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 April 19, 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
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
DSR	Design Science Research

List of Figures

List of Tables

1 Einleitung

1.1 Motivation

In the research and development of generative AI models, the computing speed and energy efficiency are increasingly becoming the focus[4] The authors of Open AI confirm, that the growth rate of machine learning models surpassed the growth of efficiency within computerchips. The required computing power of the models double each 3-4 months but the power of computerchips, after Moore's Law double only every 2 years.[5] Focusing on current problems like rising energy consumption of datacenters and the associated greenhouse gas emissions, the search for more efficient solutions is essential for the future. Worldwide energy consumption of datacaenters increase yearly around 20-40%, which means that in 2022 about 1,3% of the total global energy consumption and about 1% of energy-related global greenhouse gas emissions was caused by them.[6] However, it is not clear how large the AI share contributed to the total numbers.

1.2 Problem statement

One approach that is already well known is the use of AI accelerators based on ASICs (application-specific integrated circuits) - i.e. circuits that are used application specific, such as Google TPUs (Tensor Processing Unit).[7] This is useful because the usage of multimodels for discirminating tasks compared to task specific models are more energy intense.[8] One promissing alternative concept in research is the usage on physics-inspired hardware accelerators, that are primarly used for optimization problems because of their ability to solve problems faster and more efficient than GPUs.[9] 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.[10]

Such physics-inspired hardware accelerator 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.[11]

[4] Vgl. Luccioni/Jernite/Strubell 2023, p. 1
[5] Vgl. Dario Amodei/Danny Hernandez 2024, p. 1
[6] Vgl. Hintemann/Hinterholzer 2022, p. 1
[7] Vgl. Wittpahl 2019, p. 39
[8] Vgl. Luccioni/Jernite/Strubell 2023, p. 5
[9] Vgl. Mohseni, N./McMahon/Byrnes 2022a, p. 1
[10] Vgl. Mohseni, N./McMahon/Byrnes 2022a, p. 1
[11] Vgl. Mohseni, N./McMahon/Byrnes 2022a, p. 4

Interesting enough, despite their different applications, the energy function of the hardware accelerator that is used in Ising-machines shows big parallels to those used in Boltzmann Maschine (BM), therefore it can be suggested that Ising-machines could work well for AI.[12] Ising-machines strive to minimize their energy, which is defined by the pairwise interaction of binary variables (Spins).[13] In contrast, BMs are energy-based neuronal networks that are used for classification tasks by acllocating a skalar energy for each configuration of variables. Minimizing the total network energy is therefore equal with the solution of a optimization problem.[14] Current problems with BMs are the high complexity and high requirements for the all-to-allcommunication between the processing units, which causes the implemention on conventional digital computers to be inefficient, but also an inherently slow convergence in certain processes such as simulated annealing.[15] Theese challenges complicate the training and the usage of BMs, especially for large data volumes and complex optimization tasksl.[16] Nevertheless the similaritis 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 only few concepts that exists on how to achieve a implementation of a BM within on a Ising-machine. The paper of the authors Mahdi, Nazm, BojnordiEngin and Engin Ipek is a promissing approach, However it could not be shown how a implementation on a real accelerator chip could function.

With the given background, the following central research question for this theis arises:

1. Can Boltzmann Machines be efficiently implemented on physics-inspired Hardware accelerators by analog noise injection?
    - What is the accuracy of the AI-model on the hardware accelerator?
        - Metric: Prediction accuracy and negative Likelihood
    - Comparison between other hardware accelerators, FPGA, GPU, or CPU within the literature in terms of energy efficiency and computing speed.
        - Metrics: Throughput (Samples/Sec), Energy usage (Energy/Operation)

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

1.3 Objective

The primary objective of this bachelor thesis is the research and extension of a existing physics-inspired hardware accelerator(Ising machine) for the implementation and evaluation of BMs as an energy based AI-model. The aim is to answer the posed research question. In addition to

[12] Vgl. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 10
[13] Vgl. Wang, T./Roychowdhury 2017, p. 1
[14] Vgl. Nazm Bojnordi/Ipek 2016, p. 2
[15] Vgl. Nazm Bojnordi/Ipek 2016, p. 1
[16] Vgl. Nazm Bojnordi/Ipek 2016, p. 2

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 it is necessary to establish a simulator pipeline to the hardware accelerator that translates BM on top of it. The simulator pipeline consists of an existing machine learning library and an existing hardware accelerator that are connected to each other. With the simulator pipeline it needs to be shown 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 used for training and iterference and is comparable to conventional machine learning libraries. 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 can be validated a workload consisting of a standard data set to recognize handwritten digits will be tested. The prediction accuracy, throughput (samples/sec) and the energy consumption (energy/operation) of the Boltzmann machines on the Ising hardware accelerator will be investigated as metrics in order to thereby answering the second part of the posed research questions.

1.4 Research method

Design Science Research

1. **Problemorientierung:** DSR fokussiert auf die Lösung praktischer Probleme, wie die Forschung zur Steigerung der Effizienz und Rechengeschwindigkeit in KI-Modellen.
2. **Artefakt Entwicklung:** Zentral in DSR ist die Entwicklung innovativer Artefakte. Die Arbeit zielt darauf ab, ein solches Artefakt in Form des physikinspirierten Hardwarebeschleunigers weiterzuentwickeln und für KI-Modelle einzusetzen.
3. **Iterative Evaluation:** Durch die iterative Vorgehensweise in DSR kann die Ausarbeitung der Lösung fortlaufend verbessert und angepasst werden, was für die Entwicklung und Optimierung von KI-Systemen entscheidend ist (ebenfalls das Konzept).
4. **Beitrag zur Wissensbasis und Praxisrelevanz:** DSR unterstützt die Generierung neuer Erkenntnisse und stellt sicher, dass Forschungsergebnisse sowohl theoretisch fundiert als auch praktisch anwendbar sind, was mit den Zielen Ihres Projekts im Einklang steht. Untermethodik könnte hierbei eine Simulation sein. Variabel, je nach Verlauf der Forschung.

1.5 Aufbau der Arbeit

2 Aktueller Stand der Forschung und Praxis

2.1 Ressourcenverbrauch bei KI-Modellen

2.1.1 Ressourcenverbrauch bei KI-Modellen

substantial challenges in high consumption of computational, memory, energy, and financial resources, especially in environments with limited resource capabilities[17]

Nachhaltigkeit

Stromverbrauch

Rechenleistung begrenzt, KI-Modelle wachsen schneller als verfügbare Leistung

2.2 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.[18] Therefore, it is fair to say that neural networks are part of various important applications.[19] Particularly in the last two years, artificial intelligence has also garnered widespread interest from the public, especially regarding chatbots like ChatGPT and Google Bard.[20] An important feature of a neural network-based system that are inspired by our brain, is that they can learn and adapt to data.[21]

Internally, neural networks are computational models that consist of many simple processing units, called neurons that work together in parallel often structured within interconnected layers.[22] They consist out of a network architecture, which describes the layout and how the neurons are wired. Secondly, they have a optimization function which specifies the goals persued in the learning process.[23] Lastly, there is a training algorithm that varies all of the hyperparameters,

[17] cf. Bai et al. 2024, pp. 1–2
[18] cf. Cichy/Kaiser 2019, p. 305
[19] cf. Gawlikowski et al. 2023, p. 1513
[20] cf. Singh/Kumar, Shubham/Mehra 2023, pp. 1–2
[21] cf. Cichy/Kaiser 2019, p. 305
[22] cf. Cichy/Kaiser 2019, p. 305
[23] cf. Durstewitz/Koppe/Meyer-Lindenberg 2019, p. 1583

like connection strengths between neurons, training iterations, the learning rate, etc..[24] 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.

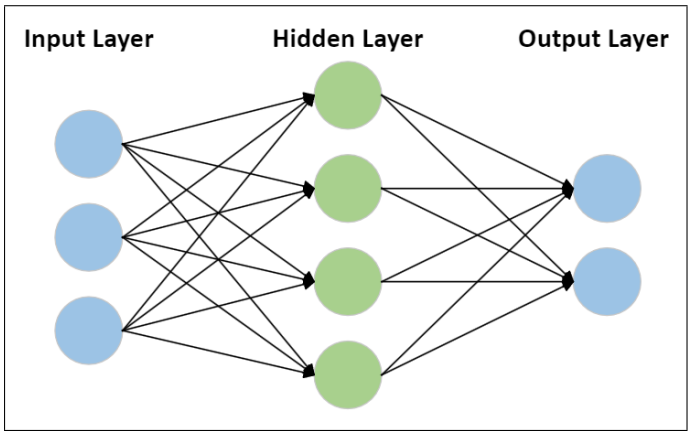

Fig. 1: figure of a neural network

Although, when these interconnected layers are stacked on top of each other, so multiple hidden layers are stacked on top of each other, the network is called deep.[25] 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.[26] 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 .[27] An example of a deep neural network is presented in figure 2 which shows the stacked layers in the middle of the network.

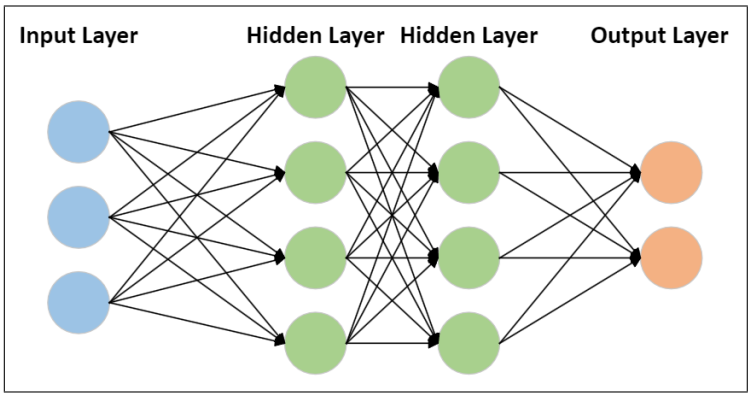

Fig. 2: figure of a neural network

Some examples for regression tasks within a DNN in the field of acomputer vision include object detection, medical image registration, head- and body-pose estimation, age estimation and visual

[24] cf. Durstewitz/Koppe/Meyer-Lindenberg 2019, p. 1583
[25] cf. Cichy/Kaiser 2019, p. 305
[26] cf. Durstewitz/Koppe/Meyer-Lindenberg 2019, p. 1583
[27] cf. Mall et al. 2023, p. 2

tracking.[28] Nowadays, verly large neural networks with millions of parameters can be created due to the research achievements made in the field of neural networks and deep learning leading to highly performing models.[29] 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.[30]

2.2.1 Energy-based models

An Energy Based Model (EBM) is a type of neural network that has special characteristics. One characteristic is that an EBM is a statistical model.[31] This probabilistic approach willingly uses uncertainty into the model calculations to draw the models inputs randomly from its underlying distribution.[32] This is done because the conventional deterministic method of backpropagation is known to potentially convert to local minimas, and requires long computation time.[33] As a result with conventional backpropagation more frequently incorrect classification would take place. The second characteristic is that an EBM is determined by an energy function that needs to be minimized in order to find the solution of the optimization problem.[34] Since 1982, those statistical neural network models have been continuously emerging in the machine learning field when J.J. Hopfield introduced the Hopfield Network.[35] Current developments include their use in reinforcement learning, potential replacements for discriminators in generative adversarial networks and for quantum EBMs.[36] 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.[37] 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.[38] Especially, it involes learning an energy function $E_\theta(x) \in \mathbb{R}$, with x representing the configuration of the network, and assigning the low energy to observed data x_i and high energy to other values x.[39]

[28] cf. Gustafsson et al. 2020, pp. 325–326
[29] cf. Marinó et al. 2023, p. 152
[30] cf. Marinó et al. 2023, p. 152
[31] cf. Huembeli et al. 2022, p. 2
[32] cf. Uusitalo et al. 2015, pp. 25–27
[33] cf. Specht 1990, p. 109
[34] cf.Huembeli et al. 2022, p. 2; cf.Ranzato, a. et al. 2006, p. 1
[35] cf. Hopfield 1982
[36] cf.Verdon et al. 2019, p. 1; cf.Du/Lin/Mordatch 2021, p. 1
[37] cf. Du/Mordatch 2020, pp. 1–2
[38] cf. Huembeli et al. 2022, p. 2
[39] cf. Gustafsson et al. 2020, p. 330

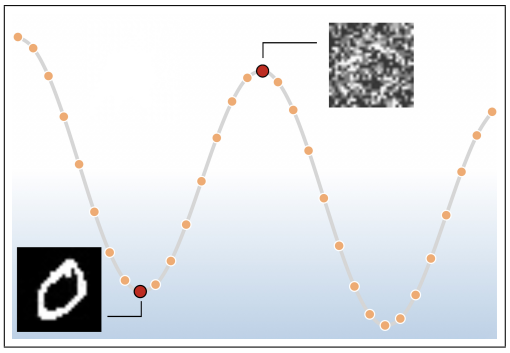

Fig. 3: Figure of a simplified energy landscape

In this figure 1 a simplified energy landscape is shown where the local minima corresponds to states that encode an MNIST digit.[40] 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.[41] The partition function Z used in 2.1 is given by summing over all possible pairs of visible and hidden vectors[42]:

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

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. What this means is that the specific aim is to adjust its parameters such that P_{model} becomes as close to P_{data} as possible, 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:

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

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

[40] cf. Huembeli et al. 2022, p. 6
[41] cf. Huembeli et al. 2022, pp. 2–3
[42] cf. Hinton, G. E. 2012b, p. 4

"dreaming".[43] In the training process the asymmetric divergence G needs to be miminized and therefore the probabiliies of the model converge close to the ones in reality. 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 probabiliies.[44]

NACH UNTEN As a side note it is worth mentioning that using the maximum likelihood estimator for Z 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.[45]

2.2.2 concept of Boltzmann Maschines

A BM is a type of symmetrical EBM consisting of binary neurons $\{0, 1\}$.[46] The neurons of the network can be split into two functional groups, a set of visible neurons and a set of hidden neurons.[47] Therefore, the BM is a two-layer model with a visible layer ("v") and a hidden layer ("h").[48] 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 and they cannot be represented by pairwise constraints.[49] The connection between the individual neurons is referred to as bidirectional, as each neuron communicates with each other in both directions.[50] 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.[51]

