New application of hardware accelerators for more sustainable AI models: Development and evaluation of the Boltzmann machines on a physics-inspired hardware accelerator

Bachelorarbeit

submitted on May 2, 2024

Faculty of Business and Health

Business Informatics - Data Science

Course WWI2021F

by

Simon Spitzer

Supervisor at the training center:

⟨ Hewlett Packard GmbH ⟩
⟨ Dr. Fabian Böhm ⟩
⟨ Research Scientist at Hewlett Packard Labs⟩

Signature of the supervisor

DHBW Stuttgart

⟨ Prof. Dr., Kai Holzweißig ⟩
⟨ der/des wissenschaftlichen Betreuerin/Prüferin ⟩

Contents

List of abbreviations

BM	Boltzmann Maschine
RBM	Restriced Boltzmann Maschine
DSR	Design Science Research
DNN	Deep Neural Network
EBM	Energy Based Model
MCMC	Markov chain Monte Carlo
DNN	Deep Neural Networks
CPU	Central Processing Unit
GPU	Graphics Processing Unit
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
TPU	Tensor Processing Unit
mem-HNN	memristor-Hopfield Neural Network
TIA	Transimpedance Amplifier
DAC	Digital Analog Converter
IT	Information Technology

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In the research and development of generative AI-models, the computing speed and energy efficiency are increasingly becoming the center of attention.[4] The authors of Open AI confirm, that the growth rate of machine learning models surpassed the growth of computing power within computerchips. The required computing power of the models double each 3-4 months but the power of computerchips, after Moore's Law the amount of transistorson a computerchip double only every 2 years.[5] Focussing on current problems like rising energy consumption of datacenters and the associated greenhouse gas emissions due to implemented AI-systems, the search for more efficient solutions is essential for the future. Worldwide energy consumption of data centers increases annually by approximately 20-40%.[6] The International Energy Agency anticipates that by 2026, global energy consumption of data centers will double to 6%, driven by factors such as AI, cryptocurrencies, and digitization.[7]

1.2 Problem statement

A well-known, more energy-efficient approach for AI workloads involves the use of AI accelerators based on Application Specific Integrated Circuit (ASIC)s. These circuits are designed specifically for computationally demanding tasks, such as linear algebra operation. A common example for such an accelerator is Google's Tensor Processing Unit (TPU).[8] This is useful because the usage of multimodels for discirminating tasks compared to task specific models are more energy intense.[9] One promissing accelerator concept in research is the usage on physics-inspired hardware accelerators. Physics-based accelerators exploit natural phenomena to perform highly efficient computation. Examples of this are the use of slime models to design train networks and search for shortest travel routes with self-assembling DNA strings.[10] These accelerators work completely different from conventional digital computing systems and have demonstrated that they can considerbaly accelerate computation at greatly reduced energy consumption.[11] For example, for solving difficult optimizaiton problems, so called Ising machines can be up to 100x more energy than Graphics Processing Unit (GPU)s.[12] This is achieved by mimicking the behavior of networks of magnets, whose behavior minimize their overall energy and thereby find solutions

[4] cf. Luccioni/Jernite/Strubell 2023, p. 1
[5] cf. Dario Amodei/Danny Hernandez 2024, p. 1
[6] cf. Hintemann/Hinterholzer 2022, p. 1
[7] cf.Anon. 2024a, p. 31-37; cf.Jackson 2024, p. 1
[8] cf. Wittpahl 2019, p. 39
[9] cf. Luccioni/Jernite/Strubell 2023, p. 5
[10] cf.Adleman 1994, p.1021; cf.Tero et al. 2010, p. 439
[11] cf. Mohseni, N./McMahon/Byrnes 2022a, p. 1
[12] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, pp. 409–418

to optimization problems. Hence, a scalable physics-inspired hardware accelerator (also called Ising-machine), that surpasses the power of existing standard digital computers, could have a large influence on practical applications for a variety of optimization problems.[13]

Such physics-inspired hardware accelerators offer, due to their special calculation method, potential for efficient processing of computationally intensive tasks. Specifically, the acceleration in contrast to digital computers is achieved by calculating the computationally intense tasks with analog signals. On top of that, the implementation on dedicated hardware offers the possibility to exlpoit the parallelization of digitwal hardware accelerators and analog computation.[14]

Interesting enough, despite their different applications, the energy function of the hardware accelerator that is used in Ising-machines is analogous to ceraitn AI-models, such as Boltzmann Maschine (BM)s and could therefore be used to accelerate the computation in AI-workloads.[15] BMs are energy-based neuronal networks that are used for classification tasks by acllocating a skalar energy for each configuration of variables. During training, the energy associated with the training data is minimized and therefore equal with the solution of a optimization problem.[16] BMs have shown to be performant AI models in a variety of AI workloads, however, their training is known to be computationally demanding.[17] For large-scale BMs, the convergence to energy minima can become prohibitively slow. These challenges complicate the training and the usage of BMs, especially for large data volumes and complex optimization tasks.[18] Nevertheless the similarities of both models implicate, that Ising-machines could be able to execute this specific AI-model with higher energy efficiency and with higher computing speed. Currently there are a few concepts that demonstrate how to achieve a implementation of a BM on a Ising-machine. However, these concepts remain primarily theoretical, with only limited analysis available that quantifies model performance, computing speed, and energy efficiency of a practical Ising machine when implementing a BM. It has not yet been demonstrated how an implementation on a real accelerator chip could function.

With the given background, the following central research question and two sub-questions arise for this thesis:

1. Can Boltzmann Machines be efficiently implemented on a physics-inspired Hardware accelerator by analog noise injection?
    - What is the accuracy of the AI-model on the hardware accelerator compared to conventional methods in terms of efficiency and accuracy?
        - Metrics: Prediction accuracy and negative Likelihood

[13] cf. Mohseni, N./McMahon/Byrnes 2022a, p. 1
[14] cf. Mohseni, N./McMahon/Byrnes 2022a, p. 4
[15] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 10
[16] cf. Nazm Bojnordi/Ipek 2016, p. 2
[17] cf. Nazm Bojnordi/Ipek 2016, p. 1
[18] cf. Nazm Bojnordi/Ipek 2016, p. 2

- How is the accelerators performance in terms of computing speed and energy efficiency compared with other hardware accelerators in literature?
  - Metrics: Throughput (Samples/Sec), Energy consumption (Energy/Operation)

It is therefore necessary to test whether this generative AI model is compatible with Ising machines machines and whether the solution is efficient or not.

1.3 Objective

The primary objective of this bachelor thesis is is to enable and perform a detailed study that quantifies the potential performance gains when implementing BMs with Ising machines Therefore, the extension of a existing physics-inspired hardware accelerator is executed with the aim of evaluating the performance of BMs as an energy based AI-model and therefore to answer the posed research question. In addition to that, it would be beneficial if rules for the influence of hyperparameters could be established since there is no data available for this new method.

To initially accomplish this objective a Simulator Pipeline needs to be developed that models an Ising machine device in connection with a machine learning library running on a digital computer. Such a simulator pipeline is needed, as Ising machines are still under development and meassurements on physical devices is often not possible yet. Hence, the Simulator Pipeline consists of an existing machine learning library and an existing hardware accelerator that need to be connected to each other. First the Simulator Pipeline needs to verify that it is possible for the hardware accelerator to realize BMs. Within the Simulator Pipeline, the activation probabilities of individual neurons are measured on the simulated hardware. If this process proves successful, it is then expanded to simulate a complete neuronal network. The final step is that the hardware accelerator can be compared against training methods running on digital computers. The Simultor includes hardware modelling results based on circuit designs that were recently developed by Hewlett Packard Labs and Forschungszentrum Jülich. This phase includes a carefully adjustment and possibly extension of the existing accelerator to be compliant with the specific requirements of BMs.

If the Simulator Pipeline is validated, performance should be evaluated using a well-known AI workload: the recognition of handwritten digits. The prediction accuracy and negative likelihood are investigated to answer the first research question. In a next step, the throughput (samples/sec) and energy consumption (energy/operation) of the BM on the Ising hardware accelerator is to be collected. These metrics aim to address the second part of the research questions posed.

1.4 Research method

The applied research methodology in this thesis is Design Science Research (DSR) by Österle et al..[19] DSR is chosen because it supports the generation of new knowledge and ensures that research results are both theoretically and practically applicable, while they also are in line with the objectives of the project. Furthermore, this allows to find a solution for practical problems through the iterative approach adding more functionalities over time but also leaving room for fixing errors and general improvements. In the initial design phase, the full Simualtor Pipeline is outlined, from which the requirements and functionalities for implementation are derived. Within the iterative design and evaluation phases of the DSR framework prototyping is used to exploratively as a fast and explorative way to implement the simulator pipeline according to G. Arthur Mihram's prototyping model.[20] Based on the developed Simualtor Pipeline, the research questions is then addressed through computer simulations. The aim of the simulation is to meassure the performance of the solution and to make the hardware accelerator comparable.

1.5 Structure of the thesis

The thesis has six chapters that are primarly structured by the guidelines given by Holzweißig.[21] The first chapter is the "introduction" focussing on the relevance and motivation for the new application of more sustainable AI-models with the novel solution of the physics-inspired hardware accelerator and what research questions should be answered. In the second chapter, the "current state of research and practice" is discussed with existing concepts but moreover all the required concepts that the hardware accelerator utilizes are explained and set into perspective to lay the foundation for the ongoing practical implementation part of this thesis. Next, the third chapter explains the applied research methodologies utilized in this thesis and why the decision is to use them. The fourth chapter covers the implementation of the Simulator pipeline and is strucuted after the DSR phases. It ends with a finished prototype, which is used to meassure the performance and therefore chapter five focusses on the diffusion and discussion of the results. Lastly, chapter six aims to give an overall critical reflexion and outlook on the thesis's applied methodology and achieved results.

[19] cf.Oesterle et al. 2010, p.1-6; cf.Österle/Otto 2010, p. 273-274
[20] cf. Mihram 1976, pp. 71–72
[21] cf. Holzweißig 2017, pp. 32–40

2 Current state of research and practice

2.1 Neural Networks and Sustainability

Over the past few years, the emergence of artificial neural networks has transformed the field of computer vision and extended its influence to other areas. These include natural language processing, game strategy development and execution (with examples in playing Atari and Go), and optimization of navigation tasks, such as determining the most efficient routes on maps.[22] Therefore, it is fair to say that neural networks are part of various important applications.[23] Particularly in the last two years, artificial intelligence has also garnered widespread interest from the public, especially regarding chatbots like ChatGPT and Google Bard.[24] An important feature of a neural network-based system that are inspired by our brain, is that they can learn and adapt to data.[25]

Internally, neural networks are computational models that consist of many simple processing units, called neurons. These neurons are interconnected, such that data can be passed between them. The connections form a network structure, which is often arranged in layers.[26] Neural networks have an optimization function which specifies the goals persued in the learning process.[27] There is a training algorithm that varies all of the hyperparameters, like connection strengths between neurons and biases, etc..[28] The following figure 1 shows a typical neural network that consists out of a input layer, a hidden layer and an output layer with dots representig the neurons wirthin the network.

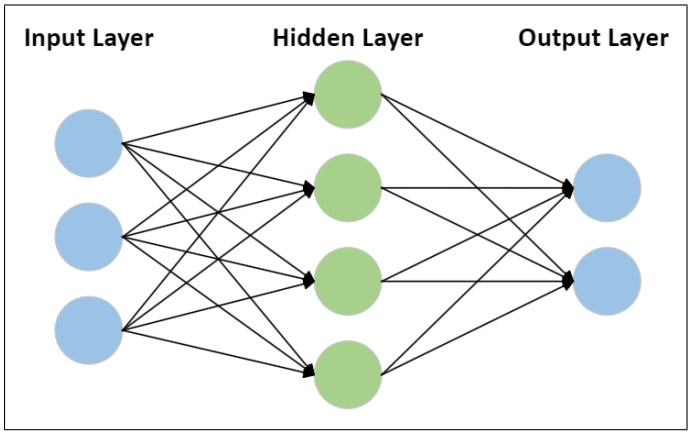

Fig. 1: figure of a neural network

[22] cf. Cichy/Kaiser 2019, p. 305
[23] cf. Gawlikowski et al. 2023, p. 1513
[24] cf. Singh/Kumar, Shubham/Mehra 2023, pp. 1–2
[25] cf. Cichy/Kaiser 2019, p. 305
[26] cf. Cichy/Kaiser 2019, p. 305
[27] cf. Durstewitz/Koppe/Meyer-Lindenberg 2019, p. 1583
[28] cf. Durstewitz/Koppe/Meyer-Lindenberg 2019, p. 1583

When several layer are stacked on top of each other the network is called deep.[29] In general, deep learning methods can be seen as subset of machine learning methods and are today's fundament of artificial intelligence allowing to solve more complex tasks.[30] Deep networks are considered more intriguing than shallow networks because they are capable of learning more complex patterns and abstract representations, enhancing their ability to perform well on a broader range of complex problems. Deep Neural Networks (DNN)s are constantly growing and currently have around 1200 interconnected layers that equal to more than 16 million neurons inside a network .[31] An example of a deep neural network is presented in figure 2 which shows the stacked layers in the middle of the network.

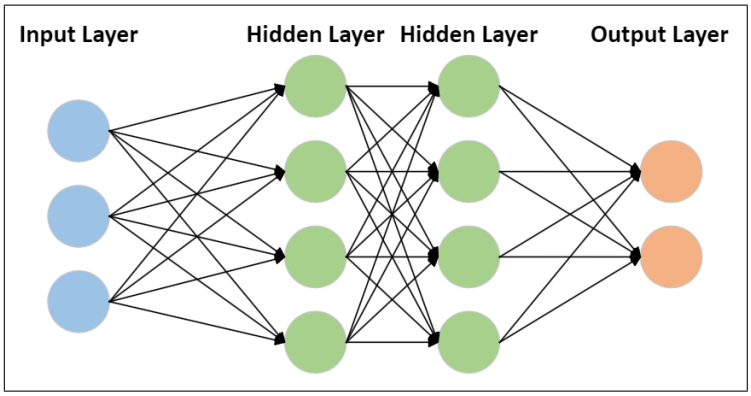

Fig. 2: figure of a neural network

Some examples of regression tasks in the field of computer vision where DNN are essential include object detection, medical image registration, head and body pose estimation, age estimation, and visual tracking.[32] As mentioned, those large neural networks with millions of parameters can be created in the field of neural networks leading to highly performing models.[33] Nonetheless, such models often have a negative effect on the environment in terms of unnecessary energy consumption and a limitation to their deployment on low-resource devices because they are excessively oversized and redundant.[34]

Currently, the exponential growth in model sizes presents substantial challenges, which includes high consumption of computational, energy, and financial resources.[35] The worldwide energy consumption of data centers increases annually by approximately 20-40% and with DNNs only running on high performance computers the future consequences are concerning.[36] The International Energy Agency estimates that global energy consumption attributed to data centers will double to approximately 6% between 2022 and 2026 due to the field of AI, cryptocurrencies, and

[29] cf. Cichy/Kaiser 2019, p. 305
[30] cf. Durstewitz/Koppe/Meyer-Lindenberg 2019, p. 1583
[31] cf. Mall et al. 2023, p. 2
[32] cf. Gustafsson et al. 2020, pp. 325–326
[33] cf. Marinó et al. 2023, p. 152
[34] cf. Marinó et al. 2023, p. 152
[35] cf. Bai et al. 2024, pp. 1–2
[36] cf. Hintemann/Hinterholzer 2022, p. 1

digitization.[37] Not only the energy consumption is a problem, but also the water footprint of AI-models consumption, which refers to the volume of fresh water used to run and train the models.[38] Hence, the water foorprint includes the water used for production (hardware, semiconductors), operation (cooling the data center), and maintenance (cooling the data center) of AI-models. The result of this is created wastewater, which contains a range of pollutants and therefore has negative environmental influences.[39] For example, the training process of ChatGPT-3 consumed approximately 700.000 liters of fresh water, equivalent to the water usage of an average American household over 20 years. In the iterference process, 500ml of fresh water are used for every 50 questions asked to the model.[40] Despite the difficulty in calculating CO2 emissions for AI-models like ChatGPT, due to the lack of publicly available data, conservative estimates can still be made. Initially, training GPT-3 required an estimated 522 tons of CO2, with conservative daily emissions ranging between 35 and 43.2 tons for current interference.[41] Furthermore, text classification emits less CO2 compared to image classification, with image generation accounting for the highest emissions.[42] All these factors motivate to use tailored neural networks designed for a specific purpose, which consume less energy and water and emit less CO2.[43]

2.1.1 Energy-based models

An Energy Based Model (EBM) is a probabilistic type of neural network where the individual neurons exhibit a random behavior.[44] Since 1982, those statistical neural network models have been continuously emerging in the machine learning field when J.J. Hopfield introduced the Hopfield Network.[45] Current developments include their use in reinforcement learning, potential replacements for discriminators in generative adversarial networks and for quantum EBMs.[46] In addition to that, Open AI showed that EBMs are useful models across a wide variety of tasks like achieving state-of-the-art out-of-distribution classification and continual online class learning to name a few.[47] This thesis shows interest in them because EBMs are hardware-friendly, can be trained easily and already are succesfully implemented on multiple hardware accelerators. The underlying idea behind EBMs is to establish a probabilistic physical system that is able to learn and memorize patterns but most importantly generalize it.[48] This probabilistic approach willingly uses uncertainty into the model calculations to draw the models inputs randomly from its underlying distribution.[49] This is done because the conventional deterministic method of backpropagation is known to potentially convert to local minimas, and requires long computation

[37] cf.Anon. 2024a, p. 31-37; cf.Jackson 2024, p. 1
[38] cf. George, A. S./George, A. S. H./Martin 2023, pp. 92–93
[39] cf. George, A. S./George, A. S. H./Martin 2023, pp. 94–96
[40] cf.George, A. S./George, A. S. H./Martin 2023, p. 94-96; cf.Anon. 2024b, p. 1
[41] cf.Anon. 2023, p. 1; cf.Chien et al. 2023, p. 2; cf.Tomlinson et al. 2024, p. 3
[42] cf. Luccioni/Jernite/Strubell 2023, pp. 1–14
[43] ibid cf. Luccioni/Jernite/Strubell 2023, pp. 1–14
[44] cf. Huembeli et al. 2022, p. 2
[45] cf. Hopfield 1982
[46] cf.Verdon et al. 2019, p. 1; cf.Du/Lin/Mordatch 2021, p. 1
[47] cf. Du/Mordatch 2020, pp. 1–2
[48] cf. Huembeli et al. 2022, p. 2
[49] cf. Uusitalo et al. 2015, pp. 25–27

time.[50] As a result with conventional backpropagation more frequently incorrect classification would take place. An EBM is characterized by an energy function $E_\theta(x) \in \mathbb{R}$, with x representing the configuration of the network. This function needs to be minimized to find the solution to the optimization problem, assigning low energy to observed data and high energy to other values.[51]

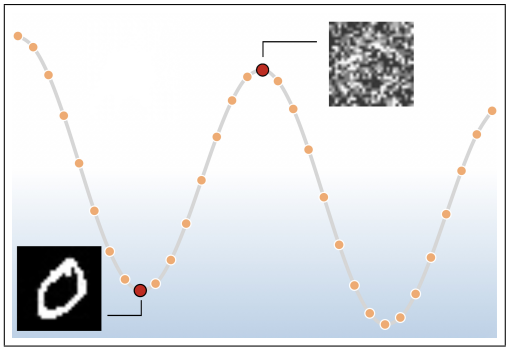

Fig. 3: Figure of a simplified energy landscape

In this figure 1 a simplified energy landscape after the training is shown where the local minima corresponds to states that encode an handwritten digit.[52] It is visible that observed data settles in the local minimum of the energy landscape, in this case a clear 0. On the other hand close to the local maxima of the energy landscape the 0 is only barely recognizable and therefore got a higher energy value assigned to it. The assumption of the underlying distribution function $P(x)$ represents the probability distribution over the input data x, indicating how likely different configurations of x are under the models learned patterns:

$$P(x) = \frac{1}{Z} \exp\left(-\frac{E(x)}{T}\right), \tag{2.1}$$

where Z is the partition function to ensure that the density function normalizes to a total probability of 1 and T is interpreted as the temperature.[53] The partition function Z used in 2.1 is given by summing over all possible pairs of visible and hidden vectors[54]:

$$Z = \sum_x \exp\left(-\frac{E(x)}{T}\right) \tag{2.2}$$

The aim of the training in an EBM is to match the true probability distribution P_{data} as closely as possible with the internal probability distribution P_{model} learned by the model. Here, P_{data} is the probability distribution when the network receives a speicific data input from the environmnet, while P_{model} represents the internal network running freely, also referred to as "dreaming".[55] The specific aim is to adjust its parameters such that P_{model} becomes as close to P_{data} as possible,

[50] cf. Specht 1990, p. 109
[51] cf. Gustafsson et al. 2020, p. 330
[52] cf. Huembeli et al. 2022, p. 6
[53] cf. Huembeli et al. 2022, pp. 2–3
[54] cf. Hinton, G. E. 2012b, p. 4
[55] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, pp. 154–155

which shows the model has learned the distribution of the real world data. A practical method to achieve this goal is to use the KL divergence. KL divergence is a mathematical meassure that helps to meassure how close the predictions are by comparing the model's learned distribution to the true distribution of the data:

$$G = \sum_{x} P_{\text{data}}(x) \ln \left(\frac{P_{\text{data}}(x)}{P_{\text{model}}(x)} \right) \tag{2.3}$$

To optimise the KL divergence the energy is adjusted, whereby data is assigned to low energy states (according to 2.1) and the training data receives high energy and therefore low probabilities.[56]

2.1.2 Boltzmann Machines within Energy Based Models

A BM is a type of symmetrical EBM consisting of binary neurons {0, 1}.[57] The neurons of the network are split into two functional groups, a set of visible neurons and a set of hidden neurons.[58] Therefore, the BM is a two-layer model with a visible layer ("v") and a hidden layer ("h").[59] The visible layer is the interface between the network and the environment. It receives data inputs during training and sets the state of a neuron to either {0, 1} which represents activated or not activated. On the other hand, the hidden units are not connected to the environment and can be used to explain underlying constraints in the internal model of input vectors.[60] The connection between the individual neurons is referred to as bidirectional, as each neuron communicates with each other in both directions.[61] As early as 1985, one of the founding fathers of artificial intelligence, Geoffrey Hinton, was aware that an BM is able to learn its underlying features by looking at data from a domain and developing a generative internal model.[62]

Most machine learning models can be categorized in either generative or discriminative models. Both are strategies to estimate a probability that an specific object can be assigned to a category.[63] Discrimivative models estimate the probability distribution based on category labels that are given to specific objects.[64] On the other hand, a generative model differ as follows. They generate a probabilistic model of the underyling probability distribution for each category, which is assumed as the basis of the data, and in a following step they use Baye's rule to identify which category is very likely to have establihed the object.[65] An real world example would be the following: to predict if a movie will be a hit, one could analyze past box office successes to model characteristics shared by hits (generative approach), or assess immediate audience reactions to

[56] cf. Zhai et al. 2016, pp. 2–3
[57] cf. Amari/Kurata/Nagaoka 1992, p. 260
[58] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 154
[59] cf. Salakhutdinov/Hinton, G. 2009, p. 448
[60] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 154
[61] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 149
[62] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 148
[63] cf. Hsu/Griffiths 2010, p. 1
[64] cf. Gm et al. 2020, p. 2
[65] cf. Hsu/Griffiths 2010, p. 1

movie trailers and reviews to predict success without modeling historical data (discriminative approach). Therefore it can be said that BMs and EBMs are generative models. In the following figure 4, a general BM is depicted, where the upper layer embodies a vector of stochastic binary 'hidden' features, while the lower layer embodies a vector of stochastic binary 'visible' variables.[66]

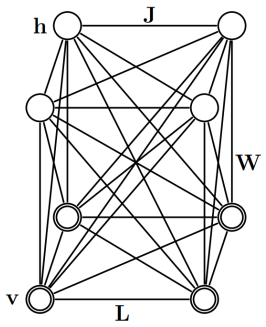

Fig. 4: figure of a general Boltzmann Machine

The model contains a set of visible units $v \in \{0,1\}$, and a set of hidden units $h \in \{0,1\}$ (see Fig. 1). The energy function of the BM with the states $\{v,h\}$ is defined as:

$$E(v,h;\theta) = -\frac{1}{2}v^T Lv - \frac{1}{2}h^T Jh - v^T Wh, \tag{2.4}$$

where $\theta = \{W,L,J\}$ are the model parameters.[67] W,L,J represent visible-to-hidden, visible-to-visible and hidden-to-hidden weights. In BM each neurons works towards minimizing the global energy by entering a particular neuron configuration representing a input to the machine and the system will find the minimum energy configuration that is similar to the given input as shown in fig.3.[68] A simple method to find a local energy minimum involes to switch into wichever of the two states (on or off) of a neuron result in a lower energy given the current state of the other neurons.[69] Inserting equation (2.4) into the earlier introduced KL-divergence (2.2) and doing gradient descend a learning rule to update the weights and biases appers.[70] The gradient descent algorithm is commonly used in machine learning and is an iterative technique that adjusts the model parameters (weights and biases).[71] It progressively acquires the gradient of the energy function, methodically advancing towards the optimal solution and ultimately achieves the minimum loss function along with adjusted parameters.[72] Consequently, this leads

[66] cf. Salakhutdinov/Hinton, G. 2009, p. 449
[67] cf. Salakhutdinov/Hinton, G. 2009, p. 448
[68] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 150
[69] cf. Fahlman/Hinton, G./Sejnowski, T. 1983, p. 110
[70] cf. Hinton, G. E. 2012b, p. 5
[71] cf. Wang, X./Yan/Zhang, Q. 2021, p. 11
[72] cf. Wang, X./Yan/Zhang, Q. 2021, p. 11

to the specific learning rule[73]:

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}) \tag{2.5}$$

The network can now update the weights W_{ij} that exist between the neurons through the training rule based on the observations that served as input.[74] In this case, the square brackets represent expected values, as the training is based on the activation probability. In addition to that, the step sizes of updates to the weights are influenced by the learning rate ϵ within the iterative training process.

Performing exact training in this model is intractable because exact computation of the activation function of $\langle v_i h_j \rangle_{\text{data}}$ and $\langle v_i h_j \rangle_{\text{model}}$ takes a time that is exponential in the number of hidden units.[75] The reason for this is using the maximum likelihood estimator for Z in (2.2) is intractable due to the requirement of summing over all possible states, which leads to an exponential increase in the number of states for larger systems.[76] When the number of hidden units is large compared to the number of visible units it is impossible to achieve a perfect model because of the totally connected network and the resulting 2^n possbilities.[77] Hereby, n represents the number of neurons in the network with each neuron being in one of the two states, the total sum of possibilities are 2^n. A specific example to demonstrate why it is intractable to calculate an activiation of a BM is the following. A fictional BM has 80 visible nodes and 120 hidden nodes and therefore the possbilities of states of neurons are 2^{200}, which is 1.61×10^{60}. To put this into perspective, the total atoms that exist on earth are only estimated to be around 1.33×10^{50}.[78] That means even if it would be possible to store one information per atom it would just not be enough. As a result, instead of directly trying to train the model sampling methods are used that are able to estimate these activation probabilities.

