Department of Physics and Astronomy

University of Heidelberg

Master thesis

in Physics

submitted by

Manuel S. Rudolph

born in Darmstadt

2020

Exploring and Benchmarking Quantum-assisted Neural Networks with Qubit Layers

This Master thesis has been carried out by Manuel S. Rudolph

at the

Kirchhoff-Institute for Physics

under the supervision of

Prof. Fred Jendrzejewski

and

Dr. Sebastian Schmitt

Honda Research Institute Europe GmbH

Erforschen und Bewerten von Quantenunterstützten Neuronalen Netzwerken mit Qubit-Schichten:

Das Ziel dieser Arbeit ist es, anwendungsbezogene Quanten- und quantenunterstützte maschinelle Lernalgorithmen zu erkunden und mögliche Vorteile von quantenunterstützten neuronalen Netzen mit Qubit-Schichten zu ermitteln.
Zwei Quanten-machinelle Lernalgorithmen werden vorgestellt, welche die Codierungskapazitäten und Samplingvorteile von Qubits demonstrieren. Als eine mögliche Erweiterung dieser generativen Modelle wird der Quantum-assisted Generator (QaG) vorgestellt, welche eine Qubit-Schicht mit nachfolgenden klassischen Schichten verknüpft. Der QaG liefert vielversprechende Anzeichen, dass Quantenphänomene die Leistung von generativen neuronalen Netzwerken verbessern können.
In dieser Arbeit werden außerdem Hamiltonian-basierte und Gatter-basierte Umsetzungen von quantenunterstützten Autoencodern vorgestellt. Unter Anwendung eines Hybrid-Trainingsansatzes, welcher Black-Box Optimierung und herkömmliche Gradientenabstiegsverfahren kombiniert, wird gezeigt, dass der Hamiltonian-basierte Autoencoder den MNIST Datensatz lernen kann und dabei gute Generalisierungseigenschaften aufweist. Mehrere Ansätze werden vorgestellt, wie die dargestellten quantenunterstützten Algorithmen erweitert werden können, um mögliche Vorteile von Quantensystemen im Zusammenhang mit künstlichen neuronalen Netzen weiter zu untersuchen.

Exploring and Benchmarking Quantum-assisted Neural Networks with Qubit Layers:

The aim of this work is to explore practical implementations of Quantum and Quantum-assisted Machine Learning algorithms and benchmark potential benefits of utilizing quantum phenomena in Quantum-assisted Neural Networks with qubit layers.
Two known approaches of generative Quantum Machine Learning algorithms are revised to demonstrate the encoding capability and sampling benefits of qubits. As one possible extension of those generative models, the Quantum-assisted Generator (QaG) is presented which implements a qubit layer coupled to subsequent classical layers. The QaG provides promising indications that quantum phenomena may enhance performance of a generative neural network.
This work also considers Hamiltonian-based and gate-based implementations of quantum-assisted Autoencoders. Using an effective hybrid training approach consisting of simultaneous application of conventional backpropagation and black-box optimization, we show that the Hamiltonian-based Autoencoder is able to learn the MNIST data set with good generalization properties. Several approaches to extend the presented quantum-assisted algorithms are proposed to further investigate potential benefits of utilizing quantum systems in combination with Artificial Neural Networks.

Contents

1 Introduction

The aim of this work is the investigation of algorithms for near-term quantum computing devices towards practical application in real-world problems. A very promising area of such applications is applying quantum computing techniques to machine learning. Therefore, this works explores a multitude of Quantum and Quantum-assisted Machine Learning algorithms to study potential benefits in implementing quantum computing devices and quantum phenomena for practical application.
This chapter provides a brief introduction to the field of quantum computation as well as potential and challenges of current quantum devices. Additionally, it introduces the necessary concepts in Machine Learning and Artificial Neural Network in particular that are required for the algorithms presented in this work. Finally, the notion of Quantum Machine Learning is introduced to establish the direction of research for this work.

1.1 Quantum Computation

Quantum computers are processing hardware which explicitly utilize quantum effects for computation. This section provides a brief introduction to the field of quantum computation, different implementations of quantum computing devices, and their potential and challenges that are relevant to this work.

1.1.1 Background & Motivation

Even though theoretical descriptions of quantum computers have been proposed and studied for roughly 40 years [11], significant progress in the practical implementation of quantum computers has come in recent years[8] [22][33]. Richard Feynman postulated in 1982 that in order to simulate quantum systems, one would need a quantum computer [23]. Following this train of thought, the perspective of quantum computers is to perform calculations which classically require computational resources that scale exponentially with the problem size [1][21]. Some quantum algorithms have been developed which prove to have a theoretical speedup compared to the best known classical algorithms. The most prominent examples are the Shor algorithm for prime factorization in polynomial time [56], the Harrow-Hassidim-Lloyd (HHL) algorithm for solving systems of linear equations with an exponential speedup [30] and Grover's algorithm for searching data bases [28]. Those would certainly have a dramatic impact current technology if implemented in practice on a quantum computers. This search for so-called *quantum supremacy* has again become an active field of research as now the experimental implementations of quantum computers

are becoming better at an impressive rate and the first steps have been made to demonstrating this in practice [8].

The reason why there is the strong belief that quantum computing could provide drastically more efficient computation is related to a few aspects of quantum machanics. Conventional computers and classical processing units work with *bits* to perform calculations or store information. Bits are binary, i.e. can be 0 or 1. The quantum analog to the bit is the *qubit* (quantum bit). It is a quantum two-level system which can be in a *superposition* of 0 and 1. In a n qubit system, the computational basis thus scales exponentially with 2^n. Additional hopes are derived from quantum *entanglement* which may allow to encode correlations between different quantities in qubits very efficiently. With precise control of the qubit system, one might be able to significantly enhance conventional computation in certain applications where an interplay of superposition and entanglement prove to be vital.
In practical implementations, a lot of expected benefits are currently made impractical because of imperfect control of qubit systems and challenges concerning efficient loading of data into a quantum computer and read-out of the qubit states [1][21].

1.1.2 Current and Near-Term Quantum Devices

The three most common types quantum devices or quantum hardware are analog quantum simulators, quantum annealers, and gate-based quantum computers. When using the term *Quantum Computer*, one usually refers to the latter gate-based devices. All approaches of implemeting quantum hardware aim to make use of fundamental quantum properties such as superposition and entanglement to perform calculations or routines that are otherwise unfeasible for classical computers.

Analog quantum simulators are highly precise quantum experiments. They implement a specific Hamiltonian with corresponding interactions and are therefore generally not suited for a wide range of problems. In other words, they are not *universal* in their calculations but can perform complex simulations with high fidelity in the case that the implemented Hamiltonian fits the task [7][16][62].

Quantum annealers use adiabatic state transfer to solve problems with discrete variables such as groundstates of spin-Hamiltonians and classical binary optimization problems. The quantum hardware has a physical temperature which leads to a thermal ensemble of solutions. Quantum annealing is a heuristic form of quantum simulation that is able to create complex quantum states but there so no conclusive evidence that it can provide a significant speedup compared to best known classical algorithms [5][10][14][60].

Gate-based quantum computers implement individual quantum two-level systems as qubits which can be addressed applying electromagnetic pulses of different frequency and duration. In reference to their classical equivalent, those operations

are called *quantum gates*. The specific implementation of the qubits, as well as the physical interactions utilized by the quantum gates, depend on the specific quantum hardware. Common implementations are Ion-trap quantum computers [24][34][36], superconducting qubit quantum computers [8][22][33] and optical lattice quantum computers [15][57]. Calculations on such gate-based quantum computers can be universal, i.e. quantum *Turing equivalent*, and are thus referred to as *universal quantum computers*.

All types of current quantum devices have advantages and disadvantages but generally suffer from the following problems:
Scalability, noise, fidelity and loss of coherence. This is especially true for gate-based quantum computers because analog quantum simulators and quantum annealers sacrifice universality for good performance on their specific application.
Most algorithms presented in this work are agnostic to the quantum hardware used as they are Hamiltonian-based. Those that are not Hamiltonian-based are gate-based. It is reasonable to assume that the algorithms can be adapted to run on analog quantum simulators using their specific operations and interactions. The algorithms are designed to be performed on near-term quantum devices and be robust against current challenges of limited qubits, significant noise and shallow quantum circuits.

1.2 Machine Learning

Machine Learning is a general term for algorithms and statistical models that perform tasks which they are not explicitly programmed for. Results or actions are based on collected data or past experience and not pre-determined behavior. Machine Learning has a wide and rapidly growing range of applications in data sciences like image- and language processing, recommendation systems, optimization, and many more. The extraction of vital experience and information is commonly done by *training* a Machine Learning model on known data. In one form or another, all machine learning approaches follow a parametrized ansatz where training consists of tuning the model's parameters to fit the training cases.
A very broad and highly useful introduction to Machine Learning specifically for Physicists can be found here [43].

1.2.1 Categorizations of Machine Learning Algorithms

One of the main categorizations of Machine Learning algorithms is into *supervised* and *unsupervised* Machine Learning algorithms. Crucial to the differentiation is the amount of a-priori or external information the model is given when learning a data set. If data X comes with a *label* Y, the learning method is called *supervised learning*. A label can be the correct content of an image, the correct grouping of data points, the right response to an external stimulus, etc. Supervised Machine

Learning models are not the focus of this work.
If the training set is not labeled, which is naturally the case in most collected data, one has to revert to *unsupervised* Machine Learning algorithms. Unsupervised Machine Learning models extract information and patterns based on the data itself. All quantum and quantum-assisted Machine Learning algorithms in this work are unsupervised. Autoencoders are popular type of unsupervised Machine Learning algorithm. They are Artificial Neural Networks that learn efficient low-dimensional representation of data [13][25][63].

Another common categorization divides Machine Learning algorithms into *discriminative* and *generative* approaches. Many discriminative algorithms are inherently supervised. The main goal of discriminative algorithms is to model the conditional probability $P(y|x)$ which defines a mapping of a data point x to a label y. This mapping creates boundaries that discriminate between data points. Usually these algorithms are trained on *training data* which are viewed as a good subset of all realistic data samples. The decision boundaries should be *generalizable* to new data in order to decide on a likely label for a previously unseen data point. *Overfitting* is a very common notion in Machine Learning and describes a model fitting its parameters too tightly to the training data such that the model does not generalize well to previously unseen data.
Generative Machine Learning algorithms aim to learn a parametrized approximation of the probability distribution $P(x)$ underlying the target data and allow to draw new samples from that learned distribution. The goal of a generative model is to be able to generate samples which are statistically indistinguishable from the training data used. Generated samples need not be a subset of the training set. In fact, it is generally a desirable property of generative models that they generalize learned systematic patterns in the training data and from that generate original samples.