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.[52] Discrimivative models estimate the probability distribution based on category labels that are given to specific objects.[53] 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.[54] An real world example would be the following: to predict if a movie will be a hit, you could analyze past box office successes to model characteristics shared by hits (generative approach), or assess immediate audience reactions to

[43] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, pp. 154–155
[44] cf. Zhai et al. 2016, pp. 2–3
[45] cf. Zhai et al. 2016, pp. 2–3
[46] cf. Amari/Kurata/Nagaoka 1992, p. 260
[47] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 154
[48] cf. Salakhutdinov/Hinton, G. 2009, p. 448
[49] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 154
[50] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 149
[51] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 148
[52] cf. Hsu/Griffiths 2010, p. 1
[53] cf. Gm et al. 2020, p. 2
[54] 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.[55]

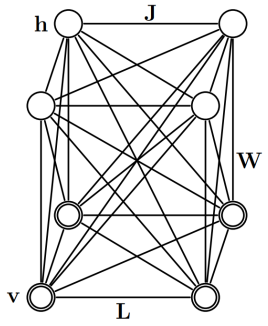

Fig. 4: figure of a general Boltzmann Machine

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.[56] 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 compatible with the given input.[57] 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.[58] Integrating the function 2.4 into the earlier introduced KL-divergence 2.2 and doing gradient descend a learning rule to update the weights and biases appers.[59] The gradient descent algorithm is commonly used in machine learning and is an iterative technique that adjusts the model parameters (weights and biases).[60] 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.[61] Consequently, this leads to the specific learning rule[62]:

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

[55] cf. Salakhutdinov/Hinton, G. 2009, p. 449
[56] cf. Salakhutdinov/Hinton, G. 2009, p. 448
[57] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 150
[58] cf. Fahlman/Hinton, G./Sejnowski, T. 1983, p. 110
[59] cf. Hinton, G. E. 2012b, p. 5
[60] cf. Wang, X./Yan/Zhang, Q. 2021, p. 11
[61] cf. Wang, X./Yan/Zhang, Q. 2021, p. 11
[62] cf. Hinton, G. E. 2012b, p. 5

The network can now update the weights "W" that exist between the neurons through the training rule based on the observations that served as input.[63] 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 data predictions and the model predictions takes a time that is exponential in the number of hidden units.[64] 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.[65] 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. This leads back to the briefly mentioned constraint of equation 2.3, that is needed to calculate an activation probability of a neuron, which is required to update a weight in the training process shown in 2.5.

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}.[66] 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 probabiliies. This enables the training of BMs and RBMs.

2.2.3 Restriced Boltzmann Machines

As a simplification of the training problem Hinton and Sejnowski proposed Gibbs sampling as an algorithm to aporoximate both expectations.[67] 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 the well-known model of a RBM establishes shown in fig.5.[68]

[63] cf. Barra et al. 2012, pp. 1–2
[64] cf. Salakhutdinov/Hinton, G. 2009, p. 449
[65] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, p. 154
[66] cf.Helmenstine 2022, p. 478-480; cf.Schlamminger 2014, p. 1
[67] cf. Ackley/Hinton, G. E./Sejnowski, T. J. 1985, pp. 158–165
[68] cf. Salakhutdinov/Hinton, G. 2009, p. 449

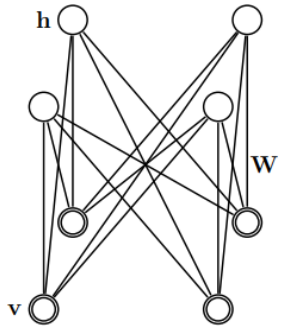

Fig. 5: Figure of a RBM

What can be recognized that no more visible-to-visible and hidden-to-hidden connections can be found in the model. The configuration of the visible and hidden units (v, h) therefore has also an updated energy function (Hopfield, 1982) given by:

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

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.[69] Despite, compared to the fully connected BM, the RBM is less complex but the advantages of training surpasses the loss in expressivity possibilities.[70] 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.[71]

Training of RBMs

The training of RBMs can be established with the use of sampling methods that estimate the activation probabilities, which are needed to update the weights. There are currently two 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.[72] In the following part both methods will be explained in depth. Especially, they are interesting since they serve as baselines to compare against the new sampling method of a hopfield network that is to be achieved in the practical part of the thesis.

[69] cf. Hinton, G. E. 2012a, pp. 3–4
[70] cf. Huembeli et al. 2022, p. 4
[71] cf. Zhang, N. et al. 2018, p. 1186
[72] cf. Patrón et al. 2024, p. 1

Contrastive Divergence: Contrastive divergence is a special Gibbs Sampling training method developed by Geoffrey Hinton for the efficient training of RBMs.[73] In traditional, Gibbs sampling would have to generate a long chain of samples, until independent samples are obtained from the observed data distribution of the model.[74] The samples are needed for each iteration of the gradient ascent on the log-likelihood resulting in large computational costs.[75] 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.[76] They way to achieve this is by initializing the Markov chain with the samples from the data distributon.[77] The outcome has been shown to heavily decrease the training time while only adding a small bias.[78] 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.[79]

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 $(\mathbf{v}_i\mathbf{h}_j)_{data}$.[80] 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:

[73] cf. Hinton, G. E. 2012b, pp. 4–5
[74] cf. Huembeli et al. 2022, pp. 5–6
[75] cf. Upadhya/Sastry 2019, pp. 7–8
[76] cf. Mocanu et al. 2016, p. 246
[77] cf. Upadhya/Sastry 2019, pp. 7–8
[78] cf. Larochelle/Bengio 2008, p. 537
[79] cf. Larochelle/Mandel, et al. 2012, p. 646
[80] cf. Hinton, G. E. 2012b, p. 5

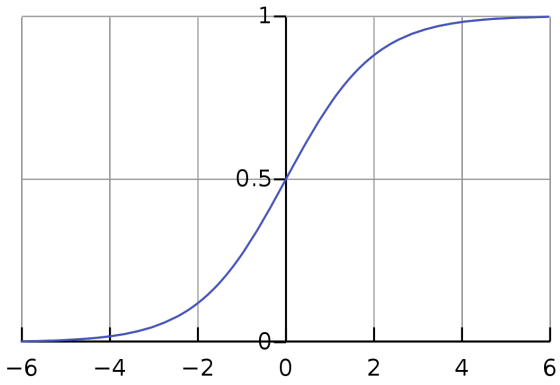

Fig. 6: Figure of a logistic sigmoid function RBM

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.[81] 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:[82]

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

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

[81] cf. Huembeli et al. 2022, p. 6
[82] cf. Hinton, G. E. 2012b, p. 6

times for a suitable chosen N the model learns better, since more steps of alternating Gibbs sampling were performed.[83]

Metropolis-Hastings: The Metropolis-Hastings algorithm, often only called Metropolis algorithm, is a technique out of Markov chain Monte Carlo (MCMC) class techniques.[84] 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.[85] 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 can proceed along this painting until all parts have been seen."[86] 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.[87] 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.[88] 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 $x_{\text{old}} = x_{\text{new}}$.[89] 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.[90] For RBMs KT is assumed to be 1.[91] In the following figure 7 the resulting probability function is shown.

[83] cf. Huembeli et al. 2022, p. 6
[84] cf. Patrón et al. 2024, p. 1
[85] cf. Metropolis et al. 1953, pp. 1087–1092
[86] cf. Robert 2016, p. 2
[87] cf. Beichl/Sullivan 2000, p. 65
[88] cf. Rosenthal 2009, p. 1
[89] cf. Patrón et al. 2024, pp. 1–2
[90] cf. Li et al. 2016, pp. 1–9
[91] cf. Hinton, G. 2014, p. 3

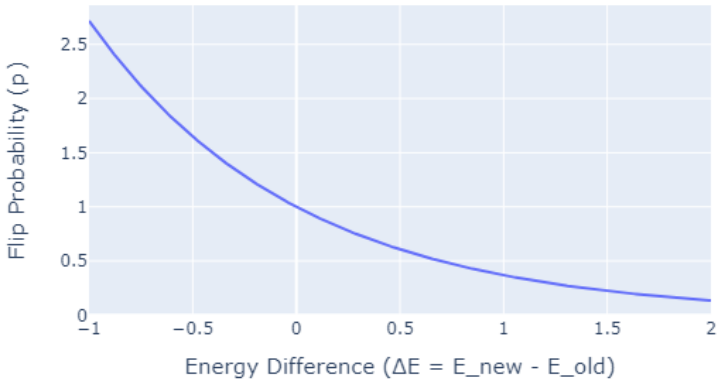

Fig. 7: Flip Probability Function in Metropolis-Hastings Algorithm

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., $x_{\text{old}} = x_{\text{new}}$).[92] Otherwise, a rejection takes place if $r > P$ (i.e., $x_{\text{old}} = x_{\text{old}}$).

Finally, the configuration x_{old} can be stored and the process repeats beginning from step 2 on.[93] After repeating enough times the activation probability for each neuron can be calculated by summing over all samples $(x_1 + x_2 + x_3 + \ldots)$ and the result is divided by the total number of samples.

2.2.4 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.[94] 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.[95] 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.[96] 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

[92] cf. Patrón et al. 2024, pp. 2–3
[93] cf. Patrón et al. 2024, p. 17
[94] cf. Fischer/Igel 2012, pp. 1–2
[95] cf. Fischer/Igel 2012, p. 2
[96] cf. Larochelle/Bengio 2008, p. 536

due to an incompatible selected temperature.[97] 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 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.[98] That is the reason why they are significant for the acceleration of this thesis and an interesting technology to look at.

2.3 Hardware accelerators

2.3.1 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.[99] This upcoming trend of growing network sizes exponentially also brings a dark side with it: An excessive increase in computational effort and memory size.[100] 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.[101] 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.[102]

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.[103] 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.[104] 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".[105] Broken

[97] cf. Huembeli et al. 2022, pp. 3–4
[98] cf. Lehnert et al. 2023, pp. 3881–3882
[99] cf. Baischer/Wess/TaheriNejad 2021, p. 1
[100] cf. Baischer/Wess/TaheriNejad 2021, pp. 1–2
[101] cf.Zhou et al. 2022, p. 1-2; cf.Baischer/Wess/TaheriNejad 2021, p. 2
[102] cf. Ahmad/Pasha 2020, p. 2692
[103] cf. Peccerillo et al. 2022, p. 2
[104] cf. Peccerillo et al. 2022, pp. 2–3
[105] cf. Peccerillo et al. 2022, p. 2

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.[106] There are different approches such as Graphics Processing Unit (GPU)s, Application Specific Integrated Circuit (ASIC)s, Field Programmable Gate Array (FPGA)s, but also new approaches like Quantum Computations or Photonic matrix multiplication are researched.[107] 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 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.[108] **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.[109] **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.[110] The following three accelerator approaches can be categorized into the first category:

2.3.2 GPU

The GPU, is by far the most common accelerator in the market with a focus at computational-complex workloads. Also GPUs enabled great achievements in DNNs due to their massive parallelization potential of computations and their high troughputs compared to FPGAs and ASICs.[111] A GPU is a manycore unit that features up to hundreds of multi-processors that consist of in-

[106] cf. Peccerillo et al. 2022, pp. 2–3
[107] Zhou et al. 2022, p. 1-2; cf.Baischer/Wess/TaheriNejad 2021, p. 2
[108] cf. Lehnert et al. 2023, p. 3883
[109] cf. Lehnert et al. 2023, p. 3883
[110] cf. Lehnert et al. 2023, p. 3883
[111] cf. Baischer/Wess/TaheriNejad 2021, p. 16

order cores which are able to exploit massive threadlevel parallelism.[112] 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.[113] With development of real-time graphics GPUs became programmable too. 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.[114] 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.[115] Well-known frameworks would be PyTorch or TensorFlow. However, compared to the other two accelerator apporaches the GPU is not as flexible and has higher latency and energy consumption.[116]

2.3.3 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.[117] They work by using reconfigurable logic blocks that can be interconnected using routing tracks with configurable switches at the intersections.[118] This combined with the use of many digital signal processors and local storage of data in the hardware enables the development of the exact hardware needed for a workload.[119] As a side note it is worth mentioning that the most energy-consuming task of a workload is 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.[120] Hence, they are utilized to design a specialized processor tailored for executing specific workloads, like machine learning, effectively.[121] Furthermore, due to their reprogrammable nature they have a lower engineering cost and faster time-to-market compared to ASICs.[122] 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[123] 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.[124]

[112] cf. Peccerillo et al. 2022, p. 2
[113] cf. Hu/Liu, Y./Liu, Z. 2022, p. 101
[114] cf. Dally/Keckler/Kirk 2021, p. 42
[115] cf. Baischer/Wess/TaheriNejad 2021, p. 16
[116] cf. Hu/Liu, Y./Liu, Z. 2022, p. 100
[117] cf. Ahmad/Pasha 2020, p. 2693
[118] cf. Babu/Parthasarathy 2021, p. 144
[119] cf. Baischer/Wess/TaheriNejad 2021, p. 19
[120] cf. Hu/Liu, Y./Liu, Z. 2022, pp. 101–102
[121] cf. Sipola et al. 2022, p. 322
[122] cf. Boutros/Betz 2021, p. 4
[123] cf. Boutros/Betz 2021, p. 4
[124] cf. Hu/Liu, Y./Liu, Z. 2022, p. 100

2.3.4 Application specific integrated circuit

ASICs can be distinguished from FPGAs because they are not programmable. In addition to that, they offer the highest degree of customization and are designed to execute a specific application with the utmost efficiency.[125] So the attributes of ASICs are the focus on specific tasks and the strong design freedom. Nowadays, the most common type of ASICs are Tensor Processing Unit (TPU)s because of their matrix vector multiplications abilities that are needed within machine learning. But conventional ASICs work by mapping neurons directly to the hardware.[126] Their design architecture enables them to outperform GPUs and FPGAs in terms of their small size, greater computation speed and high power efficiency.[127] Specifically when compared to corresponding FPGA circuits ASICs are 35x smaller and 4x faster. A more current study could show that this gap is reduced but they still are 9x smaller and also faster.[128]

Nonetheless, developing an ASIC requires expert knowledge in chip design but also within neural networks and takes a lot of time.[129] Out of the three approaches, they provide the most efficient solution, yet it comes at the expense of lacking reconfigurability, no programmability, and incurring high engineering costs.[130] With a sustainability and climate-change aspect in mind, they are the by far the best option since they represent the most power efficient approach with also the best computation speed.