2.1.3 Restriced Boltzmann Machines

As a simplification of the training problem Hinton and Sejnowski proposed Gibbs sampling as an algorithm to aporoximate both expectations.[79] Furthermore, the intralayer connections of the model got removed and the result is the so called RBM. To transform an BM into a RBM the diagonal elements L and J introduced earlier, are set to 0 and as a result an exemple of an RBMs network structure is shown in fig.5.[80]

[73] cf. Hinton, G. E. 2012b, p. 5
[74] cf. Barra et al. 2012, pp. 1–2
[75] cf. Salakhutdinov/Hinton, G. 2009, p. 449
[76] cf. Zhai et al. 2016, pp. 2–3
[77] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 154
[78] cf.Helmenstine 2022, p. 478-480; cf.Schlamminger 2014, p. 1
[79] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, pp. 158–165
[80] cf. Salakhutdinov/Hinton, G. 2009, p. 449

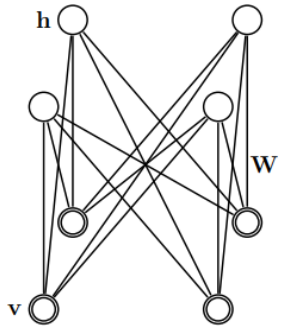

Fig. 5: Figure of a RBM

What can be recognized that no more visible-to-visible and hidden-to-hidden connections can be found in the model. The resulting energy function is given by the energy function (Hopfield, 1982):

$$E(v,h) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij}, \tag{2.6}$$

where v_i, h_j are the binary states of a visible unit i and hidden unit j, a_i and b_j are their biases and w_{ij} is the weight between them.[81] While RBMs have been shown to be easier to train, the restricted structure means a loss in expressivitivity compared to fully connected BMs.[82] The RBM has recently been drawing attention in the machine learning community beceause of its adaption and extention for various tasks such as representational learning, document modeling, image recognition and for serving as foundational components for deep networks including Deep Boltzmann Machines, Deep Belief Networks and hybrid models with CNNs.[83]

Training of BMs

The training of BMs can be established with the use of sampling methods that estimate the activation probabilities, which are needed to update the weights. Here, we consider two commonly used methods that can be chosen from: contrastive divergence and the Metropolis-Hastings algorithm. The goal of the techniques is to create a sequence of correlated steps from a random walk that, after enough iterations, makes it possible to sample a desired target probability distribution.[84] In the following part both methods will be explained in depth. Especially, Metropolis Hastings is interesting since it serve as baseline to compare against the new sampling method of a hopfield network that is to be achieved in the practical part of the thesis.

Contrastive Divergence: Contrastive divergence is a special Gibbs Sampling training method developed by Geoffrey Hinton for the efficient training of RBMs.[85] In traditional, Gibbs sampling

[81] cf. Hinton, G. E. 2012a, pp. 3–4
[82] cf. Huembeli et al. 2022, p. 4
[83] cf. Zhang, N. et al. 2018, p. 1186
[84] cf. Patrón et al. 2024, p. 1
[85] cf. Hinton, G. E. 2012b, pp. 4–5

would have to generate a long chain of samples, until independent samples are obtained from the observed data distribution of the model.[86] The samples are needed for each iteration of the gradient ascent on the log-likelihood resulting in large computational costs.[87] To solve this issue contrastive divergence minimizes an approximation of the Kullback-Leibler divergence between the empirical distribution of the training data and the distribution generated by the model.[88] They way to achieve this is by initializing the Markov chain with the samples from the data distributon.[89] The outcome has been shown to heavily decrease the training time while only adding a small bias.[90] This allows to calculate the probabilities of equation 2.5. This entails initializing the visible units using an actual data input, such as an MNIST sample, and then commencing the subsequent steps with the hidden states. Often the process can be stopped after only sampling a very small number of steps.[91]

1. Forward Pass (positive phase)

During the forward pass using the Gibbs Sampling method, the visible units are set to a completely random state. Next up the hidden units are computed. The computation of the hidden units involves calculating their activation probabilities and performing an actual sampling with their calculated activation probabilities. With the RBM it is now easy to get an analytical calculated sample of $(\mathbf{v}_i\mathbf{h}_j)_{data}$.[92] Given an input data out of the training images, v, the binary state, j, of each hidden unit, h_j, is set to 1 with following probability:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij}), \tag{2.7}$$

where $\sigma(x)$ is the logistic sigmoid function with an unbiased sample. The sigmoid function is defined as $\sigma(x) = \frac{1}{1+\exp(-x)}$ and is needed because it is the underlying activation function of each neuron. A visual representation is shown in figure 6:

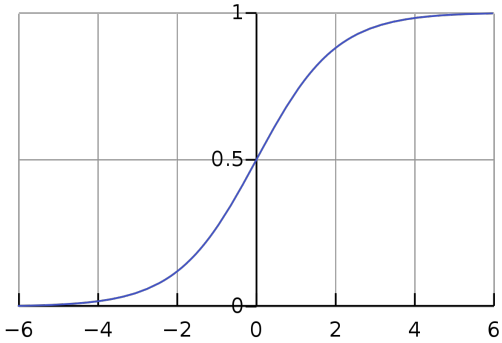

Fig. 6: Figure of a logistic sigmoid function RBM

[86] cf. Huembeli et al. 2022, pp. 5–6
[87] cf. Upadhya/Sastry 2019, pp. 7–8
[88] cf. Mocanu et al. 2016, p. 246
[89] cf. Upadhya/Sastry 2019, pp. 7–8
[90] cf. Larochelle/Bengio 2008, p. 537
[91] cf. Larochelle/Mandel, et al. 2012, p. 646
[92] cf. Hinton, G. E. 2012b, p. 5

The result is a set of probabilities that reflect how likely it is for each hidden unit to be on, wich stands for 1, or, which stands for 0, given the input data.[93] The sampling part of the positive phase uses the just calculated activation probabiliy of each hidden unit and performs a random experiment with it. That random experiment generates a uniform random number between 1 and 0 and if the random number is greater than the just calculated activiation probability the hidden unit is set to activated. Afterwards, the hidden unit is either activated or not activated and the training process continues with the new state of the hidden units.

2. Reconstruction (negative phase)

In this phase, the sampled hidden states are used to reconstruct the visible units. This is essentially a prediction of the input, which is how the model sees the input based on the just updated hidden units and is calulated with following probability:[94]

$$p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{2.8}$$

The sampling part of the negative phase uses the just calculated activation probabiliy of each visible unit and performs a random experiment, like in the positive phase. Now, the result is a prediction of the input in the visible nodes. Afterwards, a half forward pass is made to calculate the activation probability of a hidden unit again based on the activated or not activated visible units.

3. Updating the weights

Now, all the requirements to update the weights are satisfied and can be used within the equation 2.5. The delta that results is summed to the current weight and the internal model gets closer to prediciting the observed data. In total, one training iteration consists out of 1 Forward Pass, 1 Reconstruction and 0.5 Forward Pass again is accomplished. Repeating this training steps N times for a suitable chosen N the model learns better, since more steps of alternating Gibbs sampling were performed.[95]

Metropolis-Hastings: The Metropolis-Hastings algorithm, often only called Metropolis algorithm, is a technique out of Markov chain Monte Carlo (MCMC) class techniques.[96] The Metropolis-Hastings method was invented by Metropolis et al. in 1953 when they noticed, that for an intractable distribution with too many states it can be seen as a limiting distribution of Markov chains.[97] The intractable distribution to handle with the Metropolis-Hastings technique in the case of RBMs is equation 2.3. An Interpretation of the method can be expressed as: "A visitor to a museum that is forced by a general blackout to watch a painting with a small torch. Due to the narrow beam of the torch, the person cannot get a global view of the painting but

[93] cf. Huembeli et al. 2022, p. 6
[94] cf. Hinton, G. E. 2012b, p. 6
[95] cf. Huembeli et al. 2022, p. 6
[96] cf. Patrón et al. 2024, p. 1
[97] cf. Metropolis et al. 1953, pp. 1087–1092

can proceed along this painting until all parts have been seen."[98] The version already adjusted for RBMs incorporates the following functionality of the Metropolis technique:

First, select a random or given configuarion x_{old} of a RBM that holds the states of all visible and hidden neurons.[99] Secondly, the energy of the configuration, noted as E_{old}, must be calculated using Equation 2.6, as previously introduced. Subsequently, this energy value is stored. Thirdly, the configuration gets updated by picking one random neuron and changing the state of it from 0 to 1 or vice versa.[100] This new configuration is stored as x_{new}. Following that the energy of the new configuration E_{new} is calculated and stored. Now the two energy values are compared and if $E_{\text{new}} <= E_{\text{old}}$ the new configuration will be accepted and $x_{\text{old}} = x_{\text{new}}$.[101] If $E_{\text{new}} > E_{\text{old}}$ then there are some extra steps to be followed:

The flip probability is calculated as $p = \exp\left(-\frac{E_{\text{new}} - E_{\text{old}}}{kT}\right)$. KT is interpreted as the temperature in the network and with higher temperature it increases the activation probability leading to an faster exploration through the landscape but with less details.[102] For RBMs KT is assumed to be 1.[103] In the following figure 7 the resulting probability function is shown.

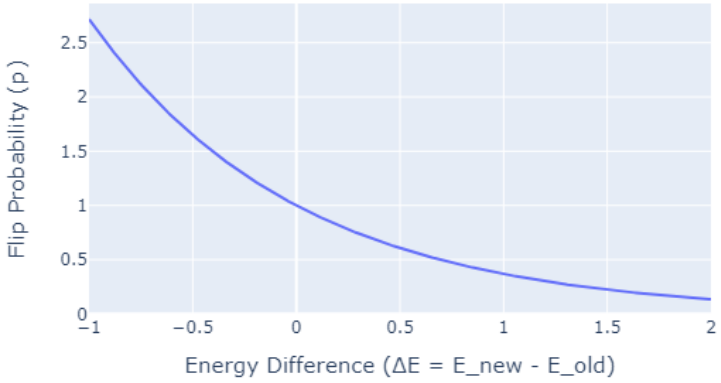

Fig. 7: Flip Probability Function in Metropolis-Hastings Algorithm

In the next step a uniform random number r between 0 and 1 is generated. After generating r the configuration will be accepted if $r \leq p$ (i.e., $x_{\text{old}} = x_{\text{new}}$).[104] Otherwise, a rejection takes place if $r > P$ (i.e., $x_{\text{old}} = x_{\text{old}}$).

[98] cf. Robert 2016, p. 2
[99] cf. Beichl/Sullivan 2000, p. 65
[100] cf. Rosenthal 2009, p. 1
[101] cf. Patrón et al. 2024, pp. 1–2
[102] cf. Li et al. 2016, pp. 1–9
[103] cf. Hinton, G. 2014, p. 3
[104] cf. Patrón et al. 2024, pp. 2–3

Finally, the configuration x_{old} is stored and the process repeats beginning with step 2.[105] After repeating enough times the activation probability for each neuron is calculated by summing over all samples ($x_1 + x_2 + x_3 + \ldots$) and the result is divided by the total number of samples.

2.1.4 Current Problems witht BMs and RBMs

One general problem that occurs in the learning process of a BM is that it is both time-consuming and difficult.[106] This is because sampling from an undirected graphical model is not straightforward and therefore RBMs make use of MCMC proposed methods like Contrastive Divergence and Metropolis Hastings.[107] In addition to that, the selection of hyperparameters can be difficult since for the training of a practical model a large hyper-parameter space needs to be explored.[108] Hyperparmeters are: the learning rate, size of the hidden layer, number of training iterations iteration count per bias (sampling step size), initializing the weight sizes in the beginning but also the method for calculating activation probabilities (Contrastive Divergence, Metropolis Hastings, etc.). As a result, establishing a RBM with perfect hyperparameters is time consuming and can be seen as art. Furthermore, training can become unstable and predictions become inaccurate due to an incompatible selected temperature.[109] A lower temperature reduces the system's possbility to explore the energy landscape thoroughly, leading to the false selection of local minima instead of finding the global minimum. Vice versa a too high temperature can cause that the energy landscape is not explored enough and has gaps between it missing some minima or skipping the global maxima. Luckily, the temperature for RBMs is expected to be 1 and only for specific use cases it makes sense to adjust internal temperature.

To accelerate the training process of a BM or RBM, it is crucial to address the most computationally demanding aspect: the matrix-vector multiplication involved in the sampling process. A possbilty of achieving this is using dedicated hardware, so called hardware accelerators for this problem. They are designed to tackle a specific task very efficiently, like matrix vector multiplications, which are widely used within most of the neural networks.[110] That is the reason why they are significant for the acceleration of this thesis and an interesting technology to look at.

[105] cf. Patrón et al. 2024, p. 17
[106] cf. Fischer/Igel 2012, pp. 1–2
[107] cf. Fischer/Igel 2012, p. 2
[108] cf. Larochelle/Bengio 2008, p. 536
[109] cf. Huembeli et al. 2022, pp. 3–4
[110] cf. Lehnert et al. 2023, pp. 3881–3882

2.2 Hardware accelerators

2.2.1 Current approaches in the field of AI and other solutions

Since Neural Networks and DNNs are growing their parameters at rapid rates they constantly achieve better and better resutls and are able to solv even more complex tasks.[111] This upcoming trend of growing network sizes exponentially also brings a dark side with it: An excessive increase in computational effort and memory size.[112] As a result Central Processing Unit (CPU)s can barely satisfy the required performance and specialized hardware accelerators are used to increase the performance of these Neural Networks.[113] In addition to that for many use cases, like autonomous driving, there are high energy, latency, and runtime predictability constraints CPUs are not able to meet.[114]

The concept of developing hardware accelerators is not new. However, their limited adoption and the fast obsolescence of even the most outstanding accelerators have made the investment in them uneconomical compared to general-purpose processors that surpassed them.[115] At present, however they are seen as promising driving force of computer architecture since they are the optimal solution to satisfy the growing computation-hungry demands of businesses, especially within machine learning workloads.[116] An hardware accelerator can be defined as "a separate architectural substructure that is architected using a different set of objectives than the base processor, where these objectives are derived from the needs of a special class of applications".[117] Broken down, they are specialized hardware, expertly optimized for the unique demands of certain application categories, freeing them from the restrictions usually placed on general-purpose processors. Moreover, a hardware accelerator doesn't replace the conventional processor, which is still used for the operating system, it rather enables specific workloads to be executed on it very efficiently.[118] There are different approches such as GPUs, ASICs, Field Programmable Gate Array (FPGA)s, but also new approaches like Quantum Computations or Photonic matrix multiplication are researched.[119] All of these methods have different use cases and get more and more application specific. The list of the sequence, sorted by application-unspecific to specific for established approches looks the following:

$$\text{CPU} \xrightarrow{\text{less flexible}} \text{GPU} \xrightarrow{\text{specialized}} \text{FPGA} \xrightarrow{\text{more customized}} \text{ASIC}$$

Currently the approaches can be segmented into three categories: **Firstly**, the design of data-driven digital circuits. It consists the shift from general-purpose GPUs to specialized dataflow

[111] cf. Baischer/Wess/TaheriNejad 2021, p. 1
[112] cf. Baischer/Wess/TaheriNejad 2021, pp. 1–2
[113] cf.Zhou et al. 2022, p. 1-2; cf.Baischer/Wess/TaheriNejad 2021, p. 2
[114] cf. Ahmad/Pasha 2020, p. 2692
[115] cf. Peccerillo et al. 2022, p. 2
[116] cf. Peccerillo et al. 2022, pp. 2–3
[117] cf. Peccerillo et al. 2022, p. 2
[118] cf. Peccerillo et al. 2022, pp. 2–3
[119] Zhou et al. 2022, p. 1-2; cf.Baischer/Wess/TaheriNejad 2021, p. 2

architectures like systolic arrays, which are used in Google's Tensor Processing Units (TPUs). These architectures are noted for their efficiency in performing deep learning operations by reducing control hardware and keeping data movement local.[120] **Secondly**, network structure optimizations. Hereby modifications to the neural networks themselves are made to improve hardware efficiency. One method is quantization, which simplifies arithmetic operations and reduces memory needs by using fixed-point representations of data and weights instad of using for example 32-bit floating points. The other one is pruning, which involves setting certain weights to zero to reduce the complexity of operations.[121] **Thirdly**, technology-driven designs. Current research into using novel circuitry and memory cells include memristive memory cells and silicon photonics, to further enhance performance and energy efficiency. They work by storing the network weights and calculating the vector multiplications with analaog signals with technologies like crossbar arrays. While these technologies promise significant advantages, their practical application is still being explored.[122] The following three accelerator approaches can be categorized into the first category:

2.2.2 GPU

The GPU, is by far the most common accelerator in the market with a focus at computational-complex workloads. A GPU is a manycore unit that features up to hundreds of multi-processors that consist of in-order cores which are able to exploit massive threadlevel parallelism.[123] They excel at performing numerous floating-point arithmetic operations for vector processing on large datasets with high degrees of data-parallelism. In practice this works by breaking down workloads into small tasks that can be processed by the enormous amount of cores in parallel.[124] The combination of programmability and floating-point performance make them very attractice for machine learning workloads and is the reson for their dominance in the market.[125] On top of that, the widespread adoption of GPUs has led to extensive support across numerous frameworks and high-level APIs commonly used in Machine Learning.[126] Well-known frameworks would be PyTorch or TensorFlow. However, compared to more specialized FPGA and ASIC apporaches the GPU is not as flexible and has higher latency and energy consumption.[127]

2.2.3 Field programmable gate arrays

In contrast, FPGAs have also demonstrated enormous parallelization capabilities due to their fast digital signal processors and on-chip memory which result in lower energy cost than GPUs.[128]

[120] cf. Lehnert et al. 2023, p. 3883
[121] cf. Lehnert et al. 2023, p. 3883
[122] cf. Lehnert et al. 2023, p. 3883
[123] cf. Peccerillo et al. 2022, p. 2
[124] cf. Hu/Liu, Y./Liu, Z. 2022, p. 101
[125] cf. Dally/Keckler/Kirk 2021, p. 42
[126] cf. Baischer/Wess/TaheriNejad 2021, p. 16
[127] cf. Hu/Liu, Y./Liu, Z. 2022, p. 100
[128] cf. Ahmad/Pasha 2020, p. 2693

They work by using reconfigurable logic blocks that can be interconnected using routing tracks with configurable switches at the intersections.[129] This combined with the use of many digital signal processors and local storage of data in the hardware enables the development of custom digital circuits for a specific workload.[130] As a side note it is worth mentioning that the most energy-consuming task of a workload is often the data transfer and not the computation iself. In the context of FPGAs, they use their on-chip memory to reduce the data transfer significantly and therefore achieve a sweet spot between computation speed and energy efficiency.[131] Hence, they are utilized to design a specialized processor tailored for executing specific workloads, like machine learning, effectively.[132] Furthermore, due to their reprogrammable nature they have a lower engineering cost and faster time-to-market compared to ASICs.[133] With FPGAs the implementation time could only be a matter of weeks and also allows to support continious upgrades and bug fixes even after the deployment which is not possible within ASICs[134] Even though the FPGA possesses all these advantages with their high flexibility, latency and low energy consumption, they are sometimes inferior in throughput compared to a GPU.[135]

2.2.4 Application specific integrated circuit

ASICs can be distinguished from FPGAs because they are not programmable and have no limited amount of building blocks. In addition to that, they offer the highest degree of customization and are designed to execute a specific application with the utmost efficiency.[136] They also have the possibility to implement analog circuits. Nowadays, a good example of ASICs are TPUs because of their matrix vector multiplications abilities that are needed within machine learning. Conventional ASICs work by mapping neurons directly to the hardware.[137] Their design architecture enables them to outperform GPUs and FPGAs in terms of their small size, greater computation speed and high power efficiency.[138] Specifically when compared to corresponding FPGA circuits ASICs a study shows that they are 9x smaller and also around 4x faster.[139] Nonetheless, developing an ASIC requires expert knowledge in chip design but also to implement neural networks on them, which both takes a lot of time.[140] Out of the three approaches, they often provide the most efficient solution, yet it comes at the expense of lacking reconfigurability and incurring high engineering costs.[141] With a sustainability and climate-change aspect in mind, they are a promising option since they represent the most power efficient approach with the best computation speed.

[129] cf. Babu/Parthasarathy 2021, p. 144
[130] cf. Baischer/Wess/TaheriNejad 2021, p. 19
[131] cf. Hu/Liu, Y./Liu, Z. 2022, pp. 101–102
[132] cf. Sipola et al. 2022, p. 322
[133] cf. Boutros/Betz 2021, p. 4
[134] cf. Boutros/Betz 2021, p. 4
[135] cf. Hu/Liu, Y./Liu, Z. 2022, p. 100
[136] cf. Baischer/Wess/TaheriNejad 2021, p. 17
[137] cf. Hu/Liu, Y./Liu, Z. 2022, p. 104
[138] cf. Baischer/Wess/TaheriNejad 2021, p. 17
[139] cf. Boutros/Betz 2021, p. 5
[140] cf. Baischer/Wess/TaheriNejad 2021, p. 17
[141] cf. Peccerillo et al. 2022, p. 4; cf.Hu/Liu, Y./Liu, Z. 2022, p. 100

2.3 Memristor Hopfield Neural Network

The so called mem-HNN is a a hardware accelerator that uses an emerging approach of combining analog signals and electronical signals to solve complex optimization problems.[142] It can be categorized into the ASIC family of hardware accelerators and its specific purpose is to solve Ising problems. A photograph of the physical mem-HNN accelerator (left side) and a microscopic view of it (right side) can be seen in following figure:

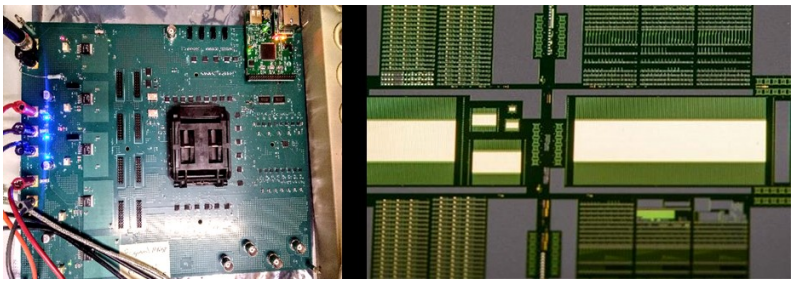

Fig. 8: Physical and microscopic view of the mem-HNN

In 1925 Ernst Ising, a german physicist, invented the Ising model which explained the interaction between ferromagnets.[143] This statistical model focuses on the state of a spin s_i (up and down; $+1$ and -1) representing the orientation of the magnetic moment. The Ising model calculates the total energy of a system though the following energy function:

$$E(\mathbf{s}) = \sum_i h_i s_i + \sum_{i,j} J_{ij} s_i s_j, \quad s_i = \pm 1, \tag{2.9}$$

where i is the label of the spins s_i, with h_i representing the external magetic field interacting with each spin, and J_{ij} is the interaction strength between pairs of spins that are connected by an edge (ij).[144] Both values h_i and J_{ij} are real-valued allowing for a wide range of possible magnetic field intensities and vary in interaction strength.[145] The Ising model is also attractive in other fields to describe the energy of a system and to transform them into an Ising problem.[146] Solving Ising problems is equal to finding the minimum energy state of a system. Hence, in practice transforming optimzation probelms into Ising problems, the optimal solution is equal to the minima of the Ising energy function. This transformation works by mapping each variable of the problem to Ising spins and designing an Ising model whose ground state represents the optimal solution.[147]

The background for the development is the current slow down or failure of Moore's law which causes slow improving computation speed, energy efficiency and computation latency of conven-

[142] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[143] cf. Ising 1925, pp. 253–258
[144] cf. Tanahashi et al. 2019, p. 2
[145] cf. Wang, T./Roychowdhury 2017, pp. 1–2
[146] cf. Tanahashi et al. 2019, pp. 2–3
[147] cf. Lucas 2014, pp. 2–3

tional semiconductor electronical technology.[148] Since the mem-HNN is engineered to solve Ising problems, therefore also called Ising machine, it can tackle various problems that fall under the category of Ising problems.[149] Originally the mem-HNN was experimental tested by the team of researches to solve nondeterministic polynomial-time hard, or NP-hard for short, Ising problems directly on the hardware.[150] NP-hard problems are amoung the toughest problems to solve and have an expontential- or even factioral time to solve (2^n, $n!$) with no efficient solution, slow to solve and to verify.[151] Well-known examples would be the travelling salesman problem, the maximum clique problem. Here, the mem-HNN outperforms both digital computer accelerators CPU and GPU by at least 10.000x in terms of energy to solution.[152]

Equivalence between the energy function of a mem-HNN and the energy function of a BM can be shown here. The energy function of the mem-HNN works by using the binary states of 1 and 0 while the BM can use $+1$ and -1 but otherwise they are completely equal. To transform the RBM into the binary states of the mem-HNN, its energy function from 2.6 needs to be modified with $\frac{x+1}{2}$ where x represents the state of the spin. The fact, that both energy functions are equal implies that the neural network of a RBM can possibly be trained on this Ising machine. Therefore, this thesis aims to develop tools that utilize mem-HNNs for implementing BMs, capitalizing on their proven efficiency in computing the Ising model with notable speed and energy savings. This will investigate the specific benefits that a mem-HNNs accelerator offers for BMs. The name mem-HNN already indicates that the Ising machine is based on the concept of a Hopfield Neural Network. All this is possible beause the update formula of the Hopfield Network is directly implemented on the hardware of the accelerator.

2.3.1 Hopfield Network

A Hopfield Network is a type of EBM and belongs to the field of recurrent neural networks.[153] The main purpose of the mem-HNN is to implement Hopfield networks in a hardware accelerator, which is used to find optimal solutions of optimization problems. Hence, the operating principle of Hopfield Networks is explained. The structure of the network consists of only one single layer with binary valued neurons.[154] The neurons state can either be $\{1, 0\}$ or $\{1, -1\}$. The connections between the neurons are symmetrical, which means that the weights of the connections are the same in either direction.[155] Initially, the primary applications of this type of network were to serve as storage for associative patterns and to facilitate pattern retrieval.[156] In practive given a query pattern a Hopfield Network can retrive a pattern that is most similar or an is an average of

[148] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 1
[149] cf. Mohseni, N./McMahon/Byrnes 2022b, p. 363
[150] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[151] cf. Izadkhah 2022, pp. 497–500
[152] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 470
[153] cf. Dramsch 2020, p. 35
[154] cf. Ahad/Qadir/Ahsan 2016, p. 7
[155] cf. Ahad/Qadir/Ahsan 2016, p. 7
[156] cf. Ramsauer et al. 2021, p. 2

similar patterns.[157] Since Hopfield networks were introduced by J.J Hopfield in 1982 the storage capacity got increased over time but the fundamentals stayed the same.[158] In following figure 6 an example of a Hopfield Network can be seen.[159]

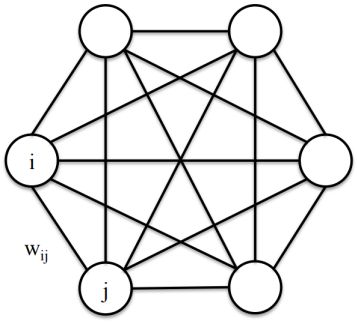

Fig. 9: Figure of a hopfield network

The exemplary network has 6 neurons and bidirectional weigths W_{ij} between the neurons. In addition to that, a Hopfield network has no input or output layer.[160] The main goal is to find the values for each neuron in the network given a specific input that minimizes the total energy of the system.[161] Hereby, the input of the network is an initial neuron configuration Given an initial input configuration, the neurons will then evolve from this configuration such that the overall energy of the Hopfield network is minimized.[162] This energy state can be calculated with the following energy equation[163]:

$$E = -\frac{1}{2}\sum_{i \neq j} T_{ij} V_i V_j. \tag{2.10}$$

This energy function invented by Hopfield has big similarities with a BM when comparing to the equation 2.4. This is one of the reasons why the execution on the mem-HNN could work out. Approximately, the activation rule for each neuron is to update its state as if it were a single neuron with the threshold activation function. The energy is minimized in an iterative update process. The state of each neuron is updated according to the following rule:[164]

$$s_i \leftarrow \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j + b \geq \theta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

The state of the neuron will be updated to 1 if the sum over all weights multiplied with the states $\{0, 1\}$ added to a bias b is greater than the threshold θ_i. In the case of our accelerator the threshold is set to 0 but in theory can be used as an hyperparameter.

[157] cf. Ramsauer et al. 2021, p. 2
[158] cf.Hopfield 1982, p. 2554-2558; cf.Ramsauer et al. 2021, p. 2
[159] cf. Yao/Gripon/Rabbat 2013, pp. 1–2
[160] cf. Yao/Gripon/Rabbat 2013, p. 3
[161] cf. Ahad/Qadir/Ahsan 2016, p. 7
[162] cf. Ahad/Qadir/Ahsan 2016, p. 7
[163] cf. Hopfield 1982, p. 2556
[164] cf. MacKay 2003, p. 506

Since every neuron's output is an input to all the other neurons the order of the updates need to be specified.[165] There is the possbility to update all neurons synchronous or asynchronous. In general, it can be highly problem specific which update mechanism performs bettert to achieve faster convergence to energy minima. The specific choice of the update method for the intended use is described later. When comparing a Hopfield Network, they seek to achieve the effect of changing node activation on the overall energy of the network but BMs replace this with the probability of a certain node being activated on the network energy.[166] The second important reason to research the hopfield networks is for their updating process because it possibly could be used to sample the intractable training of a RBM mentioned earlier.

