohar, and D. Asselmann, "Video transformer network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3163–3172, 2021.

[60] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, vol. 32, 2019.

[61] M. Schuld, I. Sinayskiy, and F. Petruccione, "The quest for a quantum neural network," *Quantum Information Processing*, vol. 13, pp. 2567–2586, 2014.

[62] S. Jeswal and S. Chakraverty, "Recent developments and applications in quantum neural network: a review," *Archives of Computational Methods in Engineering*, vol. 26, pp. 793–807, 2019.

[63] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural networks*, vol. 1, no. 1, pp. 17–61, 1988.

[64] W. J. von Eschenbach, "Transparency and the black box problem: Why we do not trust ai," *Philosophy & Technology*, vol. 34, no. 4, pp. 1607–1622, 2021.

[65] M. M. Bejani and M. Ghatee, "A systematic review on overfitting control in shallow and deep neural networks," *Artificial Intelligence Review*, pp. 1–48, 2021.

[66] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and

solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[67] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[68] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

[69] H. Kim, M. U. K. Khan, and C.-M. Kyung, "Efficient neural network compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12569–12577, 2019.

[70] M. V. DeBole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W. P. Risk, J. Kusnitz, C. O. Otero, T. K. Nayak, R. Appuswamy, *et al.*, "Truenorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019.

[71] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[72] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[73] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, pp. 241–258, 2020.

[74] E. Belouadah, A. Popescu, and I. Kanellos, "A comprehensive study of class incremental learning algorithms for visual tasks," *Neural Networks*, vol. 135, pp. 38–54, 2021.

[75] B. Maschler, H. Vietz, N. Jazdi, and M. Weyrich, "Continual learning of fault prediction for turbofan engines using deep learning with elastic weight consolidation," in *2020 25th IEEE international conference on emerging technologies and factory automation (ETFA)*, vol. 1, pp. 959–966, IEEE, 2020.

[76] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022.

[77] H. A. Yildiz, M. Altun, A. D. Gungordu, and M. R. Stan, "Analog neural network based on memristor crossbar arrays," in *2019 11th International*

Conference on Electrical and Electronics Engineering (ELECO), pp. 358–361, IEEE, 2019.

[78] R. Perfetti and E. Ricci, "Analog neural network for support vector machine learning," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1085–1091, 2006.

[79] X. Wang, X. Lin, and X. Dang, "Supervised learning in spiking neural networks: A review of algorithms and evaluations," *Neural Networks*, vol. 125, pp. 258–280, 2020.

[80] Y. Munakata and J. Pfaffly, "Hebbian learning and development," *Developmental science*, vol. 7, no. 2, pp. 141–148, 2004.

[81] H. P. Graf, L. D. Jackel, and W. E. Hubbard, "Vlsi implementation of a neural network model," *Computer*, vol. 21, no. 3, pp. 41–49, 1988.

[82] H. Chible and A. Ghandour, "Cmos vlsi hyperbolic tangent function & its derivative circuits for neuron implementation," *International Journal of Electronics and Computer Science Engineering*, vol. 2, no. 4, pp. 1162–1170, 2013.

[83] D. Mikhailenko, C. Liyanagedera, A. P. James, and K. Roy, "M^2ca: Modular memristive crossbar arrays," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2018.

[84] N. Hakim, A. Bhaduri, K. Donepudi, and S. Bodapati, "A hybrid electrical-behavioral modeling approach for pre-and post-silicon electrical validation," in *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pp. 1–5, IEEE, 2012.

[85] F. Paulů and J. Hospodka, "Geec: Graphic editor of electrical circuits," in *2017 International Conference on Applied Electronics (AE)*, pp. 1–4, IEEE, 2017.

[86] F. Paulů and J. Hospodka, "Web-based application for analysis of electrical circuits and systems," in *2018 New Trends in Signal Processing (NTSP)*, pp. 1–4, IEEE, 2018.