2.4 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.[131] It can be categorized into the ASIC family of hardware accelerators and its specific purpose is to solve Ising problems. In 1925 Ernst Ising, a german physicist, invented the Ising model which explained the interaction between ferromagnets.[132] This statistical model focuses on the state of a spin s_i (up and down; $+1$ and -1) representing their mangetical behaviour. 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

[125] cf. Baischer/Wess/TaheriNejad 2021, p. 17
[126] cf. Hu/Liu, Y./Liu, Z. 2022, p. 104
[127] cf. Baischer/Wess/TaheriNejad 2021, p. 17
[128] cf. Boutros/Betz 2021, p. 5
[129] cf. Baischer/Wess/TaheriNejad 2021, p. 17
[130] cf. Peccerillo et al. 2022, p. 4; cf.Hu/Liu, Y./Liu, Z. 2022, p. 100
[131] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[132] cf. Ising 1925, pp. 253–258

by an edge (ij).[133] 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.[134] Nowadays, the Ising modell is also attractive in other fields to describe the energy of a system and to transform them into an Ising problem.[135] 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 solution.[136]

Equivalence between the energy function of a mem-HNN and the energy function of a RBM can be shown here. The energy function of the mem-HNN works by using the binary states of $+1$ and -1 while the RBM uses $+1$ and 0 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 energy function. The fact, that both energy functions are equal implies that the neural network of a RBM could possibly be trained on this Ising machine. Moreover, other artificial intelligence models could be compatible but currently there are no apporaches on how to transform them.

Comming back to the Ising machine itself, 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 conventional semiconductor electronical technology.[137] 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.[138] 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.[139] 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.[140] Well-known examples would be the salesman problem, the maximum clique problem or the calculation of the partition function of the BM with 2^n possibilities.

In simple terms, this Ising machine is able to efficiently find the global minima of energy functions that are the underlying structure of an Ising problem. The name mem-HNN already indicates that the Ising machine is based on the concept of a Hopfield Network. All this is possible beause the update formula of the Hopfield Network is directly implemented on the hardware of the accelerator. A photograph of the physical mem-HNN accelerator (left side) and a microscopic view of it (right side) can be seen in following figure:

[133] cf. Tanahashi et al. 2019, p. 2
[134] cf. Wang, T./Roychowdhury 2017, pp. 1–2
[135] cf. Tanahashi et al. 2019, pp. 2–3
[136] cf. Lucas 2014, pp. 2–3
[137] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 1
[138] cf. Mohseni, N./McMahon/Byrnes 2022b, p. 363
[139] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[140] cf. Izadkhah 2022, pp. 497–500

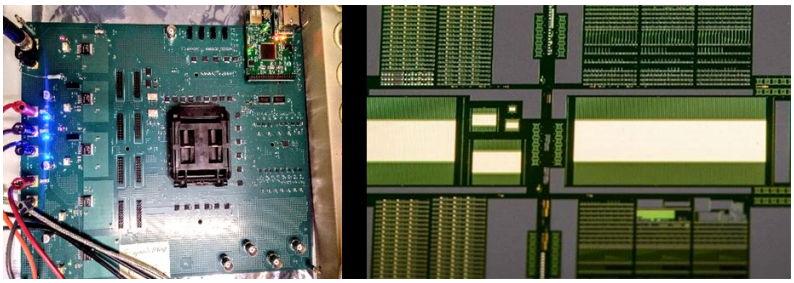

Fig. 8: Physical and microscopic view of the mem-HNN

2.4.1 Hopfield Network

A Hopfield Network is an EBM and belongs to the field of recurrent neural networks.[141] Because the Hopfield Network is the underlying structure of the physical mem-HNN accelerator this model is explained first. The structure of the network consists of only one single layer with binary valued neurons inside.[142] Therefore, 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.[143] Initially, the primary applications of this type of network were to serve as storage for associative patterns and to facilitate pattern retrieval.[144] In practive given a query pattern a Hopfield Network can retrive a pattern that is most similar or an is an average of similar patterns.[145] In this paper the Hopfield Network's update function interests us because it possibly could be used to sample the intractable training of a RBM mentioned earlier. Surprisingly, since Hopfield networks were introduced by J.J Hopfield in 1982 the storage capacity got increased over time but the fundamentals stayed the same.[146] In following figure 6 an example of a Hopfield Network can be seen.[147]

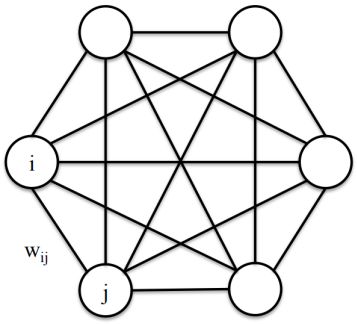

Fig. 9: Figure of a hopfield network

[141] cf. Dramsch 2020, p. 35
[142] cf. Ahad/Qadir/Ahsan 2016, p. 7
[143] cf. Ahad/Qadir/Ahsan 2016, p. 7
[144] cf. Ramsauer et al. 2021, p. 2
[145] cf. Ramsauer et al. 2021, p. 2
[146] cf.Hopfield 1982, p. 2554-2558; cf.Ramsauer et al. 2021, p. 2
[147] cf. Yao/Gripon/Rabbat 2013, pp. 1–2

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.[148] 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.[149] The minimum energy is then equal to the state where the network is able to perform as a memory item.[150] This energy state can be calculated with the following energy equation[151]:

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

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. 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.[152] The second important reason to research the hopfield networks is for their updating process. Approximately, the activation rule for each neuron is to update its state as if it were a single neuron with the threshold activation function.[153]

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

The state of the neuron will be updated to 1 if the sum over all weights multiplied with the states {1, -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.

Since every neuron's output is an input to all the other neurons the order of the updates need to be specified.[154] There is the possbility to update all neurons synchronous or asynchronous. There is no study that shows what update method leads to better results. Therefore, this paper follows the asynchronous option first and ensures to do enough sampling steps, so that every neuron has at least updated once before moving on.

2.4.2 Memristor Crossbar Array

Having set the foundational knowledge about the function of a Hopfield Network the mode of operation can be tackled. Since the mem-HNN saw the light of day in 2021, a number of

[148] cf. Yao/Gripon/Rabbat 2013, p. 3
[149] cf. Ahad/Qadir/Ahsan 2016, p. 7
[150] cf. Ahad/Qadir/Ahsan 2016, p. 7
[151] cf. Hopfield 1982, p. 2556
[152] cf. Ahad/Qadir/Ahsan 2016, p. 7
[153] cf. MacKay 2003, p. 506
[154] cf. MacKay 2003, p. 506

improvements have been made to it and at the end of 2023 the individual components can be seen in following figure[155]:

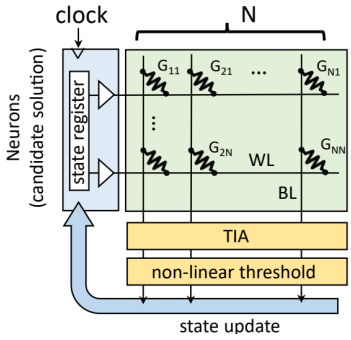

Fig. 10: Modell of the mem-HNN

The green square symbolizes the memristor crossbar array, which has the task of performing the in-memory-vector matrix multiplication. This is the core part of the architecture that is built around the update formula of the Hopfield Network. The memristor crossbar array gets his name because it 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 $G = \frac{1}{R}$. On the other hand, BL (Bitline) and WL (Wordline) represent the electical tracks. All the other parts of the model are handled in subchapter 2.4.3, so for now the spotlight belongs to the memristor crossbar array (green square). A better perspective of the memristor crossbar array gives following 3D model[156]:

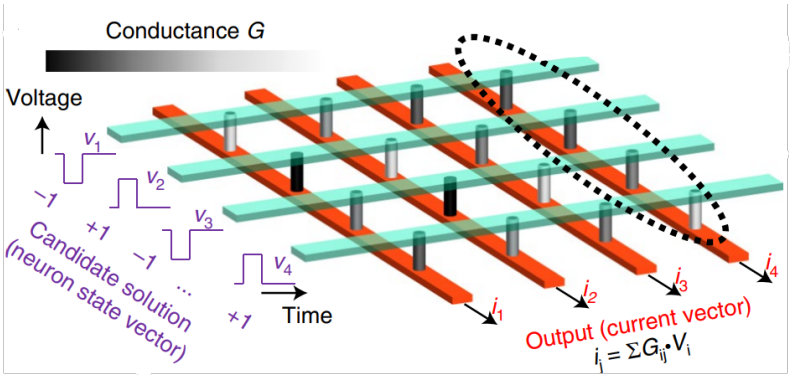

Fig. 11: 3D modell of the memristor crossbar array

To understand how this crossbar array implements the Hopfield network, it helps to have another look at the earlier introduced update formula in equation 2.11.

First of all, on the green Wordline a voltage which either is $+1$ or -1 that represent the state of the neurons in the hopfield network is applied. Then, the current is flowing towards the memristors,

[155] cf. Hizzani et al. 2023, p. 2
[156] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410

which is an electronic device that functions as a resistor, but with the ability to dynamically change its resistance while the device is being used.[157] In this model the gray cylinders represent the memristors. The memristors are suitable for matrix vector multiplication because they can use their dynamical resistance to act as weigths.[158] In addition to that, this is the reason why this kind of memory has the potential of high switching speeds, high energy efficiency and high endurance.[159]

In eletrical terminology, if the resistence of a memristor is higher, the conductance is low since the current can barely flow into the lower Bitlane. For each of the intersections, when a Wordline meets the crossbar it represents the multiplication of w_{ij} and s_j within the update formula. After the matrix vector multiplication the single currents flow in the same direction on the Bitline and are summed up together with their individual current strength.[160] The reason for this phenomenon is the 1. Kirchhoff'sche rule, which is not further explained here since it would be out of scope. To make an example for each output current i_1 has for intput voltages V_1, V_2, V_3, V_4. The current flowing into the lower Bitlane is now according to Ohm's rule $i_{out} = \frac{1}{R_{memristor}} * V_{in}$. Since $G = \frac{1}{R}$ and Kirchhoff'sche rule sums up each current i is $i_1 = G_{11} * V_1 + G_{12} * V_2 + \cdots + G_{14} * V_4$ As a result the first first part $\sum_j w_{ij} s_j$ of the updating formula of the Hopfield Network is established. Last but not least, adding the bias b to the sum is achieved by simply adding an intial current, which is worth the amount of the bias, to the the total Bitline current. Each i is the output of the current vector and equal to a neuron within the Hopfield Network with all the memristors on this lane symbolizing the weights that all the other neurons connect to it.

The way the memristors work is to imagine them as plate capacitors. The following figure shows how memristors function achieve the dynamically changing resistance.[161]

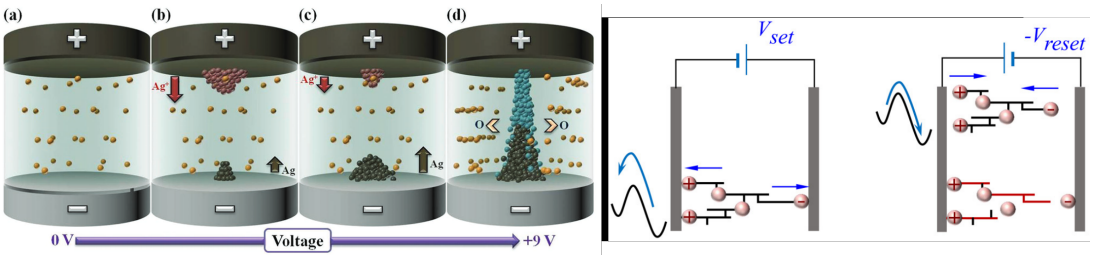

Fig. 12: Memristor-based learning

Essentially, the memristor consists out of two metal electrodes and a dielectric layer that is sandwiched between them.[162] In case of the mem-HNN it is an oxide memristors that uses tantalum oxide (TaOx) and oxygen atoms as dielectric layer.[163] Initially the metal-ions are freely

[157] cf Sung/Hwang/Yoo 2018, p. 124
[158] cf Sung/Hwang/Yoo 2018, p. 124
[159] cf. Amirsoleimani et al. 2020, p. 3
[160] cf. Amirsoleimani et al. 2020, p. 3
[161] Chang et al. 2017, p. 6; Sung/Hwang/Yoo 2018, p. 2
[162] cf. Chang et al. 2017, p. 1
[163] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 412

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.[164] 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 built.[165] 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. This in fact is the explanation on how the resistance of the memristor can change dynamically and is able to store the information about the preferred conductance. 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 would be a water pupe with a valve that can be adjusted to control 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.[166] 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 for the next iteration. 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 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.[167]

Presently, the research on the application of memristor crossbar arrays in supervised learning is sparse, indicating a clear necessity for further investigative studies to understand their potential and usability.[168] In the practical part exactly this is put to the test and training data of the RBM should be generated to give clearance about the feasibility of this idea. A possbility is to use the memristor to set the resistance V_{set} for one training iteration with the needed weigths, reset them $-V_{reset}$ and initiate the following training with the new weights that are digitally updated.