2.3.2 Memristor Crossbar Array

Having set the foundational knowledge about the function of a Hopfield Network the mode of operation is explained in the following. Since the mem-HNN saw the light of day in 2021, a number of improvements have been made to it and at the end of 2023 the individual components can be seen in following figure[167]:

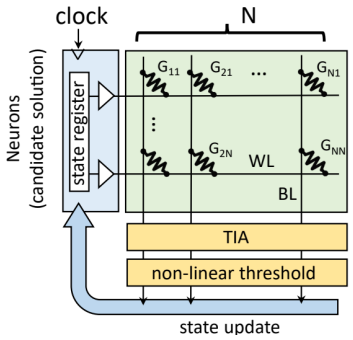

Fig. 10: Modell of the mem-HNN

The green square symbolizes the memristor crossbar array, which has the task of performing the matrix multiplication in eq.(2.11). The memristor crossbar array consists of **memristors** that connect orthogonal **electric tracks** with each other. The G_{ij} stands for conductance and represents the inverse of the resistance R of the memristors since $G = \frac{1}{R}$. BL (Bitline) and WL (Wordline) represent the electical tracks, where the WL is the input and the BL is the output of the crossbar. The other components of the model are explained in subchapter 2.4.3, as for now the focus in on the memristor crossbar array (green square). A better perspective of the memristor crossbar array gives following 3D model[168]:

[165] cf. MacKay 2003, p. 506
[166] cf. Ahad/Qadir/Ahsan 2016, p. 7
[167] cf. Hizzani et al. 2023, p. 2
[168] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410

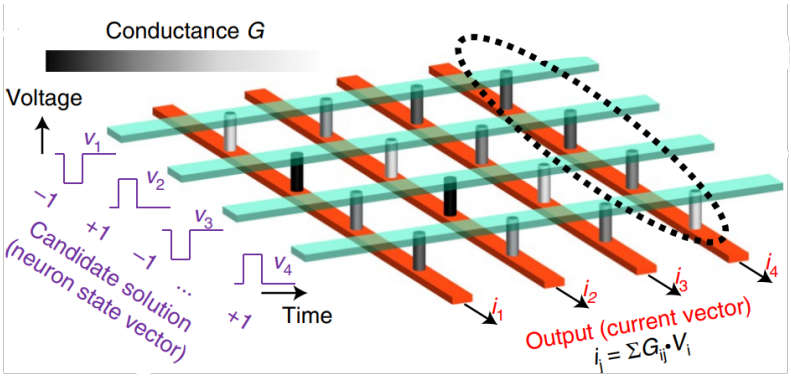

Fig. 11: 3D modell of the memristor crossbar array

To understand how this crossbar array implements the Hopfield network, it helps to have another look at the earlier introduced update formula in eq.(2.11). On the green Wordline a voltage which either is 1 or 0 that represent the state of the neurons in the hopfield network is applied. Then, the current is flowing towards the BL trough the memristors, which is an electronic device that functions as a resistor.

Unlike a resistor with fixed electrical resistance, a memristor's resistance can be programmed to change, as shown by the gray cylinders in fig.10.[169] Higher resistance in a memristor results in lower conductance, restricting current flow into the lower Bitlane. At each intersection where a Wordline meets the crossbar, it represents the multiplication of w_{ij} and s_j in the update formula. Following this, currents from different Wordlines are combined in the Bitlane, summing all currents according to Kirchhoff's first law, as demonstrated in the example crossbar in fig.10. The crossbar has the four input volatages i_1 has the four intput voltages V_1, V_2, V_3, V_4 that are applied at the WL. The current flowing into the lower Bitlane is now calculated according to Ohm's law $i_{out} = \frac{1}{R_{memristor}} * V_{in}$. Since $G = \frac{1}{R}$ and as the currents flowing into a BL are added, the output current at BL1 is $i_1 = G_{11} * V_1 + G_{12} * V_2 + \cdots + G_{14} * V_4$ As a result the summation of $\sum_j w_{ij} s_j$ can be performed by the flow of electical currents. Adding the bias b to the sum is achieved by simply adding an intial current, which is worth the bias amount, to the the total Bitline current. Analog signal processing enables vector-matrix multiplication at light speed, significantly speeding up clock cycles, making it ideal for accelerating AI workloads.

Another advantage of crossbar arrays is the ability to store the matrix weights directly in the crossbar, which avoid slow and energy intensive data movement. Memristors function like plate capacitors, dynamically changing resistance, as depicted in the following figure.[170]

[169] cf Sung/Hwang/Yoo 2018, p. 124
[170] Chang et al. 2017, p. 6; Sung/Hwang/Yoo 2018, p. 2

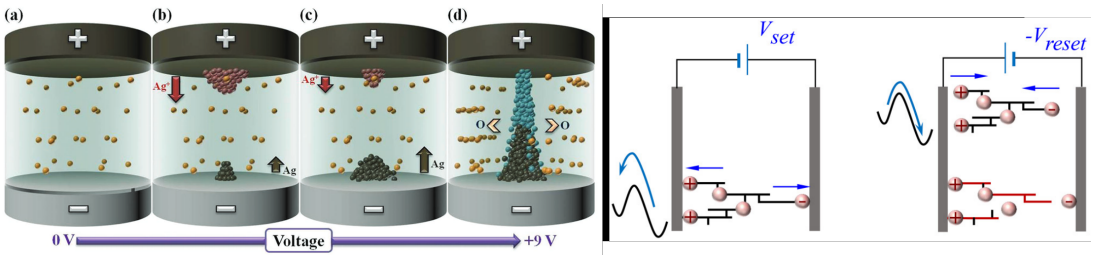

Fig. 12: Memristor-based learning

Essentially, the memristor consists out of two metal electrodes and a dielectric layer that is sandwiched between them.[171] In case of the mem-HNN it is an oxide memristors that uses tantalum oxide (TaOx) and oxygen atoms as dielectric layer.[172] Initially the metal-ions are freely moving within the dielectric layer. A visual representation can be observed in section a) on the left side of 12. In **phase 1**, the programming phase, an electrical field is applied to the plate capacitors which leads to the formation of a conductive filament within the dielectrive layer.[173] The conductive filament can be imagined as a string, which is formed out of the metal-ions. Therefore, they are able to conduct electricity once they have formed.[174] This process, visualized from $a) - d)$, can be controlled due to the voltage that is aplied to the memristor. Wigh a high enough voltage a filament can be established and increasing the voltage from there on ensures that even stronger and thicker filaments are created. The thickness of the filament determines the conductivity, so that the resistance of the memristor can be controlled by this filament growth. A schematic view of the filament and phase 1 is also shown in the right part of 12, where the label is V_{set} An everyday example is a water pipe with a valve that controls the water flow. The vavle symbolizes the memristor and the water flow the current in the electrical tracks. On the actual mem-HNN there is a controller, which is not shown in the model, that talks to a digital external computer that gives the inforation on hot to chose V_{set} for each memristor.

In **phase 2**, the performing phase, the electrical field is removed. Now, the filament has reached its final form and is not able to grow anymore but most importantly it stays the same.[175] The filament connects the top and bottom metal electrodes of the memristor with each other. The enduring presence of a filament in the memristor, even without an applied voltage, embodies its namesake combination of memory and resistor. As a result in this phase the workload can be executed and the memristor has the desired conductance.

The final **phase 3** is the dissolution of the filament to readjust the resistance. This process is called bipolar switching and a schematic view of it is shown in the right part of 12, where the label is $-V_{reset}$. Filament disonnection is performed through ionic switching, which works by swapping the plus and minus poles around. Next up, the filament wants to rearrange and

[171] cf. Chang et al. 2017, p. 1
[172] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 412
[173] cf. Chang et al. 2017, p. 3
[174] cf. Chang et al. 2017, p. 5
[175] cf. Sung/Hwang/Yoo 2018, pp. 1–2

disconnects from top and bottom of the metal electrodes. Since this process of establishing the filaments (setting the desired resistance) and disconnecting them (preparing for new desired resistanc) this process is called bipolar switching.[176]

Research on memristor crossbar arrays in supervised learning is currently limited, underscoring the need for more studies to assess their potential and usability.[177] In practice, training data for a BM is generated to test feasibility. Memristors are used by setting the resistance V_{set} for one training iteration with the necessary weights, resetting them $-V_{reset}$, and then digitally updating to new weights for subsequent training. This approach enables high switching speeds, energy efficiency, and durability.[178]

2.3.3 Output Hopfield Network

In this subchapter the other components of the mem-HNN shown in 10 are adressed. At the output of the BLs are Transimpedance Amplifier (TIA)s. The TIA is the component that converts the current i_j, which is the output of the memristor crossbar array, into a voltage.[179] Here, each BL has an individual TIA. Subsequently, to implement the update formula of the Hopfield Network in eq.(2.11), is a nonlinear threshold function on the output of the TIAs. The non-linear threshold is used to compare the $\sum_j w_{ij}s_j + b$ against the threshold θ_i to determine whether it is $\geq$ or $<$.[180] In terms of electrical components the non-linear threshold is a comparator. A voltage comparator is an analog electronical device. Comparators are able to compare a input signal, which is the converted voltage of the TIA, with a reference voltage, which is the threshold θ_i.[181] Also, the comparator is the component that transforms the analog voltage into a binary digital signal. Now, the digital signal is a binary voltage and either is $0V$ or if the sum was greater than the threshold it is a specific voltage V_{out}. The output represents the new state of a neuron in terms of the Hopfield Network and is now trasmitted to the state register.

The state register is a digital memory that is designed to store the current neuron configuarion (input vector).[182] The binary states of the neurons, which represent the voltage output of the comparator, are sent to the state register and update the old configuration.[183] For each neuron there is one TIA and an according comparator required. This not only allows for fast parallel computation but also it allows to exactly map the digital output of the comparator to the correct position within the state register.

A missing component part in figure10 is a selector that is connected to the state register. Its task is to select specific register states which can the be updated by the output signal of the

[176] cf. Sung/Hwang/Yoo 2018, p. 7
[177] cf.Amirsoleimani et al. 2020, p. 8; cf.Sung/Hwang/Yoo 2018, p. 124
[178] cf. Amirsoleimani et al. 2020, p. 3
[179] cf. Hizzani et al. 2023, p. 3
[180] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[181] cf. Chen/Zhang, M./Shen 2021, p. 28
[182] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[183] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 3

corresponding comparator. For example the mode of updating can be selected with either one neuron updated at a time asynchronously or multiple neurons synchronous. Currently, the memHNN offers two update strategies: the first updates a single random state, while the $N/2$ update randomly selects and updates about half of the neurons.[184] The white arrows next to the state register in figure10 represent a Wordline driver. Wordline Drivers are a voltage source that determines the voltage by the state of the state register ans supply a voltage to the crossbar's WLs.[185]

Notably, the mem-HNN can perform a full iteration within a single clock cycle. Thousands of sampling iterations occur within a single neural network training iteration. After each, the neuron configuration from the state register is saved on the ASIC hardware's cache and sent to an external digital computer. For instance, after 10.000 sampling iterations, the arrays of hidden and visible neurons are sent to the computer for prediction. This process illustrates how the mem-HNN implements a Hopfield Network concept.

2.3.4 Noisy Hopfield Network

Currently, the mem-HNN is also able to use noise injection to ensure the chance of finding a low energy minima of the Ising problem. This noise injection happens between the output of the Bitline and the TIA. The noise enables to escape local minmina of the energy landscape and to find lower energy minimas or even the global minima, which is equal to the solution of the optimization problem and therefore improves solution quality anf effciency.[186] This is achieved by a random number generator in the hardware that creates a random array filled with digital signals.[187] Out of this array a Digital Analog Converter (DAC) takes a subset of this array and converts them into a floating point noise signal for each neuron.[188] This noise injection uses the created floating point noise signal and adds it to the update formula. Effectively the new noisy hopfield network updating function now looks like the following:

$$s_i \leftarrow \begin{cases} 1 & \text{if } \sum_j w_{ij} + b + \mathbf{n} \geq \theta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

with n_i representing the noise.[189] Besides aiding optimization tasks, noise also creates an interesting link to BM̨s. The difference between the RBM and the Hopfield Network without noise is the activation function. As shown in equation 2.11, a simple Hopfield Network and has a binary step function as activation function which is completely deterministic. In contrast, a RBM has a statistical logistic sigmoid function as activation function shown in figure 6, which uses a temperature of 1 mentioned in chapter 2.2.4. Therefore, to successfully implement a RBM on

[184] cf. Hizzani et al. 2023, p. 3
[185] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[186] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[187] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 22
[188] cf. Hizzani et al. 2023, p. 3
[189] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410

the mem-HNN the activation behavior of the neurons need to be compatible with the activation function of the mem-HNN. A potential solution to address the issue of activation behavior involves utilizing noise from an analog noise source.[190] One relatively straightforward way to inject noise into the activation function is by adding a normal gaussian distribution $g(x)$ on top of it[191]:

$$f_g(x) = \frac{1}{\sqrt{2\pi\rho^2}} e^{-\frac{(x-\mu)^2}{2\rho^2}}, \tag{2.13}$$

with ρ representing the standard deviation and μ represents the mean of the distribuion. The visual representation of a gaussian distribution is shown in following figure. It illustrates distributions with different parameters of the standard deviation and the mean of the distribution allowing for a better understanding of the flexibility and possbilities of noise injection.

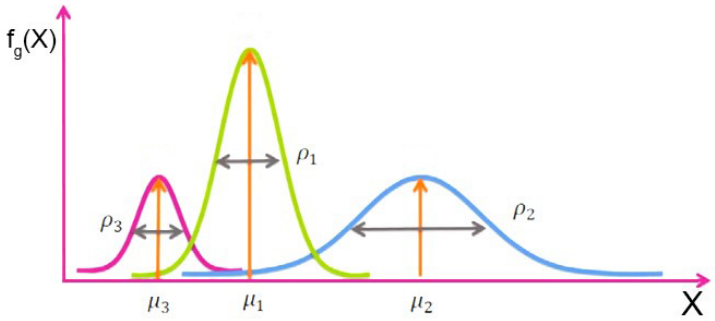

Fig. 13: Gaussian normal distribution[192]

There have been recent proposals that show a proof of concept on how to inject the noise with a gaussian distribution and make a RBM realizable.[193] In the paper by Böhm et al., they used analog singals for the neurons instead of digital signals like the mem-HNN uses. Furthermore, they used an opto-electrical Ising machine in combination with an FPGA as accelerator which works in a similar fashion to the mem-HNN.[194] The second paper by Mahmoodi/Preziosi and Strukov, also created a proof of concept in which they showed an implementation of a RBM on a memristor crosbar array. Crucially, there is no state registor or comparator involved, so all the calculations of comparing against a threshold, adding a bias and adding the noise were made by an external digital computer. The hardware computes solely the matrix vector multiplication. Hence, the additional required interfaces make the system slow and inefficient.[195]

However, beyond these initial proof of concepts, it is still an open question if the concept is feasible on the complete ASIC hardware accelerator like the mem-HNN and if it does bring an actual acceleration of the training and interference of a RBM. Therefore, the goal of this thesis is to study in detail the speed and energy consumption of memHNNs when implementing BMs.

[190] cf.Böhm et al. 2022a, p. 1-2; cf.Mahmoodi/Prezioso/Strukov 2019, p. 2
[191] cf. Böhm et al. 2022a, p. 3
[192] modified from cf. Gm et al. 2020, p. 3
[193] cf.Böhm et al. 2022a, p. 1-2; cf.Mahmoodi/Prezioso/Strukov 2019, p. 2
[194] cf. Böhm et al. 2022a, pp. 1–11
[195] cf. Mahmoodi/Prezioso/Strukov 2019, pp. 1–8

3 Objective specification and presentation of the research methodology

Having laid the groundwork with essential concepts necessary for this thesis, this chapter aims to outline the objectives of the practical segment as well as the research methodology employed to achieve them. In the first part, the specific objectives are defined. Afterwards, in a second part the used reasearch framework DSR and the two research methods Prototyping and the Simulation are explained.

3.1 Objective specification

The mentioned papers in 2.4.4 showed that the acceleration of statistical sampling in BMs theoretically work. In order to answer the research question, this thesis wants to validate these proof of concepts in a first part and see if they are feasible on the complete ASIC hardware accelerator like the mem-HNN. The second practical part goes further and has the goal of implementing a simulator pipeline with the workload of a BM and meassure if it brings an actual acceleration. Currently, such a simulation model does not exist and would be able to extract data for further research and fundamental insights. Key metrics identified in the literature that demonstrate the performance of accelerators include throughput (samples/second) and energy consumption (energy/operation).[196] Therefore, this is the goal of the data acquisition in the second practical part. The practical part is therefore simulating a current ASIC with not only the memristor crossbar array but also all the other requried hardware components. Furthermore, the thesis focuses on an actual training of a RBM and also the interference of it to answer the research question. Expanding upon the foundational work, this research explores the implementation of the N/2 synchronous update mechanism. This sampling method emphasizes a expactation of higher sampling speeds and efficiency.

At the beginning, the Information Technology (IT)-artifact to be implemented is modeled and all components, transitions and processes of the overall solution are identified. Upon this ideal overall solution a prototyp is developed that impelments a subset of the real hardware functionalities. As a result of the implementation a complete performance evaluation, including a comprehensive energy model and a latency model should be established. The following simulation aims to measure the metrics: required processing time (samples/sec) for data and the energy usage (energy/operation). The results should be compared to other sampling methods, such as Gibbs and Metropolis sampling. Lastly, this thesis performs hyperparameter tuning in oder to gather new data on how these machines should be configured for optimal performance. Since,

[196] cf.Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 409-418; cf.Ortega-Zamorano et al. 2016, p. 16-17; cf.Aadit/Mohseni, M./Camsari 2023, p. 2; cf.Belletti et al. 2009, p. 55

there is no data for hyperparameter tuning of such a concept in the literature yet this establishes foundational work on how to make use of artificial intelligence within such an accelerator. Through this process, the research seeks to determine the feasibility of a proper training with this setup, focusing on its practicality and efficiency. By benchmarking these aspects against traditional sampling methods, the thesis aims to underscore the potential of the mem-HNN in practical training of RBMs.

Hence, DSR is used as a research framework to iteratively create the IT-artifact. In addition to that, within the single iterations prototyping is used for the actual implementation of the RBM on the simulated mem-HNN with individual goals. The last iteration uses a simulation as research method because there the behaviour and performance of the system is meassured and the underlying model already finisehd with the last prototyping iteration. Since, the practical functionality can't be ensured the DSR process combined with prototyping and if successful a simulation brings flexibility and a problem-oriented structure which are emphasized for this new method.

3.2 Design Science Research

In terms of information systems DSR is a core research method within the field of business informatics that "creates and evaluates IT-artifacts intended to solve identified organizational problems".[197] Henver et al. established a DSR framework that has the goal of creating an IT-artifact which has the purpose of addressing and solving the important organizational problem.[198] This systematic DSR process lays a solid groundwork for conducting the research with rigor, offering a degree of confidence that the endeavor will yield meaningful outcomes.[199] Artifacts in DSR can be constructs, models, methods or instantiations.[200] In addition to that, Gregor and Hevner (2013) categorize the underlying IT-artifact based on their abstraction level and maturities. Hence, level 1 represents a specific, limited and less mature implementation of an artifact, level 2 are operational principles or architecture like constructs, methods or models, while level 3 represents a well-developed midrange design theory.[201] The development of the artifact is performed incrementally with specific goals for each iteration, which is beneficial for IT-artifacts that can be adjusted after every iteration.[202]

Henver et al. also introduced 7 guidelines that still today serve as framework for different DSR approaches. Arguably, the most important two guidelines are, that the research must create a viable artifact that in a next step is able to solve the organizational problem. Another important guideline is that the artifact needs to be rigorously evaluated in utility, quality and efficiency.[203]

[197] Hevner et al. 2004, p. 77
[198] Hevner et al. 2004, p. 82
[199] cf. Baskerville et al. 2018, p. 368
[200] Hevner et al. 2004, p. 77
[201] cf. Gregor/Hevner 2013, p. 342
[202] cf. Gregor/Hevner 2013, p. 343
[203] Hevner et al. 2004, p. 83

Thereupon Peffer et al. introduced a well-known DSR Process Model, which has 6 different phases: Identify problem & Motivate, Define Objectives of a solution, Design & Development, Demonstration, Evaluation and Communication.[204] Another interesting approach by Österle et. al is called design-oriented business informatics. This DSR method is used in this thesis for following reasons. His approach compresses the phases of Peffer et al. into a more compact model and also gives a more detailed explanation of each phase while still complying with the guidelines established by Henver et al..[205] On top of this promising framework they created a DSR model called consortial research. It addresses problems for collaborative research in terms of access to practical knowledge, rapid change and practical orientation and a lack of support for knowledge transfer.[206] Österle et al. aim to bridge the gap between the knowledge base of both science and practice, with a focus on evaluating and ensuring the reproducibility of research outcomes.[207] However, the individual phases of the research framework can also be implemented on their own and the best features of the research framework and especially the contents of the phases should be combined with its older framework of design-oriented business informatics. As a result following model showed in figure 14 is used:

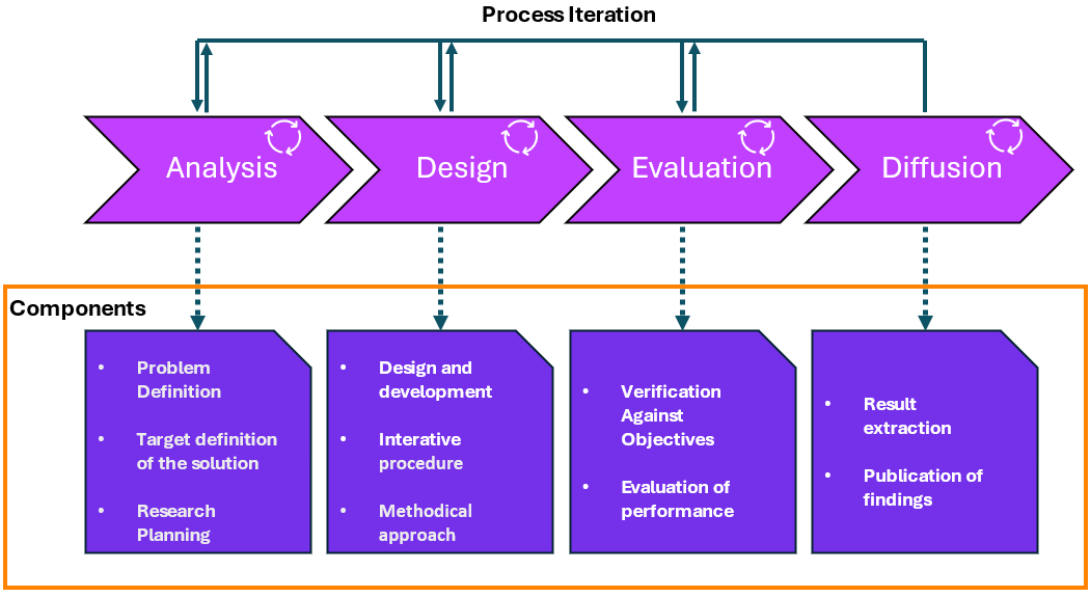

Fig. 14: DSR model by Österle et. al[208]

This model uses an iterative process for the phases that allow backwards steps to redo an already completed phase if requriements were not satisfied to an appropriate level. The four phases ideally contain the following contents:

Analysis: This phase identifies and describes the motivation in practice and formulates the

[204] cf. Peffers et al. 2007, p. 54
[205] cf. Oesterle et al. 2010, pp. 1–6
[206] cf. Österle/Otto 2010, pp. 273–274
[207] cf. Oesterle et al. 2010, p. 5
[208] inspired from cf. Österle/Otto 2010, p. 278

desired research objectives. In addition to that, a vague research plan is introduced which can hold goals of the project but also underlying constraints. Components of the research plan could be external stakeholders, funding, a timetable and of course a concept of the solution. If possible also a research method should be selected.[209]

Design: The IT-artifact is designed and developed with regard to the selected research methology. Specific processes must be justified and comprehensible with consideration to the outcomes of the analysis phase. The Design process can take multiple iterations on its own with the chance of making adjustments until the requirements. The outcomes is a functional IT-artifact that fulfills the set goals.[210]

Evaluation: Here the estasblished IT-artifact needs to be validated against the earlier specified goals. Furthermore, it can be validated against the chosen research method. This means they must be applicable and they must provide the expected benefit. If the artifact can not be testet with for example, an pilot application, it is poosible to persue expert interviews to validate the outcome.[211]

Diffusion: In this phase the results are generally made available to the public. Therefore, the results need to be prepared for publication for individual communities. Methods for publication could be teaching at universities and colleges and through their publication in books and specialist journals. Diffusion in practice also includes the implementation in companies and public administration which the solution was initially developed for.[212]

3.3 Prototyping

Given that the aim of the implementation is to create a new IT-artifact with a focus on rapid development, prototyping has been selected as the methodology. Generally, prototyping is a fundamental practice in designing tools, applications or user interfaces and defining requirements within the framework of agile software development. It belongs to the agile requirements engineering practice and allows to gather feedback on requirements in a light-weight fashion.[213] Prototypes are created to assist in the analysis and design of proposed systems. A prototype can be defined as "a simplified model of a proposed system, that is built for a specific purpose", which can apply to various kind of systems like software, hardware or even people.[214] It can be seen as an early increment, model, or release that implements some features of the desired product or model and therefore represents it.

At the core of prototyping it comes down to the exploration of the solution space through experimenting with ideas, collecting feedback and communicating product requirements in an

[209] cf. Oesterle et al. 2010, p. 4
[210] cf. Österle/Otto 2010, p. 279
[211] cf. Österle/Otto 2010, p. 279
[212] cf. Oesterle et al. 2010, p. 5
[213] cf. Bjarnason/Lang/Mjöberg 2021, p. 1
[214] Luqi/Steigerwald 1992, p. 470

iteratively detailing process. Hence, prototyping can deliver new requirements that are elicited through exploration and can later be validated by testing technical feasibility or business viability.[215] A few benefits of prototyping are early construction with low development costs and no large upfront investments of either time or money. In addition to that, it can promote innovation due to early results that can be communicated and if viable researched further.[216] The reason to chose prototyping can have various reasons. This thesis uses this method for the design phase within DSR due to the just named benefits and the possibility of fast results and testing feasibility of the model.

In specific G. Arthur Mihram's prototyping model is chosen because it suits well to the DSR framework and has overlaps with it. There are five steps to Mihram's prototyping process. The first step, setting the "modelling goals" is already completed with the analysis phase of the DSR analysis phase.[217] Within each iteration in the design phase a subselection of the goals are chosen to be prototyped. Furthermore, the previous established prototype is used as basis for the following iteration phase allowing to implement more and more features. In a second step "systemic analysis", the prototype can be categorized to set the prototypes behavioural mechanisms.[218] As a guideline to categorize this behaviour the thesis uses the House of Prototyping Guidelines by Ahmed and Demirel. These guidelines shown in 1 introduce four different dimensions used to categorize prototypes: Type of Prototype(1), Fidelity Level(2), Complexity(3), Scale(4) and Number of Iterations(5).[219]

The third step "model synthesis" requires a description and a chronological sequences of the prcoesses.[220] Furthermore, this is the phase of exploration and ends when the complete set of entities and the environment have been developed in a computer-directed language and the data is provided in machine-readable formats.[221]

The last steps of Mihram's model: "model confirmation" and "scientific interferenced" are not considered since they overlap with the DSR phases evaluation and diffusion. Simply this prevents a duplication of work. Therefore, the categorization of the prototype and afterwards the "model synthesis" is executed per iteration in the prototyping model used in this thesis.

3.4 Simulation

Simulation has been chosen as the methodology for the evaluation phase of the DSR framework to collect data, due to the unavailability of accelerators and the fact that the current chip design can be modeled more quickly and still accurate in software when done properly. A simulation model can be defined as computerized representation of a given model capturing its dynamic

[215] cf. Bjarnason/Lang/Mjöberg 2021, p. 8
[216] cf. Nelson et al. 2016, p. 25
[217] cf. Mihram 1976, p. 71
[218] cf. Mihram 1976, pp. 71–72
[219] cf. Ahmed/Demirel 2021, pp. 6–7
[220] cf. Mihram 1976, pp. 71–72
[221] cf. Mihram 1976, pp. 75–76

behaviour. The primary motivations for establishing a simulation model or using any other modeling method like prototyping is that it is an cheap and fast way to gain important insights without being exposed to following constraints: costs, risks or logistics of manipulating the real system.[222] Furthermore, the gathered data helps with decision making for strategical and operational levels.[223] For example, with results of a simulation it can be decided if the new hardware works like expected and can be set up for production. This are the reasons why the simulation methodology is chosen for the evaluation phase of the prototype.