1.2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) consist of a network of *nodes* and *edges*. Commonly, the nodes are structured in *layers* which are inter- but not intra-connected by edges. Fig. 1.2.1 shows a basic ANN architecture which is the foundation for the quantum-assisted Neural Networks described in this work with annotated nomenclature of the most significant components. The nodes of an ANN can have a variable value which is called *activation*. In a *feed forward* neural network, the first layer is initiated with an input that is propagated through the rest of the network. This is done by applying non-linear transformations between layers. In the most basic ANNs, the activation $\mathbf{v}$ of the subsequent layer with values v_i is calculated via

$$\begin{aligned} \mathbf{v} &= \zeta\big(\mathbf{W}\mathbf{u} + \mathbf{b}\big) \\ v_i &= \zeta\big(\sum_j W_{ij} u_j + \mathbf{b}\big) \end{aligned} \tag{1.1}$$

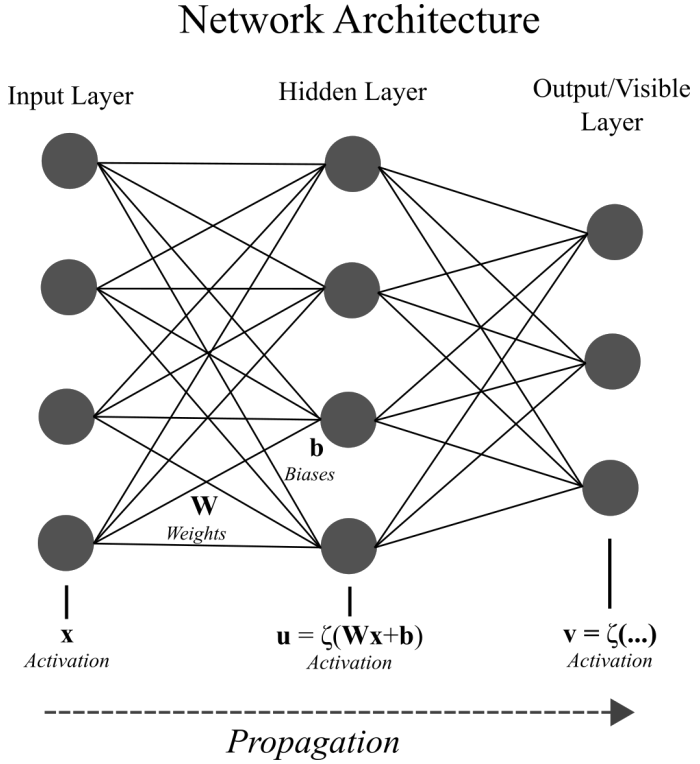

Figure 1.2.1: General example of the network architecture of an Artificial Neural Network (ANN). Activations for each layer depend on the weights **W** of the edges between the layer and the previous layer, biases **b** of the nodes, and a non-linear activation function ζ. Propagation of the activation here happens from left to right. Not every ANN has an input layer but always an output layer. Additional intermediate layers do not necessarily bare direct interpretation in terms of the data but they allow for more complex non-linear transformations and are generally expected to learn essential features of the data.

where ζ is a non-linear function called the *activation function* and the input to that activation function is calculated as a weighted sum of the activations **u** in the previous layer plus an individual bias b_i. The weights W_{ij} correspond to the edges between the nodes i and j in the network. Most of the 'magic' of ANNs lies in the non-linear transformation of data between layers. The choice of activation function closely depends on the type of ANN and the purpose of the layer.

Training an Artificial Neural Network consists of fine-tuning the network parameters, namely the weights **W** and the biases **b**, such that the output is as desired. Consequently, one needs to define a cost- or error-function which measures how good the result is. The resulting high-dimensional cost/ fitness landscape is used to optimize the parameters. Typically, the parameters of an ANN receive a *gradient descent* update in each iteration of training where the gradient of the cost function with respect to the respective parameter is calculated via *backpropagation*. Backpropagation is a term for the application of the chain rule in the calculation of the

derivative in order to the define the gradient of the cost/error in deeper layers of the network.

The training of a Neural Network can be viewed as high-dimensional point fitting. The parameters of the network are tuned such that they provide the best mapping between input and output. It is simple to understand that given enough parameters, any number of data point with arbitrary structure can be fitted. This is called *overfitting*. As a consequence of overfitting, models usually perform exceptionally well on the training set arbitrarily but poorly on unseen data. Overfitting thus reduces *generalizability* which is one of the intrinsic motivations to performing Machine Learning.
Common practices to prevent overfitting, for example regularization and dropout, weaken the model during training such that it has to focus on the most essential structural information of the training data with which it still performs well on new data. Regularization usually penalizes large network parameters which would indicate overfitting [38][58]. This reduces the expressivity of a model but can also help convergence because it also reduces complexity. Another kind of Regularization is called *dropout*. In dropout, random weights are set to zero during any training iteration so that the network learns a more robust model of the data [64].

A lot of choices are to be made when setting up and training an Artificial Neural Network, namely the number of layers, the amount of nodes per layer, connectivity between the layers, the learning rate for the gradient descent, number of training iterations, dropout rate, and many more. Those external parameters are called *hyperparameters*. The choice of hyperparamters can largely impact the training performance of an ANN and its generalization capability.

1.3 Quantum Machine Learning

With the growing interest in Machine Learning in current years, many physicists have moved towards implementing Machine Learning algorithms in order to enhance results of quantum experiments or improve our understanding of physical systems [17][20][54][59]. This type of *Quantum Machine Learning* is not studied in this work. Instead, in this work we focus on studying the implementation of quantum systems in order to enhance conventional Machine Learning algorithms. It has been proposed that an interplay with *Deep Learning* and other Machine Learning algorithms is one of the most promising applications for current and near-term quantum computing devices [1][4][9][12][21][42]. The goal with such Quantum or Quantum-assisted Machine Learning algorithms is not to perform large calculations on quantum devices but to only perform crucial sub-routines of a larger overall algorithm that would otherwise be very hard or intractable.

The general direction of this work is to make use of fundamental quantum properties that near-term quantum devices offer while respecting their limitations. For that, we implement qubits in unsupervised Quantum Machine Learning algorithms and more specifically in Artificial Neural Networks (ANNs). The networks may consist of just a singular qubit layer or a larger overall network with classical layers and one qubit layer at an essential position.
One fundamental property of a quantum system is that when measured, the system always collapses by random projection onto one basis state. This proposes quantum systems as prime candidates in generative Machine Learning algorithms where complex probability distributions can be encoded in quantum wavefunctions and then sampled by measurement of the state. Classical sampling is usually, and without a priori knowledge of the target distribution, either global but inefficient or efficient but local [12][42]. Recently, a claim of quantum supremacy with a quantum random number generator on 53 qubits has been published [8]. Though it is not yet clear how strong the quantum advantage is for that case, it demonstrates the growing capabilities of quantum computers and their application in generative tasks.
A somewhat different motivation for implementing qubits in ANNs is to potentially benefit from quantum advantages which arise from entanglement in the system. Conventional ANNs rely on a large number of linear and non-linear transformations between layers and a quantum system might aid the model in extracting essential structure by being able to represent highly-correlated data. Also, by implementing a small number qubits in a deep layer of an ANN, one minimizes the number of qubits required to put near-term quantum devices to practical use.

2 Techniques

This chapter showcases the techniques and concepts that re-occur throughout this work. It shows the spin-Hamiltonian on which all Hamiltonian-based Quantum Machine Learning algorithms in this work based are on. The Variational Quantum Eigensolver (VQE) is a classical-quantum hybrid algorithm which allows to find approximations of eigenvalues and eigenstates of Hamiltonians on near-term gate-based quantum computers. We also show numerical models which are used to implement and simulate the Hamiltonian-based models and the VQE. Many Quantum Machine Learning algorithms studied in this work are trained with the help of the *Covariance Matrix Adaptation Evolution Strategy* (cma-es) [29] optimization algorithm on training sets of different types which is also described in this chapter. The code and numerical models that have been used for this work can be found here [51][52].

2.1 Spin-Hamiltonian

Spin is the intrinsic angular momentum of elementary quantum mechanical particles and it can take two possible discrete measurement states for spin-1/2 particles. Here, we denote these two states by $s \in \{0, 1\}$. In the classical interpretation of spins, they can be viewed as vectors in euclidean space whereas in quantum mechanics, spins are hermitian operators which relate to the observable of spin expectations. They thus encode a quantum two-level system which in this work are interpreted as qubits (quantum bits).
In a n-spin system, the basis states are $\mathbf{s} \in |\{0, 1\}^{\otimes n}\rangle$, e.g. in a 2-spin system $\mathbf{s} \in \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. The quantum state of the spin system is a general superposition of the spin basis states $\mathbf{s}_i$ like

$$|\psi\rangle = \sum_i c_j |\mathbf{s}_j\rangle. \tag{2.1}$$

with $\sum_i |c_j|^2 = 1$. The coefficients c_j therefore encode a normalized probability distribution over the basis states. When measured, the quantum state collapses onto one of the basis states $|\mathbf{s}_i\rangle$ with probability $|c_j|^2$. The random projection upon measurement is a fundamental quantum property and utilized in every implementation of Quantum Machine Learning algorithms in this work.
Quantum states are called *entangled* if the total state of the quantum state $|\psi_{tot}\rangle$ cannot be written as a product of the individual quantum states like $|\psi_{tot}\rangle = |\psi_1\rangle|\psi_2\rangle$. In that case, the measurement outcome of $|\psi_1\rangle$ depends on the measurement $|\psi_2\rangle$ and vice-versa. For a more detailed read on Quantum Mechanics and spin systems, we refer to books in German and English language [27][46].