[164] cf. Chang et al. 2017, p. 3
[165] cf. Chang et al. 2017, p. 5
[166] cf. Sung/Hwang/Yoo 2018, pp. 1–2
[167] cf. Sung/Hwang/Yoo 2018, p. 7
[168] cf.Amirsoleimani et al. 2020, p. 8; cf.Sung/Hwang/Yoo 2018, p. 124

2.4.3 Output Hopfield Network

In this subchapter the left over components of the mem-HNN shown in 10 are adressed. To remain in the correct sequence, after the memristor crossbar array the Transimpedance Amplifier (TIA) is addressed next. The TIA is the component that converts the current i_j, which is the output of the memristor crossbar array, into a voltage.[169] Since each output current symbolizes one neuron, there are also that many TIAs required.

Subsequently, to fulfill the update formula of the Hopfield Network, that the mem-HNN impersonates, is the non-linear threshold. 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 $<$.[170] 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.[171] In addition to that, the comparator is the component that transforms the analog voltage into a binary digital signal. 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).[172] 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.[173] 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. Now, a new sampling step can begin and neurons states can be updated.

A missing component part in figure10 is a selector that is connected to the state register. Its task is to allow updating through unlockickt the correct positions inside the state registor. For example the mode of updating can be selected with either one neuron updated at a time asynchronously or multiple neurons synchronous. A promissing and interesting sampling strategy could be to update $N/2$ of all neurons.[174] So when there are 100 neurons in the neural network with each sampling step 50 neurons are updated.

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. They are required to enable a parallel activation of Wordlines and to start the new sampling step.[175]

[169] cf. Hizzani et al. 2023, p. 3
[170] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[171] cf. Chen/Zhang, M./Shen 2021, p. 28
[172] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18
[173] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 3
[174] cf. Hizzani et al. 2023, p. 3
[175] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 18

Finally, it is worth to mention one sampling step can happen within one clock cycle of the. There are thousands of sampling iterations that are performed within one training iteration of a neural network. After each sampling iteration the neuron configuration in the state register is saved either in a cache directly on the ASIC hardware or sent to a digital external computer. For example after 10000 sampling iterations the arrays of the hidden- and visible neurons are sent to the computer to perform a prediction after the iteration. This completes the explanation of how the mem-HNN implements the concept of a Hopfield Network.

2.4.4 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.[176] This is achieved by a random number generator in the hardware that creates a random array filled with digital signals.[177] 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.[178] 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:

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

with n representing the noise.[179] Besides aiding optimization tasks, noise also creates an interesting link to BMs. Without noise the Hopfield Network is only able to do perform optimization tasks and no sampling, which is needed to train a RBM. 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 has no temperature 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 2.2.4. Therefore, to successfully implement a RBM on 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 and enhance compatibility involves utilizing noise from an analog noise source.[180] One relatively straightforward way to inject noise

[176] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[177] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Liu, R., et al. 2019, p. 22
[178] cf. Hizzani et al. 2023, p. 3
[179] cf. Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, p. 410
[180] cf.Böhm et al. 2022a, p. 1-2; cf.Mahmoodi/Prezioso/Strukov 2019, p. 2

into the activation function is by adding a normal gaussian distribution $g(x)$ on top of it[181]:

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

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.

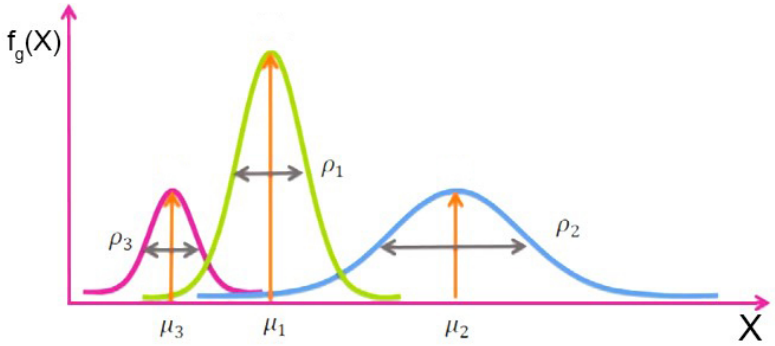

Fig. 13: Gaussian normal distribution[182]

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.[183] 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.[184]

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.[185]

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.

[181] cf. Böhm et al. 2022a, p. 3
[182] modified from cf. Gm et al. 2020, p. 3
[183] cf.Böhm et al. 2022a, p. 1-2; cf.Mahmoodi/Prezioso/Strukov 2019, p. 2
[184] cf. Böhm et al. 2022a, pp. 1–11
[185] 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).[186] 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,

[186] 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".[187] 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.[188] 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.[189] Artifacts in DSR can be constructs, models, methods or instantiations.[190] 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.[191] 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.[192]

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.[193]

[187] Hevner et al. 2004, p. 77
[188] Hevner et al. 2004, p. 82
[189] cf. Baskerville et al. 2018, p. 368
[190] Hevner et al. 2004, p. 77
[191] cf. Gregor/Hevner 2013, p. 342
[192] cf. Gregor/Hevner 2013, p. 343
[193] 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.[194] 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..[195] 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.[196] 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.[197] 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:

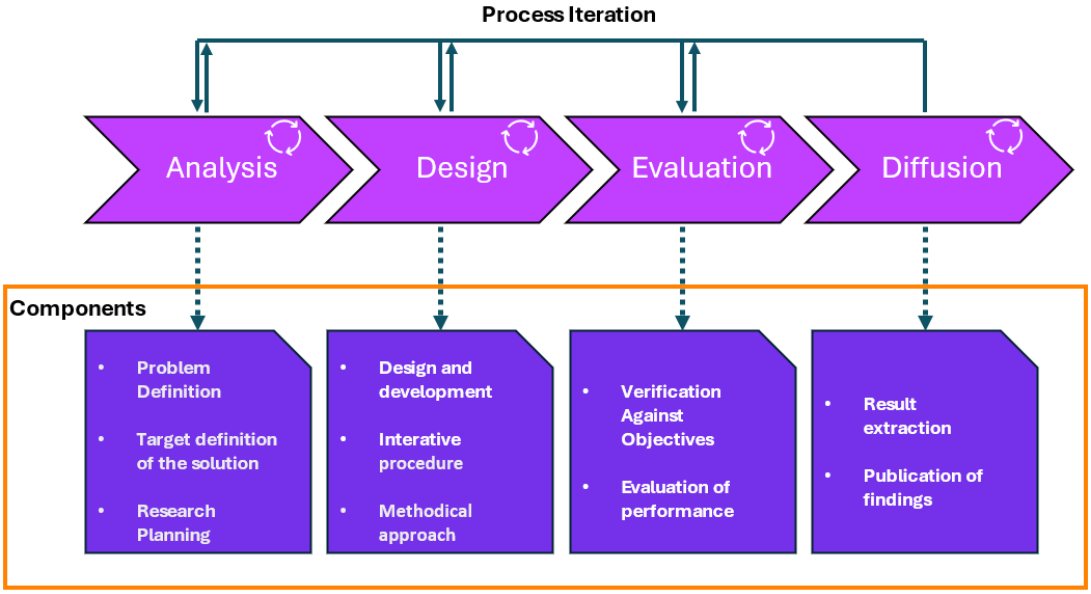

Fig. 14: DSR model by Österle et. al[198]

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

[194] cf. Peffers et al. 2007, p. 54
[195] cf. Oesterle et al. 2010, pp. 1–6
[196] cf. Österle/Otto 2010, pp. 273–274
[197] cf. Oesterle et al. 2010, p. 5
[198] 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.[199]

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.[200]

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.[201]

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.[202]

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.[203] 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.[204] 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

[199] cf. Oesterle et al. 2010, p. 4
[200] cf. Österle/Otto 2010, p. 279
[201] cf. Österle/Otto 2010, p. 279
[202] cf. Oesterle et al. 2010, p. 5
[203] cf. Bjarnason/Lang/Mjöberg 2021, p. 1
[204] 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.[205] 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.[206] 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.[207] 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.[208] As a guideline to categorize this behaviour the thesis uses the House of Prototyping Guidelines by Ahmed and Demirel. These guidelines shown in 2 introduce four different dimensions used to categorize prototypes: Type of Prototype(1), Fidelity Level(2), Complexity(3), Scale(4) and Number of Iterations(5).[209]

The third step "model synthesis" requires a description and a chronological sequences of the prcoesses.[210] 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.[211]

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

[205] cf. Bjarnason/Lang/Mjöberg 2021, p. 8
[206] cf. Nelson et al. 2016, p. 25
[207] cf. Mihram 1976, p. 71
[208] cf. Mihram 1976, pp. 71–72
[209] cf. Ahmed/Demirel 2021, pp. 6–7
[210] cf. Mihram 1976, pp. 71–72
[211] 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.[212] Furthermore, the gathered data helps with decision making for strategical and operational levels.[213] 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.[214] 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.[215] 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.

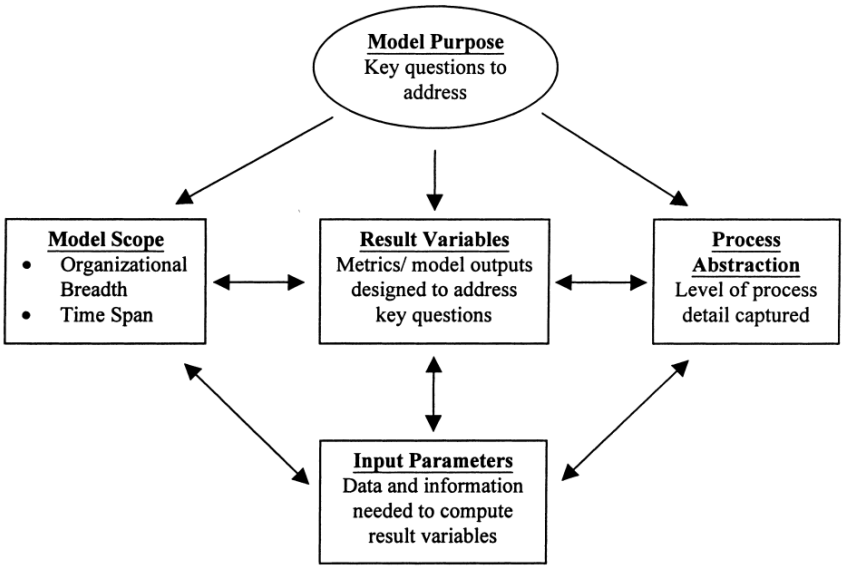

Fig. 15: general simulation model[216]

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

[212] cf. Kellner/Madachy/Raffo 1999, p. 92
[213] cf. Kellner/Madachy/Raffo 1999, p. 93
[214] cf. Abar et al. 2017, pp. 13–14
[215] cf. Sarhadi/Yousefpour 2015, pp. 470–471
[216] Kellner/Madachy/Raffo 1999, p. 95

selection of the **model scope**.[217] 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).[218]

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.[219]

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.[220] 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.[221]

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.[222] 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.

[217] cf. Kellner/Madachy/Raffo 1999, p. 95
[218] cf. Kellner/Madachy/Raffo 1999, p. 96
[219] cf. Kellner/Madachy/Raffo 1999, pp. 96–97
[220] cf. Kellner/Madachy/Raffo 1999, p. 97
[221] cf. Kellner/Madachy/Raffo 1999, pp. 97–98
[222] 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 Zielsetzung und Forschungsmethodik

Upon establishing the precise research methodology, this chapter delves into the practical application of the previously mentioned methods. First, the analysis phase of the DSR process is executed with the goal to establish a model of the research plan which the requirements and framework conditions of the IT-solution can be derived from. Next, the practical implementation is performed during the iterative design phase and uses the method of prototyping. In the end of the design phase is a functional IT-artifact, which fulfills the set requirements. The evaluation phase in this chapter uses the method of simulation to answer the second part of the research question; to see how efficient the mem-HNN can utilize the AI-model in terms of throughput and energy usage.

4.2 Analysis phase

4.2.1 General conditions

The first phase of the DSR-cycle has the goal of specifying the objective and establishing an according research outline and the requirements of the artifact. Additionally, the research outline should be visualized as a model of the overall solution concept.[223] The objective of the practical part is already specified in chapter 3.1. 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. This is tested by implementing the concept in software that is also part of the ASIC design process.[224]

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..[225] 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 RBMs and has a lower learning curve compared with e.g. Tensorflow.[226]

It should also be clarified that the hardware, which the analog mem-HNN-accelerator consists of, is implemented in software. This design decision is made out of time constraints of this thesis and part of the ASIC design process before buildng an actual accelerator. Nonetheless, the complete

[223] cf. Österle/Otto 2010, pp. 278–279
[224] cf.Rao 2024, p. 1; cf. *ASIC Design Flow for VLSI Engineering Teams [GUIDE] - Xinyx Design* 2024, p. 1
[225] cf. *Discrete and Continuous Models and Applied Computational Science* 2024, pp. 306–307
[226] cf. Raschka/Patterson/Nolet 2020, pp. 5–6

hardware is realizable in software without taking compromisses within their functionality. The simulation data gathered later on is close to the real energy efficiency and throughput.[227]

Lastly, the energy model used in the simulation can, depending on a specific input, calculate the amount of energy required from the hardware to perform computations. This energy model is developed by HPE and the Forschungszentrum Jülich. The explanation for this model is out of scope for this thesis but core parameters are explained to understand the data generation for the energy values. A seperate paper will be published about the energy model in the near future but kindly for this thesis the model can be used in advance.

4.2.2 Requirements

In order to set requirements for the IT-artifact an ideal overall solution architecture is of importance. Hence, a complete solution is modelled, which surpasses both proof of concepts introduced in 2.4.4. It shows the components required for an acceleration of a BM in hardware with the statistical sampling performed by an Hopfield Network, which was never done and tested before. Furthermore, the following figure16 contains the interaction between the digital computer and the analog mem-HNN Accelerator with five different steps.