Computer simulation involves adjusting a computer-based model to better analyze how a system behaves and to evaluate approaches for their operation, either for descriptive or predictive purposes.[224] In the case of th mem-HNN, there is the need for the evaluation of software performance in combination with the hardware to gather proper data. The reason for this is that with only using a functional software simulation without considering the hardware specifications it results in a decreased price and time but with a significant precision loss.[225] However, precision and efficiency are a key part towards beeing able to answer the research question.

A general simulation model by Kellner/Madachy/Raffo published a model that can be seen as overview of the work in the simulation field. It consists out of the following entities: (0) model purpose, (1) model scope, (2) result variables, (3) process abstraction and (4) input parameters.

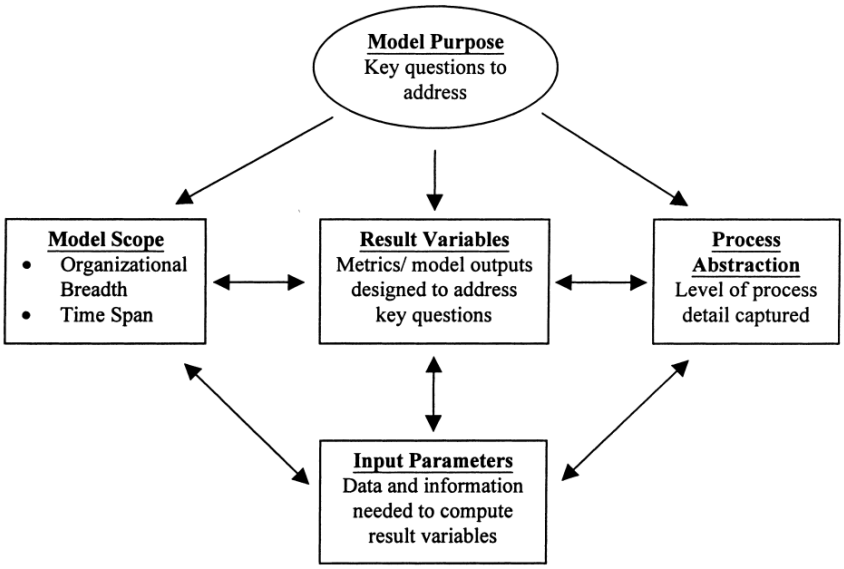

Fig. 15: general simulation model[226]

The general **model purpose** is based on the specific research question that needs to be answered. It is crucial to thoroughly understand the effects based on this key question to ensure the correct

[222] cf. Kellner/Madachy/Raffo 1999, p. 92
[223] cf. Kellner/Madachy/Raffo 1999, p. 93
[224] cf. Abar et al. 2017, pp. 13–14
[225] cf. Sarhadi/Yousefpour 2015, pp. 470–471
[226] Kellner/Madachy/Raffo 1999, p. 95

selection of the **model scope**.[227] This can be an iterative process which includes the selection of a scope, for example a development project, long-term product evolution etc. and within this scope an estimated timespan (short(less than 12 months), medium (12-24 months), long (more than 24months)) meeds to be selected. In addition to that, the organizational simulation breadth is set (less than one project team, one project team, multile project teams).[228]

The **result variables** are information elements that are the central key entity of the simulation model. They change based on the main question being asked, representing the crucial signs of a successful simulation. Typical metrics for software simulation could be: effort/cost, throughput/productivity, queue lengths in the backlog, energy efficiency, return on investment Furthermore, with one simulation mutiple result variables can be gathered simultanously.[229]

Process abstraction includes inner structure of simulation model. Therefroe, all the processes, vital ressources, dependencies and iteration looops need to be considered to achieve the desired result variables and to answer the key quesions.[230] Lastly the **input parameters** consider all the parameters that are needed to produce vaulable outcomes. This can range up to hundreds of data parameters achieve the desired results. In theory, these parameters can also be extendetd to human ressources like software engineers that are needed for their skills in programming knowledge.[231]

This general usable simulation model should is not only part of the DSR evaluation phase but also part of the ASIC design process. Therefore, the model is modified to match the needs for a performance simlation of the mem-HNN. The simulation model is part of the architecture and high level design of the ASIC design process. It involves selecting key components like processors, memory blocks, and communication interfaces and carrying out a functional verification through a suitable simulation.[232] The modification to the model is expressed through the actual energy model of the mem-HNN, which is added to the simulation model, that can compute energy usage per clock cycle. Hence, depending on a specific input it can calculate how much energy was required to do computations that are the output and used for the next cycle.

[227] cf. Kellner/Madachy/Raffo 1999, p. 95
[228] cf. Kellner/Madachy/Raffo 1999, p. 96
[229] cf. Kellner/Madachy/Raffo 1999, pp. 96–97
[230] cf. Kellner/Madachy/Raffo 1999, p. 97
[231] cf. Kellner/Madachy/Raffo 1999, pp. 97–98
[232] cf.Rao 2024, p. 1; cf. *ASIC Design Flow for VLSI Engineering Teams [GUIDE] - Xinyx Design* 2024, p. 1

4 Implementation of the mem-HNN

4.1 Objectives and research methodology

Upon establishing the precise research methodology, this chapter delves into the implementaiton of the simualtor pipeline. First, the analysis phase of the DSR process is executed with the goal to establish a model of the simulator pipeline, which the requirements and conditions can be derived from. Next, a practical prototype is developed in iterative design cycles to fulfill the target requirements. In the evaluation phase, the simulator pipeline is used to assess the performance of the mem-HNN in the training of an examplary machine learning workload. This thesis utilizes performance metrics collected from the simulation to address the central research question, which explores potential speed and efficiency enhancements of the mem-HNN compared to digital computers.

4.2 Analysis phase

4.2.1 General conditions

Following the first phase of the DSR-cycle described in Chapter 3, the research outline is initially established from which the requirements for the simulator pipeline are derived.[233] This analysis begins by describing the general conditions specified in Section 3.1. Hereby, general conditions are permanent design decisions that are used as foundation for the implementation of the simulator pipleine. The underlying motivation hereby is to research if the known proof of concepts are feasible on the complete mem-HNN and evaluate if that brings an actual acceleration, which is equivalent to answering the research question of this thesis.

The implementaton is executed in the programming language Python since it offers a variety of third party libraries that are useful for machine learning that are state of the art, like pytorch, scikit learn etc..[234] Furthermore, Scikit Learn is chosen as machine learning library since it is one of the industry standards for classical machine learning, has a broad variety of features in terms of BMs and has a lower learning curve compared with e.g. Tensorflow.[235] For simplicity and to save time, an RBM is used as a test case with handwritten digit classification as workload.

The complete mem-HNN is being simulated based on a design that has been developed by the Forschungszentrum Jülich and HPE.[236] This design describes an ASIC design that realizes the noisy Hopfield Network shown in figure10. It includes an energy model based on low-level circuit-simulations, which can derive the average energy consumption per clock cycle of the mem-HNN.

[233] cf. Österle/Otto 2010, pp. 278–279
[234] cf. *Discrete and Continuous Models and Applied Computational Science* 2024, pp. 306–307
[235] cf. Raschka/Patterson/Nolet 2020, pp. 5–6
[236] cf. Hizzani et al. 2023, pp. 3–4

In addition to that, the model includes latency estimations of the mem-HNN to perform a full iteration. This simualtion approach is chosen as the device is still under development and hardware devices are not available yet. Nonetheless, the complete hardware can be realized in software without compromising its functionality. Such a simulation based performance evaluation is quite common in the ASIC design flow.[237] A in-depth explanation of this model is out of scope for this thesis but core parameters are explained to understand the gathered energy values. Lastly, the simulation is performed on a notebook. Due to the limitations of the built in CPU[238], efficient coding is required to ensure simulations are performed within an acceptable timeframe.

4.2.2 Requirements

To evaluate the performance of mem-HNNs in training and inference of BMs, a full simulator pipeline has to be modeled. In Fig.16 the envisioned model is shown, in which an mem-HNN chip can be used to implement BMs. Here, a digital computer is used to implement and train machine learning models on vairous datasets. The mem-HNN chip is then used to perform the sampling during the BM inference or training. Training and inference on this system then involves the following five steps, which are handled by different components and desribe the iteraction between the digital computer and the analog mem-HNN chip:

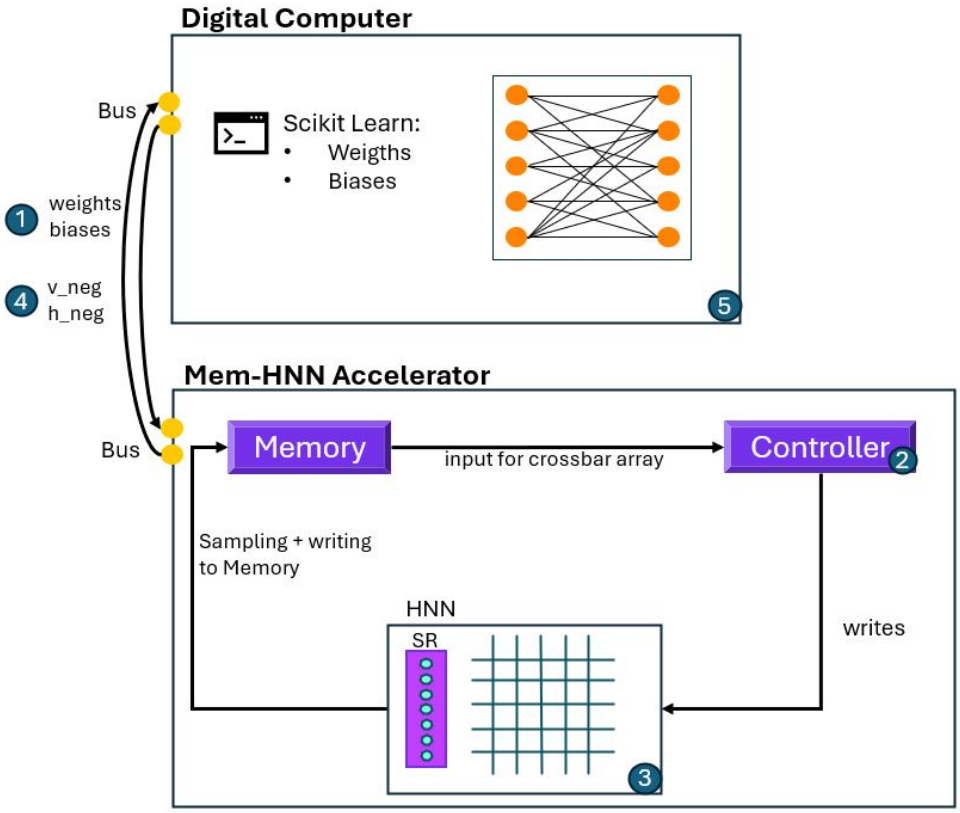

Fig. 16: proposed solution architecture

[237] cf.Rao 2024, p. 1; cf. *ASIC Design Flow for VLSI Engineering Teams [GUIDE] - Xinyx Design* 2024, p. 1
[238] `Intel i7-10610U, 1.80GHz, 2304 MHz, 4 Cores, 8 Logical Processors`

1. initializes the machine larning model (neural network) including setting the weights and biases of the BM.

2. starts with the transfer of the weights and biases to the mem-HNN accelerator via a bus-system. The local memory saves the data and forwards them to the controller. The controller is able to programm the memristors in the crossbar array.

3. is the Hopfield Neural Network (HNN), which contains the memristor crossbar array and the state register (SR). Here, mem-HNN performs a pre-defined amount of iterations, where a sample configuration is stored in the on-chip memory after each iteration.[239] The state register includes the current neuron configuration and can lock und unlock specific neurons, so that it is possible to update neurons synchronously.[240] This enables the possibility of the promissing N/2 update strategy.

4. After the mem-HNN has performed all iterations, the stored sample configurations of the visible v_{neg} and the hidden h_{neg} neurons are transferred back to the digital computer via the bus-system.

5. With the sample configurations, the digital computer calcualtes the activation probabilities and performs the updates to the weights and biases according to the trainign rule in equation2.5. These training updates are repeatley performed starting again from step 2 until the model achieves sufficient performance. Furthermore, the model can be evaluated in its performance in terms of chosen metrics like prediction accuracy or the negative likelihood etc..

In the simulator pipeline the behavior of the mem-HNN in fig.16 is mimicked, where the simulator acts as a drop-in replacement until hardware devices become availbale. The simulator models the behavior of the chip, so that performance predictions are possible long before a hardware device can eventually be used. Here, it is important to stress that the current hardware model described in the previous section currently does not contain modeling results for the on-chip memory, the controller and the bus system. The simualtor is therefore solely focused on modeling the sampling step 3. With more modeling results becoming available, accurate simualtions of step 2 and 4 can be added to the simualtor. The next step is to derive the requirements and to establish the research outline. The aim of generating requirements is to generate good quality, requirements that offers an acceptable level of risk to start the project.[241] These requirements need to cover the functions of the mem-HNN, which then must be implemented by the respective software components. Also, requirements may evolve over time and require adjustments when outcomes differ from initial expectations. As a result, considering the research question and objectives, the following software requirements emerged:

Digital Computer

- Initialize a BM

[239] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[240] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[241] cf. Ebert 2008, p. 11

- Utilization of any training data
- Initializing of other ML models that are used in conjucntion with the BM for evaluation
- Training of the BM with either Gibbs Sampling or Metropolis Hasting
- Setting individual parameters: sampling steps, training iterations, noise level, learning rate
- Evaluation of the model's performance using prediction accuracy and negative likelihood

Simulated Mem-HNN Accelerator

- Using any BM as input
- Implementing the noisy Hopfield Neural Network update mechanism described in 2.12
- Return sampled output of the neuron configurations
- Selectively switching between the N/2 half updating method instead of single spin updates described in 2.4
- Simulation of computation speed (throughput) and energy consumption of the mem-HNN chip

Furthermore, the python program should be split logically into the different modules and components to enable well structured code. With set requirements it is now possible to begin the iterative design and evaluation cycle with focusing on some requirements per iteration.

4.3 First Design and Evaluation phase

This **Design phase** has the goal of implementing the parts of the simulator pipeline that is executed on the digital compuer in fig.16 requirements. Accordingly the resulting prototyp of this iteration has to satisfy all requirements listed under digital computer in section 4.2. The implementation is verfied with a test case based on classification of handwritten digits. The first step in the described prototyping methodology within 3.3 is to perform the systemic analysis to categorize the prototype. In the realm of prototyping, the following categorizations are made: the prototype type (1) is computational, and its fidelity level (2) is high, as it aims to model all functionalities closely to reality. Furthermore, the complexity (3) is considered moderate because not all hardware components can be modeled in software. Additionally, the scale (4) remains constant, and there are multiple iterations (5) executed sequentially to train and infer the RBM. This process of categorizing the protoype helps to make prototypes comporable as prototyping can be used very vague. The second step, is to set up the Scikit Learn machine learning library as explained in 4.2. Scikit Learn includes built-in models for BMs that enable rapid development also includung popular datasets, delivering results that are comparable with those found in literature. This is useful to answer the research question in a timely manner.

Especially, the test case used for the BM is based on an example of the official scikit learn documentation.[242]

The following task is to set parameters like the learning rate, iterations, size of the hidden layer. With having a look in the literature and through testing a learning rate of 0.2, 10 training epochs with 72 iterations in one epoch, and an hidden layer of 100 neurons is chosen.[243] The size of the visible layer is automatically recognized by Scikit Learn, so for example to 64 to correspond with 8x8 images of a dataset. Also, the dataset implementation can be customized as needed, such as for a breast cancer classification workload.

The training of the RBM is performed in the `.fit` method and for the functionality to select the preferred sampling algorithm an additional sampling method need to be added. This process includes modifying the `_rbm.py` file in the basic scikit learn library. The predefined sampling method is gibbs sampling and there is no option to access metropolis hasting within the basic library. Therefore, the metropolis hasting alhorithm, explained in 2.2.3 needs to be manually implemented. The according adjustments are included from the code availability of a paper.[244] This decision is made because the algorithm used there is the original metropolis algorithm by Metropolis et.al..[245] Furthermore, the implementation is performant with many numpy functions. To utilize this sampling method, some minor adjustments are made for the user friendliness. First, one function is fixed that has a small error, which produced an erroneous empty configuration as the first sampling iteration. The user friendliness is achieved by introducing a new parameter `sampling_method` that dynamically allows to change the sampling method. Another change is the approach of evaluating the performance of the neural network after an x amount of iterations to meassure its performance on the test data while training. A complete code overview of the metropolis hastings sampling algorithm can be found in the `mcmc2.py` file as part of the digital delivery with all adjusted methods for the training of the RBM in `_rbm.py`, while the overall execution takes place in the `playground.py`.

To evaluate the results and functionalities the **Evaluation phase** in this iteration validates the functionalities through a training of the RBM. with each sampling method and extract their prediction accuracy. Scikit learn offers a variety of datasets that are already in a polished format, ready to use. The decision is to use a classification workload of handrwritten digits as a test case for validating the implementation. One reason for this is that the "load digits dataset" is similar to the well known MNIST dataset representing an industry standard problem, but has a smaller resolution of 8x8 pixels and features around 1800 samples that can be categorized in 10 classes (integers 0-9).[246] The second reason is that the workload is already optimized and therefore can deliver relevant data for the research question. In this case additionally a nudging of the data is chosen to create more samples, by a factor of five, and to bring more complexity in the workload. The split in the dataset is selected to be divided into the conventional 80% training data and

[242] cf. *Restricted Boltzmann Machine Features for Digit Classification* 2024, p. 1
[243] cf.Hinton, G. E. 2012b, p. 11-12; cf.Böhm et al. 2022a, p. 1
[244] cf. Böhm et al. 2022a, pp. 11–12
[245] cf. Metropolis et al. 1953, pp. 1087–1092
[246] cf. *Sklearn.Datasets.Load_ digits* 2024, p. 1

20% test data.[247] For training, an RBM is selected as the feature extractor and paired with a logistic regression classifier for prediction, thereby establishing a cohesive pipeline. Following figure17 shows the training results using the gibbs sampling approach.

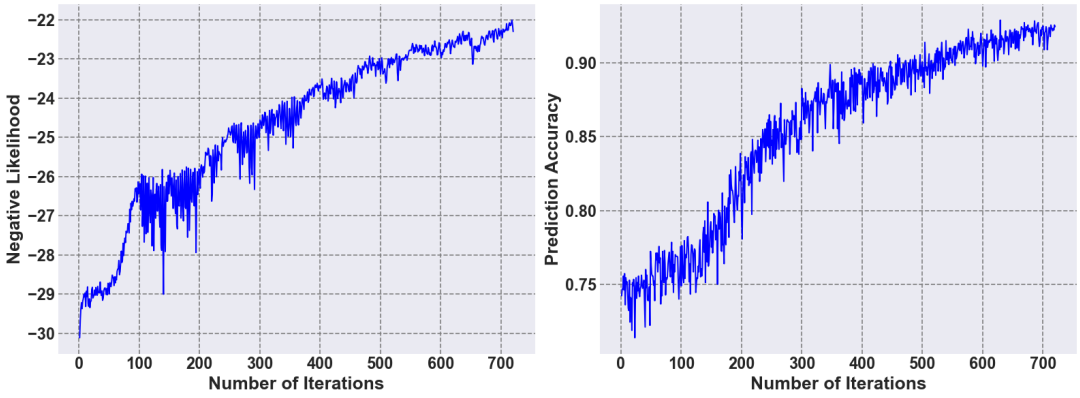

Fig. 17: Gibbs sampling baselines

The right plot shows that the initial prediction accuracy starts at 75%, akin to that of just a single linear regression model. This suggests that the untrained RBM at the input of the classifier initially does not effect the model's classification performance. Data points are colected after every iteration across the span of 720 iterations. After 650 iterations the accuracy slowly stagnates and has a maximum prediction accuracy of 92.29%. In the left plot picturing the negative likelihood, which is a measure of how well a statistical model represents the observed data. When training a model the aim is to minimize the negative log-likelihood, which means that the model maximizes the probability of generating the observed data. Hence, it is visible that in the beginning the model learns more rapidly and in steadily grows its knowledge with some smaller break-ins at the end. The best value is a negative likelihood of -22.01.

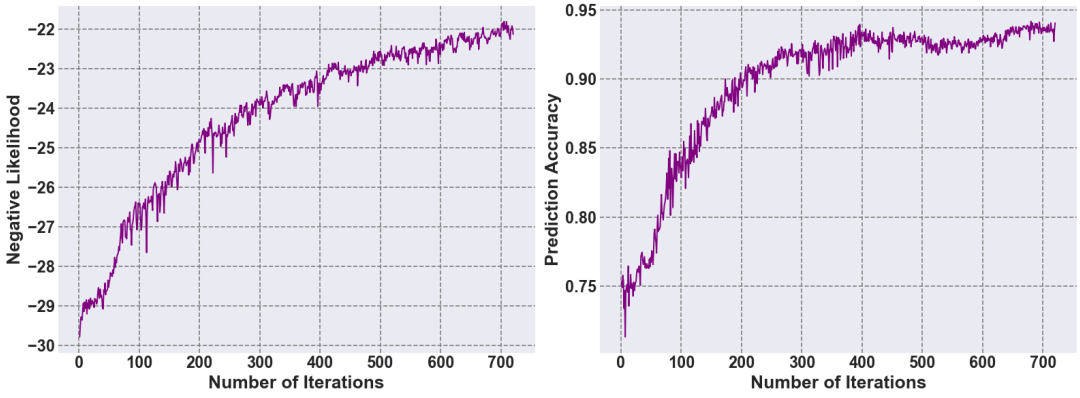

Fig. 18: Metropolis sampling baselines

In contrast, the fig.18 used the sampling algorithm of metropolis hasting to train the RBM.

[247] cf.Charitha et al. 2022, p. 1-2; cf.Supri et al. 2023, p. 1

Noteworthy is that the prediction accuracy in the right plot has a faster increase in the beginning but already starts to stagnate at around 400 iterations. Here, the maximum prediction accuracy achieves 94.15%. The negative likelohood is also growing faster than in Gibbs sampling and exhibits less variablitiy showing a more continuously learning rate. Here, the best negative likelihood value is -21.80. Furthermore, to validate the accuracy of the results, they are compared against similar results to those from the literature.[248] Therefore, a good agreement is visible, suggesting that the implementation produces valid results and can thus be considered correct. As a result, with each sampling method successfully undergoing a training, all the functionalities can be proven right and the prototype can be passed into the next design iteration. The gathered data can later on be used as baseline against the desired new updating mehtod of sampling with a Hopfield Network.

4.4 Second Design and Evaluation phase

This **design phase** iteration has the goal of implementating the simulator for mem-HNN chip shown in fig.16. The functionality of this chip is that of a noisy Hopfield Neural Network, which has previously been described in 2.4.4. The update mechanism is based of he update formula in eq.2.12, where noise is injected to imitate the stochastic behaviour of the neurons. Hence, a BM can be modelled by correctly tuning the noise of the Hopfield Neural Network. Following subfunctionalities need to be established: drawing random neurons to update, correct injection of the gaussian noise scale, calculating the weighted sum, comparing the weighted sum + bias + noise against the threshold and saving the new neuron configuration. Furthermore, the possibility of selectively switching between the N/2 half updating method instead of single spin updates is to be included in this design phase. Hence, this iteration aims to break new ground as it involves implementing the Simulator Pipeline, which has not yet been validated, and integrates the noisy Hopfield Network with the capability for N/2 half updating.

The Hopfield Network is initialized with a size of just one neuron and an sampling iteration counter of 1500 iterations with a thermalization of 100 sampling steps before the neuron is updated. Thermalization is included to allow the network to perform independent sampling steps and get into a flow to ensure unbiased sampling steps. The threshold as defined in the update formula is 0. As experimented the update formula for the implementation of the Hopfield Network looks the following:

```
for x in range(self.iterations_per_theta):

    self.neuron_index = np.random.randint(0, self.size) #pick a random neuron in
        the network
    # Calculate the weighted sum for the neuron, excluding its own state
    weighted_sum = sum(self.weights[self.neuron_index][j] * self.configuration[j]
        for j in range(len(self.configuration)) if j != self.neuron_index)
```

[248] cf.Böhm et al. 2022b, p. 5; cf. *Restricted Boltzmann Machine Features for Digit Classification* 2024, p. 1

```
        self.new_configuration = deepcopy(self.configuration) #copying the old
            configuration to create a new one and update it
        if (weighted_sum + self.bias + np.random.normal(0, scale=1.75)) >=
            self.threshold_theta:
            self.new_configuration[self.neuron_index] = 1
        else:
            self.new_configuration[self.neuron_index] = 0


        self.configuration = deepcopy(self.new_configuration) #Cloning current
            configuration and updating the cloned version to the new configuration
            after comparing with threshold


        if x >= self.thermalization:
            self.summedConfigurations =
                self.sum_configurations(self.summedConfigurations,
                self.new_configuration)
            self.iterationcounter += 1

    self.activationProbabilityPerNeuronDict[self.bias] =
        self.divide_array_elements(self.summedConfigurations, self.iterationcounter)
    self.bias += 0.025
```

The code represented here shows the update meachnism for the single spin update. In the beginnig a random neuron is drawn to be updated, which currently everytime is neuron number one because the network size is initialized with one neuron. Calculating the weighted sum can be seen as the core of the update formula and is executed first. Therefore, the weight matrix is selected (indexed by the random drawn neuron) and multiplied by the current configuration resulting in the weighted sum that also ignores the weights connected from the drawn neurons. Afterwards for the comparison against the threshold the according bias of the neuron is added together with the injected noise (scale). To achieve the injection of noise, a Gaussian normal distribution is added which can modify the activation function, making it compatible with the sigmoid function.[249] Technically this is performed by adding `np.random.normal(0, scale=1)` to the weighted sum and the bias, with 0 the mean of the distribuion and the scale representing the standard deviation. Hereby, it is important to find a standard deviation that is very close to the true activation probability is important, otherwise the training of the RBM would not work. In addition to that, the standard deviation changes with neuron size and needs to be readjusted if changes are made to the networks structure. Next, if the weighted sum plus bias and noise exceeds the threshold, the corresponding neuron state is set to 1; otherwise, it is set to 0. Lastly, after enough iterations when the thermalization is exceeded the new configurations are summed up to enable calculating the activation probability of the neurons.

For the sake of readability the N/2 half update mechaism is seperated but the used selective method with an according parameter is available in attachment 4. Hence, in a next step the pos-

[249] cf.Böhm et al. 2022a, p. 1-2; cf.Mahmoodi/Prezioso/Strukov 2019, p. 2

sibility of the N/2 half updating methdod should be implemented as already mentioned in 2.4.3 and 3.1. N/2 half is updating neurons synchronously instead of the conventional asynchronously (only one neuron is chosen and updated) used in the Hopfield Network updating mechanism.[250] Following adsjustments are made in the code to achieve this behaviour:

```
self.neuron_index = np.random.randint(0, 2, self.size) #pick complete random
    neurons in the network, result [0,1,1,0,...]

            weighted_sum = np.dot(self.weights[:, :], self.configuration)
            self.new_configuration = deepcopy(self.configuration)
            bias = self.bias

            for i in range(len(self.neuron_index)):
                #updating function comparing against threshold
                if self.neuron_index[i] > 0:
                    if (weighted_sum[i] + bias[i] + np.random.normal(0,
                        scale=self.scale)) >= self.threshold_theta:
                        self.new_configuration[i] = 1
                    else:
                        self.new_configuration[i] = 0
```

The randomly drawn neuron index now assigns either a 1 or a 0 for each neuron over the size of the network. Hereby, a 1 means that the sum of this neuron is calculated and will be compared against the threshold. As a result, in a completely random process this would lead to about 25% of all neurons updated ($50\%drawn * 50\%updated$).

Coming back to the last two lines of the first code block, the resulting activation function is obtained by summing all configurations within a single bias configuration. In a next step, the configurations counted are divided by the number of total sampling iterations within the bias configuration (in this case 1500 iterations).