[87] M. Cantelon, M. Harter, T. Holowaychuk, and N. Rajlich, *Node.js in Action*. Manning Greenwich, 2014.

[88] G. Bierman, M. Abadi, and M. Torgersen, "Understanding typescript," in *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*, pp. 257–281, Springer, 2014.

[89] H. Vogt, M. Hendrix, P. Nenzi, and D. Warning, "Ngspice user's manual version 34 (ngspice release version)," 2021.

[90] M. Reljan-Delaney and J. Wall, "Solving the linearly inseparable xor problem with spiking neural networks," in *2017 Computing Conference*, pp. 701–705, IEEE, 2017.

[91] J. Rajpoot and S. Maheshwari, "High performance four-quadrant analog multiplier using dxccii," *Circuits, Systems, and Signal Processing*, vol. 39, pp. 54–64, 2020.

[92] F. Paulů and J. Hospodka, "New concept of analogue adaptive filter based on fully analogue artificial neural network," in *2022 New Trends in Signal Processing (NTSP)*, pp. 1–4, 2022.

[93] F. Paulů and J. Hospodka, "High-speed adaptive analog filter based on fully analog artificial neural network," *International Journal of Circuit Theory and Applications*, 2023.

[94] A. Carusone and D. Johns, "Analogue adaptive filters: past and present," *IEE Proceedings Circuits Devices and Systems*, vol. 147, no. 1, pp. 82–90, 2000.

[95] M. Jafaripanah, B. M. Al-Hashimi, and N. M. White, "Application of analog adaptive filters for dynamic sensor compensation," *IEEE Transactions on Instrumentation and Measurement*, vol. 54, no. 1, pp. 245–251, 2005.

[96] M. Hakonen, H. Piitulainen, and A. Visala, "Current state of digital signal processing in myoelectric interfaces and related applications," *Biomedical Signal Processing and Control*, vol. 18, pp. 334–359, 2015.

[97] M. Z. Jamal, D.-H. Lee, and D. J. Hyun, "Real time adaptive filter based emg signal processing and instrumentation scheme for robust signal acquisition using dry emg electrodes," in *2019 16th International Conference on Ubiquitous Robots (UR)*, pp. 683–688, IEEE, 2019.

[98] K. Belwafi, O. Romain, S. Gannouni, F. Ghaffari, R. Djemal, and B. Ouni, "An embedded implementation based on adaptive filter bank for brain–computer interface systems," *Journal of neuroscience methods*, vol. 305, pp. 1–16, 2018.

[99] D. W. Schobben, "Real-time adaptive concepts in acoustics: Blind signal separation and multichannel echo cancellation," 2002.

[100] I. Boyko, E. Glushankov, D. Kirik, K. Korovin, and E. Rylov, "Algorithms for multiple signals adaptive processing in radio engineering systems antenna arrays," in *2021 Systems of Signal Synchronization, Generating and Processing in Telecommunications*, pp. 1–6, IEEE, 2021.

[101] L. Shen, Y. Zakharov, B. Henson, N. Morozs, and P. D. Mitchell, "Adaptive filtering for full-duplex uwa systems with time-varying self-interference channel," *IEEE Access*, vol. 8, pp. 187590–187604, 2020.

[102] S. J. Visser, A. S. Dawood, and J. A. Williams, "Fpga based real-time adaptive filtering for space applications," in *2002 IEEE International Conference on Field-Programmable Technology, 2002.(FPT). Proceedings.*, pp. 322–326, IEEE, 2002.

[103] C. Safarian, T. Ogunfunmi, W. J. Kozacky, and B. K. Mohanty, "Fpga implementation of lms-based fir adaptive filter for real time digital signal processing applications," in *2015 IEEE International Conference on Digital Signal Processing (DSP)*, pp. 1251–1255, IEEE, 2015.

[104] H. Serra, R. Santos-Tavares, and N. Paulino, "Background on switched-capacitor filters," in *Optimization Methodologies for the Automatic Design of Switched-Capacitor Filter Circuits for IoT Applications*, pp. 7–18, Springer, 2022.