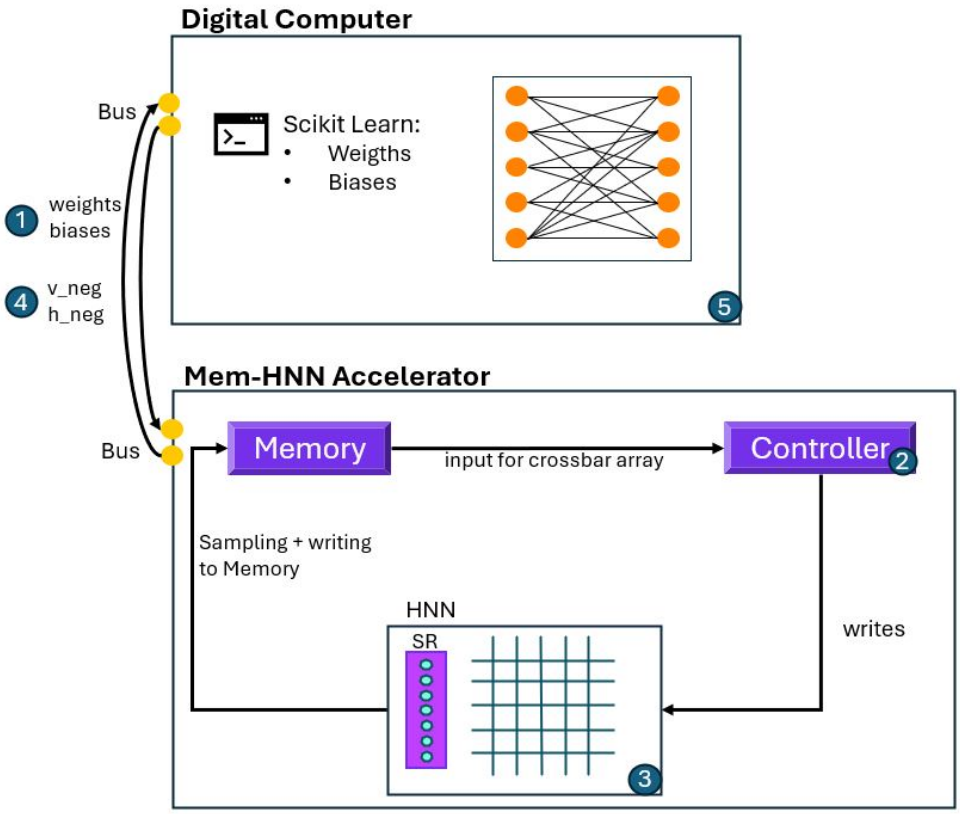

Fig. 16: proposed solution architecture

[227] cf. Hizzani et al. 2023, pp. 3–4

1. contains the initialization of the neural network. Hence, the weights and biases are assigned and the structure is build.

2. starts with the transfer of the weigths 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, communicates with the computer and sets a counter and reset for the amount of sampling steps.

3. is the Hopfield Neural Network (HNN), which contains the memristor crossbar array and the state register (SR). The state register includes the current neuron configuration.[228] For each of the neurons a TIA and comparator are required in the hardware. Furthermore, the state register can lock und unlock specific neurons, so that it is possible to update neurons synchronously. This enables the possibility of the promissing N/2 update strategy.

4. After iterating over steps two and three multiple times until enough sampling are executed the, all the configurations of the visible neurons v_{neg} and the hidden neurons h_{neg} are transferred from the controller via the bus-system to the digital computer.

5. contains the update of the weights and biases. Furthermore, the model can be evaluated in its performance in terms of chosen metrics like prediction accuracy or the negative likelihood etc.. In this case a logistic regression is used for the classification task and the RBM is used for the trainng of the neural network. After the evaluation the iteration is completed and the new weghts and biases can be transmitted for the following training iteration.

Since such chips do not yet exist, the objective is to construct a simulator pipeline that integrates these individual steps into a simulator, effectively replicating this hardware platform. Only the bus system, the memory and the controller are not mappable since the local machine already has the hardware. With this underlying model deriving the requirements is the next step in establishing the research outline. The aim of generating requirements is to generate good quality, not perfect, requirements that offers an acceptable level of risk to start the project.[229] Hence, these requirements need to cover the functions of the mem-HNN, which then must be implemented by the respective software components. Despite this, requirements may evolve over time and occasionally require adjustments when outcomes differ from initial expectations. As a result, the analysis of the model in conjunction with the reasearch question and under thought of the defined objective, the following requirements for the software emerged:

Digital Computer

- Defining a RBM
- Utilization of any training data
- Training a RBM

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

- Establishing a pipeline for the classification
- Possibility of conventional sampling algorithms: gibbs sampling and metropolis hasting
- Setting individual parameters: sampling steps, training iterations

Simulated Mem-HNN Accelerator

- Using any RBM as input
- Correctly using the Hopfield Network update algorithm
- Return sampled output of the neuron configurations
- Correctly using the Hopfield Neural Network as sampling method
- Possibility to use N/2 half updating method instead of asynchronously update of states
- Component for measuring the parameters required for evaluation: Speed (throughput) and energy consumption

Fruthermore, 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 all requirements of the digital computer resulting in a first prototyp iteration. 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. The first step, is to chose one of the machine learning libraries like Ternsorflow, Pytorch or Scikit Learn. In this case Scikit Learn is chosen since it is one of the industry standards for classical machine learning, has a broad variety of features in terms of RBMs and has a lower learning curve compared with e.g. Tensorflow.[230] This allows fast development with already well balanced hyperparameters within the library that can be used. Another reason is that the available workloads of datasets are popular and devliver results that are comparable with literature. This is useful to answer the research question in a timely manner with already available functionalities for RBMs. Especially, the implementation of the RBM is inspired by an example of the official scikit learn documentation.[231] In general,

[230] cf. Raschka/Patterson/Nolet 2020, pp. 5–6
[231] cf. *Restricted Boltzmann Machine Features for Digit Classification* 2024, p. 1

the RBM model is modelled as the feature extractor in combination with a logistic regression classifier for the prediction.

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. One reason for this is that the load digits dataset is similar to the well known MNIST dataset but has a smaller resolution of 8x8 pixels and features around 1800 samples that can be categorized in 10 classes (integers 0-9).[232] The second reason is that the workload is already optimized and therefore can deliver relevant data for the research question. Also, the dataset can be changed as desired with for example a breast cancer classification workload; in general all datasets that have two states like the activation of neurons in the neural network are compatible. 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 20% test data.[233]

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. Noteworthy, the size of the visible layer is automatically recognized by scikit learn but since the resolution of the pictures is 8x8 it requires 64 visible neurons to process an input.

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, which are published in Nature Communiations.[234] This decision was made because the algorithm used there is the original metropolis algorithm by Metropolis et.al..[235] Furtherore, 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 many zero arrays at the beginning of the sampling. 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`.

[232] cf. *Sklearn.Datasets.Load_ digits* 2024, p. 1
[233] cf.Charitha et al. 2022, p. 1-2; cf.Supri et al. 2023, p. 1
[234] cf. Böhm et al. 2022a, pp. 11–12
[235] cf. Metropolis et al. 1953, pp. 1087–1092

Hence, the possibility of different sampling methods are possible and the training of the RBM is possible. 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 and negative likelihood for each iteration. Following figure17 shows the training results using the gibbs sampling approach.

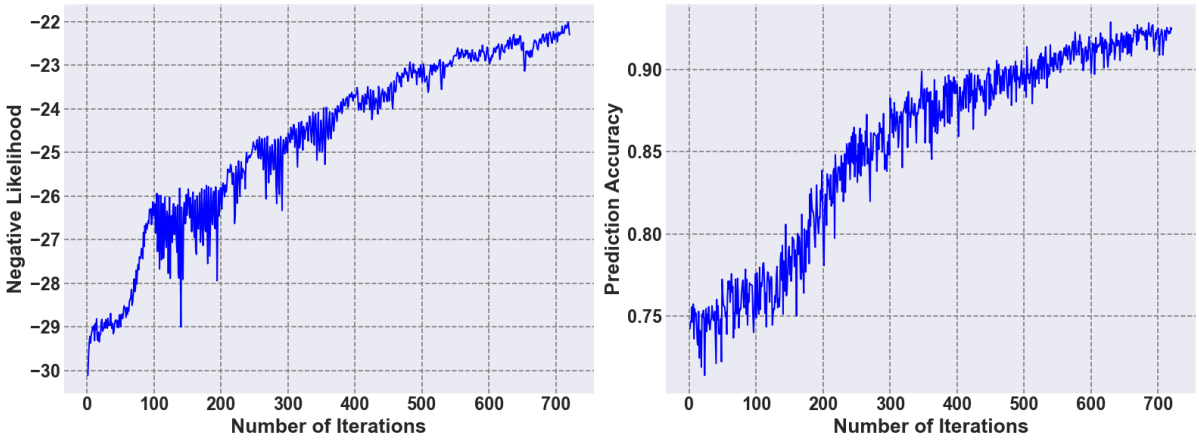

Fig. 17: Gibbs sampling baselines

The right plot shows that the initial prediction accuracy starts at 75%, akin to that of a linear regression model. This suggests that without training a linear regression alone could account for this level of accuracy when an untrained RBM is included in the pipeline. Data points are colected after every iteration across the span of 720 iterations. It is noteworthy that after 650 iterations the accuracy tagnated and had 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.

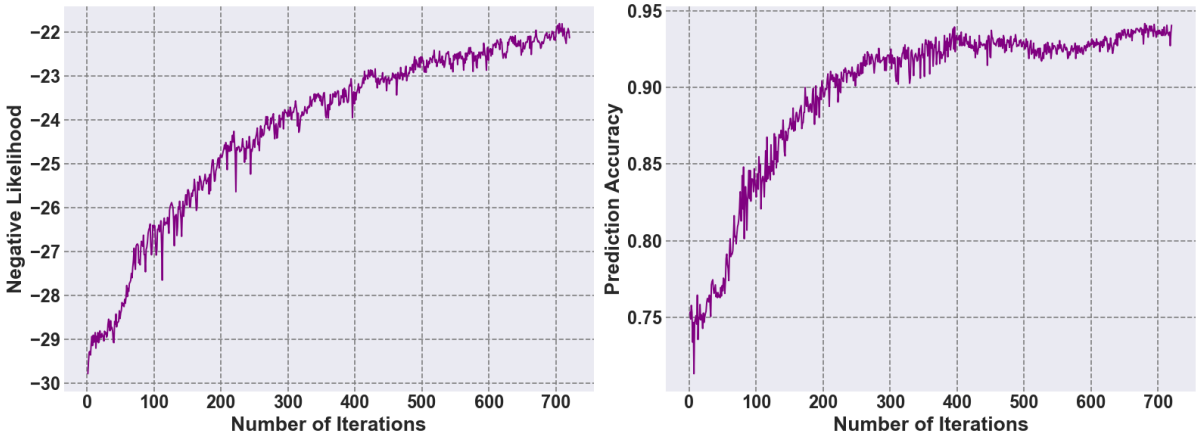

Fig. 18: Metropolis sampling baselines

In contrast, the figure18 used the sampling algorithm of metropolis hasting to train the RBM. 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 already returned 94.15%. The negative likelohood is also growing faster than in gibbs sampling and has less spikes in the beginning showing a more continuously learning rate. Here, the best negative likelihood value is -21.80. The gathered data can later on be used as baseline against the desired new updating mehtod of sampling with a Hopfield Network. Furthermore, the data is comparable and show similar results to those from the literature and are therefore to be considered correct.[236] 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.

4.4 Second Design and Evaluation phase

This **design phase** iteration has the goal of implementating the noisy Hofpield Network as described in the chapter 2.4.4. The functionilities of a RBM can be reproduced by the Hopfield Neural Network by correctly tuning the noise. Following subfunctionalities need to be established: imitating the RBM activation function, drawing random neurons to update, correct injection of the gaussian noise scale, calculating the weighted sum, comparing the weighted sum + bias + noise agaisnt the threshold and saving the new neuron configuration. Hence, this iteration is accessing new ground since the implementation of the simulator pipline is not validated yet.

First, a new file `_hopfield_network_v1.py` is created that aims to first establish a 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

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

sigmoid activation function of the RBM. 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. The reason for the thermalization is, so that the internal network can get into a flow and do some sampling steps to be statistical independent in order to have un unbiased sampling. The threshold as defined in the update formula is 0. As experimented the update formula for 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)