With the resulting file `_hopfield_network_v1.py` the aim is to **Evaluate** the establihed noisy Hopfield Network with a single neuron and meassure its activation function. Like mentioned in 2.4.4, a Hopfield Network has a binary activation function that needs to be made compatible with the sigmoid activation function of the RBM. The decision to use a single neuron enables fast iteration times and clear results on how the network behaves, facilitating the measurement of activation probability. Once the noise injection proves stable and effective for a single neuron, it proves that it will work for coupled neurons. The value of the bias ranges from $-6; 6$ in step sizes of 0.025. After completing all sampling iterations beginning with -6 the step size is added to the bias until all iterations are made. This is sufficient to completely cover an ordinary Hopfield Network with its range of the neuron activation function. That following figure 19 visualizes the resulting activation probability of the single neurons.

[250] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, pp. 23–24

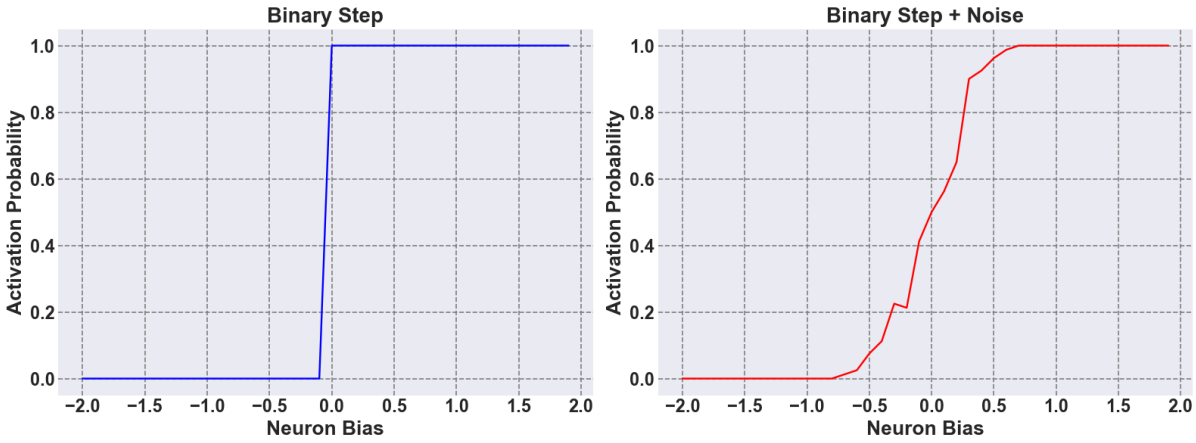

Fig. 19: Modification of the the Hopfield Network binary step activation function

In the left plot, visualized in blue, the activation probability of the neuron is shown without adding noise to the plot. The behaviour is like one would expect it; once the bias reaches 0 the neuron is activated all the time. In the right figure with the the red line a noise of $sigma = 0.3$ is added. The resulting activation is probabilistic and follows a rudimentary sigmoid shape, very similar to the activation function of a BM shown in fig.6. Is is visible that the noise injection works, even though it doesn't perfectly copy the sigmoid function. Hence, the standard deviation of the noise has a direct influence on the shape of the activation function, similar to the temperature in eq.2.12. To correctly mimic the behavior of a BM, it is therefore important to select the noise strength such that it corresponds the temparature of a BM, which is $T = 1$.[251] In following fig.20 the standard deviation of the noise injection is optimized $scale = 1.75$ and compared to the sigmoid activation function of a BM. The result verifys that the mem-HNN can correctly imitate the behaviour of a BM and its activation function:

[251] cf. Hinton, G. 2014, p. 3

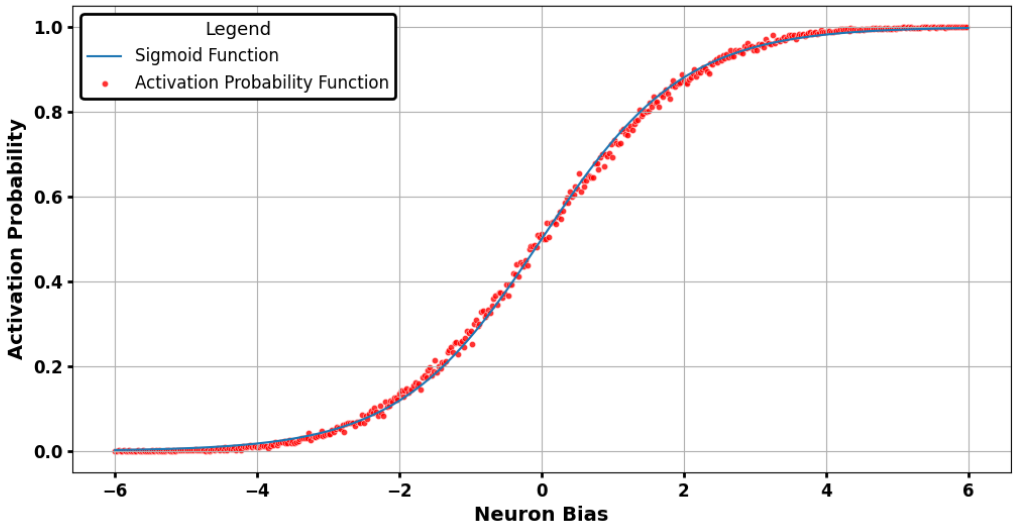

Fig. 20: Noisy activation function of the Hopfield Network imitating the RBM

4.5 Third Design and Evaluation phase

Now, that the proof of concept has been validated, the third **Desing Phase** has the goal integrating the mem-HNN simulator into Scikit Learn. This integration enables the execution of the full Simulator Pipeline shown in fig.16 and allow for the simulation of training a BM. This includes using the weights and biases of the BM as an input and performing the sampling with the input. Finally, the sampled output of the visible and hidden neuron configurations need to be returned, so that the digital computer can update the weights. With these subgoals the total goal is to enable a complete training of the BM with the sampling method "Hopfield Network". The first technical step is to extend the `_rbm.py` to fit the new sampling method:

```
if sampling_method == SamplingMethod.GIBBS:
        v_neg = self._sample_visibles(self.h_samples_, rng)
        h_neg = self._mean_hiddens(v_neg)

elif sampling_method == SamplingMethod.METROPOLIS_HASTING:
    h_neg,v_neg=mcmc_sample(10000,len(self.components_))

elif sampling_method == SamplingMethod.HOPFIELD_NETWORK:
    # Hopfield Network Sampling
    v_neg, h_neg = interface_hopfield_sampling(self.components_,
        self.intercept_visible_, self.intercept_hidden_, iterations_per_theta,
        N2_HALF=False)
```

Here, the compontents represent the weights of the neurons in the network, while intercept_visible and intercept_hidden represent the bias of the neurons. Within the, `hopfield_network_interface_v2.py` the parameters are taken and an object of the class is initiated. Furthermore, the boolean N2_HALF can be assigned to determine which updating approach is desired.

```
def interface_hopfield_sampling(components_, intercept_visible_,
    intercept_hidden_):

    H_net = Hopfield_Net(components_, intercept_visible_, intercept_hidden_)
    H_net.update_network_state()

    return H_net.v_neg , H_net.h_neg
```

Inside the class, the initialization of all the parameters and weights are performed. The update formula needs to calculate the weighted sum, which necessarily requires to know all the weights between the neuron itself, to all the other neurons. The decision is to create a weight matrix shown in attachment 2. The function begins by defining the total number of hidden and visible neurons based on the class properties parameters used as input. These quantities dictate the dimensions of the weight matrix, which, in this instance, results in a matrix of size $(100, 64)$. This square matrix represents the fully interconnected network, where each neuron can potentially connect to every other neuron, including itself. Again the RBM is used as simple test case and therefore as mentioned in 2.2.3, the diagonal elements (self-connections) are set to zero.

It is important to maintain the model's symmetry in the weight matrix, which is crucial for the energy-based nature of RBMs and the dynamics of Hopfield networks. Hence, for this reason matrix is initialized as a symmetric matrix using NumPy's np.zeros function. This ensures all initial weights are set to zero before explicitly being defined through the components weights. Scikit Learn randomly initializes the weights by default close to zero, while the biases are set to zero. This small weights allow to support an effective gradient distribution which protects against rapid saturation or inefficient learning, while the bias set to zero allows the network to begin in a neutral position and learn on its own.

The subsequent nested loops iterates over the indices for hidden and visible neurons to fill the weight matrix. For each pair of hidden and visible neurons, the corresponding weights are extracted from the components matrix. This matrix essentially serves as the template for the interactions between hidden and visible layers. Indexing within the weight matrix is handled carefully, to respect the structure of the BM. Therefore, the decision is to set weights between a hidden neuron i and a visible neuron j at positions `[i, j+num_hidden]` and `[j+num_hidden, i]` to ensure symmetry.

With some more adjustments necessary to the code like transposing the returned neuron configurations, the first training with the sampling method of the Hopfield Network is possible. As in the Design Phase 1 the same test case of RBM predicting on a handwritten digits classification

dataset is chosen to make results comparable with the earlier established baselines. The Hyperparmeters first are set to a scale of 1.75, thermalization of 100 as used for the single neuron in fig.20 but with more sampling steps (5000) than before. The results of the training are visualized in fig.21.

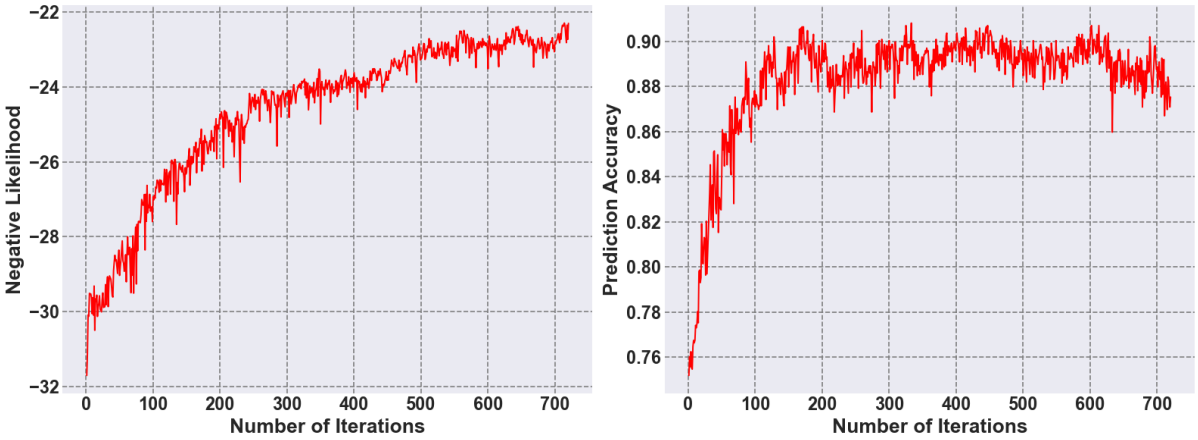

Fig. 21: Hopfield Network sampling baselines

It can be seen that the negative likelihood is far more stable as Gibbs sampling and rather has similarities with Metropolis Hastings sampling. Here, the best value is a likelihood of -22.3, which is slightly worse than Metropolis Hastings and Gibbs sampling. The right plot, which displays the prediction accuracy, shows the steepest ascent among the three graphs. This suggests that good results can be achieved with fewer iterations compared to the other two methods. The best prediction value is 90.81% but no hyperparameter tuning has been done yet.

As a result, the following hyperparameters are tuned to possibly received better outcomes. Specifically, the standard deviation used for noise injection requires tuning. In addition to that the amount of sampling iterations within a single training iteration is tuned. Additional hyperparameters that can be tuned are the learning rate, the total amount of sampling/training iterations and the thermalization. These are not optimized as part of this study because otherwise there is no appropriate benchmark against the other two sampling methods. First the single neuron update Hopfield Network's hyperparameter are tuned. Since, the training takes around 40 minutes to complete tuning too many hypaerparameters takes too much time for the period of this thesis. The first hyperparameter researched is the influence of the standard deviation (scale) to the maximum prediction accuracy. Given that the Hopfield Network operates as a statistical sampling method, the standard deviation, average and maximum are analyzed for the last 50 training iteations, as the training has stagnated at this point. To optimze the standard deviation, it is swept from sigma=1 to sigma=2 with a stepsize of 0.05, totalling to 21 single trainings. Lastl, only the prediction accuracy is analyzed because the negative likelihood, which represents the learning rate, is significantly steeper than the other sampling methods, and the model's final performance is of greater interest. The result is shown in figure22:

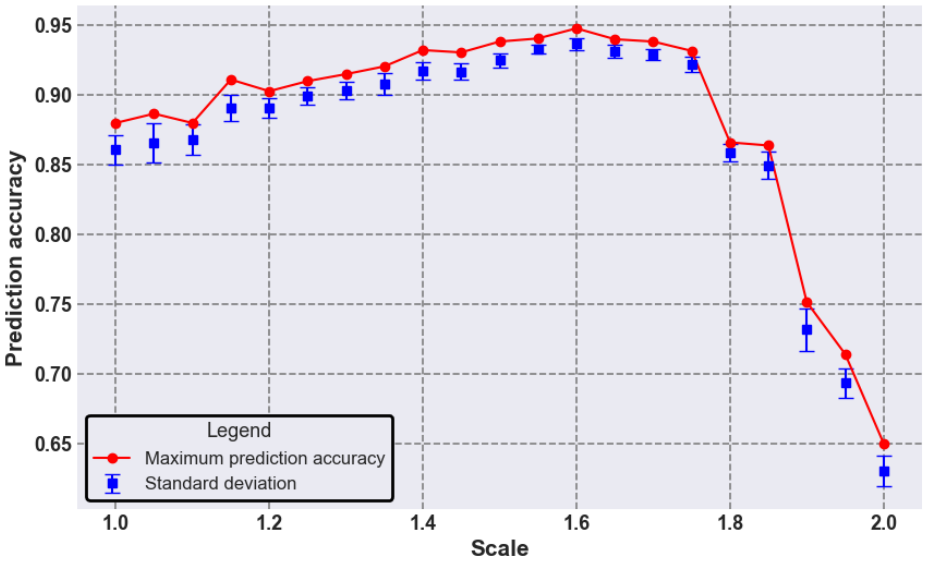

Fig. 22: Hopfield Network Hyperparametertuning scale

The result shows that beginning with a scale of 1.0 the prediction accuracy slowly rises until the scale of 1.6 Here, the maximum value is 94.77% and with that surpasses the performance of both metropolis hastings and gibbs sampling. After a scale of 1.75 the prediction accuracy rapidly declines. This shows that with adjustments to the standard deviation a good prediction accuracy is achievable. The average standard deviation follows the maximum prediction accuracy pretty closely and has no outliers. Close to the sclae of 1.0 the deviation is slightly higher compared to the rest of the plot, indicating that the scale does not model the sigmoid function correctly.

In a next step, the best fit with a scale of 1.6 is fixed as for the optimization of the number of sampling steps, as the two parameters are independent of each other. Hence, the decision is to begin with 1000 sampling iteration continiunig with an increase of 1000 iterations until 15000 iterations are reached. With that the training shows that the interesting range is around 1000 to 4000 iteration and that the step size of 1000 is too big for that. Therefore, additional trainings at sampling iterations 1500 ad 2500 are completed, totalling to 17 trainings performed. The values are extracted as before, by considering the last 50 iterations and then calculating both the maximum value and the standard deviation from this subset. The visualized results can be found in following figure23:

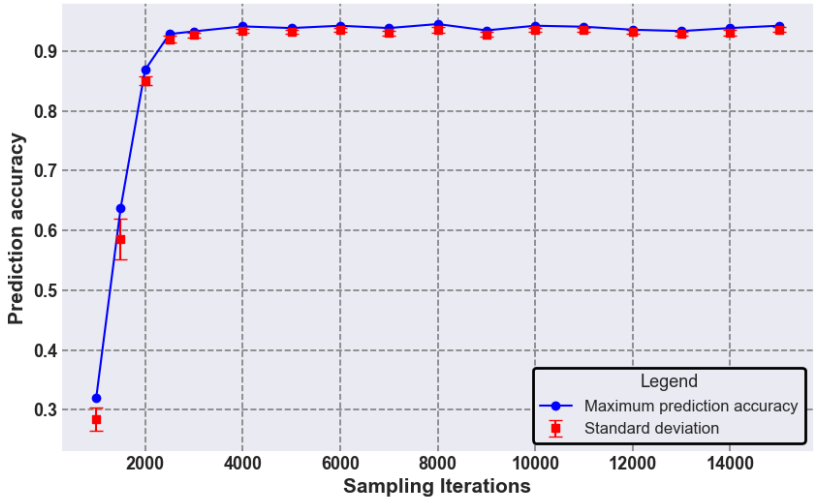

Fig. 23: Hopfield Network Hyperparametertuning sampling iterations

The line of maximum prediction accuracy starts at the first iteration with a value close to 0.3 and rises rapidly to reach a value just above 0.92 after around 2500 iterations. From the point of 4000 iterations onwards, the accuracy remains largely constant with slight fluctuations. The best value is at 15000 iterations with an prediction accuracy of 94.5%, while at 4000 iterations the accuracy is at 93.35%. The error bars indicating the standard deviation are large at the beginning of the graph. With the number of iterations increasing, the error bars become smaller resulting in a more stable accuracy. From this, it can be concludeded that increasing the number of iterations beyond 4000 has no additional benefit to the prediction accuracy. In a second step the scale of N/2 Half update mechanism is analyzed visualized in figure24 :

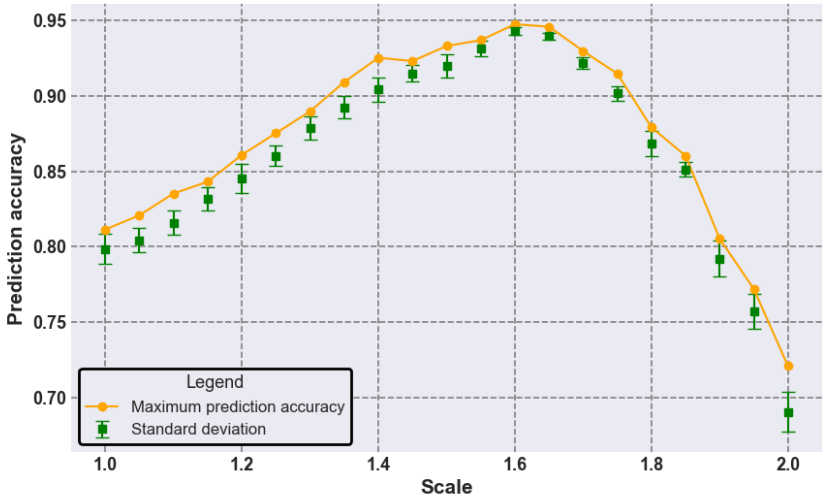

Fig. 24: Hopfield Network Hyperparametertuning scale for N/2

In this method, 21 complete training sessions are also carried out for this purpose On a first glance, the prediction accuracy beginning from left to right is constantly rising until it reaches the scale of 1.6. Here, the maximum prediction accuracy tops out at 94.76%. What is interesting, that in 22 the exact same scale has also the best performance. With increasing the scale after the top of 1.6, the accuracy declines rapidly. Noteworthy, is that in comparison with 22 the N/2 half updating method has nearly equal prediction accuracy but is more sensitive. This means that the performance decreases more rapidly as the noise is detuned from its optimum value. The standard deviation represented in the error bars is high at the lower scale values but also similar for too high values beginning at a scale of 1.8.

The second hyperparameter tuned for the N/2 Half update mechanism are the amount of sampling iterations. The decision is to start at iteration 201 (1st iteration after the 200 thermaliation steps) and ending with a sampling iteration of 250 with a step size of 1. This totals to 50 complete training sessions. The results are illustrated in figure25:

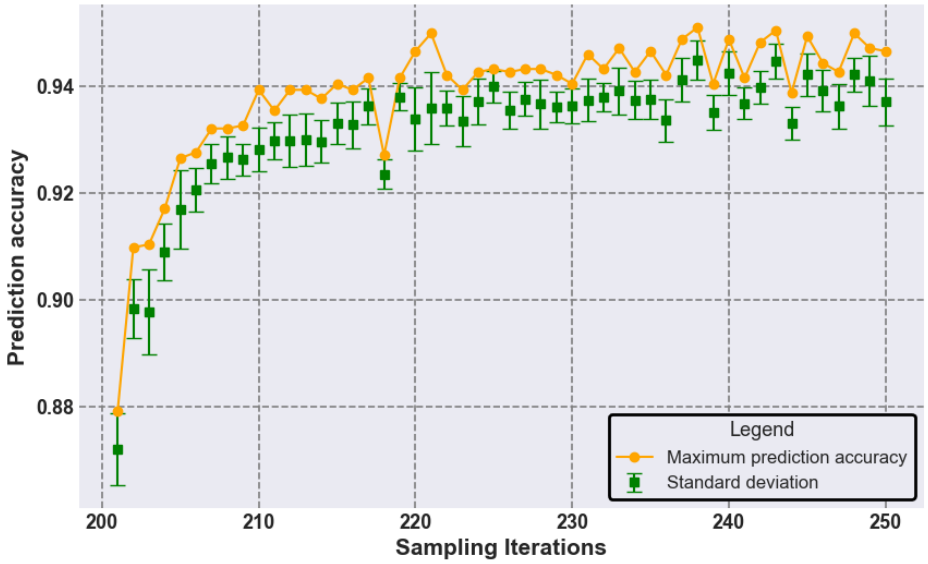

Fig. 25: Hopfield Network Hyperparametertuning sampling iterations for N/2 half

The figure shows that with only 10 sampling iterations after the thermalization good prediction accuracies of 94% can be achieved. The maximum prediction value is 95.10% at 221 sampling iterations, which surpasses the earlier achieved results. Nonetheless, the key result is, that far less sampling iterations are required within one big training iteration of the RBM to achieve good results. When comparing the number of 221 to the around 4000 sampling iterations required in 23 it would be more efficient by a factor of $18.09x$. These results of the third design and evaluation pase validate that the BM, including its weights and biases, are used as input to perform a complete training. Furthermore, the neuron configuration is returned to the digital computer to evaluate and predict the performance while updating weights and biases. Hence,

all the functionalities of the prototype are implemented and the prototype itself is complete and detailed code is available within the digital delivery.

4.6 Fourth Design and Evaluation phase

In this final Design and Evaluation phase the goal is to enhance the mem-HNN simulator to predict the performance of a future mem-HNN chip. This equals with the goal ultimately answer the research question "Can Boltzmann Machines be **efficiently** implemented on physics-inspired Hardware accelerators by analog noise injection?". The **Design** phase enables this by implementing the energy model and an autocorrelation metric on top of the functional Simulator. Since the key word is efficiently, this happens in the methodology of a simulation, which requires to specify the parameters meassured for evaluation. With the model purpose set, the model scope(time frame) of the simulation introduced in 3.4 is to be considered as short with only mutliple weeks and only one person working on the simulation. The next step is to define the result variables. The two variables that are of interest in literature to meassure the performance of the mem-HNN accelerator are troughput (samples/sec) and energy consumption (energy/operation). These two metrics are widely used in literature and can create a good comparison.[252] Hereby, throughput can be defined as the time needed per Hopfield cycle (sampling iteration) per second.[253] Meanwhile, the energy consumption is defined by summing over all the single energy consumptions within one sampling iteation. The resulting unit is called energy/operation.

Now that the result variables are set the input parameters neeed to be clarified. For the throughput knowledge about the autocorrelation of the sampling is required. Autocorrelation is a statistical measure that captures the degree of correlation between successive configurations generated by timeseries, in this case sampling algorithm for the training of a RBM.[254] Long correlations between configurations can reduce the effective sample size and lead to inefficiency impacting the precision of the model. This is important for the result variable "troughput" because it allows to know when the sampling is statistically independent and ready to use and therefore how many sampling iterations neeed to be done for an effective training. Combined with the specifications of the chip it can be calculated what the resulting throughput is. Here, covariance is implemented and used to measure the correlation between two successive samples:

$$K_{XX}(t_1, t_2) = \mathbb{E}[(X_{t_1} - \mu_{t_1})(X_{t_2} - \mu_{t_2})] = \mathbb{E}[X_{t_1} X_{t_2}] - \mu_{t_1}\mu_{t_2}, \tag{4.1}$$

with t_1, t_2 being two distinct points in time and X_{t1}, X_{t2} are random variables representing the values of the stochastic process at the distinct time points. μ_1, μ_2 are the mean (expected) values of the random variables X_{t1}, X_{t2}. The $\mathbb{E}$ is an expectation operator and is used to calculate the expected value of the expression within the brackets. Also, the cycle speed of the mem-HNN

[252] cf.Belletti et al. 2009, p. 54-55; cf.Aadit/Mohseni, M./Camsari 2023, p. 1-2; cf.Ortega-Zamorano et al. 2016, p. 16-17
[253] cf. Böhm et al. 2022a, pp. 6–7
[254] cf. Tanaka/Tomiya 2017, pp. 1–6

accelerator is needed. The clock frequency is 700MHz and therefore can sample 700 million times per second, which is 1.44ns for a single clock cylce.[255] With the simulation model specified, the autocorrelation is implemented into the prototype. The changes made for the implementation of the autocorrelation is available in the interface version 5 and part of the digital delivery.

Next, the only input variable to meassure the second metric **energy consumption** is the energy model, which is introduced in 4.2.1 and is developed by HPE in combination with the Forschungszentrum Jülich. The implementation of it allows to meassure each of the individual energy consumptions that are configured to the specifications of the mem-HNN hardware accelerator.[256] The energy function is provided in the form of a python library, that returns the average energy consumed for a single clock cycle. Specifically, it includes functions to calculate the energy usage of the probabilistic random number generator, the register, the crossbar and the digital analog converter that generates the noise signal. These functions are based on circuit simulations that were performed on a 28nm technology node commercial manufacturial. As input, the energy model requires the number of neurons, the average activation rate and the output pattern rate during a single clock cycle. Hence, the mentioned energy model needs to be implemented into the Hopfield Network interface. The first implementation is to **intialize the energy model** with the size of 164 neurons as this impacts the size of the crossbar array and influences the overall energy consumption. An increase in the number of neurons causes a higher energy consumption due to the enlarged crossbar structure.

Afterwards, the two parameters "a_pattern" and "a_WL" need to be calculated. Hereby, a_pattern refers to the currents that flow through the respective bitlines (output pattern rate). The current for each bitline is determined by the average of the weighted sum and is accumulated with each iteration. Subsequently, it is divided by the number of sampling iterations to obtain an average for the respective training iteration. For an correct calculation and imitation of the mem-HNN, there is one more restriction to solve. The digital computer generates negative and positive weights and biases, which is not possible in the hardware. Therefore, the weight matrix is adjusted to handle positive and negative weights seperately. Lastly, the memristors can be tuned to a limited amount of discrete conductance levels. For the mem-HNN design, the memristors can be set to around 32 different levels, which correspond to a 5-bit resolution. Due to this, the coupling matrix of the BM is descretized to this resolution, where the lowest conductance level corresponds to the highest weight value. In earlier design phases, the values are calculated with perfect resolution but for the energy model this is not possible anymore. A_Wl is the average configuration change (average activation rate). This parameter tracks the average changes in configuration within the wordline. Each time the state changes from 0 (no current) to 1 (current flows), energy is consumed by the switching process. To calculate the average energy consumed by such switching processes, the average number of rising bits is measured between subsequent iterations. Subsequently, an average change rate is calculated by dividing by the number of iterations. This approach quantifies the energy cost associated with state transitions within the

[255] cf. table1 Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, pp. 7–8
[256] cf. Hizzani et al. 2023, pp. 1–5

network configuration. All the implementations for the energy model within the interface can be found in the version 5 of the hopfield interface as part of the digital delivery.