[105] B. Rumberg and D. W. Graham, "A low-power and high-precision programmable analog filter bank," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 4, pp. 234–238, 2012.

[106] J. Bičák and J. Hospodka, "Design of dual bandpass and bandreject lc ladder filters," *Radio Engineering*, vol. 15, pp. 26–30, 2006.

Chapter 9
List of author's publications

9.1 Publications related to the topic of the thesis

9.1.1 Publications in impacted journals

[A1] F. Paulů and J. Hospodka, "High-speed adaptive analog filter based on fully analog artificial neural network," *International Journal of Circuit Theory and Applications*, 2023. [50%]

[A2] F. Paulů and J. Hospodka, "Design of fully analogue artificial neural network with learning based on backpropagation," *Radioengineering*, vol. 30, no. 2, pp. 357–369, 2021. [85%]

9.1.2 Publications in conferences indexed in WoS

[A3] F. Paulů and J. Hospodka, "New concept of analogue adaptive filter based on fully analogue artificial neural network," in *2022 New Trends in Signal Processing (NTSP)*, pp. 1–4, 2022. [85%]

[A4] F. Paulů and J. Hospodka, "Web-based application for analysis of electrical circuits and systems," in *2018 New Trends in Signal Processing (NTSP)*, pp. 1–4, IEEE, 2018. [50%]

[A5] F. Paulů and J. Hospodka, "GEEC: Graphic editor of electrical circuits," in *2017 International Conference on Applied Electronics (AE)*, pp. 1–4, IEEE, 2017. [85%]

9.1.3 Publications in conference proceedings

[A6] L. Bernau, F. Paulů, and J. Voves, "Electronic equivalent of consciousness with elementary mental process model," in *Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference*, pp. 487–496, Springer, 2020. [30%]

[A7] F. Paulů and J. Hospodka, "Návrh analogové neuronové sítě," in *LDD'17 Letní doktorandské dny*, Praha: Czech Technical University in Prague, pp. 31–37, 2017. [85%]

[A8] F. Paulů and J. Hospodka, "Metody měření parametrů tekutin," in *VI. Letní doktorandské dny 2016*, Praha: ČVUT FEL, Katedra teorie obvodů, pp. 31, 2016. [90%]

9.2 Other publications unrelated to the topic of the thesis

[A9] K. Janecek, L. Bernau, T. Benka, and F. Paulu, "Mathematical algorithm for understanding numbers and their operations integrated into the cornerstone of learning in a pelmanism game: Mathesso," *ECE Official Conference Proceedings*, 2022.

9.3 Citation of author's publications

The publication [A2] is cited in

- I. A. Wisky, M. Yanto, Y. Wiyandra, H. Syahputra, and F. Hadi, "Machine learning classification of infectious disease distribution status," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 27, no. 3, p. 1557, 2022.
- A. Ellery, "Bootstrapping neural electronics from lunar resources for in-situ artificial intelligence applications," in *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pp. 83–97, Springer, 2022.
- H. Chu, "Research on expert system of japanese auxiliary teaching based on bp neural network," *Mobile Information Systems*, vol. 2022, 2022.

The publication [A4] is cited in

- B. Baharuddin, S. Sulasteri, A. M. Ainun, S. Suharti, and M. R. Rasyid, "Pengembangan bahan ajar berbantuan software maple pada mata kuliah kalkulus I," *EDUKATIF: Jurnal Ilmu Pendidikan*, vol. 3, no. 6, pp. 3898–3904, 2021.

The publication [A5] is cited in

- F. Dhiabi, M. L. Megherbi, A. Saadoune, R. Carotenuto, and F. Pezzimenti, "PyAMS: A new software for modeling analog elements and circuit simulations," *International Journal of Electrical and Electronic Engineering & Telecommunications*, vol. 10, no. 4, pp. 233–242, 2021.