    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
```

In the beginnig a random neuron is drawn to be updated, which currently everytime is neuron number one because the network size is one. This allows to meassure the activation probability and fast iteration time with a clear result on how the network behaves. Calculating the weighted sum can be seen as the core of the update formula and is done first. Afterwards to compare against the threshold an bias is added. The value of the bias ranges from $-6; 6$ in step sizes of 0.025. After completing all samplin iterations beginning with -6 the step size is added to the bias until all iterations are made. This is sufficient for the neuron activation function of an ordinary Hopfield Network and results in the binary step. 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). That following figure 19 **Evaluates** the resulting activation probability of the single neurons.

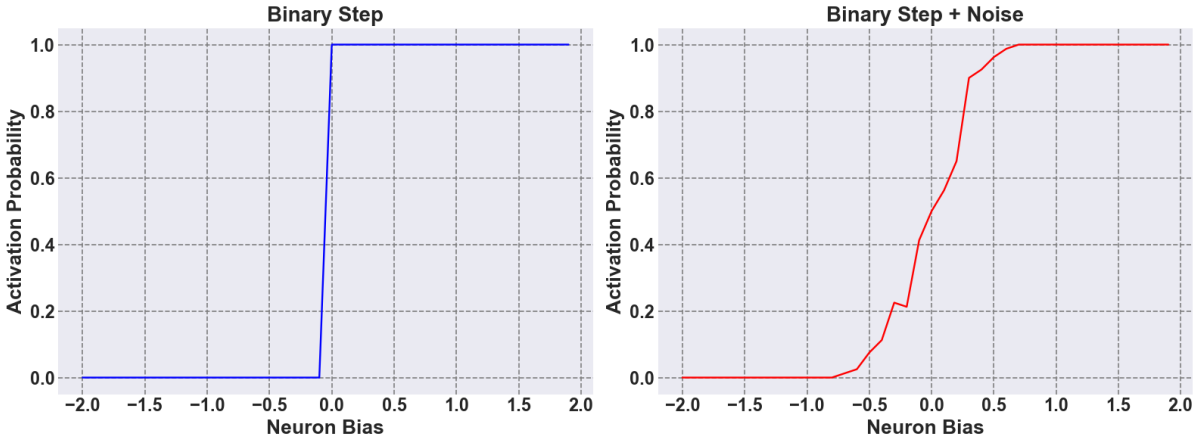

Fig. 19: Modification of the the Hopfield Network binary step activation function

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 sme would expect it; once the bias reaches 0 the neuron is activated all the time. Far more interesting is the red plot on the right side, which introduces some noise into the binary step function, bringing it closer to the logistic sigmoid activation function used in the RBM. To achieve this behavior, injecting noise through a Gaussian normal distribution can modify the activation function, making it compatible with the sigmoid function.[237] Technically this is performed by adding `np.random.normal(0, scale=1.75)` to the weighted sum and the bias, with 0 beeing 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.

The right plot shows that the noise injection works, even though it doesn't perfectly copy the sigmoid function. Concluding from this the sigmoid function is not achieved and the trainig of an RBM probably fails. Therefore, to completely evaluate and ensure that the injected noise fits to the logistic sigmoid function, the function itself is plotted as index, to have a visual comparison. Now, finding the correct standard deviation is the goal. In following figure20 the hyperparameter is tuned and the resulting scale is visualized. It verifys that a noisy activation function of a Hopfield Network can imitate the sigmoid function of a RBM correctly:

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

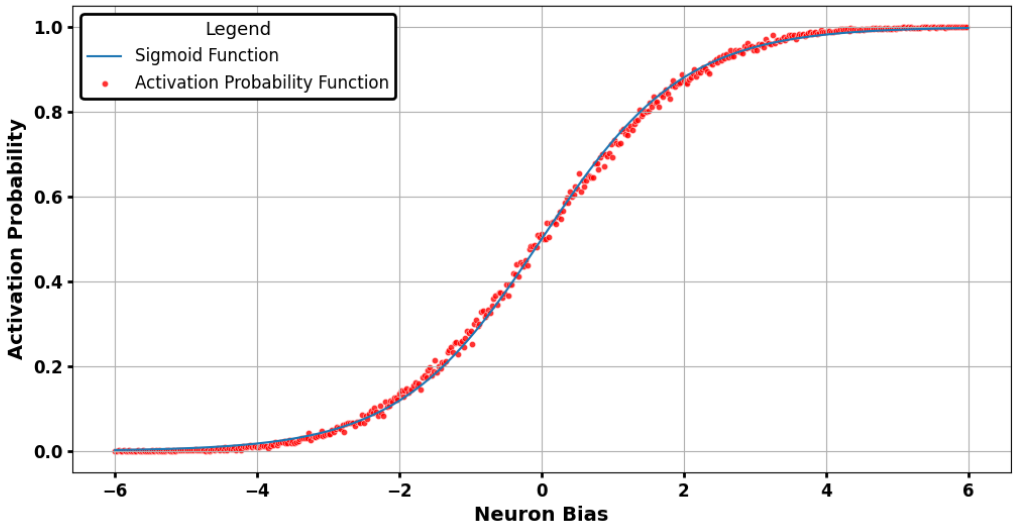

Fig. 20: Noisy activation function of the Hopfield Network imitating the RBM

4.5 Third Design and Evaluation phase

Now, that the proof of concept has been validated, this **Desing Phase** has the goal of connecting the Hopfield Network as a sampling method to the RBM and therefore enable a complete training. This includes using the weights and biases of the RBM 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 RBM with the sampling method of the Hopfield Network. Furthemrore, if the training is successful, the possibility of the N/2 half updating method instead of asynchronously updating of the states should be implemented. 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.

```
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 3. 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. As mentioned in 2.2.3, the diagonal elements (self-connections) are set to zero in RBMs.

Another important aspect is to maintain the model's symmetry, which is crucial for the energy-based nature of RBMs and the dynamics of Hopfield networks. Hence, for this reason and simplicity, the weight matrix is initialized as a symmetric matrix using NumPy's np.zeros function. This ensures that all initial weights are set to zero before being explicitly 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 RBM. 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 the first training of the RBM with the sampling method of the Hopfield Network can begin. The Hyperparmeters first are set to the same scale(1.75) and same thermalization(100) as for a single neuron but with more sampling steps (5000 sampling steps) than before. The results of the traning are visualized in figure21.

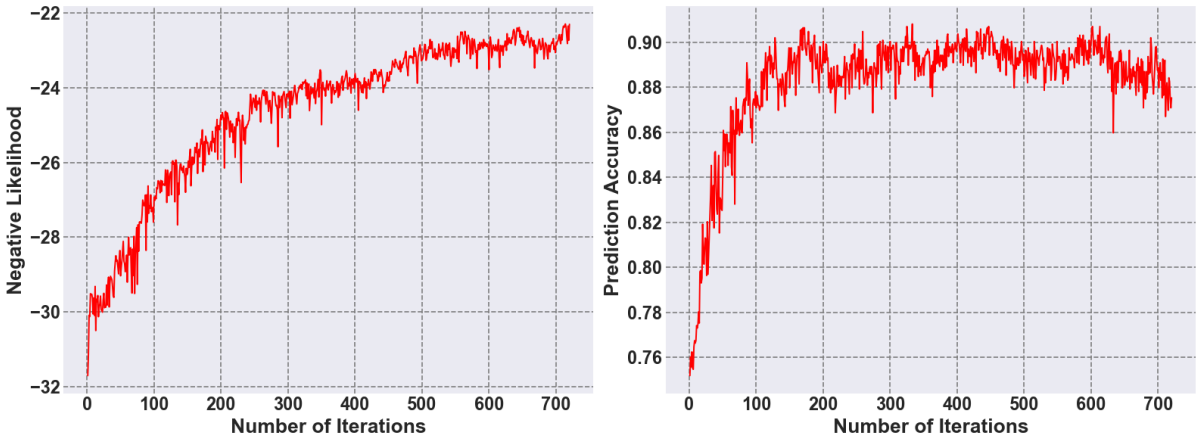

Fig. 21: Hopfield Network sampling baselines

First of all, it can be recognized that the training was successful, which was uncertain. Noteworthy, that with this method the output had to be transposed to be able to parse the data Scikit Learn. In a second look 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, so slightly worse than metropolis hastings and gibbs sampling. The right plot showing the prediction accuracy has the highest ascent of all the three graphs, which proofs that the less iterations are required to receive good results compared to the other two methods. Still, The best prediction value is 90.81% and therefore worse than gibbs and metropolis hastings. The reason for this can be the hyperparameter tuning. Since the noisy Hopfield Network method can become sensitive fast or is not adjusted correctly for this amount of neurons in the network, receiving this result without further adjustments already looks promissing.

As a result, the following hyperparameters are tuned to possibly received better outcomes. Specifically, the scale, which represents the standard deviation and is used as noise is tuned. In addition to that the amount of sampling iterations within a single training iteration is tuned. One extended parameter could be the learning rate and the total iterations but to have an appropriate benchmark against the other two sampling methods these parameters are locked in. Last but not least, the possibility of tuning the thermalization could help out too even if slightly less significant than the other two hyperparameters. Since the training takes around 40 minutes to complete tuning too many hypaerparameters takes too much time for the period of this thesis. First hyperparameter researched is the influence of the standard deviation (scale) to the maximum prediction accuracy. Especially the maximum prediction accuracy of the last 50 training iterations are gathered since this is the relevant area for inference. Given that the Hopfield Network operates as a statistical sampling method, the standard deviation is also calculated for these final 50 training iterations. This is done for the scale range of 1.0 to 2.0 within step sizes of 0.05, totalling to 21 single trainings done. The result is illustrated in figure22:

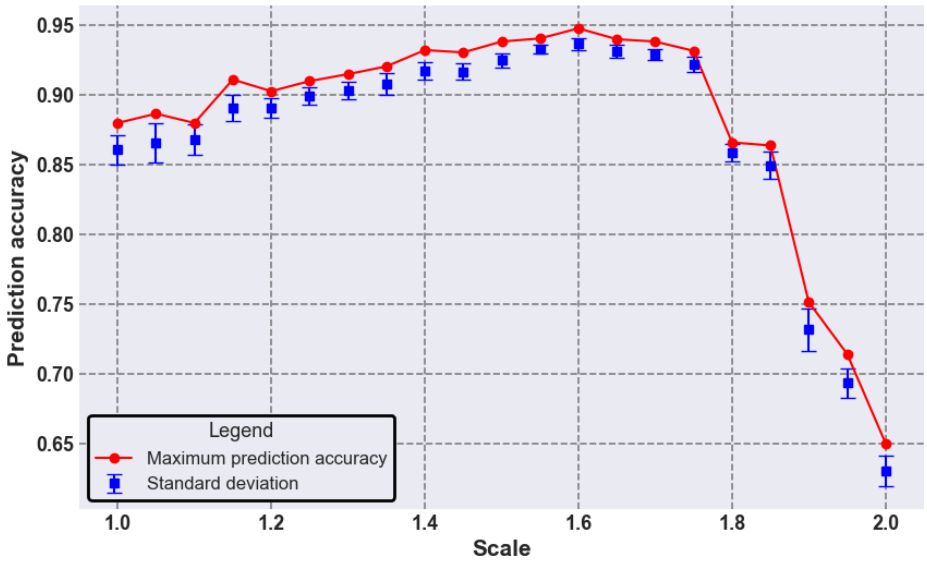

Fig. 22: Hopfield Network Hyperparametertuning scale

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 falls off a cliff indicating that the activation function of the RBM is not modelled correctly anymore. This shows that depending an network size and workload the adjustment within this method is important to achieve good results. The standard deviation follows the maxmimum prediction accuracy pretty closely and has no outliers indicating that the prediction accuracy is only a lucky random training. Close to the sclae of 1.0 the deivation is slightly lower compared to the rest of the plot, showing that the scale could be too low to model the sigmoid function correctly.

In a next step, the best fit with a scale of 1.6 is used for the second hyperparameters. The sampling iterations within one training iterations are now tuned. Hence, the decision is to begin with 1000 sampling iteration contiiunig with an increase of 1000 iterations until 15000 iterations are reached. With that the training showed that the interesting are is around 1000 to 4000 iteration and that the step size of 1000 is too big for that. To be more granular two extra trainings with iterations 1500 ad 2500 werde 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:

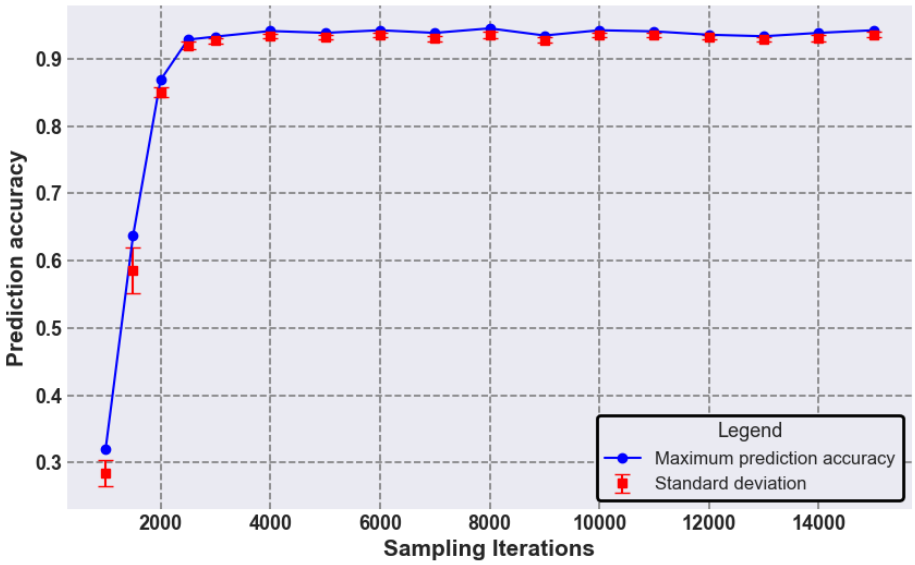

Fig. 23: Hopfield Network Hyperparametertuning sampling iterations

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, which indicates that not enough neurons were updated in the sampling process. However, with the number of iterations increasing, the error bars become smaller resulting in a more stable accuracy. Key take away is that after 4000 iterations there are no significant changes to the outcome of the prediction accuracy.

With the successful training and tuned hypaerparameters in the next step the possbility of the N/2 half updating methdod should be researched 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.[238] 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] 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)):
```

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

```
#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 neuron index now assigns either a 1 or a 0 to each neuron, with 1 meaning that the sum of this neuron is calculated and will be compared against the threshold. In a completely random process this would about lead to 25% of all neurons updated ($50\%drawn * 50\%updated$). To create a good comparison, the same parameters scale and sampling iterations are tuned for this updating method. In this method, 21 complete training sessions are also carried out for this purpose. Beginning with the scale the result is visualized in figure24 :

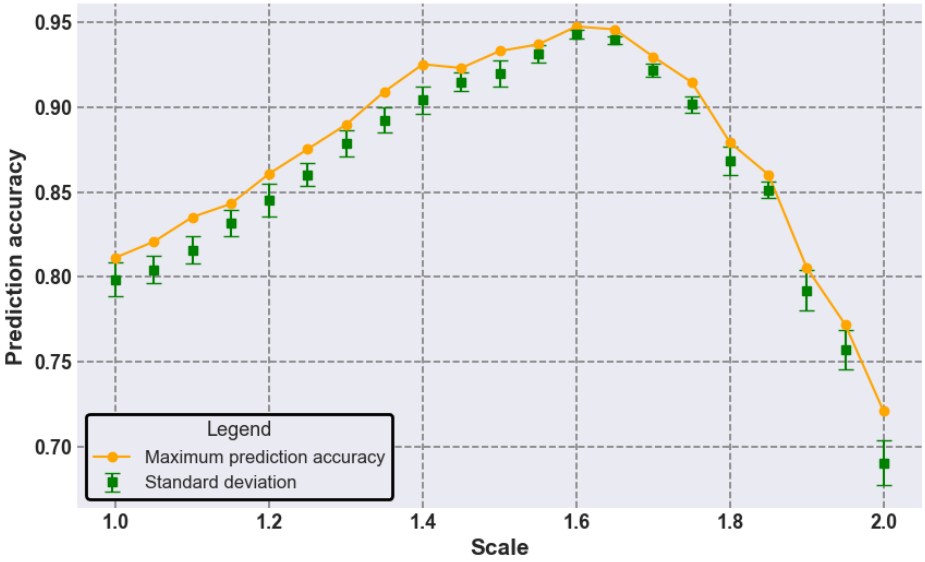

Fig. 24: Hopfield Network Hyperparametertuning scale for N/2

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 falls off a cliff. 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 configuration of the hyperparameters is tougher for this updating method and results can get worse more easily. The standard deviation represented in the error bars is high at the lower scale values validating that the sigmoid function is not properly mapped. This is similar for too high values beginning at a scale of 1.8.