Of course, for both the energy model and the autocorrelation the methods one input is the finished prototype, that mirrors the functionalities of the ASIC on a high level. The first **Evaluation** focuses on the throughput, which is decided to average the values of the output configurations from row to row to an average of 60. This allows to extract a smoother autocorrelation plot. In the attachment3 the full implementation of the autocorrelation function is available. To compare the performance of the sampling methods in terms of the autocorrelation a threshold is required. Here, 1/e is chosen since it is inspired by many fields, like physics(diffusion length), chemistry(half-life as a threshold) etc..[257] Following results can be **evaluated** and are shown in following figure26:

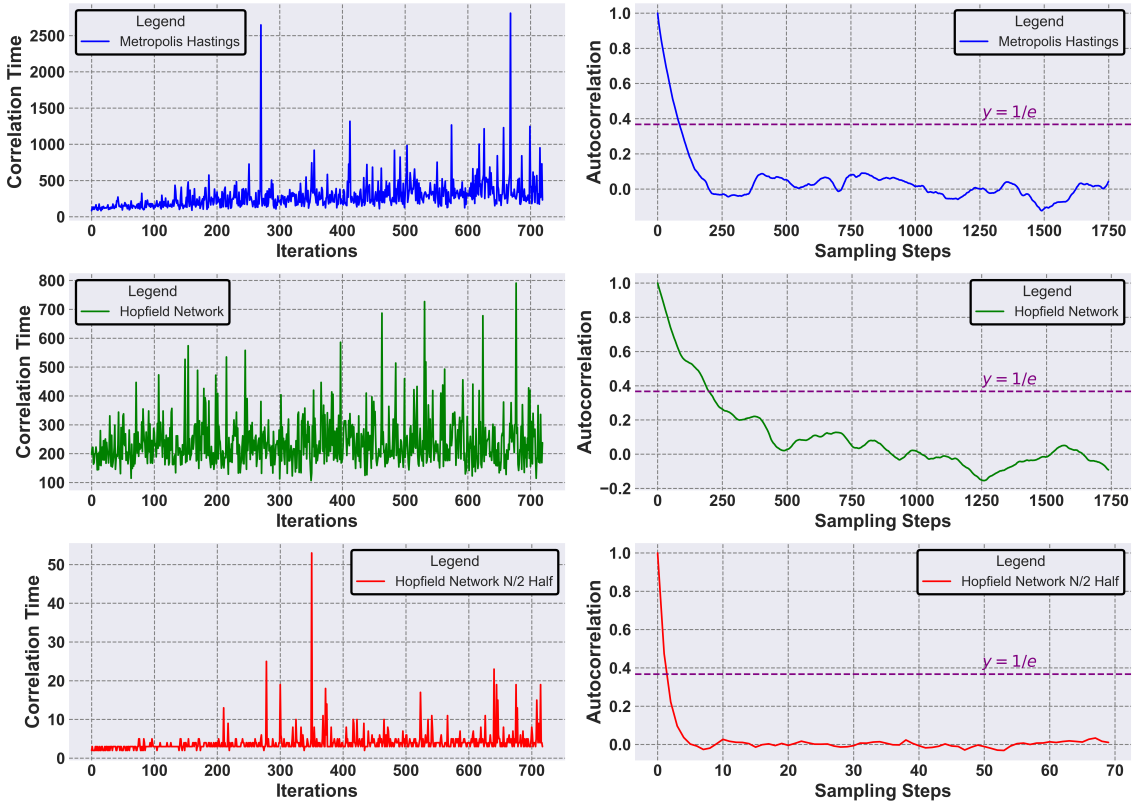

Fig. 26: Autocorrelation for the three sampling methods

The first row shows the conventional **Metropolis Hastings** sampling method. On the right plot the autocorrelation and the according sampling steps are visualized for the first training iteration. It can be seen, that the value falls below the threshold at around **100 sampling steps**. Falling below the threshold of 1/e symbolizes that statistical independent samples are generated. The left plot therefore meassures the correlation time time within each training iteration, where these samples are first generated. It can be seen that the the scale on the x-axis in the left plot

[257] cf.Archie 1985, p. 624-630; cf.Böhm et al. 2022a, p. 7-13

is higher compared to the other two sampling approaches, meaning that metropolis hastings is **more sensitive**. The second row is the **single update Hopfield Network** sampling approach. In the right plot it shows that the autocorrelation threshold is reached around after around **200 sampling steps**. Even if the value is **2x worse** than with metropolis Hastings the correlation time for the whole training is **more stable**. So even if the initial autocorrelation takes longer over a whole training period the end result has an **better average** than Metropolis Hastings.

The last approach is the **N/2 half Hopfield Network**. This method only needs about **3 iterations** to surpass the threshold in the right plot. Furthermore, the methods correlation time in the left plot shows that it is by far more **stable** than the other 2 approaches. When setting this into perspective even with a conservative average of 5 iterations as correlation time, N2/Half updating therefore performs **40x better**, than the single neuron Hopfield Network and **20x better** than Metropolis Hastings. When comparing the correlation at the end of the training iterations, the performance even increases: **34x better** than the single neuron Hopfield Network (correlation time value of 170) and **46,6x better** than Metropolis Hastings (correlatin time value of 233). Surprisingly for this updating method large statistical dependency could be seen in two out of the 720 training iterations. This means that for these two iterations the autocorrelation **doesn not fall under the threshold** of 1/e. The training was attempted three times, and in each instance, the phenomenon occurred between iterations 300 and 500. Still, this has no impact on the performance of the training and therefore can be seen as outlier. It is open for further research to identify why this doesn not happen with the other two approeaches and what is the cause. As the corresponding correlation time for the outliers is infinite, the scale has been reduced here for better visualzation of other the other datapoints.

Based on this implementation, the desired "throughput" of the simulator, which reflects the rate at which statistical independent samples are generated, is calculated. This is done by combining the autocorrelation time of both Hopfield Network approaches (with N/2 Half and without) and mutlitplying them with the cycle speed of the mem-HNN. Then, the inverse of the result is calculated resulting in the desired throughput metrics " samples/second". The cycle time is based on the simulation shown in ref[258], where a single Hopfield update cycle is performed in a single clock cycle with a frequency of 700 MHz, resulting in a cycle time of 1.44ns. The simulation was conducted using a network of 111 neurons. However, it is assumed that for the 164 neurons utilized in this thesis, the clock frequency is reduced. This is because a larger crossbar has a longer delay. As a first assumption the cycle time scales linearly with the number of neurons and therefore reaches a clock frequency around **2ns** for 164 neurons. Because the network in this thesis has 164 neurons, the time used for the calculation is estimated to be about 2ns, which can be seen as conservative estimate. The subsequent visualization27 shows the throughput for the Hopfield Network:

[258] cf. Hizzani et al. 2023, p. 4

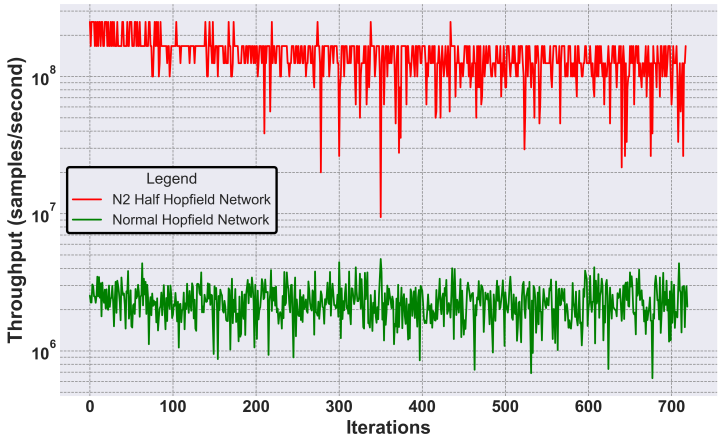

Fig. 27: throughput

The following results of two networks show that the Hopfield Network (green line) maintains stable throughput at over $\mathbf{10^6}$ samples per second, indicating predictable performance. In contrast, the N2 Half Hopfield Network (red line) shows greater variability, typically ranging from $\mathbf{10^8}$ **to** $\mathbf{10^9}$ samples, but fluctuates more than the single update approach. Calculating the average throughput, the N/2 Half Hopfield Network achieves **144 megasamples/second**, significantly outperforming the Hopfield Network's **2,3 megasamples/second**. This means that the computation speed of the N/2 Half method **is faster** by a factor of **62.72x**. In general, this shows that the N/2 Half update can considerably increase the computing speed in the sampling of neuron activation probabilities and implies a low energy consumption. The next **evaluation** focuses on the energy consumption, which is shown in following fig.28:

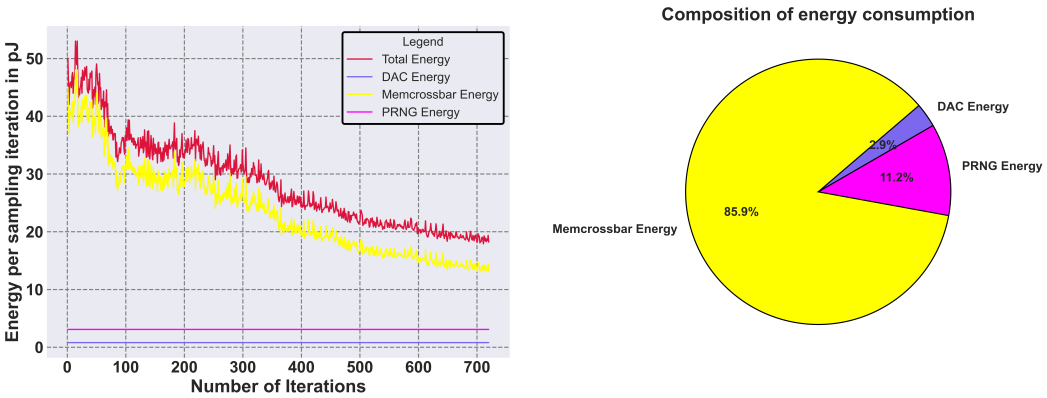

Fig. 28: Energy consumption of the mem-HNN

The left plot shows the energy consumed per sampling iteration in piko Joules. In the beginning of the training the average clock cycle has an total energy consumption of around 45 piko Joules. Further in the training at atround 100 training iterations the consumption drops to around 35

piko Joules. In the end of the training the value falls just below the 20 piko Joules mark. This decrease may be attributed to the network's weights reducing on average, impacting a_pattern. The Memcrossbar is the primary variable energy consumer, while the digital-to-analog converter and pseudo-random number generator remain constant. A composition of the energy consumption is shown in the right plot. It is important to note that the plot does not include all hardware components, specifically the communication with the memory, the controller of the mem-HNN, and the updating of weights in the digital computer. While the sampling is the computationally most demanding task in the training, it can still be expected that these components consume additional energy.

In a next step the power consumption of the training is targeted as this delivers a good comparison to other hardware components like CPUs, GPU, ASICs or FPGAs. Therefore, the energy per sampling iteration is divided by the time of one clock cycle (2ns) since $P = \frac{\Delta E}{\Delta t}$. Furthermore, the right plot in fig.29 is based on the power and cumulates the power multiplied with the sampling iteration to vissualize how much energy the training the neural network consumes for this workload.

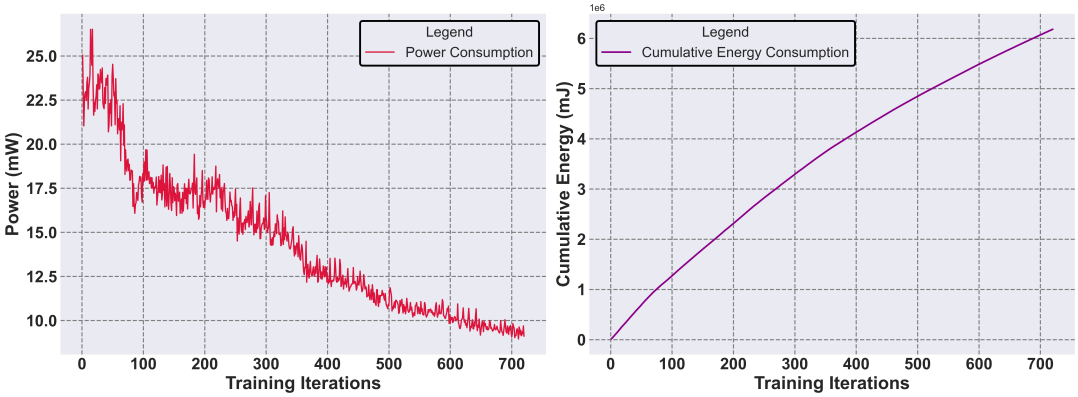

Fig. 29: Power consumption of the mem-HNN

Figure 29 reveals that training requires a maximum of 22.5 mW, substantially less than a basic CPU (20-40W); but for an accurate comparison, OS overhead should be excluded and measurements conducted using tools like Powertop.[259] Again with training iterations continuing to grow the power consumption decreases. At the end of the training around 620 iterations, an power consumption of around 10mW is reached. The power consumption is mostly stable with some minor fluctuations in the beginning. When looking at the right plot, which shows the cumulative energy required for the training in mJ. Here, it is visible that a complete training with 720 iterations requires around 6.2 mJ. With that the evaluation phase can successfully verify the models purpose of the simulation, which is to meassure the performance of the mem-HNN. The gathered data can be used in the next chapter to compare the efficiency with conventional hardware but the first part of the research question is already answered within the iterative design and evaluation phases of the DSR framwork.

[259] cf. *Powertop - ArchWiki* 2024, p. 1

5 Diffusion and discussion of results

This chapter aims to summarize the results of this thesis and to critically discuss them. Moreover, the goal is to contextualize the results with existing literature to highlight their significance and facilitate comparisons. This approach directly addresses the primary research question posed in this thesis: "Can Boltzmann Machines be efficiently implemented on a physics-inspired hardware accelerator through analog noise injection?" It also specifically responds to the secondary sub-question: "How does the performance of the mem-HNN accelerator compare to other hardware accelerators documented in the literature?" While a detailed benchmark of conventional computing concepts is not part of this thesis, wherever possible a generalization of results should be established for an initial comparison. Guided by this, more detailed comparisons can then be performed as part of future research

5.1 Solution architecture and Hyperparameter tuning

The first result is the developed Simulator Pipeline shown in fig.16 as overall solution architecture. The required components for the combination of the digital computer with the mem-HNN accelerator are universally valid and can be used to train Boltzmann Machines with the Hopfield Neural Network as sampling method. This is evaluated on the resulting IT-artifact of the DSR process by a test workload, where the mem-HNN achieves successful training with good prediction accuracies comparable to conventional training methods.

The hardware-aware Simulation Pipeline implemented as part of the incremental IT-artifact allows to predict the speed and energy consumption of the hardware long before any chips become available. Such simulations are commonly used as part of the ASIC design process and are known to produce relevant performance estimates. Crucially, the simulator acts as a drop-in replacement, where the simulator correctly imitates the hardware's components and can be replaced by the actual hardware once it becomes availbale.

In the next step, the baselines for training are established using Gibbs sampling and the Metropolis Hastings algorithm. **Gibbs Sampling** is the most **sensitive** method and has a prediction accuracy of **92.29**%, while **Metropolis Hastings** is far more stable with a faster learning curve and an total prediction accuracy of **94.15**%. With the created Baselines, the implementation of the noisy injected Hopfield Network is achieved, which is considered the main result for the thesis. A complete training of the test workload is accomplished using a noisy mem-HNN, where noise is introduced by injecting a random Gaussian distribution. This process generates the stochastic activation function of the RBM, as depicted in fig.20. Then, the performance of training is evaluated on the basis of the prediction accuracy of a machine learning pipeline consisting of an RBM combined with a linear classifier. Primarly, the metric "predition accuracy" is of

interest, since this helps to compare the performance with the other two conventional sampling methods.

Hence, the **asynchronously Hopfield Network** achieved an baseline of **90.81**% without Hyperparameter tuning but has a more stable learning rate than Gibbs Sampling and is equal with Metropolis Hastings. Here, the Hyperparameter tuning can be highlighted due to the fact that until now there was no data available for such an IT-artifact and especially not for the Hopfield Network sampling method. With some adjustments made to the standard deviation, which represents the standard deviation of the injected noise the stability, the performance is meassured. The outcome is at a scale of 1.6 the prediction accuracy is the highest of all with a value of **94.77**% shown in 22. In comparison, the **N/2 Half Hopfield Network** has a scale that is **more sensitive** than the asynchronously approach 24. On the other Hand, its predition accuracy of **94.76**% is very similar. In contrast, noticable differences can be seen when looking at the second Hyperparamer "sampling iterations". Here, the **asynchronously Hopfield Network** approach requires at least **4000 iterations** to achieve good results topping out with a prediction accuracy of **94.5**% at 15000 sampling iterations. Meanwhile, the **N/2 Half Hopfield Network** updates, on average, 50% of all neurons in the network per sampling iteration. This results in achieving good prediction accuracy after just **221 sampling iterations**. Hence, the best prediction accuracy with **95.1**% is the best out of all approaches with the least sampling iterations 25. Compared to the asynchronous update it is about **18.09x** more efficient. Furthermore, compared to Metropolis Hastings, which uses 10000 sampling iterations it is **45.24x** more efficient. Therefore, the N/2 Half approach is promising and the Hyperparameter findings from 4.5, especially for the sampling iterations, can be **generalized**. Hence, the N/2 Half Hopfield Network updating approach for a Boltzmann Machine is at least equal in prediction accuracy but uses significant less sampling iterations to the single spin update or Metropolis Hastings. In general, this shows Boltzmann Machines can be implemented on the physics-inspired hardware accelerator by analog noise injection.

It is important to keep in mind that this is only comparable for the workload of the handrwitten digit recognition by Scikit Learn that was tested on. It is to assume that the performance for other workloads and datasets would perform similar but need to be evaluated further. Hence, a literature comparison is difficult since parameters and data differ.

5.2 Throughput

The second goal of this thesis's research question is to find out about the performance of the solution in terms of the computing speed (throughput) and energy efficiency. For the throughput the result of the **autocorrelation is important** to determine when a sampling method produces statistical independent configurations shown in fig.26. The result of comparing Metropolis Hastings, asynchronous Hopfield Network and N/2 Half Hopfield Network sampling the results are the followig: Metropolis Hastings can achieve independent samples after **100 sampling**

iterations, while the asynchronous Hopfield approach requires around **200 sampling iterations** and therefore is **2x worse**. On the other Hand the **Hopfield Network** is **more stable** than Metropolis Hastings and continiously needs around 200 iterations while Metropolis Hastings needs around 400-500 iterations at the end of the training and is **more sensitive**. Next, the **N/2 Half Hopfield Network** approach only requires **3 sampling iterations** to be statistical independant from the previous sample. Furthermore, it is **the most stable** out of all approaches. In comparison, N/2 Half updating is **40x** faster than the single neuron update and **20x** faster than the Metropolis Hastings. Taking the average correlation time, the performance even increases to **34x** faster than the single neuron Hopfield Network and $\mathbf{46,6x}$ faster than Metroplis Hastings. Lastly, it is worth to mention that some outliers in the N/2 Half updating approach are found where the correlation does not fall below 1/e. This needs further reasearch but effectively does not impact the training performance.

When combining the autocorrelation with the technical specifications of the mem-HNN accelerator following computing speed results arise: The asynchronous Hopfield Network reaches a Throghput of $\mathbf{10^6}$ samples per second with a high consistency over the whole training period. Meanwhile, the N/2 Half Hopfield Network is around $\mathbf{10^8}$ to $\mathbf{10^9}$ samples per second with a more sensitive throughput and less consistency but overall higher throughput. In numbers, the troughput of the N/2 Half Hopfield Network has an average of **144 megasamples/second** while the asynchronous update Hopfield Network has an average of **2,3 megasamples/second**. The result shows that the N/2 Half approach is $\mathbf{62,72x}$ faster.

This can be put into comparison with literature values of FPGAs since compared to a CPU or GPU it is the fastest accelerator. One example of an FPGA accelerator with 300MHZ used for Monte Carlo Simulations on Ising Models performs one monte carlo step in **26.6ns** with a lattice size of 128 or **106.6ns** for a lattice size of 256. A realistic estimate of 80ns for camparison is chosen. A monte carlo step in the paper is defined as once all spins have been touched.[260] For the N/2 Half method a script `touch_all_neurons.py` is used, which is part of the digital delivery, that calculates the average of iterations required to touch all neurons in the network. With the calculated 8.79 iterations, the result is $\mathbf{2ns * 8.79 = 17.58ns}$ per sampling step, which now is equivalent to one monte carlo step in the paper. Therefore, the mem-HNN is **4.55x** faster for one sampling step than the FPGA used. Furthermore, the workload in this thesis is more complex than the 2D Ising Model in the paper and therefore an estimation for the same workload would be even more advantageous for the mem-HNN. The frequency 300MHZ used by the FPGA can be transformed to 3.33ns per clock cycle and with that a third graph can be added to figure27 showing the Metropolis Hastings throughput. This is achieved by using the 3.33ns and combining them with the results of 26. The resulting throughput comparison is shown in subsequent figure30:

[260] cf. Ortega-Zamorano et al. 2016, p. 4

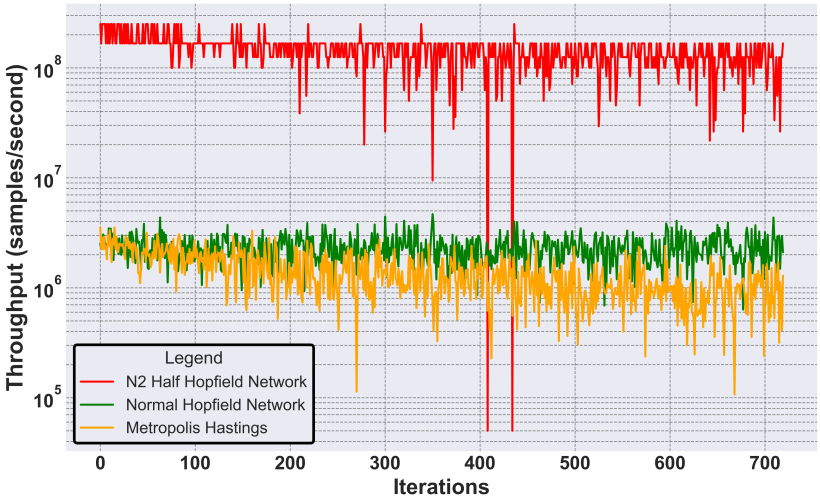

Fig. 30: Comparison throughput literature

It is visible that the single spin Hopfield Network initially has a similar throughput compared to Metropolis Hastings but over time Metropolis Hastings falls off. The average is **1.37 megasamples per second** compared to to the 2.3 megsamples per second of the single spin Hopfield Network, which therefore is **1.67x** faster. Lastly, another paper established a comparison of probalistic hardware accelerators. Here, "time per sweep(ns)" is used, wich measures the duration required once all spins have touched like in the other paper.[261] A visual representation of the probabilistic accelerators can be seen in following figure31:

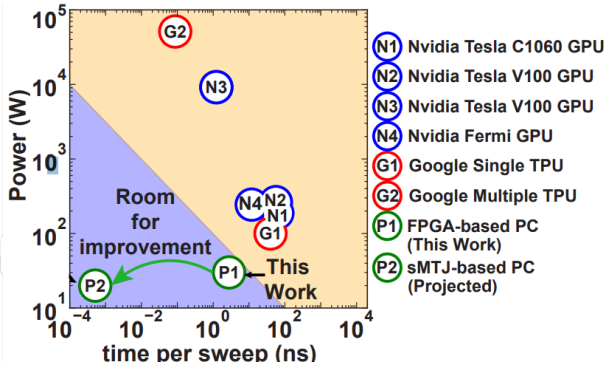

Fig. 31: Comparison throughput with Metropolis Hastings

With a clock cycle of 2 ns and the script used to ensure all 900 neurons in the network are updated, which requires 11.2 iterations, the resulting mem-HNN time per sweep is **22.4ns**. This is **slower** than the FPGA-based PC (5.83ns) and Google's multiple TPU (no exact value). It can be said that the research question is answered for throughput since the performance of the mem-HNN

[261] cf. Aadit/Mohseni, M./Camsari 2023, p. 2

is competitive. Although the time per sweep is competitive and surpasses some probabilistic accelerators it is important to consider that the workloads differ and a direct comparison is not possible requiring future work.

5.3 Energy consumption

Although the computation speed matches other probabilistic accelerators, this model's energy efficiency is distinctive. Energy consumption per sampling iteration ranges from 50 to 20 picojoules, reducing as training progresses. Power requirements start at **22.5mW** and decrease to just below **10mW**, allowing the neural network to complete training with only **6mJ** across 720 iterations. Although it is a simplistic comparison, one might consider, for instance, measuring the time it takes to run RBM training using the Metropolis Hastings algorithm on a CPU. The training on a CPU[262], takes around 30 minutes and has a thermal design power of 15W. Therefore, the training used **27.000Joules**, which consumes **460.000x** more energy compared to the mem-HNN. For a fairer comparison the energy consumption of the mem-HNN regarding the communication, memory and controller would need to be included, which in total makes the result slightly worse but is part of future work. The results in fig.31 display performance for a system with 7200 neurons. To project energy for this neuron counts, it is assumed that power consumption scales linearly with neuron number, due to the implementation which divides into smaller subproblems corresponding to a single small-scale mem-HNN. Hence, the power consumption of the mem-HNN is multiplied by the factor of the neurons $\mathbf{7200/164 = 43}$.[263]. Thus, the estimated power is $\sim$ **1W**, which is between **10x** and **1000x** more effcient than an FPGA-, GPU- and TPU-based sampling methods. In total the mem-HNN has an competitive computing speed but has an significant advantage when comparing the energy consumption. Therefore, Boltzmann Machines can indeed be efficiently implemented on the physics-inspired Hardware accelerator by analog noise injection answering the research question ultimately. Since the aim is to achieve new applications for more sustainable AI models on this ASIC accelerator, this shows there can be large upside potential for running workoads on mem-HNN chips.

5.4 diffusion

Now that the results have been processed and are comparable, the diffusion phase of the DSR framework requires that results are made available to the public. The results and implementation are handed over to Hewlett Packard Labs and are used as bases for further research. The goal is to further develop the complete implementation of the model on the physical hardware accelerator. Furthermore, it is planned to take the results and include them in an upcoming paper that is presented at TechCon, HPE's internal research conference. In addition to that, the results are

[262] `Intel i7-10610U, 1.80GHz, 2304 Mhz, 4 Core(s), 8 Logical Processor(s)`
[263] cf. Aadit/Mohseni, M./Camsari 2023, p. 2

planned to be included and shown at scientific conferences that HPE plans on going in near future. The results will also be shared with HPE's research partners within the context of the mem-HNN. Lastly, this bachelor thesis is submitted to DHBW-Stuttgart for assessment.

6 Critical reflexion and outlook

This chapter concludes the research work by summarizing the key findings and provide a critical reflection on these outcomes. The goal is to retrospectively evaluate the core structure and methodology of this study, discussing how effectively it addressed the posed research question. Additionally, this section aims to illustrate the practical and theoretical consequences of the artifacts and insights developed through this work. By doing so, it underscores the relevance of the study's contributions to both academic research and real-world applications. Lastly, this chpaer offers an outlook on future research, improvements and implications in general.

6.1 Mission of the work

The general mission of the work is the implementation of BMs on a physics-inspired hardware accelerator by analog noise injection. For that purpose, a simulation framework is developed that, that for the first time, enables a detailed study of the performance of such accelerators in AI workloads. This research is motivated by ever increasing energy consumption for artificial intelligence models and stagnating computing speeds.

Therefore, the newly developed mem-HNN ASIC hardware accelerator, currently being developed and studied by HPE, should be further researched to explore its potential in efficiently implementing this neural network during training and inference tasks. This thesis first explains all the relevant concepts. Here, topics like Neural Networks and sustainability is tackled and within these energy-based models, BMs and RBMs are at the center of attention. After discussing neural networks, the focus shifts to the hardware accelerator, providing a comparison of current developments and research that lead to the functionality of the mem-HNN accelerator. Hence, the concept of Hopfield Networks which originates out of physics, but also technical elements like the crossbars are explained to ultimately bring the idea of noise injection to the reader. The noise jenction makes it possible to train and interfer the BMs on the ASIC accelerator.

The theoretical research theory is to be implemented in the practical part of the thesis. This is done by following the scientific DSR process by Österle et al. to create a Simulator Pipeline, which is developed iteratively and consists out of four design and evaluation phases. The design phase itself follows the methodology of prototyping by G.A. Mihram that allows rapid and explorative proceed. Initially, the prototyping process creates a catalog of functions and distributes these across various design phases. The focus lies on using the Hopfield Network as sampling method to train the BM and therefore show the possibility of the realizability on the physics-inspirated mem-HNN accelerator. The implementation of the prototype matches well with the DSR framework and is done icrementally to develop the prototyp further and use the previous output as the new input for the next iterative phase. The last evaluation phase uses the finished prototype with desirable all functionalities as input and aims to meassure the performance of the

successful established Simulator Pipeline. Afterwards it is compared to the established intrinsic baselines of the conventialnal approaches Gibbs sampling and Metropolis Hastings. Specifically, this is done by performing the methodology simulation by Kellner/Madachy/Raffo and setting the focus on the metrics throughput and energy consumption. All the single methodologies were chosen to fit well into the iterative DSR framework in order to best create the desired IT-artifact with a high scientific procedure. The desired result is a functioning representation of the mem-HNN with all the functionalities included, evaluated and meassured performance with additional information about the Hyperparamter tuning.