The spin-Hamiltonian that is used throughout this work is

$$H = \sum_{ij} \sigma_i^z J_{ij} \sigma_j^z + \sum_i h_i^z \sigma_i^z + \sum_i h_i^x \sigma_i^x. \tag{2.2}$$

Here, σ^z, σ^x are the z- and x- Pauli matrices respectively, J are pair-wise interactions of the spins and h^z, h^x local fields along the z- or x- axis.
Without the transverse field term, the Hamiltonian is diagonal in z-basis, which is the computational basis in this work, and represents a classical Ising model [40] with all-to-all interactions. Consequently, eigenstates of the diagonal Hamiltonian are single basis states, e.g. $|1100\rangle$ in a 4-spin system. The Hamiltonian groundstate is the spin configuration with the lowest energy and next higher eigenstates are spin configurations with the next higher energies. This model does not allow for superposition inside eigenstates and they are thus not entangled in computational basis. With the transverse field, the Hamiltonian is no longer diagonal in computational basis. Eigenstates are in general superpositions of basis elements like in Eq. (2.1) and are generally entangled.

For finite-temperature, the spin system tends towards the groundstate but the temperature induces excitations to higher eigenstates. In this case, the probability for each spin configuration $\mathbf{s}$ follows the *Boltzmann distribution*

$$P_\beta(\mathbf{s}) = \frac{e^{-\beta E(\mathbf{s})}}{Z} \tag{2.3}$$

where $\beta = \frac{1}{k_B T}$ is the inverse temperature and $Z = \sum_{\mathbf{s}} e^{-\beta E(\mathbf{s})}$ the partition function.

2.2 The Variational Quantum Eigensolver and Exited States

The Variational Quantum Eigensolver (VQE) is a classical-quantum hybrid algorithm to find upper bounds on groundstate energies of Hamiltonians [22][33] or optimize classical cost-functions on gate-based quantum computers [49]. The qubit state that minimizes the energy of a Hamiltonian is therefore an approximation of the Hamiltonian's groundstate. The VQE implements quantum resources in conjunction with classical variables to solve optimization problems with shallow quantum circuits on noisy near-term quantum devices.

The goal of the VQE is to variationally minimize an objective function $\mathcal{C}$ over a parametrized wavefunction ansatz $|\psi(\boldsymbol{\theta})\rangle$. The variational functional reads

$$\mathcal{E} = \frac{\langle\psi(\boldsymbol{\theta})|\ \mathcal{C}\ |\psi(\boldsymbol{\theta})\rangle}{\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta})\rangle} \tag{2.4}$$

State preparation for the qubits is performed by a parametrized quantum circuit ansatz $U(\boldsymbol{\theta})$ which can consist of arbitrary single- or multi-qubit gates:

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})\,|0\rangle^{\otimes n} \tag{2.5}$$

For a given set of parameters $\boldsymbol{\theta}$, the quantum state is prepared and measured to calculate qubit expectations and correlations. Those expectations are used to evaluate an the objective function $\mathcal{C}$. In this work, the objective function is a spin-Hamiltonian $\mathcal{C} \equiv H$ as described in Sec. 2.1. An external classical feedback loop performs the optimization of the circuit parameters $\boldsymbol{\theta}$ such that the the solution to $\mathcal{E}$ is the state of lowest energy, i.e. the groundstate of the Hamiltonian H. Here, the VQE is implemented using the *Simultaneous pertubation stochastic approximation* (SPSA) optimization algorithm [22] which uses only two function evaluations per VQE iteration. This is very beneficial for practical use on near-term quantum devices.

A common VQE ansatz is to implement quantum circuits is

$$|\psi\rangle = \prod \left[U_s^{(j)}(\boldsymbol{\theta}^{(j)})\; U_E\right]_{j=1}^{d} U_s^{(0)}(\boldsymbol{\theta}^{(0)})\; |0\rangle^{\otimes n}. \tag{2.6}$$

It consist of layers of parametrized single-qubit gates U_s and entangling gates U_E. A depth parameter d controls the expressivity and complexity of the state preparation by setting the number of rotation parameters and entangling operations.
The single-qubit rotation unitaries U_s in this work consist of parametrized Y and Z rotations $U_s(\theta) = R_z R_y R_z$ and the entangling operations U_E are pair-wise CNOT gates in a linear chain between neighboring qubits. The set of gates does not necessarily have to be universal for an approximate or even exact solution. The entangling gates can be any gates which act on more than one qubit as long as they generate "enough" entanglement. This again depends on the problem. The exact gates used are arbitrary and most likely hardware-dependent, i.e. which gates are most naturally implemented on the quantum device.

On a qubit quantum computer, it is also possible to iteratively find the k^{th} eigenstate $|\psi_k\rangle$ of a Hamiltonian with the VQE [48]. This is done by variationally minimizing the energy of the state and adding an overlap constraint to previously found lower eigenstates

$$\mathcal{E}_k = \frac{\langle\psi_k|H|\psi_k\rangle}{\langle\psi_k|\psi_k\rangle} + \sum_i^{k-1} \beta_i |\langle\psi_k|\psi_i\rangle|^2 \tag{2.7}$$

The first exited state is the state that minimizes $\mathcal{E}_1$ and is orthogonal to the groundstate. This can be repeated iteratively for each higher eigenstate as long as $\beta_i > 0$ is larger than the energy gap between the k^{th} and the i^{th} eigenstate. For a diagonal Hamiltonian, eigenstates of the Hamiltonian contain only one basis element with no

phase-dependence. Therefore, overlap with lower eigenstates can be calculated by measuring expectations in each eigenstate. For a non-diagonal Hamiltonians, this is no longer possible and one needs to apply the inverse state preparation unitary of lower eigenstates and measure the expectation of the $|0\rangle^{\otimes n}$ state.

With this method, it is in principle possible to find multiple eigenstates simultaneously. Instead of starting successive searches for higher eigenstates, one can start all at once and modify the orthogonality constraint to the current state $|\psi_i^{(t)}\rangle$ of the other searches on iteration t. $\mathcal{E}_k$ in eq. 2.7 is therefore changed to and iteration-dependent energy functional $\mathcal{E}_k^t$ Initial tests have shown that this method generally works but it is not further pursued in this work.

2.3 Numerical Models

The Quantum Machine Learning algorithms in this work are not implemented on quantum devices. We use two numerical models in order to simulate the algorithms which are designed respecting the capabilities and limitations of near-term quantum computers. The models used are the *Qasm simulator* from IBM Q's *qiskit* framework [3], and a self-built simulator which is based on exact diagonalization with the Python *quspin* package [61].

The qiskit *Qasm simulator* is a noisy quantum circuit simulator backend. In this work, it is used for two types of applications: To implement algorithms which directly implement quantum circuits, and to find eigenstates of Hamiltonians with the Variational Quantum Eigensolver (Sec. 2.2). The *Qasm simulator* is close to current capabilities of gate-based quantum computers, especially when incorporating gate fidelities, read-out error and effects of de-coherence. For this work, only Gaussian noise on the gates and read-out are included in the simulation. A review of other quantum simulation platforms can be found here [37]

For most of the Hamiltonian-based algorithms in this work, we designed a numerical model of the qubit system which relies on the *quspin* Python package to exactly calculate Hamiltonians eigenstates. Full VQE searches are either slow or unreliable because they require a varying number of iterations until convergence. By utilizing this exact method for calculating eigenstates, we shift the focus of research from the VQE algorithm to studying the properties of the Quantum Machine Learning models directly. With the *quspin* package, Hamiltonians with $n < 8$ can be diagonalized fast enough for the simulations presented here.
It has to be noted that the results gained from exact diagonalization are of higher quality than the expected results of the VQE, especially given limited circuit depth and gate fidelity. Still, they are in principle not unreasonable for near-term quantum devices that directly implement Hamiltonian systems.

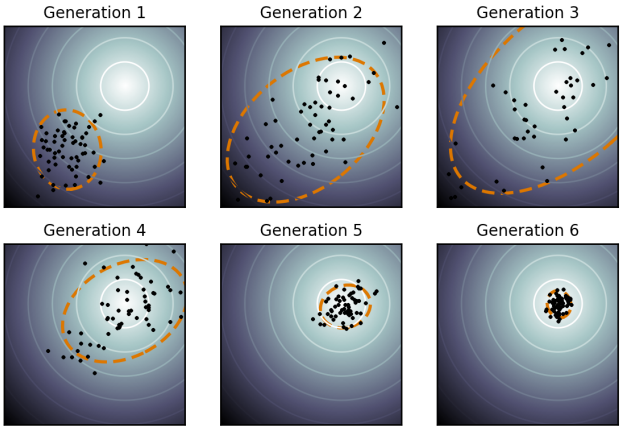

Figure 2.4.1: Schematic illustration of the *cma-es* algorithm on a convex objective-function. The population of the evolutionary strategy are represented as black dots and the multivariate normal distribution from which they are sampled is sketched in orange. Taken from [55].

The code used for the simulations can be viewed on github [51][52]. Generally, to simulate the measurement of qubits that populate the eigenstates of a Hamiltonian, we construct the probability of a measurement or quantum sample $\mathbf{s}$ through the probability $P(\lambda)$ that the qubits are in eigenstate λ and the conditional probability $P(\mathbf{s}|\lambda)$ for the measurement $\mathbf{s}$ given that the system is in eigenstate λ:

$$P(\mathbf{s}) = \sum_{\lambda} P(\mathbf{s}|\lambda)P(\lambda). \tag{2.8}$$

$P(\lambda)$ is calculated with the density matrix ρ which describes the quantum system via $P(\lambda) = \langle\lambda|\rho|\lambda\rangle$. $P(\mathbf{s}|\lambda) = |\langle\mathbf{s}|\lambda\rangle|^2$ is given by the amplitudes in eigenstate $|\lambda\rangle$ which are calculated by *quspin*. Finally, we construct the probability distribution by assigning each probability $P(\mathbf{s})$ an interval $\subset (0, 1)$ and drawing a random number $\in (0, 1)$ to draw a quantum sample $\mathbf{s}$ with the correct probability.

2.4 Optimization Algorithm cma-es

The optimizer used in this work is the *Covariance Matrix Adaptation Evolution Strategy* (cma-es) [29]. *cma-es* is an evolutionary algorithm for difficult non-linear non-convex black-box optimization. As an evolutionary strategy, it is a stochastic and derivative-free numerical optimization algorithm which is based on mutation and re-combination of previous evaluations.
For a given objective- or cost-function $\mathcal{C}$ over a parameter space $\mathbb{R}^N$, *cma-es* draws λ stochastic samples according to a multivariate normal distribution in $\mathbb{R}^N$. The λ function evaluations are called the *population* of the evolutionary algorithm. Depending on the performance of the population members, the means and variances

of the multivariate normal distribution are adapted to provide faster convergence.