The second hyperparameter are the amound of sampling iterations required to achieve good results. In the training process the plan is to begin with the same step sizes like earlier on. Soon it was clear, that the granularity needs to be much finer too achieve good results. Therefore, 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 new results are illustrated in figure25:

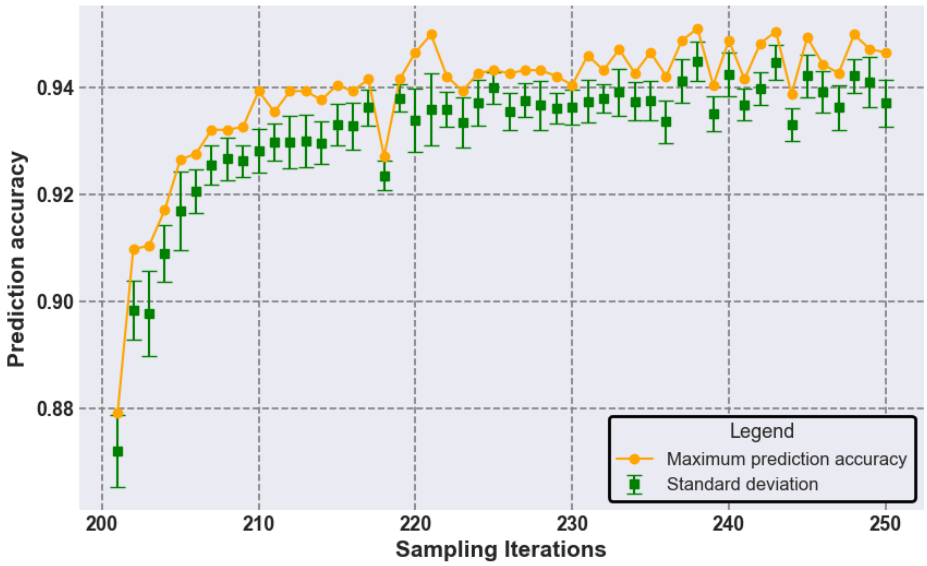

Fig. 25: Hopfield Network Hyperparametertuning sampling iterations for N/2 half

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 message here is that far less sampling iterations are needed 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. In return, this updating methods saves time and also has less energy consumption required.

With this third design and evaluation phase ending all the functionalities of the prototype are implemented and the prototype itself is complete.

4.6 Final Evaluation phase

In this final evaluation phase the functional prototype should now be validated within the evaluation phase of the DSR framework. The goal is to answer the research question “Can Boltzmann Machines be **efficiently** implemented on physics-inspired Hardware accelerators by analog noise

injection?". Since the key word here is efficiently, a simulation will be performed to meassure the parameters required 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.[239] Hereby, throughput can be defined as the time needed per Hopfield cycle (sampling iteration) per second.[240] 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.[241] 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. The equation to calculate the statistical dependency of 2 successive samples is the auto-covariance function:

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

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 accelerator is needed. The clock frequency is around 700MHz and therefore can sample 700 million samples per second, which is about 1.44ns for a single clock cylce.[242] Noteworthy, these numbers are takem from a neural network with 111 neurons. On the other hand, the only input variable for the energy consumption is the energy model of the mem-HNN, which is introduced in 4.2.1 and is developed by HPE in combination with the Forschungszentrum Jülich. This allows to meassure each of the individual energy consumptions that are configured to the specifications of the mem-HNN hardware accelerator.[243] For both of the methods one input of course is the finished prototype, that mirrors the functionalities of the ASIC on a high level.

With the simulation model specified, the autocorrelation can be implemented into the prototype.

[239] 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
[240] cf. Böhm et al. 2022a, pp. 6–7
[241] cf. Tanaka/Tomiya 2017, pp. 1–6
[242] cf. table1 Cai/Kumar, Suhas/Van Vaerenbergh/Sheng, et al. 2020, pp. 7–8
[243] cf. Hizzani et al. 2023, pp. 1–5

The decision is 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 that is not too biased in terms of the statistical approach. In the attachment4 the full implementation of the autocorrelation function is available. To compare the performance of the individual 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 and mathematics(half-life as a threshold) etc..[244] Folliwng results are visualized in following figure26:

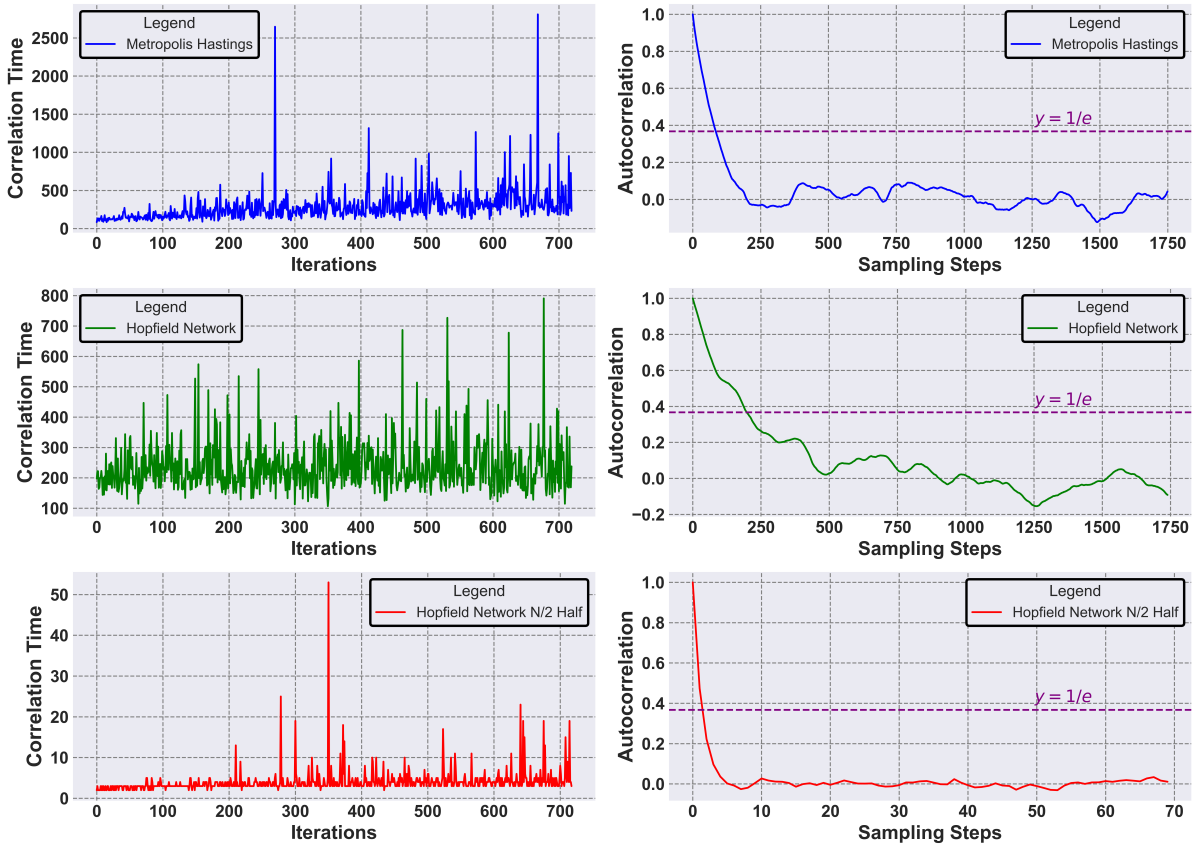

Fig. 26: Autocorrelation for the three sampling methods

The first row shows the conventional **Metropolis Hastings** sampling method. On the right plot the autocorrelation and the according sampling steps are visualized. 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 first sampling step within each training iteration that falls below this threshold. Here, named correlation time. It can be seen that the the scale on the x-axis in the left plot is higher compared to the other two sampling approaches, meaning that metropolis hastings is **more sensitive**.

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

The second row is the **single neuron 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**. The result verifys the new methods right to exist with only 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 traing 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. Since the correct value for not falling under the threshold would be infinity the scale in the plot would be too large. Therefore, the same plot with these two outliers can be found in the attachment5.

After identifying the autocorrelation of the desired metric, "throughput," it can be calculated by combining the autocorrelation with the cycle speed of the mem-HNN. This is done by taking the autocorrelation time of both Hopfield Network approaches (with N/2 Half and without) and mutlitplying them with the cycle time of the mem-HNN. In addition to that the inverse of the result is calculated resulting in the desired throughput metrics " samples/second". Noteworthy, the cycle time for the mem-HNN is 1.44ns for 111 neurons in the network. 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:

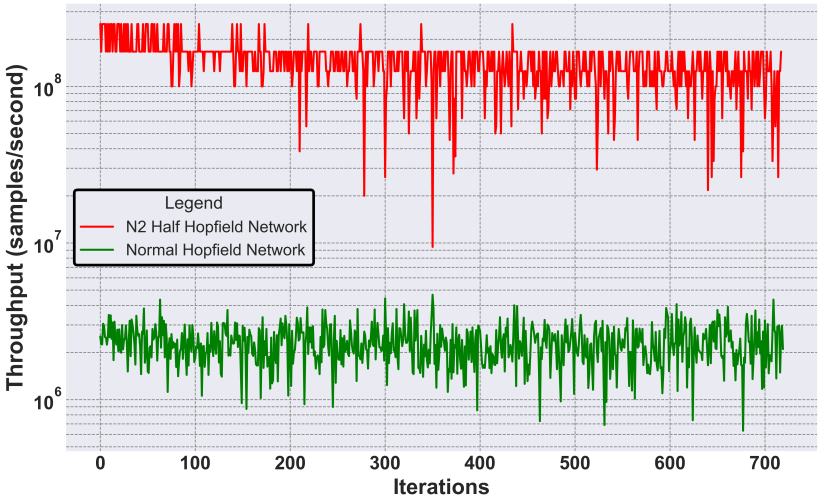

Fig. 27: throughput

The following results can be gathered when comparing the two lines in terms of efficiency and stability: The Normal Hopfield Network (green line) displays a stable throughput performance throughout the iterations, consistently above $\mathbf{10^6}$ samples per second. This suggests a stable and predictable behavior. The N2 Half Hopfield Network (red line) exhibits significant variability in throughput. While it generally hovers around $\mathbf{10^8}$ **to** $\mathbf{10^9}$ the throughput is more sensitive and fluctuates more compared to the single update approach. To get a better feeling of the data, the average of the throughput is calculated: N/2 Half Hopfield Network has an average of **144 megasamples/second**, while the Normal Hopfield Network has an average of: **2,3 megasamples/second**. This means that the computation speed of the N/2 Half method **is faster** by a factor of **62.72x**. Also in the attachment6 is a version with the outliers included symolizing a throughput of zero for these two iterations.

The results of the autocorrelation implies a low computing time and low energy consumption than with conventional hardware. To confirm this theory, the second metric to simulate is the **energy consumption** in energy/operation. Hence, the mentioned energy model needs to be implemented into the Hopfield Network interface. The first adaptation is to **intialize the energy model** with the size of the neural network (164) as it 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. 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 needs to be adjusted to handle positive and negastive weights seperately. Specifically, the weight matrix is discretized to the according bit resolution of the mem-HNN, which is 5bit. In the ealier design phases, the values were calculated with perfect resolution but for the energy model of the accelerator this is possible. Despite the low bit resolution there are many papers that proof that even with a small resolution a good performance can be achieved.[245] Since only positive weights can exist within the accelerator, these are separated and written into their own matrix. This modification is essential to accommodate the hardware constraints that prevent the use of negative weights.

On the other hand, A_Wl is the average configuration change. This means 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 to initially let the metal ions flow. Therefore, every position in the configuration must be compared with the following configuration position wise. 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 network configuration. An estimate for this, especially for N/2 half is that 25% of the neurons are updated in one training iteration (50% randomly drawn and 50% of that updated). All the adjustments for the energy model to the interface can be found in the version 5 of the hopfield interface as part of the digital delivery. The result of the energy consimption is illustrated in following figure28:

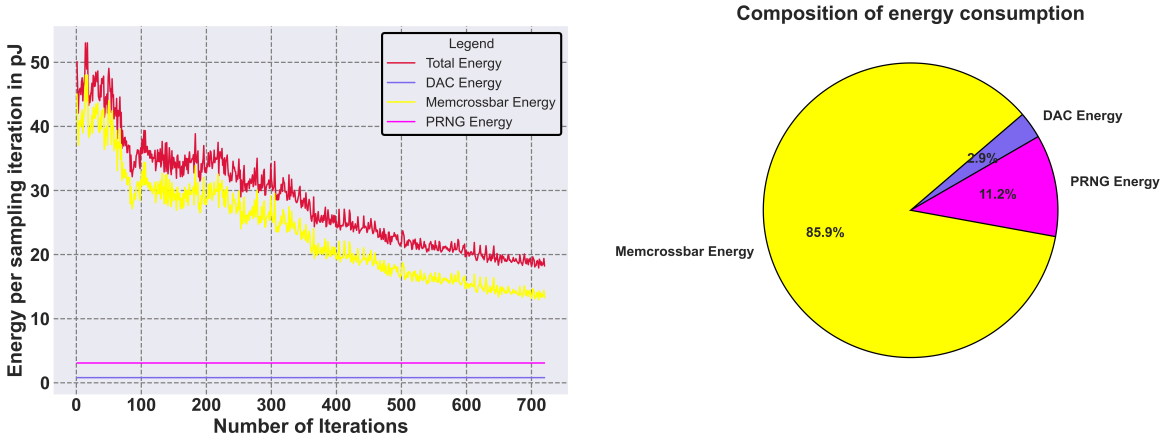

Fig. 28: Energy consumption of the mem-HNN

The left plot shows the energy consumed per sampling iteration (eq. to 1 clock cylce of the mem-HNN) 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. The decrease of the energy consumption over the time

[245] 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

comes from the weights of the neural netowotk that get better and better configured over time. Noteworthy, the Memcrossbar energy is the only variable energy consumer that actively changes, while the digital-to-analog converter and the pseudo-random number generator stay horizontal. A composition of the energy consumption is shown in the right plot. Here 85% of the energy is consumed entirely by the Memristor Crossbar and the support entities consume around 15% of the energy. The results of the energy model an the energy consumption can be verified by the following paper.[246]

Finally, it is important to mention that not all hardware components are included. and especially the communication and updating of For example the memory and the controller of the mem-HNN and are not included in the plot. Also to consider are the weights in the digital computer, that probably consume 10x the energy of the training on the analog mem-HNN.