6.2 Critical reflection on the results and methodology

This thesis represents a significant advancement in the field of energy-efficient AI models by introducing the first-ever Simulator Pipeline designed for BMs. Existing proof of concenpts were taken further and with the succesfully implementation the research question is answered. Furthermore, an intrinsic comparison between the mem-HNN ASIC and the conventional methods of Gibbs Sampling and Metropolis Hastings is established. The results of this thesis, while promising, warrant a thoughtful examination in terms of applied methodologies and meassured results.

Reflection on Results: The introduction of the Simulator Pipeline has not only demonstrated the possibility of the implementation of BMs with a Hopfield Network as sampling method but also the results regarding the energy efficiency and computational throughput are very positive. Therefore, the overall research question has been successfully answered, resulting in a novel tool that has not been explored in previous studies. This tool also offers adaptability for various workloads in future research. In an second part, the evaluation of the energy efficiency has been partially addressed. This is because the metric energy consumption does not include all hardware compontents, already mentioned in chapter 4.6. An more correct energy consumption of the overall solution architecture is part of future work with approximately one year development time. Therefore, the current energy model for the mem-HNN is considered a first attempt but with the generalizable Simulator Pipeline the energy model can be extended trouble-free for example a bus system, controller or digital computer. Still, these initial results suggest that a mem-HNN chip could be more energy efficient (15x), even when compared to sampling methods implemented on specialized hardware acceelrators such as FPGAs. A relevant study comparing electronic with analog hardware reveals that the energy required by the analog solution, even when accounting for currently missing components such as the bus system, memory, and the controller, remains substantially lower than that of its digital counterparts.[264] Although the inclusion of these components might raise energy consumption by about 30%, the analog approach is still estimated to be five times more energy efficient overall, with the additional energy contribution likely being insignificant. Furthermore, the current Simulator Pipeline uses double precision for weights and biases, whereas the resolution would be much less in the mem-HNN chip. In the

[264] cf. Demirkiran et al. 2023, pp. 12–13

energy model, the bit-resolution is considered, but not for the results of the prediction accuracy. Despite this, various studies indicate that the effect of reduced resolution is manageable for a lot of AI workloads and that 5 bit-resolution can still yield sufficient performance.[265] The training of a RBM testcase is limited due to its simplicity. As stated in chapter 2, the ultimate aim is to harness accelerator technology for more powerful BMs. Although current findings suggest that BMs can be implemented with similar performance gains, a more comprehensive study is required as future work.

Reflection on Methodology: Utilizing DSR as the applied research methodology facilitates the incremental and continuous progress of the Simulator Pipeline as IT-artifact. This approach is well-suited for the implementation of the novel Simulator Pipeline especially since it requires agility. Despite the result being specific for the mem-HNN, the DSR procedure can be generalized for other AI-models or other data sets on different ASIC or FPGA accelerators. The protyping methodology enables a rapid and explorative implementation and therefore as well suits the thesis's goal. Although performance is not a criterion for prototyping, the training of the model is conducted on a notebook with a CPU[266], which takes approximately 30-40 minutes per training session. Therefore, optimization emerges as a critical area for further development. Next up, simulations can only provide estimates of the performance of an actual mem-HNN chip. Once the hardware is available real-world results can be different making it essential to build and validate a physical hardware accelerator based on these initial analyses. Differences may arise in energy efficiency, throughput, and prediction accuracy. Still, the functionalities are modelled with great effort and the existing pipeline can potentially be adapted to also interface with a mem-HNN chip later on.

6.3 Extrusion of results for theory and practice

The resulting Simulator Pipeline for the mem-HNN established in this thesis achieved promising results of relevance for the scientific community and practical applications. The intrinsic comparison with other accelerators revealed that the mem-HNN offers competitive throughput while enhancing energy efficiency, thereby highlighting its potential as a transformative tool in the realm of energy-efficient computing. Also, a first estimation is done by reviewing other analog probalistic accelerators and comparing the energy efficiency and throughput which result in promising and competitive estimates. Additionally, as the Simlator Pipeline is generalizable, in the furure different workloads can be chosen as starting point for research with AI-models like BMs. Hence, even greater improvements could be achieved for these workloads. Certainly, the results from the intrinsic comparison provide valuable insights but also bridge a research gap in this novel sampling method. Moreover, they demonstrate the effectiveness of the solution through successful hyperparameter tuning and improved performance. As the tool is developed

[265] cf.Ma et al. 2024, p. 1; cf. *GitHub - Htqin/QuantSR* 2024, p. 1; cf.Rouhani, B. D. et al. 2023, p. 1; cf.Rouhani, B. et al. 2023, p. 1

[266] `Intel i7-10610U, 1.80GHz, 2304 MHz, 4 Cores, 8 Logical Processors`

using scientific methods, it is easy to track why a specific program section, for example, has certain properties. Also it ensures fewer errors after iteratively correcting them. For further research within HPE the results validate that training for BMs is possible and that in a next step this it is planned to research further with other workloads, architectures and changing from RBM to a general BM. Furthermore, the current plan is to use the results of this thesis and include them into an upcoming paper that will be submitted to TechCon, which is an HPE internal research conference. Lastly, the results are also planned to be presented at public research conferences and to HPE's research partners in the mem-HNN research project.

6.4 Outlook

The future directions for the mem-HNN Simulator Pipeline focuses on the improvement of missing functionalities and refining its accuracy. Key topics are the bit-resolution of the ASIC, which is planned to be integrated, while at the same time the energy model is planned to be modified. Currently, the energy consumption of the bus system to the digital computer, the controller and the memory on the mem-HNN are not included in the energy model. The digital computer that updates the weights and possibly require the highest energy consumption is also not modelled due to missing energy values. In general, further research plans to gather a nearly real-world energy consumption of the Simulator Pipeline. Further research will also extend beyond the current use of the RBM to implement the general BM model and focus on more complex workloads. This allows to research the tools robustness and flexibility in order to verify the performance across a broader range of AI tasks and ensuring scalability. Lastly, the planned model should not only support more complex workloads but also streamline the training process to be more time-efficient and enable faster trainings. Overall, the ongoing improvements and research into the mem-HNN Simulator Pipeline are positioned to solidify its status as a cutting-edge tool in the field of energy-efficient AI technologies.

Appendix

List of appendices

Appendix 1: House of Prototyping Guidelines: Prototyping Dimensions

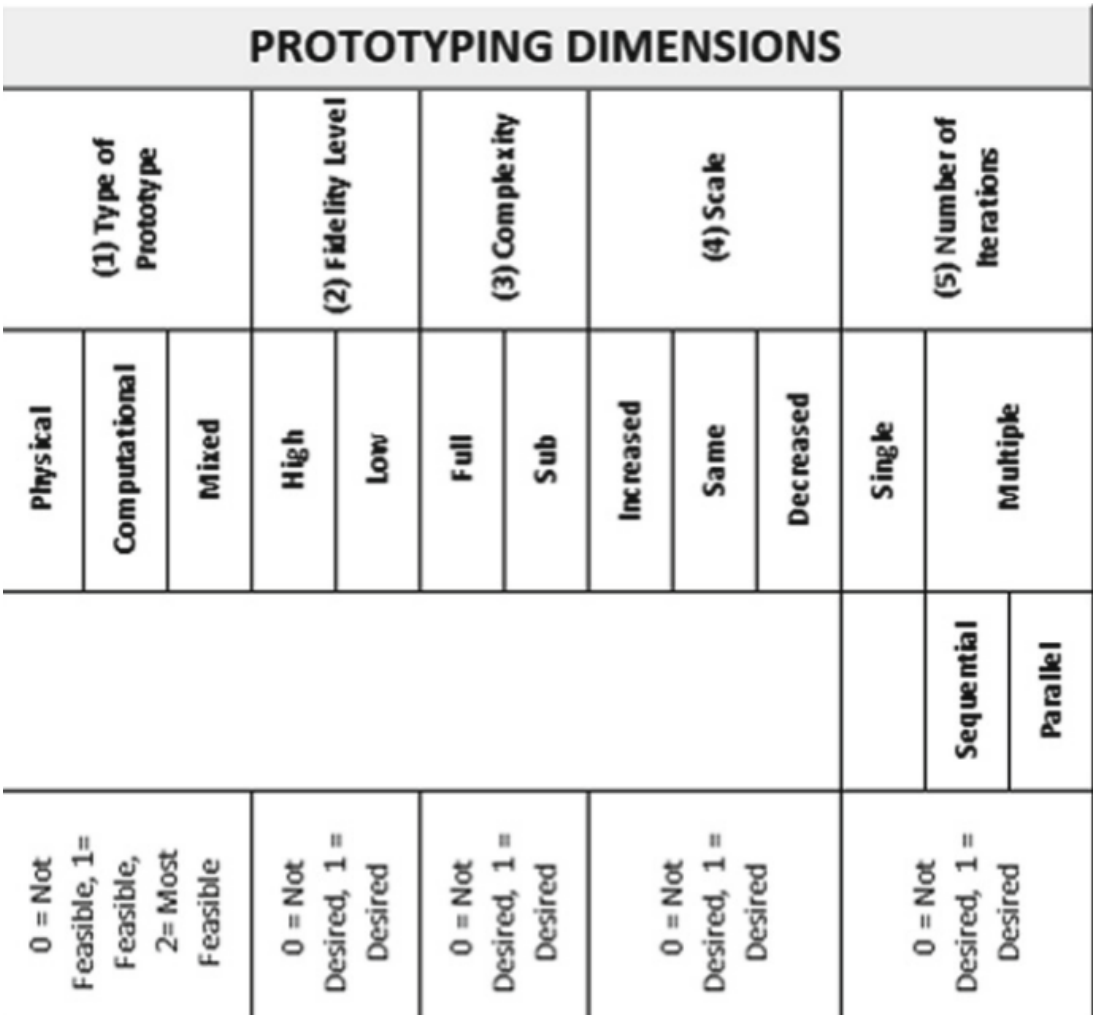

Fig. 32: Prototyping Dimensions to categorize prototypes

Appendix 2: Weight matrix within the Hopfield Network

```
def initialize_weights_and_biases(self, components):
  num_hidden = self.num_hidden_neurons
  num_visible = self.num_visible_neurons

  #Result size:  100,64

  # Initialize a symmetric weight matrix for simplicity
  self.weights = np.zeros((num_hidden + num_visible, num_hidden + num_visible))
```

```
    # Fill in the weights from components for connections between hidden and visible
        layers
    # for connections between hidden and visible layers
    for i in range(num_hidden): # Looping through hidden neurons
        for j in range(num_visible): # Looping through visible neurons
            hidden_index = i
            visible_index = j + num_hidden

            # Additional safeguard:  Ensure 'j' is within the bounds of 'components'
                second dimension
            if j < len(components[0]):
                self.weights[hidden_index, visible_index] = components[i][j]
                self.weights[visible_index, hidden_index] = components[i][j]
            else:
                print(f"Attempted to access components[{i}][{j}], which is out of
                    bounds.")

    return self.weights
```

Appendix 3: Autocorrelation function within the prototype

```
def autocorr(self, x):
        # print("das ist das shape:", x.shape)
        # print("das ist das shape:", x.shape[1])
        average = 60
        leng= 8999-average
        autocorr = np.zeros(leng)

        for i in range(0, leng):
            for k in range (0, average):
                autocorr[i] += np.dot(x[:, k]-np.mean(x[:,k]), x[:,
                    k+i]-np.mean(x[:,k+1]))

        return autocorr / autocorr[0]
```

Appendix 4: Code for Hopfield Networking upate mechanism with possbility for N/2 Half with outliers for all sampling methods

```
    for x in range(self.iterations_per_theta):

        if self.N2_Half == False:
            self.neuron_index = np.random.randint(0, self.size) #pick a random neuron in
                the network
```

```
        # Calculate the weighted sum for the neuron, excluding its own state
        weighted_sum = np.dot(self.weights[self.neuron_index, :], self.configuration)

        self.new_configuration = deepcopy(self.configuration) #copying the old
            configuration to create a new one and update it
        bias = self.bias[self.neuron_index]

        if (weighted_sum + bias + np.random.normal(0, scale=self.scale)) >=
            self.threshold_theta:
            self.new_configuration[self.neuron_index] = 1
        else:
            self.new_configuration[self.neuron_index] = 0

    else:
        self.neuron_index = np.random.randint(0, 2, self.size) #pick complete random
            neurons in the network, result [0,1,1,0] and so on for size of the
            network
        weighted_sum = np.dot(self.weights[:, :], self.configuration)

        self.new_configuration = deepcopy(self.configuration)
        bias = self.bias

        for i in range(len(self.neuron_index)):

            #updating function comparing against threshold
            if self.neuron_index[i] > 0:
                if (weighted_sum[i] + bias[i] + np.random.normal(0,
                    scale=self.scale)) >= self.threshold_theta:
                    self.new_configuration[i] = 1
                else:
                    self.new_configuration[i] = 0

    self.configuration = deepcopy(self.new_configuration) #Saving the new
        configuration as basic configuration, so for the next iteration it works

    if x >= self.thermalization:
        self.v_neg[:, self.iterationcounter]= self.new_configuration[100:]
        self.h_neg[:, self.iterationcounter]= self.new_configuration[:100]

        self.iterationcounter += 1

    self.activationProbabilityPerNeuronDict[self.bias] =
        self.divide_array_elements(self.summedConfigurations, self.iterationcounter)
    self.bias += 0.025
```

List of references

Aadit, N. A./Mohseni, M./Camsari, K. Y. (2023): Accelerating Adaptive Parallel Tempering with FPGA-based p-Bits. In: *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. 2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits), pp. 1–2. DOI: 10.23919/VLSITechnologyandCir57934.2023.10185207. URL: https://ieeexplore.ieee.org/abstract/document/10185207 (retrieval: 04/10/2024).

Abar, S./Theodoropoulos, G. K./Lemarinier, P./O'Hare, G. M. P. (2017): Agent Based Modelling and Simulation Tools: A Review of the State-of-Art Software. In: *Computer Science Review* 24, pp. 13–33. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2017.03.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013716301198 (retrieval: 04/02/2024).

Ackley, D. H./Hinton, G. E./Sejnowski, T. J. (1985): A Learning Algorithm for Boltzmann Machines. In: *Cognitive Science* 9.1, pp. 147–169. ISSN: 0364-0213. DOI: 10.1016/S0364-0213(85)80012-4. URL: https://www.sciencedirect.com/science/article/pii/S0364021385800124 (retrieval: 02/16/2024).

Adleman, L. M. (1994): Molecular Computation of Solutions to Combinatorial Problems. In: *Science* 266.5187, pp. 1021–1024. DOI: 10.1126/science.7973651. URL: https://www.science.org/doi/10.1126/science.7973651 (retrieval: 05/01/2024).

Ahad, N./Qadir, J./Ahsan, N. (2016): Neural Networks in Wireless Networks: Techniques, Applications and Guidelines. In: *Journal of Network and Computer Applications* 68, pp. 1–27. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2016.04.006. URL: https://www.sciencedirect.com/science/article/pii/S1084804516300492 (retrieval: 02/28/2024).

Ahmad, A./Pasha, M. A. (2020): Optimizing Hardware Accelerated General Matrix-Matrix Multiplication for CNNs on FPGAs. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.11, pp. 2692–2696. ISSN: 1558-3791. DOI: 10.1109/TCSII.2020.2965154. URL: https://ieeexplore.ieee.org/abstract/document/8954788?casa_token=FflZ-99s30MAAAAA:l-4hcDRIsH2x9gRN3bGMy8BAo1nbQbrJEhqZpdRnAR5IJSe2naviSLmKFiAuYV_yuWVlAPPPdb (retrieval: 03/18/2024).

Ahmed, S./Demirel, H. O. (2021): A Prototyping Framework for Human-Centered Product Design: Preliminary Validation Study. In: *Design, User Experience, and Usability: UX Research and Design*. Ed. by Marcelo M. Soares/Elizabeth Rosenzweig/Aaron Marcus. Cham: Springer International Publishing, pp. 3–14. ISBN: 978-3-030-78221-4. DOI: 10.1007/978-3-030-78221-4_1.

Amari, S./Kurata, K./Nagaoka, H. (1992): Information Geometry of Boltzmann Machines. In: *IEEE Transactions on Neural Networks* 3.2, pp. 260–271. ISSN: 1941-0093. DOI: 10.1109/72.125867. URL: https://ieeexplore.ieee.org/abstract/document/125867 (retrieval: 02/16/2024).

Amirsoleimani, A./Alibart, F./Yon, V./Xu, J./Pazhouhandeh, M. R./Ecoffey, S./Beilliard, Y./Genov, R./Drouin, D. (2020): In-Memory Vector-Matrix Multiplication in

Monolithic Complementary Metal–Oxide–Semiconductor-Memristor Integrated Circuits: Design Choices, Challenges, and Perspectives. In: *Advanced Intelligent Systems* 2.11, p. 2000115. ISSN: 2640-4567. DOI: 10.1002/aisy.202000115. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202000115 (retrieval: 02/15/2024).

Anon. (2023): A Closer Look at The Carbon Footprint of ChatGPT. Piktochart. URL: https://piktochart.com/blog/carbon-footprint-of-chatgpt/ (retrieval: 04/28/2024).

– **(2024a)**: Electricity 2024 - Analysis and Forecast to 2026. In.

– **(2024b)**: AI Programs Consume Large Volumes of Scarce Water. URL: https://news.ucr.edu/articles/2023/04/28/ai-programs-consume-large-volumes-scarce-water (retrieval: 04/28/2024).

Archie, J. W. (1985): Statistical Analysis of Heterozygosity Data: Independent Sample Comparisons. In: *Evolution* 39.3, pp. 623–637. ISSN: 1558-5646. DOI: 10.1111/j.1558-5646.1985.tb00399.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1558-5646.1985.tb00399.x (retrieval: 04/16/2024).

ASIC Design Flow for VLSI Engineering Teams [GUIDE] - Xinyx Design (2024). URL: https://www.xinyxdesign.com/resources/asic-design-flow-for-vlsi-engineering-teams/ (retrieval: 04/03/2024).

Babu, P./Parthasarathy, E. (2021): Reconfigurable FPGA Architectures: A Survey and Applications. In: *Journal of The Institution of Engineers (India): Series B* 102.1, pp. 143–156. ISSN: 2250-2114. DOI: 10.1007/s40031-020-00508-y. URL: https://doi.org/10.1007/s40031-020-00508-y (retrieval: 03/20/2024).

Bai, G./Chai, Z./Ling, C./Wang, S./Lu, J./Zhang, N./Shi, T./Yu, Z./Zhu, M./Zhang, Y./Yang, C./Cheng, Y./Zhao, L. (2024): Beyond Efficiency: A Systematic Survey of Resource-Efficient Large Language Models. DOI: 10.48550/arXiv.2401.00625. arXiv: 2401.00625 [cs]. URL: http://arxiv.org/abs/2401.00625 (retrieval: 02/23/2024). preprint.

Baischer, L./Wess, M./TaheriNejad, N. (2021): Learning on Hardware: A Tutorial on Neural Network Accelerators and Co-Processors. DOI: 10.48550/arXiv.2104.09252. arXiv: 2104.09252 [cs]. URL: http://arxiv.org/abs/2104.09252 (retrieval: 03/18/2024). preprint.

Barra, A./Bernacchia, A./Santucci, E./Contucci, P. (2012): On the Equivalence of Hopfield Networks and Boltzmann Machines. In: *Neural Networks* 34, pp. 1–9. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2012.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608012001608 (retrieval: 02/16/2024).

Baskerville, R./Baiyere, A./Gregor, S./Hevner, A./Rossi, M. (2018): Design Science Research Contributions: Finding a Balance between Artifact and Theory. In: *Journal of the Association for Information Systems* 19.5. ISSN: 1536-9323. URL: https://aisel.aisnet.org/jais/vol19/iss5/3.

Beichl, I./Sullivan, F. (2000): The Metropolis Algorithm. In: *Computing in Science & Engineering* 2.1, pp. 65–69. ISSN: 1558-366X. DOI: 10.1109/5992.814660. URL: https://ieeexplore.ieee.org/document/814660 (retrieval: 02/27/2024).

Belletti, F./Cotallo, M./Cruz, A./Fernandez, L. A./Gordillo-Guerrero, A./Guidetti, M./Maiorano, A./Mantovani, F./Marinari, E./Martin-Mayor, V./Munoz-Sudupe,

A./Navarro, D./Parisi, G./Perez-Gaviro, S./Rossi, M./Ruiz-Lorenzo, J. J./Schifano, S. F./Sciretti, D./Tarancon, A./Tripiccione, R./Velasco, J. L./Yllanes, D./Zanier, G. (2009): Janus: An FPGA-Based System for High-Performance Scientific Computing. In: *Computing in Science & Engineering* 11.1, pp. 48–58. ISSN: 1558-366X. DOI: 10.1109/MCSE.2009.11. URL: https://ieeexplore.ieee.org/document/4720223 (retrieval: 04/10/2024).

Bjarnason, E./Lang, F./Mjöberg, A. (2021): A Model of Software Prototyping Based on a Systematic Map. In: *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ESEM '21. New York, NY, USA: Association for Computing Machinery, pp. 1–11. ISBN: 978-1-4503-8665-4. DOI: 10.1145/3475716.3475772. URL: https://dl.acm.org/doi/10.1145/3475716.3475772 (retrieval: 04/01/2024).

Böhm, F./Alonso-Urquijo, D./Verschaffelt, G./Van der Sande, G. (2022a): Noise-Injected Analog Ising Machines Enable Ultrafast Statistical Sampling and Machine Learning. In: *Nature Communications* 13.1 (1), p. 5847. ISSN: 2041-1723. DOI: 10.1038/s41467-022-33441-3. URL: https://www.nature.com/articles/s41467-022-33441-3 (retrieval: 02/15/2024).

– **(2022b)**: Noise-Injected Analog Ising Machines Enable Ultrafast Statistical Sampling and Machine Learning. In: *Nature Communications* 13.1 (1), p. 5847. ISSN: 2041-1723. DOI: 10.1038/s41467-022-33441-3. URL: https://www.nature.com/articles/s41467-022-33441-3 (retrieval: 02/15/2024).

Boutros, A./Betz, V. (2021): FPGA Architecture: Principles and Progression. In: *IEEE Circuits and Systems Magazine* 21.2, pp. 4–29. ISSN: 1558-0830. DOI: 10.1109/MCAS.2021.3071607. URL: https://ieeexplore.ieee.org/abstract/document/9439568 (retrieval: 03/20/2024).

Cai, F./Kumar, Suhas/Van Vaerenbergh, T./Liu, R./Li, C./Yu, S./Xia, Q./Yang, J. J./Beausoleil, R./Lu, W./Strachan, J. P. (2019): Harnessing Intrinsic Noise in Memristor Hopfield Neural Networks for Combinatorial Optimization. DOI: 10.48550/arXiv.1903.11194. arXiv: 1903.11194 [cs]. URL: http://arxiv.org/abs/1903.11194 (retrieval: 02/15/2024). preprint.

Cai, F./Kumar, Suhas/Van Vaerenbergh, T./Sheng, X./Liu, R./Li, C./Liu, Z./Foltin, M./Yu, S./Xia, Q./Yang, J. J./Beausoleil, R./Lu, W. D./Strachan, J. P. (2020): Power-Efficient Combinatorial Optimization Using Intrinsic Noise in Memristor Hopfield Neural Networks. In: *Nature Electronics* 3.7, pp. 409–418. ISSN: 2520-1131. DOI: 10.1038/s41928-020-0436-6. URL: https://www.nature.com/articles/s41928-020-0436-6 (retrieval: 03/21/2024).

Chang, C.-F./Chen, J.-Y./Huang, C.-W./Chiu, C.-H./Lin, T.-Y./Yeh, P.-H./Wu, W.-W. (2017): Direct Observation of Dual-Filament Switching Behaviors in Ta2O5-Based Memristors. In: *Small* 13.15, p. 1603116. ISSN: 1613-6829. DOI: 10.1002/smll.201603116. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smll.201603116 (retrieval: 03/22/2024).

Charitha, C./Devi Chaitrasree, A./Varma, P. C./Lakshmi, C. (2022): Type-II Diabetes Prediction Using Machine Learning Algorithms. In: *2022 International Conference on Computer Communication and Informatics (ICCCI)*. 2022 International Conference on Computer Communication and Informatics (ICCCI), pp. 1–5. DOI: 10.1109/ICCCI54379.2022.9740844. URL: https://ieeexplore.ieee.org/document/9740844 (retrieval: 04/12/2024).

Chen, Y./Zhang, M./Shen, X. (2021): Application of Voltage Comparator and Its Multisim Simulation. In: *2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*. 2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), pp. 28–30. DOI: 10.1109/ICPICS52425.2021.9524134. URL: https://ieeexplore.ieee.org/abstract/document/9524134 (retrieval: 03/25/2024).

Chien, A. A./Lin, L./Nguyen, H./Rao, V./Sharma, T./Wijayawardana, R. (2023): Reducing the Carbon Impact of Generative AI Inference (Today and in 2035). In: *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. HotCarbon '23: 2nd Workshop on Sustainable Computer Systems. Boston MA USA: ACM, pp. 1–7. ISBN: 9798400702426. DOI: 10.1145/3604930.3605705. URL: https://dl.acm.org/doi/10.1145/3604930.3605705 (retrieval: 04/28/2024).

Cichy, R. M./Kaiser, D. (2019): Deep Neural Networks as Scientific Models. In: *Trends in Cognitive Sciences* 23.4, pp. 305–317. ISSN: 1364-6613, 1879-307X. DOI: 10.1016/j.tics.2019.01.009. pmid: 30795896. URL: https://www.cell.com/trends/cognitive-sciences/abstract/S1364-6613(19)30034-8 (retrieval: 02/23/2024).

Dally, W. J./Keckler, S. W./Kirk, D. B. (2021): Evolution of the Graphics Processing Unit (GPU). In: *IEEE Micro* 41.6, pp. 42–51. ISSN: 1937-4143. DOI: 10.1109/MM.2021.3113475. URL: https://ieeexplore.ieee.org/abstract/document/9623445 (retrieval: 03/20/2024).

Dario Amodei/Danny Hernandez (2024): AI and Compute. URL: https://openai.com/research/ai-and-compute (retrieval: 02/15/2024).

Demirkiran, C./Eris, F./Wang, G./Elmhurst, J./Moore, N./Harris, N. C./Basumallik, A./Reddi, V. J./Joshi, A./Bunandar, D. (2023): An Electro-Photonic System for Accelerating Deep Neural Networks. In: *ACM Journal on Emerging Technologies in Computing Systems* 19.4, pp. 1–31. ISSN: 1550-4832, 1550-4840. DOI: 10.1145/3606949. arXiv: 2109.01126 [cs]. URL: http://arxiv.org/abs/2109.01126 (retrieval: 04/25/2024).

Discrete and Continuous Models and Applied Computational Science (2024). Discrete/Continuous Models/Applied Computational Science. URL: http://journals.rudn.ru/miph (retrieval: 04/08/2024).

Dramsch, J. S. (2020): 'Chapter One - 70 Years of Machine Learning in Geoscience in Review'. In: *Advances in Geophysics*. Ed. by Ben Moseley/Lion Krischer. Vol. 61. Machine Learning in Geosciences. Elsevier, pp. 1–55. DOI: 10.1016/bs.agph.2020.08.002. URL: https://www.sciencedirect.com/science/article/pii/S0065268720300054 (retrieval: 02/28/2024).

Du, Y./Lin, T./Mordatch, I. (2021): Model Based Planning with Energy Based Models. DOI: 10.48550/arXiv.1909.06878. arXiv: 1909.06878 [cs, stat]. URL: http://arxiv.org/abs/1909.06878 (retrieval: 02/19/2024). preprint.

Du, Y./Mordatch, I. (2020): Implicit Generation and Generalization in Energy-Based Models. DOI: 10.48550/arXiv.1903.08689. arXiv: 1903.08689 [cs, stat]. URL: http://arxiv.org/abs/1903.08689 (retrieval: 02/23/2024). preprint.