In this work, the *cma-es* optimization algorithm is used to train parameters in Quantum Neural Networks or Quantum-assisted Neural Networks. It is applied on models with a small or moderate number of parameters, i.e. ≤ 50, or a subset of parameters in a bigger model. Generally, the black-box optimizer is used in cases where efficient calculation of derivatives with respect to model parameters is not possible.

2.5 Relevant Training Sets

The Quantum Machine Learning models throughout this work are trained on different training sets. Training sets are chosen accordingly to characterize a model's adequacy to learn data with different complexity and structure. The training data presented in this section, range from random binary samples to 28x28 continuous-valued images of hand-written digits.
For the generative Quantum Machine Learning algorithms described in this work, the distribution of training samples in the training set acts as target distribution for the model. In the case of the input-dependent algorithms, each training set sample is input separately with a distinct expected model output.

2.5.1 Random Training Set

To benchmark a model's ability to tune its parameters and learn arbitrary data distributions, we generate random training sets. Random training sets are not expected to show any systematic structure which could be learned so one is essentially measuring a model's ablity to perform *overfitting* (see Sec. 1.2.2). In practice, overfitting not generally viewed as a desirable property of a model but in this work it is used to quantify a models expressivity and flexibility.

The random training samples are drawn from a flat distribution of binary or continuous values. In the case of a binary training set D for a model with 4 output nodes, the training samples d, are generated by drawing random numbers $d' \in [0, 15]$ and converting them into base-2 samples $d \in [0000, 0001, ..., 1111]$.
For continuous-valued random samples, four random numbers $d_i \in (0, 1), i = 0, 1, 2, 3$ are drawn to construct $d = [d_0, d_1, d_2, d_3]$. The values can be interpreted as gray-scale where 0 signifies black and 1 is white.

2.5.2 Binomial Training Set

In this work, the binomial training set is used for generative models described. Contrary to the random data in Sec. 2.5.1, this training set arguably requires less degrees of freedom to fully characterize and learn. When considering an ordered histogram

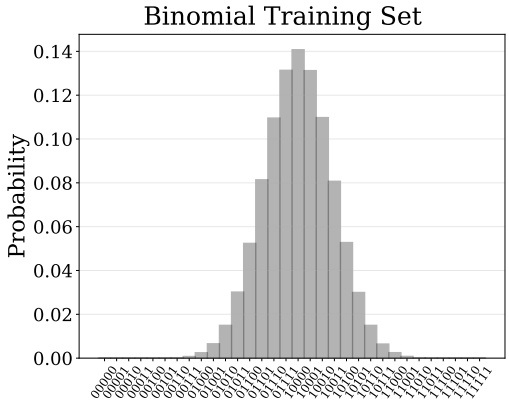

Figure 2.5.1: Probability distribution of a binomial training set for a model with 5 output nodes. Samples are pulled from a binomial distribution around the middle of the binary range $(0, 2^5\text{-}1)$.

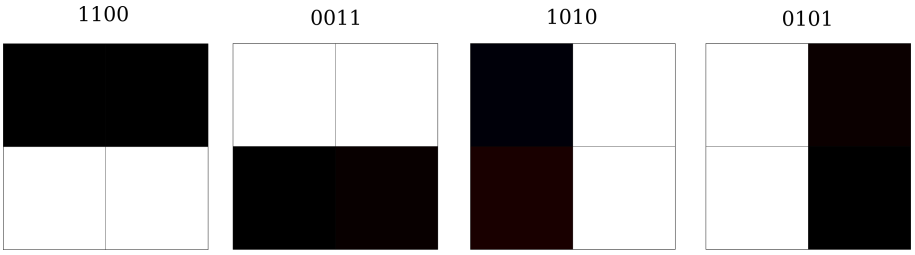

Figure 2.5.2: Visual interpretation of *Bars-and-Stripes* (BAS) patterns. The pixels of the BAS patterns form horizontal or vertical 'Bars' and 'Stripes'. This training set has 25% probability for each BAS pattern and zero probability for every other sample. It can only be used for Machine Learning models with 4 output nodes.

of n-bit binary samples, the training set contains a binomial distribution around the middle of the histogram. Fig. 2.5.1 shows the probability distribution for a binomial training set for a model with 5 output nodes. 01111 is the most frequent sample in that training set with others being less frequent.
To generate a n-bit binomial training set, integers are drawn from a binomial distribution with a mean of $\frac{2^n}{2}$ and then converted to n-bit binary samples.
It is also possible to generate several binomial distributions in one training set with different means and amplitudes. This generally preserves systematic structure in the training set while increasing the degrees of freedom required to parametrize the multi-modal distribution.

2.5.3 Bars-and-Stripes Patterns

The *Bars-and-Stripes* (BAS) patterns can be used as a training set for Machine Learning models with 4 output nodes. The BAS training set consists of the samples $\text{BAS} = \{1100, 0011, 1010, 0101\}$. When visualized in an ordered 2x2 pixel image like in Fig. 2.5.2, they give rise to vertical or horizontal lines. Unlike for the random or

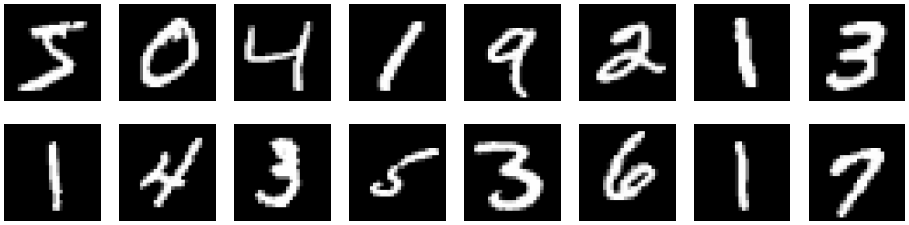

Figure 2.5.3: First 16 images of the MNIST hand-written digits training data set [39]. Each image consists of 28x28 continuous gray-scale valued pixels. The MNIST set is a wide-spread standard training data set for benchmarking and comparing many kinds of Machine Learning algorithms.

binomial training sets in Secs. 2.5.1 & 2.5.2, the BAS training set has exactly zero probability for 12 out of 16 4-bit samples and is thus a training set with distinct gaps in the distribution. Interestingly, it is also a symmetric training set where every pixel in the images is equally likely to be black or white. The determining factor in the BAS patterns which has to be learned are the correlations between the pixels. The BAS training set is a deceptively complex data set with rich possibilities and challenges for the Quantum Machine Learning algorithms in this work.

One can include the samples 0000 and 1111 into the BAS set as they technically include zero- and two- bars and stripes respectively but they are not included in this work.

2.5.4 MNIST Data Set

The MNIST data set is a large data set of hand-written digits [39]. It is one of the standart data sets for benchmarking Artificial Neural Networks and Machine Learning algorithms in general. It contains 60.000 training samples and 10.000 test samples with a 28x28 pixel resolution and gray-scale values $\in [0, 1]$. The first 16 images of the MNIST training data set can be viewed in Fig. 2.5.3. The MNIST samples have a lot of structure which is intuitive for humans who are used to the Arabic numerals and that is also learnable by Machine Learning algorithms. Even though there is only ten different digits in the MNIST data set, it offers a lot of variation in the written style. The task of any Machine Learning algorithm trained on this data set is to identify the most significant structural elements for each digit.

The largest quantum-assisted Machine Learning algorithms in this work are trained with a subset or even the the full $28^2 = 784$ pixel gray-scale images. For networks with a smaller number of output nodes, the images are re-sized by the Python *scipy.misc.imresize()* function.
Any training set consisting of MNIST samples can be continuous-valued or binary. If binary, the pixel values are calculated by thresholding at the value of 0.5.

3 Single Qubit-Layer Generative Algorithms

This chapter revises and studies two known generative Quantum Machine Learning algorithms which implement qubits. The motivations for using qubits in a generative Machine Learning algorithm are discussed in Sec. 1.3 and are mainly reflected in the following two questions: Will the use of qubits make sampling learned probability distributions more efficient, and can the model utilize quantum effects to gain a performance benefit?
The algorithms presented in this chapter follow two different implementations of generative modeling in a single qubit layer. The first model, the *Direct Variational Generator* (DVG) [9], is a simple generative quantum algorithm and in this work is used to demonstrate the sampling benefits one can expect from implementing qubits in generative models. The second algorithm presented is the *Quantum Boltzmann Machine* (QBM)[6]. It is a widely studied generative algorithm in the field of Quantum Machine Learning [10][42] on which essential parts of this work are based on.

3.1 The Direct Variational Generator

The first generative quantum algorithm studied in this chapter is the *Direct Variational Generator* (DVG). It was first introduced as *data-driven quantum circuit learning* (DDQCL) model [9]. The aim of this algorithm is to learn a parametrized qubit wavefunction which approximates the probability distribution of a target data set. The qubits can then be measured to provide efficient sampling of the encoded distribution. With this model, we show improved sampling capabilities of generative quantum models and improve our understanding of state preparation in gate-based quantum devices. The relevant code for the simulations in this section can be seen and tested on github [51][52].