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 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.

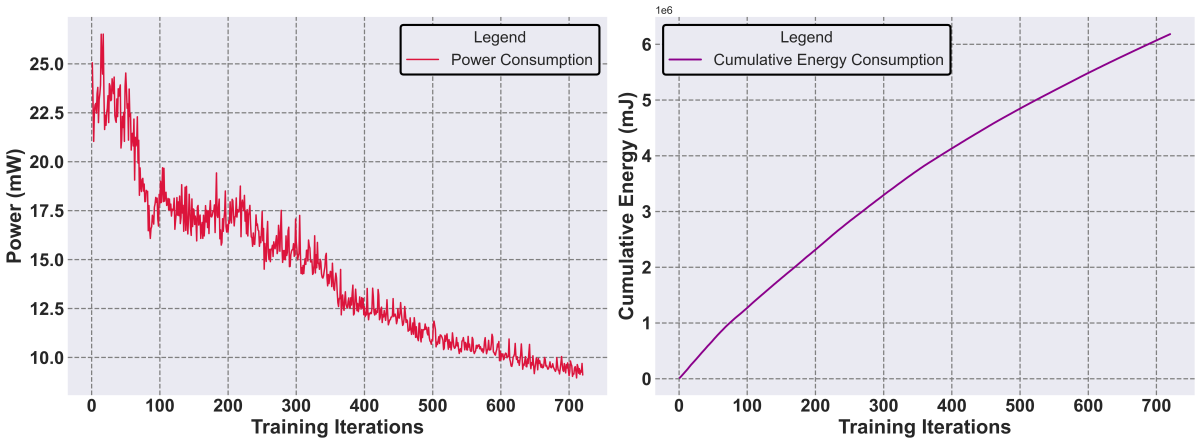

Fig. 29: Power consumption of the mem-HNN

As shown in figure29 the power required to train is around 22.5 mW, which compared to a basic CPU(20-40W) is significant less. 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 entire power consumption is mostly stable with some larger fluctuatins in the beginning of the training areound 0 to 70 iterations. When looking at the right plot, which symbolizes 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. Again it is important to

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

keep in mind that these numbers consist of only the mem-HNN and do not include the digital computer.

With that the evaluation phase can successfully verify that the models purpose of the simulation, which is to answer the reasearch question, is achieved. Therefore, Boltzmann Machines can indeed be efficiently implemented on the physics-inspired Hardware accelerator by analog noise injection.

5 Diffusion and discussion of results

von Holzweißig: Dieses dient dazu die Ergebnisse des eigenen Beitrags zusammenzufassen und kritisch zu diskutieren. Auch kann hier der Versuch einer Ergebnisverallgemeinerung erfolgen. Vergleich mit der Literatur Evaluation der Erkenntnisse in Bezug auf die Zielsetzung intrinsisch

Was sind miene Hauptergebnisse?

5.1 Solution architecture and Hyperparameter tuning

5.1.1 Gibbs sampling

evtl. weglassen und nur als Baseline benutzen

5.1.2 Metropolis Hastings

evtl. weglassen und nur als Baseline benutzen

5.1.3 Single update Hopfield Network

Hyperoarameter Scale und Iterations

5.1.4 N/2 Half update Hopfield Network

Hyperparameter Scale und Iterations

5.2 Throughput

Vergleich mit CPU usw. für N/2 Half

5.3 Energy consumption

Vergleich mit CPU usw. für N/2 Half

5.4 diffusion

Inspiration, kann ähnlich sein Veröffentlichung in den Labs, Paper veröffnetlichung, auf messen gehen mit Ergebnissen, Weiterentwicklung für Komplette umsetzung des Modells auf dem richtigen hardwarebeschleuniger, und Einreichung der Bachelorarbeit an die DHBW

6 Mission of the work

6.1 Critical reflection on the results and methodology

6.2 Extrusion of results for theory and practice

6.3 Outlook

Appendix

List of appendices

Appendix 1: So funktioniert's

Um den Anforderungen der Zitierrichtlinien nachzukommen, wird das Paket `tocloft` verwendet. Jeder Anhang wird mit dem (neu definierten) Befehl `\anhang{Bezeichnung}` begonnen, der insbesondere dafür sorgt, dass ein Eintrag im Anhangsverzeichnis erzeugt wird. Manchmal ist es wünschenswert, auch einen Anhang noch weiter zu unterteilen. Hierfür wurde der Befehl `\anhangteil{Bezeichnung}` definiert.

Appendix 2: House of Prototyping Guidelines: Prototyping Dimensions

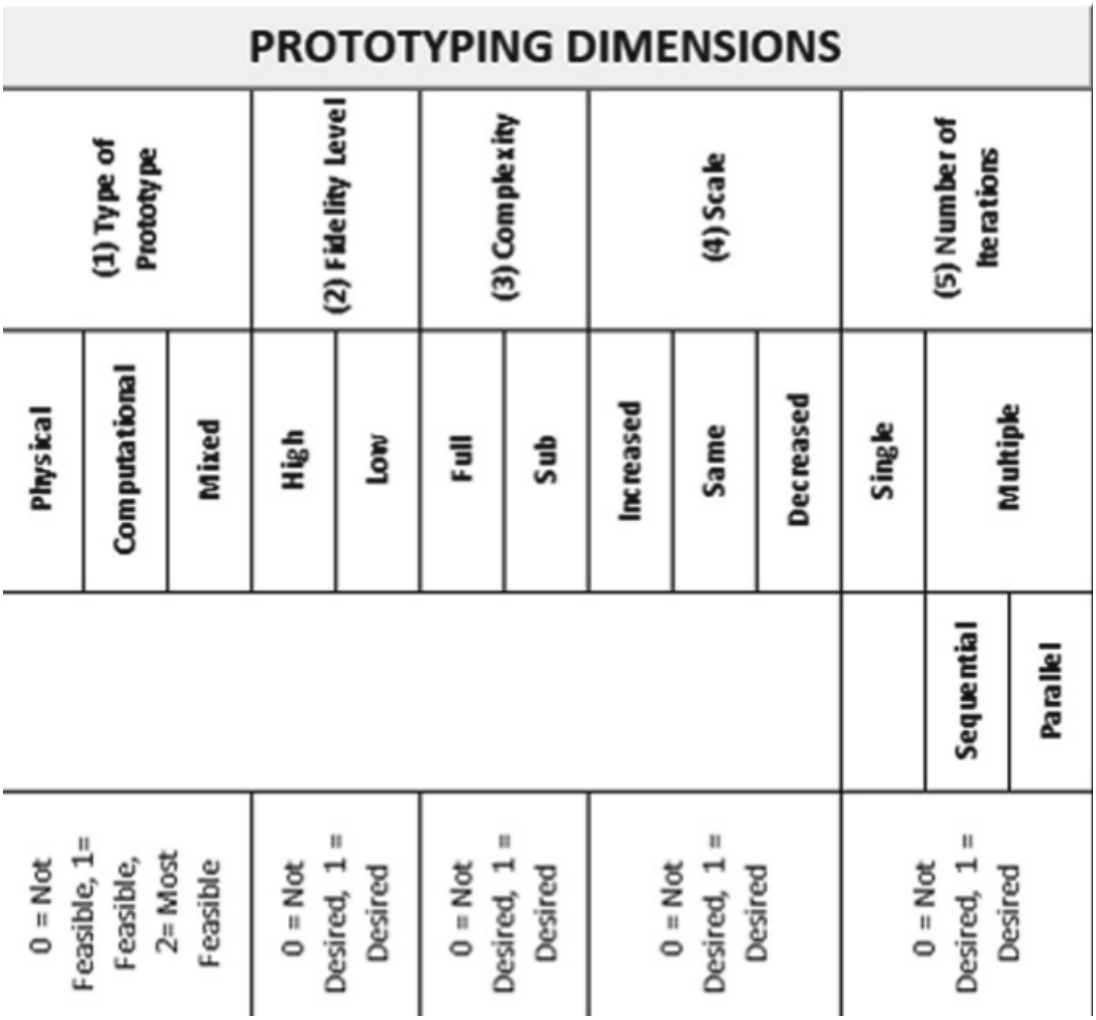

Fig. 30: Prototyping Dimensions to categorize prototypes

Appendix 3: 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 4: 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 5: Autocorrelation for all sampling methods with outliers

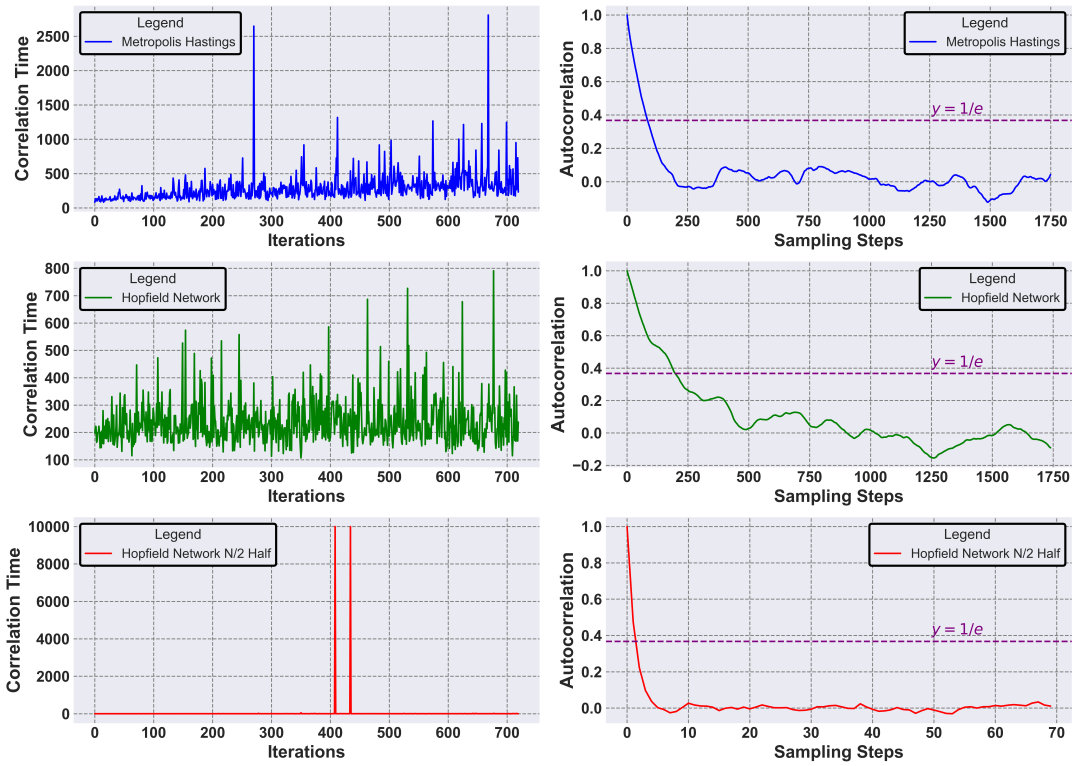

Fig. 31: Autocorrelation for the three sampling methods with outliers

Appendix 6: Throughput with outliers for all sampling methods

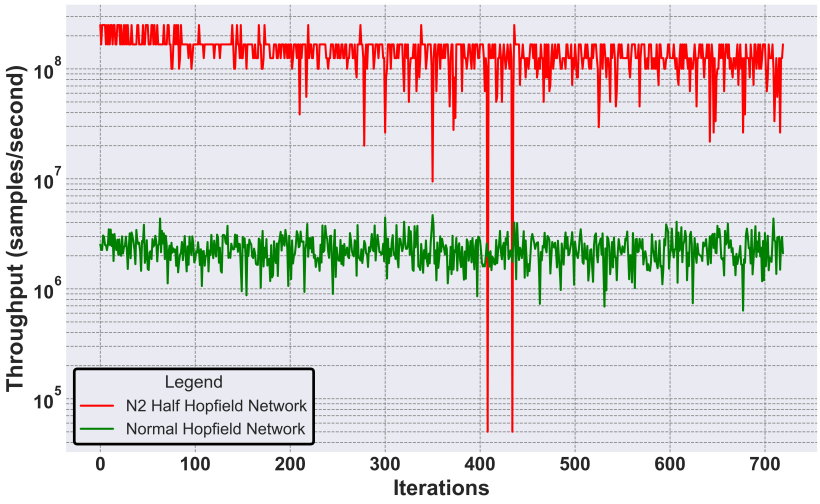

Fig. 32: Throughput with outliers

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).

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).

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).

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).

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).

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.

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).

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 Chemi-*

cal 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.

– **(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).

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.

Ranzato, M. a./Poultney, C./Chopra, S./Cun, Y. (2006): Efficient Learning of Sparse Representations with an Energy-Based Model. In: *Advances in Neural Information Processing Systems*. Vol. 19. MIT Press. URL: https://proceedings.neurips.cc/paper/2006/hash/87f4d79e36d68c3031ccf6c55e9bbd39-Abstract.html (retrieval: 03/07/2024).

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./Mel-

nick, 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.

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)