Durstewitz, D./Koppe, G./Meyer-Lindenberg, A. (2019): Deep Neural Networks in Psychiatry. In: *Molecular Psychiatry* 24.11 (11), pp. 1583–1598. ISSN: 1476-5578. DOI: 10.1038/s41380-019-0365-9. URL: https://www.nature.com/articles/s41380-019-0365-9 (retrieval: 02/23/2024).

Ebert, C. (2008): Systematisches Requirements Engineering Und Management - Anforderungen Ermitteln, Spezifizieren, Analysieren Und Verwalten (2. Aufl.). ISBN: 978-3-89864-546-1.

Fahlman, S./Hinton, G./Sejnowski, T. (1983): Massively Parallel Architectures for AI: NETL, Thistle, and Boltzmann Machines., p. 113. 109 pp.

Fischer, A./Igel, C. (2012): An Introduction to Restricted Boltzmann Machines. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Ed. by Luis Alvarez/Marta Mejail/Luis Gomez/Julio Jacobo. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 14–36. ISBN: 978-3-642-33275-3. DOI: 10.1007/978-3-642-33275-3_2.

Gawlikowski, J./Tassi, C. R. N./Ali, M./Lee, J./Humt, M./Feng, J./Kruspe, A./Triebel, R./Jung, P./Roscher, R./Shahzad, M./Yang, W./Bamler, R./Zhu, X. X. (2023): A Survey of Uncertainty in Deep Neural Networks. In: *Artificial Intelligence Review* 56.1, pp. 1513–1589. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10562-9. URL: https://doi.org/10.1007/s10462-023-10562-9 (retrieval: 02/23/2024).

George, A. S./George, A. S. H./Martin, A. S. G. (2023): The Environmental Impact of AI: A Case Study of Water Consumption by Chat GPT. In: *Partners Universal International Innovation Journal* 1.2 (2), pp. 97–104. ISSN: 2583-9675. DOI: 10.5281/zenodo.7855594. URL: https://puiij.com/index.php/research/article/view/39 (retrieval: 04/28/2024).

GitHub - Htqin/QuantSR (2024): GitHub - Htqin/QuantSR: This Project Is the Official Implementation of Our Accepted NeurIPS 2023 (Spotlight) Paper QuantSR: Accurate Low-bit Quantization for Efficient Image Super-Resolution. GitHub. URL: https://github.com/htqin/QuantSR (retrieval: 04/19/2024).

Gm, H./Gourisaria, M. K./Pandey, M./Rautaray, S. S. (2020): A Comprehensive Survey and Analysis of Generative Models in Machine Learning. In: *Computer Science Review* 38, p. 100285. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2020.100285. URL: https://www.sciencedirect.com/science/article/pii/S1574013720303853 (retrieval: 03/08/2024).

Gregor, S./Hevner, A. R. (2013): Positioning and Presenting Design Science Research for Maximum Impact. In: *MIS Quarterly* 37.2, pp. 337–355. ISSN: 0276-7783. JSTOR: 43825912. URL: https://www.jstor.org/stable/43825912 (retrieval: 03/28/2024).

Gustafsson, F. K./Danelljan, M./Bhat, G./Schön, T. B. (2020): Energy-Based Models for Deep Probabilistic Regression. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi/Horst Bischof/Thomas Brox/Jan-Michael Frahm. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 325–343. ISBN: 978-3-030-58565-5. DOI: 10.1007/978-3-030-58565-5_20.

Helmenstine, A. (2022): How Many Atoms Are in the World? Science Notes/Projects. URL: https://sciencenotes.org/how-many-atoms-are-in-the-world/ (retrieval: 02/21/2024).

Hevner, A. R./March, S. T./Park, J./Ram, S. (2004): Design Science in Information Systems Research. In: *MIS Quarterly* 28.1, pp. 75–105. ISSN: 0276-7783. DOI: 10.2307/25148625. JSTOR: 25148625. URL: https://www.jstor.org/stable/25148625 (retrieval: 03/28/2024).

Hintemann, R./Hinterholzer, S. (2022): Data Centers 2021: Data Center Boom in Germany Continues - Cloud Computing Drives the Growth of the Data Center Industry and Its Energy Consumption. DOI: 10.13140/RG.2.2.31826.43207.

Hinton, G. (2014): 'Boltzmann Machines'. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut/Geoffrey I. Webb. Boston, MA: Springer US, pp. 1–7. ISBN: 978-1-4899-7502-7. DOI: 10.1007/978-1-4899-7502-7_31-1. URL: https://link.springer.com/10.1007/978-1-4899-7502-7_31-1 (retrieval: 03/16/2024).

Hinton, G. E. (2012a): 'A Practical Guide to Training Restricted Boltzmann Machines'. In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon/Geneviève B. Orr/Klaus-Robert Müller. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 599–619. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_32. URL: https://doi.org/10.1007/978-3-642-35289-8_32 (retrieval: 02/15/2024).

– **(2012b)**: 'A Practical Guide to Training Restricted Boltzmann Machines'. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon/Geneviève B. Orr/Klaus-Robert Müller. Vol. 7700. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 599–619. ISBN: 978-3-642-35288-1 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_32. URL: http://link.springer.com/10.1007/978-3-642-35289-8_32 (retrieval: 02/15/2024).

Hizzani, M./Heittmann, A./Hutchinson, G./Dobrynin, D./Van Vaerenbergh, T./Bhattacharya, T./Renaudineau, A./Strukov, D./Strachan, J. P. (2023): Memristor-Based Hardware and Algorithms for Higher-Order Hopfield Optimization Solver Outperforming Quadratic Ising Machines. DOI: 10.48550/arXiv.2311.01171. arXiv: 2311.01171 [cs]. URL: http://arxiv.org/abs/2311.01171 (retrieval: 02/15/2024). preprint.

Holzweißig, K. (2017): Wissenschaftliches Arbeiten. Leanpub. URL: https://leanpub.next/wawinfo (retrieval: 04/26/2024).

Hopfield, J. J. (1982): Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In: *Proceedings of the National Academy of Sciences* 79.8, pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554. URL: https://www.pnas.org/doi/10.1073/pnas.79.8.2554 (retrieval: 02/19/2024).

Hsu, A./Griffiths, T. (2010): Effects of Generative and Discriminative Learning on Use of Category Variability. In: *Proceedings of 32nd Annual Conference of the Cognitive Science Society*.

Hu, Y./Liu, Y./Liu, Z. (2022): A Survey on Convolutional Neural Network Accelerators: GPU, FPGA and ASIC. In: *2022 14th International Conference on Computer Research and Development (ICCRD)*. 2022 14th International Conference on Computer Research and Development (ICCRD), pp. 100–107. DOI: 10.1109/ICCRD54409.2022.9730377. URL: https://ieeexplore.ieee.org/abstract/document/9730377 (retrieval: 03/19/2024).

Huembeli, P./Arrazola, J. M./Killoran, N./Mohseni, M./Wittek, P. (2022): The Physics of Energy-Based Models. In: *Quantum Machine Intelligence* 4.1, p. 1. ISSN: 2524-4914. DOI: 10.1007/s42484-021-00057-7. URL: https://doi.org/10.1007/s42484-021-00057-7 (retrieval: 02/19/2024).

Ising, E. (1925): Beitrag zur Theorie des Ferromagnetismus. In: *Zeitschrift für Physik* 31.1, pp. 253–258. ISSN: 0044-3328. DOI: 10.1007/BF02980577. URL: https://doi.org/10.1007/BF02980577 (retrieval: 03/21/2024).

Izadkhah, H. (2022): 'P, NP, NP-Complete, and NP-Hard Problems'. In: *Problems on Algorithms: A Comprehensive Exercise Book for Students in Software Engineering*. Ed. by Habib Izadkhah. Cham: Springer International Publishing, pp. 497–511. ISBN: 978-3-031-17043-0. DOI: 10.1007/978-3-031-17043-0_15. URL: https://doi.org/10.1007/978-3-031-17043-0_15 (retrieval: 03/21/2024).

Jackson, A. (2024): AI Boom Will Cause Data Centre Electricity Demand to Double. URL: https://datacentremagazine.com/data-centres/ai-boom-will-cause-data-centre-electricity-demand-to-double (retrieval: 04/26/2024).

Kellner, M. I./Madachy, R. J./Raffo, D. M. (1999): Software Process Simulation Modeling: Why? What? How? In: *Journal of Systems and Software* 46.2, pp. 91–105. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(99)00003-5. URL: https://www.sciencedirect.com/science/article/pii/S0164121299000035 (retrieval: 04/02/2024).

Larochelle, H./Bengio, Y. (2008): Classification Using Discriminative Restricted Boltzmann Machines. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. New York, NY, USA: Association for Computing Machinery, pp. 536–543. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390224. URL: https://dl.acm.org/doi/10.1145/1390156.1390224 (retrieval: 02/22/2024).

Larochelle, H./Mandel, M./Pascanu, R./Bengio, Y. (2012): Learning Algorithms for the Classification Restricted Boltzmann Machine. In: *The Journal of Machine Learning Research* 13, pp. 643–669.

Lehnert, A./Holzinger, P./Pfenning, S./Müller, R./Reichenbach, M. (2023): Most Resource Efficient Matrix Vector Multiplication on FPGAs. In: *IEEE Access* 11, pp. 3881–3898. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3234622. URL: https://ieeexplore.ieee.org/document/10007836?denied= (retrieval: 03/16/2024).

Li, G./Deng, L./Xu, Y./Wen, C./Wang, W./Pei, J./Shi, L. (2016): Temperature Based Restricted Boltzmann Machines. In: *Scientific Reports* 6.1 (1), p. 19133. ISSN: 2045-2322. DOI: 10.1038/srep19133. URL: https://www.nature.com/articles/srep19133 (retrieval: 02/27/2024).

Lucas, A. (2014): Ising Formulations of Many NP Problems. In: *Frontiers in Physics* 2. ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. URL: https://www.frontiersin.org/articles/10.3389/fphy.2014.00005 (retrieval: 03/22/2024).

Luccioni, A. S./Jernite, Y./Strubell, E. (2023): Power Hungry Processing: Watts Driving the Cost of AI Deployment? DOI: 10.48550/arXiv.2311.16863. arXiv: 2311.16863 [cs]. URL: http://arxiv.org/abs/2311.16863 (retrieval: 02/15/2024). preprint.

Luqi, L./Steigerwald, R. (1992): Rapid Software Prototyping. In: *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*. Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences. Vol. ii, 470–479 vol.2. DOI: 10.1109/HICSS.1992.183261. URL: https://ieeexplore.ieee.org/abstract/document/183261 (retrieval: 04/02/2024).

Ma, S./Wang, H./Ma, L./Wang, L./Wang, W./Huang, S./Dong, L./Wang, R./Xue, J./Wei, F. (2024): The Era of 1-Bit LLMs: All Large Language Models Are in 1.58 Bits. arXiv: 2402.17764 [cs]. URL: http://arxiv.org/abs/2402.17764 (retrieval: 04/19/2024). preprint.

MacKay, D. J. C. (2003): Information Theory, Inference and Learning Algorithms. Cambridge University Press. 694 pp. ISBN: 978-0-521-64298-9. Google Books: AKuMj4PN_EMC.

Mahmoodi, M. R./Prezioso, M./Strukov, D. B. (2019): Versatile Stochastic Dot Product Circuits Based on Nonvolatile Memories for High Performance Neurocomputing and Neurooptimization. In: *Nature Communications* 10.1, p. 5113. ISSN: 2041-1723. DOI: 10.1038/s41467-019-13103-7. URL: https://www.nature.com/articles/s41467-019-13103-7 (retrieval: 03/26/2024).

Mall, P. K./Singh, P. K./Srivastav, S./Narayan, V./Paprzycki, M./Jaworska, T./Ganzha, M. (2023): A Comprehensive Review of Deep Neural Networks for Medical Image Processing: Recent Developments and Future Opportunities. In: *Healthcare Analytics* 4, p. 100216. ISSN: 2772-4425. DOI: 10.1016/j.health.2023.100216. URL: https://www.sciencedirect.com/science/article/pii/S2772442523000837 (retrieval: 02/23/2024).

Marinó, G. C./Petrini, A./Malchiodi, D./Frasca, M. (2023): Deep Neural Networks Compression: A Comparative Survey and Choice Recommendations. In: *Neurocomputing* 520, pp. 152–170. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2022.11.072. URL: https://www.sciencedirect.com/science/article/pii/S0925231222014643 (retrieval: 02/23/2024).

Metropolis, N./Rosenbluth, A. W./Rosenbluth, M. N./Teller, A. H./Teller, E. (1953): Equation of State Calculations by Fast Computing Machines. In: *The Journal of Chemical Physics* 21.6, pp. 1087–1092. ISSN: 0021-9606. DOI: 10.1063/1.1699114. URL: https://doi.org/10.1063/1.1699114 (retrieval: 02/27/2024).

Mihram, G. A. (1976): Simulation Methodology. In: *Theory and Decision* 7.1, pp. 67–94. ISSN: 1573-7187. DOI: 10.1007/BF00141103. URL: https://doi.org/10.1007/BF00141103 (retrieval: 04/02/2024).

Mocanu, D. C./Mocanu, E./Nguyen, P. H./Gibescu, M./Liotta, A. (2016): A Topological Insight into Restricted Boltzmann Machines. In: *Machine Learning* 104.2, pp. 243–270. ISSN: 1573-0565. DOI: 10.1007/s10994-016-5570-z. URL: https://doi.org/10.1007/s10994-016-5570-z (retrieval: 02/22/2024).

Mohseni, N./McMahon, P. L./Byrnes, T. (2022a): Ising Machines as Hardware Solvers of Combinatorial Optimization Problems. DOI: 10.48550/arXiv.2204.00276. arXiv: 2204.00276 [physics, physics:quant-ph]. URL: http://arxiv.org/abs/2204.00276 (retrieval: 02/15/2024). preprint.

Mohseni, N./McMahon, P. L./Byrnes, T. (2022b): Ising Machines as Hardware Solvers of Combinatorial Optimization Problems. In: *Nature Reviews Physics* 4.6, pp. 363–379. ISSN: 2522-5820. DOI: 10.1038/s42254-022-00440-8. URL: https://www.nature.com/articles/s42254-022-00440-8 (retrieval: 03/22/2024).

Nazm Bojnordi, M./Ipek, E. (2016): Memristive Boltzmann Machine: A Hardware Accelerator for Combinatorial Optimization and Deep Learning, p. 13. 1 p. DOI: 10.1109/HPCA.2016.7446049.

Nelson, S. D./Fiol, G. D./Hanseler, H./Crouch, B. I./Cummins, M. R. (2016): Software Prototyping. In: *Applied Clinical Informatics* 07.1, pp. 22–32. ISSN: 1869-0327. DOI: 10.4338/ACI-2015-07-CR-0091. URL: http://www.thieme-connect.de/DOI/DOI?10.4338/ACI-2015-07-CR-0091 (retrieval: 04/02/2024).

Oesterle, H./Becker, J./Hess, T./Karagiannis, D./Krcmar, H./Loos, P./Mertens, P./Oberweis, A./Sinz, E. (2010): Memorandum Zur Gestaltungsorientierten Wirtschaftsinformatik. In: *http://www.alexandria.unisg.ch/Publikationen/71074* 62. DOI: 10.1007/BF03372838.

Ortega-Zamorano, F./Montemurro, M. A./Cannas, S. A./Jerez, J. M./Franco, L. (2016): FPGA Hardware Acceleration of Monte Carlo Simulations for the Ising Model. In: *IEEE Transactions on Parallel and Distributed Systems* 27.9, pp. 2618–2627. ISSN: 1045-9219. DOI: 10.1109/TPDS.2015.2505725. arXiv: 1602.03016 [physics]. URL: http://arxiv.org/abs/1602.03016 (retrieval: 04/10/2024).

Österle, H./Otto, B. (2010): Konsortialforschung. In: *WIRTSCHAFTSINFORMATIK* 52.5, pp. 273–285. ISSN: 1861-8936. DOI: 10.1007/s11576-010-0238-y. URL: https://doi.org/10.1007/s11576-010-0238-y (retrieval: 03/29/2024).

Patrón, A./Chepelianskii, A. D./Prados, A./Trizac, E. (2024): On the Optimal Relaxation Rate for the Metropolis Algorithm in One Dimension. DOI: 10.48550/arXiv.2402.11267. arXiv: 2402.11267 [cond-mat, physics:math-ph]. URL: http://arxiv.org/abs/2402.11267 (retrieval: 02/27/2024). preprint.

Peccerillo, B./Mannino, M./Mondelli, A./Bartolini, S. (2022): A Survey on Hardware Accelerators: Taxonomy, Trends, Challenges, and Perspectives. In: *Journal of Systems Architecture* 129, p. 102561. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2022.102561. URL: https://www.sciencedirect.com/science/article/pii/S1383762122001138 (retrieval: 03/19/2024).

Peffers, K./Tuunanen, T./Rothenberger, M. A./Chatterjee, S. (2007): A Design Science Research Methodology for Information Systems Research. In: *Journal of Management Information Systems* 24.3, pp. 45–77. ISSN: 0742-1222, 1557-928X. DOI: 10.2753/MIS0742-1222240302. URL: https://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302 (retrieval: 03/29/2024).

Powertop - ArchWiki (2024). URL: https://wiki.archlinux.org/title/powertop (retrieval: 04/30/2024).

Ramsauer, H./Schäfl, B./Lehner, J./Seidl, P./Widrich, M./Adler, T./Gruber, L./Holzleitner, M./Pavlović, M./Sandve, G. K./Greiff, V./Kreil, D./Kopp, M./Klambauer, G./Brandstetter, J./Hochreiter, S. (2021): Hopfield Networks Is All You Need.

DOI: 10.48550/arXiv.2008.02217. arXiv: 2008.02217 [cs, stat]. URL: http://arxiv.org/abs/2008.02217 (retrieval: 02/28/2024). preprint.

Rao, R. (2024): The Ultimate Guide to ASIC Design: From Concept to Production. URL: https://www.wevolver.com/article/the-ultimate-guide-to-asic-design-from-concept-to-production,%20https://www.wevolver.com/article/the-ultimate-guide-to-asic-design-from-concept-to-production (retrieval: 04/03/2024).

Raschka, S./Patterson, J./Nolet, C. (2020): Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. In: *Information* 11.4 (4), p. 193. ISSN: 2078-2489. DOI: 10.3390/info11040193. URL: https://www.mdpi.com/2078-2489/11/4/193 (retrieval: 04/08/2024).

Restricted Boltzmann Machine Features for Digit Classification (2024). scikit-learn. URL: https://scikit-learn/stable/auto_examples/neural_networks/plot_rbm_logistic_classification.html (retrieval: 04/10/2024).

Robert, C. P. (2016): The Metropolis-Hastings Algorithm. DOI: 10.48550/arXiv.1504.01896. arXiv: 1504.01896 [stat]. URL: http://arxiv.org/abs/1504.01896 (retrieval: 02/27/2024). preprint.

Rosenthal, S. (2009): Optimal Proposal Distributions and Adaptive MCMC. In: *Handbook of Markov Chain Monte Carlo*.

Rouhani, B./Zhao, R./Elango, V./Shafipour, R./Hall, M./Mesmakhosroshahi, M./More, A./Melnick, L./Golub, M./Varatkar, G./Shao, L./Kolhe, G./Melts, D./Klar, J./L'Heureux, R./Perry, M./Burger, D./Chung, E./Deng, Z./Naghshineh, S./Park, J./Naumov, M. (2023): With Shared Microexponents, A Little Shifting Goes a Long Way. arXiv: 2302.08007 [cs]. URL: http://arxiv.org/abs/2302.08007 (retrieval: 04/19/2024). preprint.

Rouhani, B. D./Zhao, R./More, A./Hall, M./Khodamoradi, A./Deng, S./Choudhary, D./Cornea, M./Dellinger, E./Denolf, K./Dusan, S./Elango, V./Golub, M./Heinecke, A./James-Roxby, P./Jani, D./Kolhe, G./Langhammer, M./Li, A./Melnick, L./Mesmakhosroshahi, M./Rodriguez, A./Schulte, M./Shafipour, R./Shao, L./Siu, M./Dubey, P./Micikevicius, P./Naumov, M./Verrilli, C./Wittig, R./Burger, D./Chung, E. (2023): Microscaling Data Formats for Deep Learning. arXiv: 2310.10537 [cs]. URL: http://arxiv.org/abs/2310.10537 (retrieval: 04/19/2024). preprint.

Salakhutdinov, R./Hinton, G. (2009): Deep Boltzmann Machines. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. PMLR, pp. 448–455. URL: https://proceedings.mlr.press/v5/salakhutdinov09a.html (retrieval: 02/16/2024).

Sarhadi, P./Yousefpour, S. (2015): State of the Art: Hardware in the Loop Modeling and Simulation with Its Applications in Design, Development and Implementation of System and Control Software. In: *International Journal of Dynamics and Control* 3.4, pp. 470–479. ISSN: 2195-2698. DOI: 10.1007/s40435-014-0108-3. URL: https://doi.org/10.1007/s40435-014-0108-3 (retrieval: 04/02/2024).

Schlamminger, S. (2014): A Cool Way to Measure Big G. In: *Nature* 510.7506 (7506), pp. 478–480. ISSN: 1476-4687. DOI: 10.1038/nature13507. URL: https://www.nature.com/articles/nature13507 (retrieval: 02/21/2024).

Singh, S. K./Kumar, Shubham/Mehra, P. S. (2023): Chat GPT & Google Bard AI: A Review. In: *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*. 2023 International Conference on IoT, Communication and Automation Technology (ICICAT), pp. 1–6. DOI: 10.1109/ICICAT57735.2023.10263706. URL: https://ieeexplore.ieee.org/abstract/document/10263706?casa_token=JMHwBzQgxnwAAAAA:70OnfYs5ECetZhuq8D_F3QXyua1Xu65rL0a_Ywve3mch00UAeSsOyVjhWCUvDuBpMX83NAbpUpM (retrieval: 02/23/2024).

Sipola, T./Alatalo, J./Kokkonen, T./Rantonen, M. (2022): Artificial Intelligence in the IoT Era: A Review of Edge AI Hardware and Software. In: *2022 31st Conference of Open Innovations Association (FRUCT)*. 2022 31st Conference of Open Innovations Association (FRUCT), pp. 320–331. DOI: 10.23919/FRUCT54823.2022.9770931. URL: https://ieeexplore.ieee.org/abstract/document/9770931 (retrieval: 03/18/2024).

Sklearn.Datasets.Load_digits (2024). scikit-learn. URL: https://scikit-learn/stable/modules/generated/sklearn.datasets.load_digits.html (retrieval: 04/10/2024).

Specht, D. F. (1990): Probabilistic Neural Networks. In: *Neural Networks* 3.1, pp. 109–118. ISSN: 0893-6080. DOI: 10.1016/0893-6080(90)90049-Q. URL: https://www.sciencedirect.com/science/article/pii/089360809090049Q (retrieval: 03/07/2024).

Sung, C./Hwang, H./Yoo, I. K. (2018): Perspective: A Review on Memristive Hardware for Neuromorphic Computation. In: *Journal of Applied Physics* 124.15, p. 151903. ISSN: 0021-8979. DOI: 10.1063/1.5037835. URL: https://doi.org/10.1063/1.5037835 (retrieval: 02/15/2024).

Supri, B./Rudianto/Abdurohim/Mawadah, B./Ali, H. (2023): Asian Stock Index Price Prediction Analysis Using Comparison of Split Data Training and Data Testing. In: *JEMSI (Jurnal Ekonomi, Manajemen, dan Akuntansi)* 9.4 (4), pp. 1403–1408. ISSN: 2579-5635. DOI: 10.35870/jemsi.v9i4.1339. URL: http://journal.lembagakita.org/index.php/jemsi/article/view/1339 (retrieval: 04/12/2024).

Tanahashi, K./Takayanagi, S./Motohashi, T./Tanaka, S. (2019): Application of Ising Machines and a Software Development for Ising Machines. In: *Journal of the Physical Society of Japan* 88.6, p. 061010. ISSN: 0031-9015. DOI: 10.7566/JPSJ.88.061010. URL: https://journals.jps.jp/doi/full/10.7566/JPSJ.88.061010 (retrieval: 03/21/2024).

Tanaka, A./Tomiya, A. (2017): Towards Reduction of Autocorrelation in HMC by Machine Learning. DOI: 10.48550/arXiv.1712.03893. arXiv: 1712.03893 [cond-mat, physics:hep-lat, stat]. URL: http://arxiv.org/abs/1712.03893 (retrieval: 04/16/2024). preprint.

Tero, A./Takagi, S./Saigusa, T./Ito, K./Bebber, D. P./Fricker, M. D./Yumiki, K./Kobayashi, R./Nakagaki, T. (2010): Rules for Biologically Inspired Adaptive Network Design. In: *Science* 327.5964, pp. 439–442. DOI: 10.1126/science.1177894. URL: https://www.science.org/doi/10.1126/science.1177894 (retrieval: 05/01/2024).

Tomlinson, B./Black, R. W./Patterson, D. J./Torrance, A. W. (2024): The Carbon Emissions of Writing and Illustrating Are Lower for AI than for Humans. In: *Scientific Reports* 14.1, p. 3732. ISSN: 2045-2322. DOI: 10.1038/s41598-024-54271-x. URL: https://www.nature.com/articles/s41598-024-54271-x (retrieval: 04/28/2024).

Upadhya, V./Sastry, P. (2019): An Overview of Restricted Boltzmann Machines. In: *Journal of the Indian Institute of Science* 99. DOI: 10.1007/s41745-019-0102-z.

Uusitalo, L./Lehikoinen, A./Helle, I./Myrberg, K. (2015): An Overview of Methods to Evaluate Uncertainty of Deterministic Models in Decision Support. In: *Environmental Modelling & Software* 63, pp. 24–31. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2014.09.017. URL: https://www.sciencedirect.com/science/article/pii/S1364815214002813 (retrieval: 03/07/2024).

Verdon, G./Marks, J./Nanda, S./Leichenauer, S./Hidary, J. (2019): Quantum Hamiltonian-Based Models and the Variational Quantum Thermalizer Algorithm. arXiv: 1910.02071 [quant-ph]. URL: http://arxiv.org/abs/1910.02071 (retrieval: 02/19/2024). preprint.

Wang, T./Roychowdhury, J. (2017): Oscillator-Based Ising Machine. DOI: 10.48550/arXiv.1709.08102. arXiv: 1709.08102 [physics]. URL: http://arxiv.org/abs/1709.08102 (retrieval: 02/15/2024). preprint.

Wang, X./Yan, L./Zhang, Q. (2021): Research on the Application of Gradient Descent Algorithm in Machine Learning. In: *2021 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. 2021 International Conference on Computer Network, Electronic and Automation (ICCNEA), pp. 11–15. DOI: 10.1109/ICCNEA53019.2021.00014. URL: https://ieeexplore.ieee.org/document/9603742 (retrieval: 03/08/2024).

Wittpahl, V., ed. **(2019)**: Künstliche Intelligenz: Technologie | Anwendung | Gesellschaft. Berlin, Heidelberg: Springer. ISBN: 978-3-662-58041-7 978-3-662-58042-4. DOI: 10.1007/978-3-662-58042-4. URL: http://link.springer.com/10.1007/978-3-662-58042-4 (retrieval: 02/15/2024).

Yao, Z./Gripon, V./Rabbat, M. (2013): A Massively Parallel Associative Memory Based on Sparse Neural Networks. In.

Zhai, S./Cheng, Y./Lu, W./Zhang, Z. (2016): Deep Structured Energy Based Models for Anomaly Detection. In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1100–1109. URL: https://proceedings.mlr.press/v48/zhai16.html (retrieval: 02/19/2024).

Zhang, N./Ding, S./Zhang, J./Xue, Y. (2018): An Overview on Restricted Boltzmann Machines. In: *Neurocomputing* 275, pp. 1186–1199. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.09.065. URL: https://www.sciencedirect.com/science/article/pii/S0925231217315849 (retrieval: 02/15/2024).

Zhou, H./Dong, J./Cheng, J./Dong, W./Huang, C./Shen, Y./Zhang, Q./Gu, M./Qian, C./Chen, H./Ruan, Z./Zhang, X. (2022): Photonic Matrix Multiplication Lights up Photonic Accelerator and Beyond. In: *Light: Science & Applications* 11.1, p. 30. ISSN: 2047-7538. DOI: 10.1038/s41377-022-00717-8. URL: https://www.nature.com/articles/s41377-022-00717-8 (retrieval: 03/18/2024).

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Mein Titel* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum) (Unterschrift)