3.1.1 Model Description

The Direct Variational Generator aims to learn a parametrized qubit wavefunction ψ such that the distribution of measured quantum samples $\mathbf{s}$ approximates a target data distribution. The model distribution for the DVG is

$$P_\psi(\mathbf{s}) = |\langle \mathbf{s}|\psi\rangle|^2 \tag{3.1}$$

The quantum circuit to prepare the quantum state $|\psi\rangle$ is a variational state preparation protocol equivalent to the state preparation in the Variational Quantum Eigen-

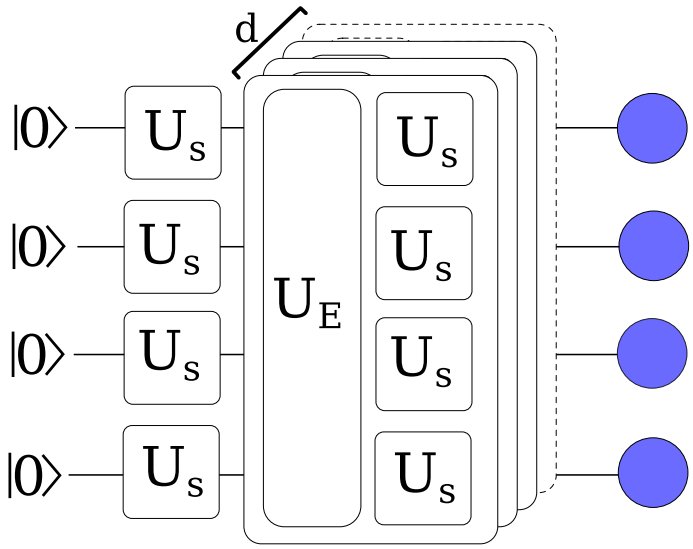

Figure 3.1.1: Visual representation of the variational state preparation circuit of the Direct Variational Generator. The state preparation is analogous to the approach in the Variational Quantum Eigensolver (Sec. 2.2). One layer of the circuit consists of single-qubit rotations and entangling operations between qubits. With increasing circuit depth d, as more single-qubit rotations and richer entanglement becomes available, complex quantum wavefunctions can be approximated with increasing quality. In the Direct Variational Generator, the projection of the qubit wavefunction encodes the data distribution.

solver (see Sec. 2.2). State preparation is performed with a quantum circuit consisting of parametrized single-qubit gates and entangling gates:

$$|\psi\rangle = \prod \left[U_s^{(j)}(\boldsymbol{\theta}^{(j)})\ U_E \right]_{j=1}^{d} U_s^{(0)}(\boldsymbol{\theta}^{(0)})\ |0\rangle^{\otimes n} \tag{3.2}$$

Fig. 3.1.1 offers a visual representation of the state preparation circuit for the DVG. The single-qubit unitary U_s may consist of any single-qubit gates which offer sufficient degrees of freedom for the quantum state and the entangling operation U_E can consist of any sequence of pair-wise, multi-qubit or global entangling gates that create entanglement between qubits. The circuit depth d determines expressivity of the model and the degree of potential approximation of a target distribution. In practice, the circuit depth is limited by systematic errors accumulating over time and with a large number of imperfect gates.

In this work, the Direct Variational Generator is implemented on the qiskit *Qasm simulator* backend (see Sec. 2.3). The single-qubit unitaries U_s are implemented through a Z-rotation, a Y-rotation and another Z-rotation, i.e. their matrix exponential is $U_s^{(j)} = e^{-i\theta^{(j,2)}\sigma^z} e^{-i\theta^{(j,1)}\sigma^x} e^{-i\theta^{(j,0)}\sigma^z}$ where σ^z, σ^x are Pauli matrices. These parametrized rotations offer full flexibility for each qubit's state. The first Z-rotation of the first layer is not implemented because the initial $|0\rangle^{\otimes n}$ state is an eigenstate of the operator and does not change when applied. The single-qubit gates used should

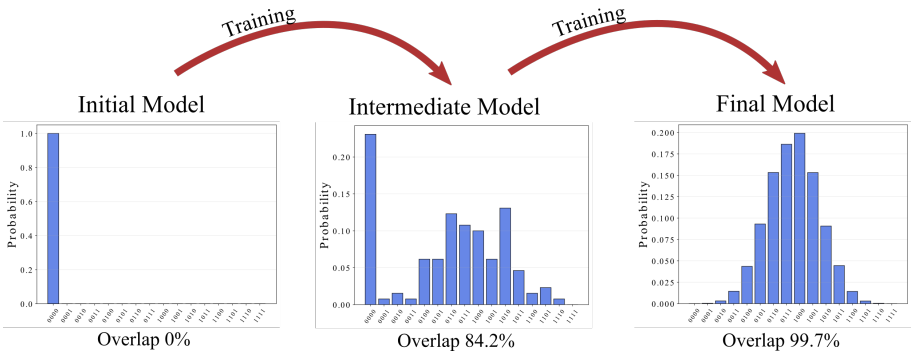

Figure 3.1.2: Sketch tomographies of an initial, an intermediate and a fully trained model distribution of a Direct Variational Generator. The model is initialized in $|0\rangle^{\otimes n}$ state and the rotation parameters of the variational state preparation are optimized during training to learn a wavefunction whose projection approximates the target distribution. In this case, the taget distribution is a binomial distribution.

generally be gates that are naturally implemented in the quantum hardware that the algorithm is run on. The entangling operations U_E here are a linear chain of pair-wise CNOT gates between consecutive qubits. Also the entangling gates and connectivity between qubits should be adapted to a given quantum device.

3.1.2 Training

Training the Direct Variational Generator consists of tuning the rotation parameters $\boldsymbol{\theta}$ in U_s such that the model distribution P_ψ best approximates a target data distribution P_D. This is commonly done by minimizing a *cost-function* which for the DVG is the *negative log-likelihood* (NLL)

$$\mathcal{L} = -\sum_{\mathbf{s}\in D} P_D(\mathbf{s}) \log \max(\epsilon, P_\psi(\mathbf{s})). \tag{3.3}$$

The target data distribution P_D is represented by a training set which is assumed to follow the same distribution. The singularity of the logarithm at $P_\psi = 0$ is bounded by an $\epsilon = 10^{-8}$. The NLL provides is a measure for the difference between two distributions. Minimizing it ensures that the data distribution P_D is well parametrized by the model distribution P_ψ. In other words, the distribution of quantum samples $\mathbf{s}$ in the DVG is close to the distribution of $\mathbf{s}$ in the training set.
The training protocol consists of a classical optimization loop which proposes and optimizes the rotation parameters in the quantum circuit in order to minimize the NLL. The optimizer used for training the DVG is the *cma-es* algorithm (see Sec. 2.4). In fact, the training procedure is equivalent to the work-flow of the VQE algorithm (Sec. 2.2).

Fig. 3.1.2 shows sketch tomographies of an initial DVG model in the initial $|0\rangle^{\otimes n}$ state, an intermediate model and a fully trained DVG model. The final distribution

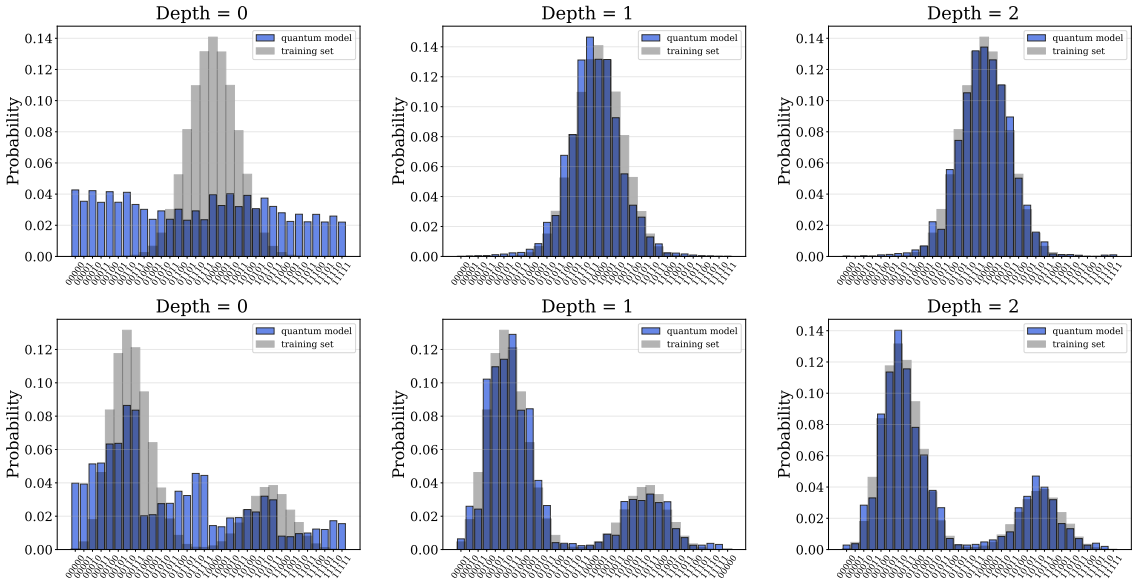

Figure 3.1.3: Trained Direct Variational Generator models by the Variational Quantum Eigensolver algorithm and different circuit depths. More depth translates in more entanglement and degrees of freedom for the quantum state. Entangling operations in the quantum circuit are definitely required for good approxmiation. In both cases, the binomial distribution and the bi-modal distribution, the deeper quantum circuit provides more degrees of freedom and smother distributions.

is a good approximation of a binomial distribution. In practice, the single-qubit rotation angles are not initialized as zero but instead for example randomly.

Given a large enough depth d for the variational state preparation, the wavefunction is highly expressive and is expected to model arbitrary distributions. For a more shallow VQE curcuit, samples that are not in the training set will appear as artifacts of insufficuent entanglement. Fig. 3.1.3 shows the effects of different circuit depths d in a DVG with 5 qubits for two different training set distributions - a single-binomial in the middle of the tomography and double-binomials at different locations and different height (see Sec. 2.5.2 for details). It is apparent that a larger depth parameter d increases the quality of the approximation of the target distribution. Deeper circuits offer more degrees of freedom through more parameters and entanglement. Without entangling operations ($d = 0$), non of the data sets can be learned with high quality. It has to be noted, that the model adapts to the degrees of freedom available and finds the best solution for it, as seen for the double-binomial. The single-binomial distribution in Fig. 3.1.3 is nicely learned by a circuit of depth = 1 which offers just enough degrees of freedom for the symmetric target tristribution. The double-binomial training set is also learned in somewhat low resolution. $d = 2$ already offers detail and higher resolution for both training sets.

This generative model is not expected to generalize from training data in a way that is generally desirable for Machine Learning models (see Sec. 1.2). It is possible that low-resolution approximations of distributions with shallow circuits are useful for generalizability and to learn only the most essential features in data though it has to be stressed that the exact structure of the result is highly dependent on the encoding of the data, e.g. the sorting of the binary bit strings. The effects resulting from lacking circuit depth depend on the respective product state artifacts which may change strongly with a different bit-representation of the data.

Generally, more errors accumulate in deeper state preparation circuits with large number of gates. For our simulations on the *Qasm simulator* (Sec. 2.3), those effects are not respected. Another factor not respected in these simulations is the loss of coherence in the quantum state over time. A final consideration in implementing a DVG is that the number of parameters increases linearly with d. A classical optimizer will consequently have increasing difficulty training the algorithm.

3.1.3 Sampling Benefits & BAS Random Walk

One main motivation for implementing qubits in generative models is to utilize the sampling benefits [21][42]. 'Sampling' means drawing data samples from the model where the distribution of the samples follows a given probability distribution. Classical sampling is not efficient for general probability distributions, in particular in high dimensions, for multi-modal distributions, or for highly structured distributions [12]. One method for parametrizing high-dimensional probability distributions is given by the Restricted Boltzmann Machine (RBM)[53]. Performing *Gibb's sampling* [43], the RBM can generate samples directly from its model distribution without calculation overhead but this method suffers from the so-called *slow mixing* problem. It is a *local* sampling method and relies on a Markov-chain process which can take a long time to reach every mode of the distribution. The problem is amplified for distant parts of distributions which are separated by gaps of low density. In some cases, parts of the probability distribution can barely be sampled at all.

To demonstrate the slow-mixing of the local sampling technique, we train a RBM [18] with $n_v = 4, n_h = 3$ and 10^5 training iterations on the *Bars-and-Stripes* (BAS) training set (see Sec. 2.5.3) which is a training set with a very discrete distribution and large gaps. Fig. 3.1.4 shows the samples that are generated by the trained RBM. When drawing 100 from the model distribution using Gibb's sampling, only one mode of the distribution is explored. Orders of magnitude of additional sampling steps are required to cumulatively sample the model distribution.
In practice, one would not select every sample produced by Gibb's sampling to be a valid sample but instead empirically choose every $10^1\text{th} - 10^5\text{th}$ sample, depending on the distribution. Additionally, one can perform restarts of the sampling procedure in order to avoid skipping many samples. Drawback here is that the first initial input to the Gibb's sampling method needs to be close to a learned visible state to

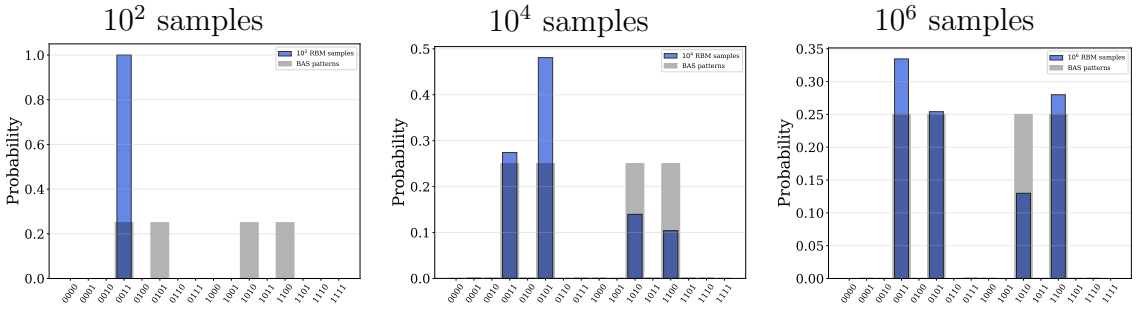

Figure 3.1.4: Tomographies of samples generated by a RBM which was trained on the BAS training set (Sec. 2.5.3). The model distribution is very discrete and not well sampled by the Gibb's sampling method which is the conventional technique for sampling RBMs. With 100 sampling steps, only one mode of the distribution has been explored. Orders of magnitude more sampling steps are required to cumulatively sample the entire model distribution.

generate valid samples. That requires certain knowledge of the distribution.

A probability distribution encoded in a qubit wavefunction does not have this issue. Every measurement of the qubit state causes a random global projection of the wavefunction onto one basis state with the true probability as encoded in the qubit state.

To further visualize this point, we encode the BAS patterns into instructions for a 2D random walk and inspect the 'randomness' of a walk that results from samples generated by a generative model.
The BAS patterns directly encode the instructions for two random walkers like $|\Delta x_1 \Delta y_1 \Delta x_2 \Delta y_2\rangle$ where $\Delta x_i, \Delta y_i$ are the next steps in x and y direction for walker $i = 1, 2$. A '1' signifies a step of $+1$ along the dimension and '0' a step of -1. By the nature of the BAS patterns, all valid steps are diagonal in the x-y-plane. For the four BAS patterns, the four possible steps are

$$\begin{aligned} |1100\rangle &= \nearrow_1 \ \swarrow_2 \\ |0011\rangle &= \swarrow_1 \ \nearrow_2 \\ |1010\rangle &= \searrow_1 \ \searrow_2 \\ |0101\rangle &= \nwarrow_1 \ \nwarrow_2 \end{aligned} \tag{3.4}$$

Both walkers start at the origin in two dimensional space $\vec{x}_1(0) = \vec{x}_2(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. The position of random walker i after t steps is determined by the sequence of consecutive

samples that are generated by the trained generative model

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \begin{pmatrix} \Delta x_i \\ \Delta y_i \end{pmatrix} \tag{3.5}$$

Interestingly, the mean of both BAS random walkers and their relative distance are de-coupled in the BAS patterns. The samples $|1010\rangle$ & $|1010\rangle$ influence only the mean (both walkers walk diagonally in the same direction), and $|1100\rangle$ & $|0011\rangle$ influence only the distance (both walkers walk apart diagonally). Non of the BAS samples influences both values at once. Consequently, we can interpret the mean of the walkers and the distance between them as independent 1D random walks.

We will now derive the expected drift of the mean $m = |\frac{\vec{x}_1+\vec{x}_2}{2}|$ to the origin and the expectation of the relative distance $d = |\vec{x}_1 - \vec{x}_2|$. For the calculation, we assume each BAS pattern to be equally likely with a probability of $\frac{1}{4}$. The expected absolute distance to the origin of a 1D random walker with $\frac{1}{2}$ probability to walk either left or right is well known in literature [44] and given by

$$d_{1D}(t) = \sqrt{\frac{2t}{\pi}}. \tag{3.6}$$

For the 2D BAS random walk considered here, each distance is scaled with $\sqrt{2}$ because the BAS walkers move strictly diagonally. Since only half of the BAS samples contribute to each 1D random walk, the expected step size per step is halved. This rule change in the 1D random walk contributes with the square-root and thus all expectations are scaled with $\frac{1}{\sqrt{2}}$. For now, the intermediate theoretical results are back to Eq. (3.6) which in fact is the expected results for the mean of the walkers. The distance between the walkers has another scaling factor of 2 because the relevant BAS samples move the walkers apart and thus the distance between them increases by 2 units.
The final results for a theoretical BAS random walk are

$$m_{BAS}(t) = d_{1D}(t) \cdot \frac{\sqrt{2}}{\sqrt{2}} = \sqrt{\frac{2t}{\pi}} \tag{3.7}$$

$$d_{BAS}(t) = d_{1D}(t) \cdot 2\frac{\sqrt{2}}{\sqrt{2}} = \sqrt{\frac{8t}{\pi}} \tag{3.8}$$

for the drift of the mean of the two BAS random walkers and their relative distance respectively. Generating samples that are not part of the BAS patterns will systematically change the behavior of the walkers with the same being true for a non-random sequence of samples, even if they are BAS patterns. This offers an extension to the *qBAS* score [9] which was proposed to evaluate the quality of a trained generative model. Our method not only measures the fidelity of the learned

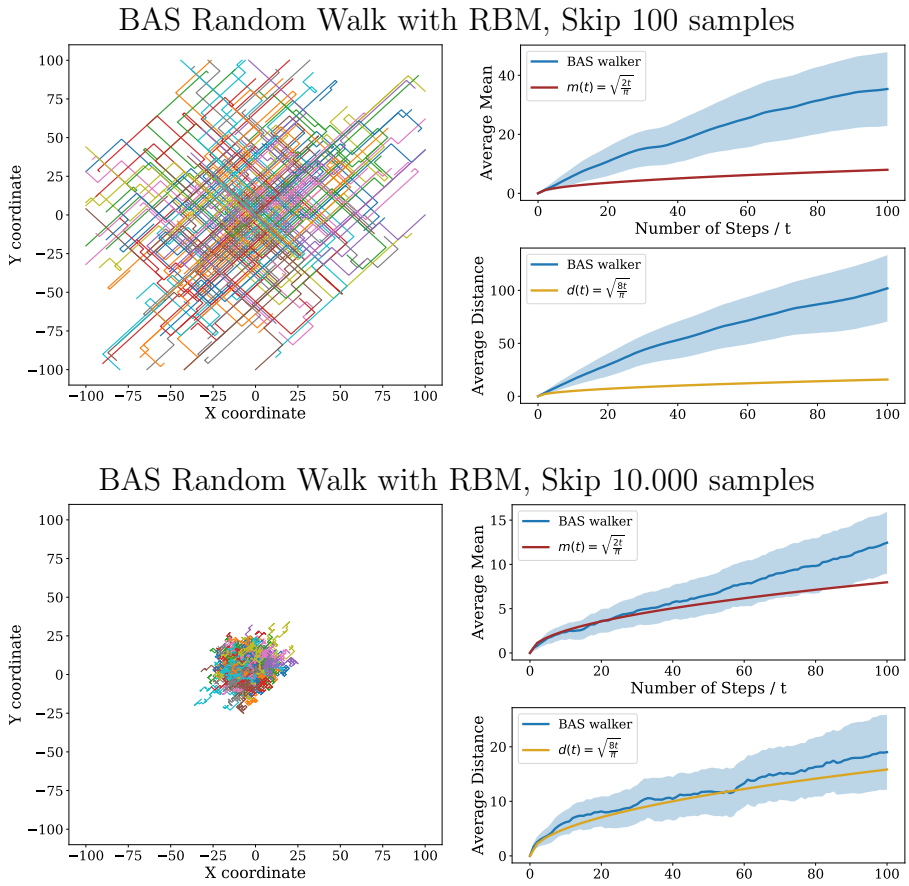

Figure 3.1.5: Visualization of 100 repetitions of the BAS random walk with 100 steps each generated by a RBM which was trained on the BAS training set. Average mean and distance of the two BAS random walkers are compared to expected values for a true BAS random walk. Randomness of the BAS random walk increase with an increasing number of skipped samples of the RBM before the next one is used for a step in the walk. Residual deviation to the expected quantities are likely caused by imperfectly learned probabilities for the four BAS samples.

distribution but also the randomness of the generated samples.

The models compared here are a RBM and the Direct Variational Generator. The DVG is implemented on the qiskit *Qasm simulator* (see Sec. 2.3). The RBM and Direct Variational Generator are trained until full convergence on the BAS training set. The experiment is carried out over 100 repetitions with 100 steps each. Measured are the paths of all random walkers, their drift of the mean in each run and their eucledian distance. Because of the results shown in Fig. 3.1.4, we choose to skip 10^2 or 10^4 samples generated by the RBM before the next valid sample which contributes the random walk. We expect the randomness of the walk to increase when skipping more samples.
Fig. 3.1.5 shows a systematically wrong behavior for the RBM with 100 skipped

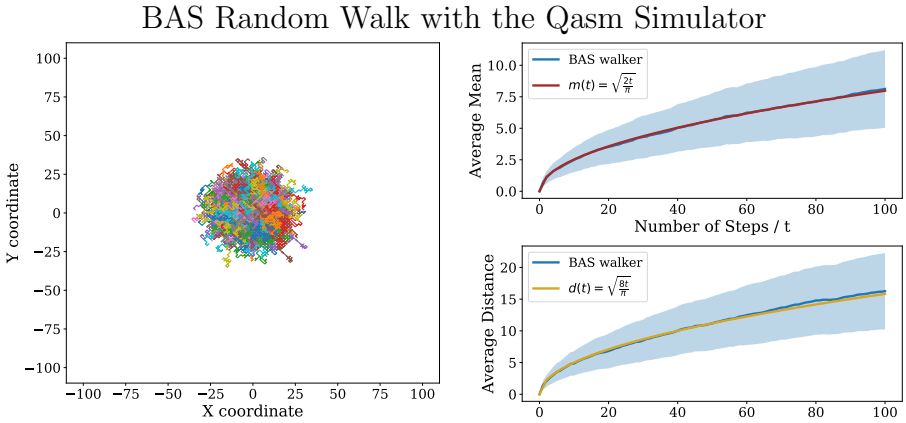

Figure 3.1.6: Visualization of 100 repetitions of the BAS random walk with 100 steps each generated by a DVG which was trained on the BAS training set. Average mean and distance of the two BAS random walkers are compared to expected values for a true BAS random walk. The DVG is implemented on the qiskit *Qasm simulator* which provides simulated sampling results of the qubits. The observed random walks are very good compared to the expected results.

samples where the walkers move into the same direct for many steps. For 10.000 skipped samples, the model approaches a better approximation of a BAS random walk. The quality of the RBM results depends on its network architecture. With 4 instead of 3 hidden units, the the same measurements become significantly worse as the already distant modes in the distribution become more distant.
In contrast to the complications of classical the classical sampling technique, Fig. 3.1.6 shows 100 BAS random walks generated by DVG samples. The average mean and distance of the two BAS random walkers agree exactly with the expected quantities. Note, that these measurements are not performed on a physical quantum devices but on the qiskit *Qasm simulator*.

Residual discrepancies of a generated BAS random walk relative to the expected values, even when efficient sampling of the true encoded distribution s performed, are due to the following reasons. A significant error can arise from imperfect learning of the four different BAS patterns where not all have an exact 25% likelihood. Additionally, the probability of non-BAS patterns is $> 0\%$ which will systematically change the behavior of the random walk, e.g. couple mean and distance and introduce movement that is not along the diagonal axes.

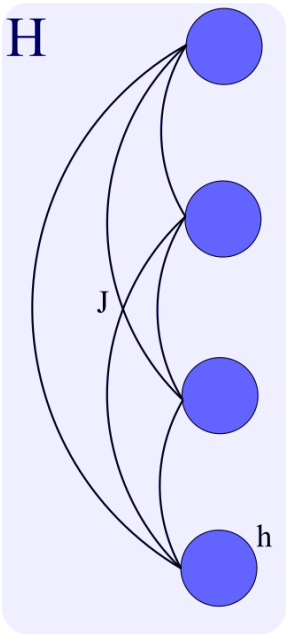

Figure 3.2.1: Schematic representation of the network architecture of a Quantum Boltzmann Machine (QBM). The QBM is a generative Quantum Machine Learning algorithm which implements a Boltzmann distribution of quantum states on a spin-Hamiltonian. The Hamiltonian can either be diagonal of non-diagonal. The distribution of the QBM is parametrized by the Hamiltonian parameters J, h^z and potentially h^x.

3.2 The Quantum Boltzmann Machine

Another single-layer generative Quantum Machine Learning algorithm is the *Quantum Boltzmann Machine* (QBM) [6]. It is a quantum extension of the Boltzmann Machine [31] and a Hamiltonian-based generative Machine Learning model. A QBM implements a Boltzmann distribution over its states with the goal to approximate a target distribution P_D over a set of measured quantum states $\{\mathbf{s}\}$. The relevant code for the simulations in this section are provided on github [51][52].

3.2.1 Model Description

The general concept of a Quantum Boltzmann Machine is to assign each qubit state $\mathbf{s}$ in computational basis, the z-basis, an energy and thus a probability according to the Boltzmann distribution. The Hamiltonian over which the thermal ensemble is constructed is the spin-Hamiltonian introduced in Sec. 2.1

$$H = \sum_{ij} \sigma_i^z J_{ij} \sigma_j^z + \sum_i h_i^z \sigma_i^z + \sum_i h_i^x \sigma_i^x \tag{3.9}$$

The model distribution of a QBM can be written as

$$P_\rho(\mathbf{s}) = \langle \mathbf{s} | \rho_t | \mathbf{s} \rangle \tag{3.10}$$

where the density matrix describes a Boltzmann distributed ensemble which reads

$$\rho_t = \sum_{\lambda=0}^{t} \frac{e^{-E_\lambda}}{\mathcal{Z}} |\lambda\rangle\langle\lambda|. \tag{3.11}$$

$|\lambda\rangle$ and E_λ are the eigenvectors and eigenenergies of the Hamiltonian H and $\mathcal{Z} = \sum_{\lambda=0}^{t} e^{-E_\lambda}$ is the partition function. In the exact theory, $t = 2^n$ - 1 includes all eigenstates of the Hamiltonian but we allow for a truncation after t higher eigenstates.

In the form without the h_x term, the Hamiltonian in Eq. (3.9) is diagonal in the computational basis and without entanglement in the eigenstates. We will refer to this as a *classical Hamiltonian*. In a classical Hamiltonian, each eigenstate contains exactly one n-qubit binary basis state, e.g. 0011,1010 etc. for 4 qubits. By allowing transverse local biases h_x, which are orthogonal to the computational basis, the Hamiltonian is no longer diagonal and we consider this as a *quantum Hamiltonian*. In a quantum Hamiltonian, eigenstates can be highly entangled and in complex superpositions of basis states.

One interesting feature of the QBM is the possibility of *reconstruction*. Assuming a corrupted BAS pattern '11xx' where only the first two bits are correctly given, by clipping the fist two h^z fields to large values, e.g. 5 times the maximum of $\mathbf{h}^z$, we strongly encourage the first two qubits to be 1 and the J_{ij} interactions will determine the state of the the remaining two. This way, we can reconstruct the most likely pattern according to the learned correlations in the Hamiltonian. This is not further pursued in this work.

3.2.2 Training

The training of a Quantum Boltzmann Machine consists of tuning the parameters J, h^z and potentially h^x of the Hamiltonian in Eq. (3.9) such that the model distribution P_ρ best approximates a target data distribution P_D. This is commonly done by minimizing a cost-function which measures the distance between P_ρ and P_D. The preferred cost-function for training the QBM is the *negative log-likelyhood* (*NLL*)

$$\mathcal{L} = -\sum_{\mathbf{s}\in D} P_D(\mathbf{s}) \ \ln P_\rho(\mathbf{s}) \tag{3.12}$$

The target data distribution P_D is represented by a training set which is assumed to follow the same distribution.

For a QBM with diagonal Hamiltonian, the gradients of the NLL with respect to the Hamiltonian parameters are well-established and can be calculated via

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial J_{ij}} &= \langle s_i^z s_j^z \rangle_D - \langle s_i^z s_j^z \rangle_\rho \\ \frac{\partial \mathcal{L}}{\partial h_i^z} &= \langle s_i^z \rangle_D - \langle s_i^z \rangle_\rho \end{aligned} \tag{3.13}$$

where $\langle\cdot\rangle_D$ describes the expectation in the training data set and $\langle\cdot\rangle_\rho$ in the qubit system. The gradients on training iteration m are applied in a simple step of gradient descent

$$\begin{aligned} J_{ij}^{(m+1)} &= J_{ij}^{(m)} + \eta \frac{\partial \mathcal{L}}{\partial J_{ij}^{(m)}} \\ h_i^{(m+1)} &= h_i^{(m)} + \eta \frac{\partial \mathcal{L}}{\partial h_i^{(m)}} \end{aligned} \tag{3.14}$$

where η is an external parameter called *learning rate*.

In the case of a QBM with non-diagonal Hamiltonian, i.e. with transverse fields h^x, the gradients of the Boltzmann distribution with respect to the Hamiltonian parameters are exponentially expensive to calculate. This is because the Pauli matrices and their derivatives not longer commute with the Hamiltonian which is required to simplify the derivative with respect to the Hamiltonian parameters [6]. As a response, in this work, all parameters of a QBM with non-diagonal Hamiltonian are trained with a black-box optimization algorithm directly minimizing the cost-function in Eq. (3.12) with no explicit gradients being calculated. There are many robust and established algorithms but we choose the *cma-es* algorithm (see Sec. 2.4) for its reliable performance on non-convex objective functions.

3.2.3 Comparing Gradient Training and Optimizer Training

To compare the training approaches for the QBM, three different models are trained on the Bars-and-Stripes training set (see Sec. 2.5.3). One diagonal QBM is trained with the gradients shown in Eq. (3.13), another diagonal is trained with the *cma-es* optimizer minimizing the NLL, and a non-diagonal QBM is also trained with the optimizer. All models have $n = 4$ qubits and follow the full Boltzmann density matrix in Eq. (3.11) with no truncation. The eigenstates of the respective Hamiltonians are calculated using exact diagonalization (see Sec. 2.3). The training procedure is repeated 100 times.

Fig. 3.2.2 shows a comparable training progress for all three models. The gradient-trained diagonal model initially follows a clearer gradient but with a constant learning rate of $\eta = 0.1$ converges only asymptotically. The *cma-es* optimizer with initial step size of 0.1 converges fast to the global minimum of the NLL after some initial iterations. It seems like the QBM with non-diagonal Hamiltonian is harder to train for the optimizer, potentially because of four additional h^x parameters in the Hamiltonian which need to be optimized. It also surpasses the gradient-trained model before 100 iterations.

These results show the merit in using gradients for training simple Quantum Machine Learning models but also legitimize the use of the *cma-es* optimization algorithm for training the models.

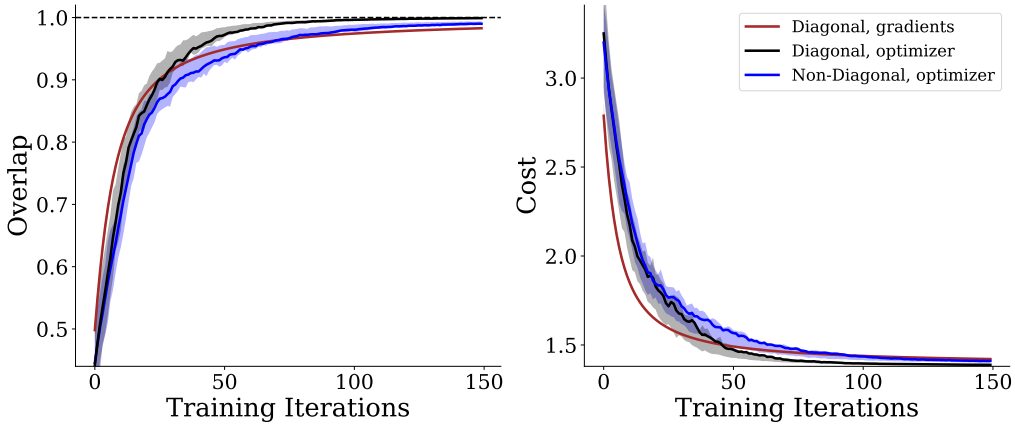

Figure 3.2.2: Evaluating the average training progress for three Quantum Boltzmann Machines on the Bars-and-Stripes training set over 100 repetitions. Two QBMs follow a diagonal Hamiltonian and are trained using gradient descent and the *cma-es* optimizer respectively. The QBM with non-diagonal Hamiltonian cannot be trained efficiently with gradients and is trained with the *cma-es* optimizer. Learning rate for the gradient descent and the initial step size for the optimizer are 0.1. Gradients initially receive a strong training signal but the optimizer coverges quickly after some initial iterations. The QBM with non-diagonal Hamiltonian is slightly harder to train because of additional Hamiltonian parameters.

3.2.4 Error in Truncating the Boltzmann Density Matrix

Constructing the full Boltzmann density matrix in Eq. (3.11) on a near-term qubit quantum computer is currently impractical. Each eigenstate needs to be calculated iteratively using the orthogonalization technique described in Sec. 2.2. Thus, we study the expressivity and training performance of a Quantum Boltzmann Machine with a truncated Boltzmann density matrix. For the simulation, we use $t = 0, 1, 2$ and $t = 15$ in a $n = 4$ qubit system and compare a QBM with diagonal and non-diagonal Hamiltonian. We consider the diagonal model to be classical and the non-diagonal to be quantum. To fully benchmark their expressivity, the models are trained on random training sets (see Sec. 2.5.1) consisting of 10 random binary samples each. The training progress is averaged over 40 repetitions with a different random training set for each repetition.
The eigenstates of the Hamiltonians are calculated by the exact diagonalization framework discussed in Sec. 2.3.

Fig. 3.2.3 clearly shows that a QBM with classical Hamiltonian cannot be trained appropriately with a small number of eigenstates contributing to the Boltzmann distribution in Eq. (3.11). This is because each eigenstate of a diagonal Hamiltonian contains exactly one basis state with amplitude 1 and thus all 2^n eigenstates are required to learn n-qubit wavefunctions with 2^n flexible amplitudes. More surprisingly, the non-diagonal Hamiltonian performs well with any truncation t. Even just

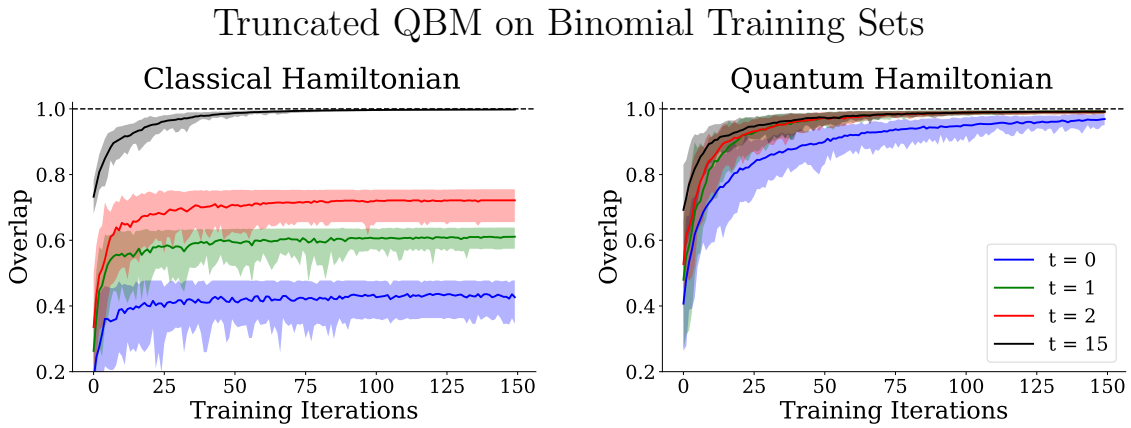

Figure 3.2.3: Measuring the truncation error-dependence of $n = 4$ QBMs with classical and quantum Hamiltonian on random training sets. The construction of the Boltzmann density matrix is truncated at $t = 0, 1, 2, 15$. As each eigenstate of a classical diagonal Hamiltonian only consists of a single basis state, the classical model requires full access to all 16 eigenstates in order to learn random distribution. Conversely, even a single quantum groundstate can be trained to approximate random distributions.

the groundstate of a quantum Hamiltonian can be trained appropriately to approximate a random target distribution. This indicates that a strict Boltzmann Machine framework might not be required with a quantum Hamiltonian and instead, one can perform generative machine learning with just a quantum groundstate - we call this case *groundstate training*. Additional eigenstates increase the training performance per iteration but don't seem to be strictly necessary. Already with $t = 1$ or $t = 2$ there is little difference compared to the training performance with $t = 15$.
The groundstate training framework has different implications for different quantum hardware. For a qubit quantum computer, it makes the implementation of a Hamiltonian-based model feasible as it is generally inefficient to find a substantial number of eigenstates with high accuracy. Quantum hardware with finite temperature such as quantum annealers, naturally implement approximate Boltzmann distributions [42] and don't benefit from the groundstate training. Appendix C studies an initially promising approach of the *Thermofield Double state* to implement a thermal state in qubits. Unfortunately, it cannot be used for this application as it creates a pure state and not a thermal ensemble. Training performance is thus equivalent to that of a quantum groundstate.

Another significant result of these simulations is that both QBMs, with diagonal or non-diagonal Hamiltonian, always achieve a final overlap with the training set of practically 100%. This indicates that giving a Quantum Boltzmann Machine flexible access to all of its Hamiltonian eigenstates provides the QBM model with very high expressivity. In this experimental setup, the training sets consist of 10 out of 16 random samples. Not all 16 degrees of freedom are required to learn those distributions. Appendix A shows a quantitative performance comparison between a

QBM with diagonal an non-diagonal Hamiltonian. Both models are always able to learn any training set distribution. For practical application on quantum devices, it is not clear whether this conclusion holds.

3.2.5 Challenges for Groundstate Training

Motivated by the results in Sec. 3.2.4, when implementing a Quantum Boltzmann Machine on a gate-based quantum computer, we would like to truncate the density matrix in Eq. (3.11) with a truncation parameter t that is as small as possible. Additional eigenstates are expensive to find with the Variational Quantum Eigensolver and, as we have seen, not strictly necessary for a QBM with non-diagonal quantum Hamiltonian.

A crucial consideration when nesting the VQE eigenstate search into the training of a Quantum Boltzmann Machine is that the energy functional $\mathcal{E} = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle}$ for the VQE and the negative log-likelihood in Eq. (3.12) are qualitatively different cost-functions with different objectives. The VQE is a classical-quantum hybrid algorithm for gate-based quantum computers to compute an upper bound to the lowest eigenvalue of a Hamiltonian. Note, that it is not explicitly designed to find the groundstate of a Hamiltonian. The groundstate of a Hamiltonian is not always the only state with lowest energy. In a diagonal Hamiltonian with degenerate eigenstates which all have the lowest energy, the VQE finds any of those eigenstates with equal probability, or, depending on if entangling gates are used, an arbitrary superposition of the eigenstates. For example, assuming a diagonal Hamiltonian H where $|\lambda_0\rangle = |00\rangle$ and $|\lambda_1\rangle = |11\rangle$ are degenerate eigenstates with the lowest energy such that $\hat{H}|00\rangle = E_0|00\rangle$ and $\hat{H}|11\rangle = E_0|11\rangle$. The variational energy functional $\mathcal{E}$ is minimal with $\langle 00|\hat{H}|00\rangle = \langle 11|\hat{H}|11\rangle = E_0$. An entangled state $|\lambda'\rangle = a|00\rangle + b|11\rangle$ solves the variational energy functional $\mathcal{E}$ for any values $a, b \in \mathbb{C}$ where $|a|^2+|b|^2 = 1$:

$$\begin{aligned}\langle\lambda'|\hat{H}|\lambda'\rangle &= |a|^2\langle 00|\hat{H}|00\rangle + |b|^2\langle 11|\hat{H}|11\rangle \\ &= |a|^2 E_0 + |b|^2 E_0 = E_0.\end{aligned} \tag{3.15}$$

The solution space of the VQE algorithm is thus of dimension > 0 with many equally plausible solutions. The VQE with entangling gates in the wavefunction ansatz will not reliably find a solution with fixed amplitudes a and b but an arbitrary superposition. Those superpositions are of course judged very differently by the negative log-likelihood for training the QBM as it strongly depends on the exact values of a and b for the correct distribution. Varying amplitudes will result in unstable and unreliable training signal.

To demonstrate the argument, a $t = 2$ diagonal QBM is trained on the Bars-and-Stripes (BAS) training set (see Sec. 2.5.3). The QBM is implemented with the qiskit *Qasm simulator* (see Sec. 2.3) and each eigenstate is found using the VQE

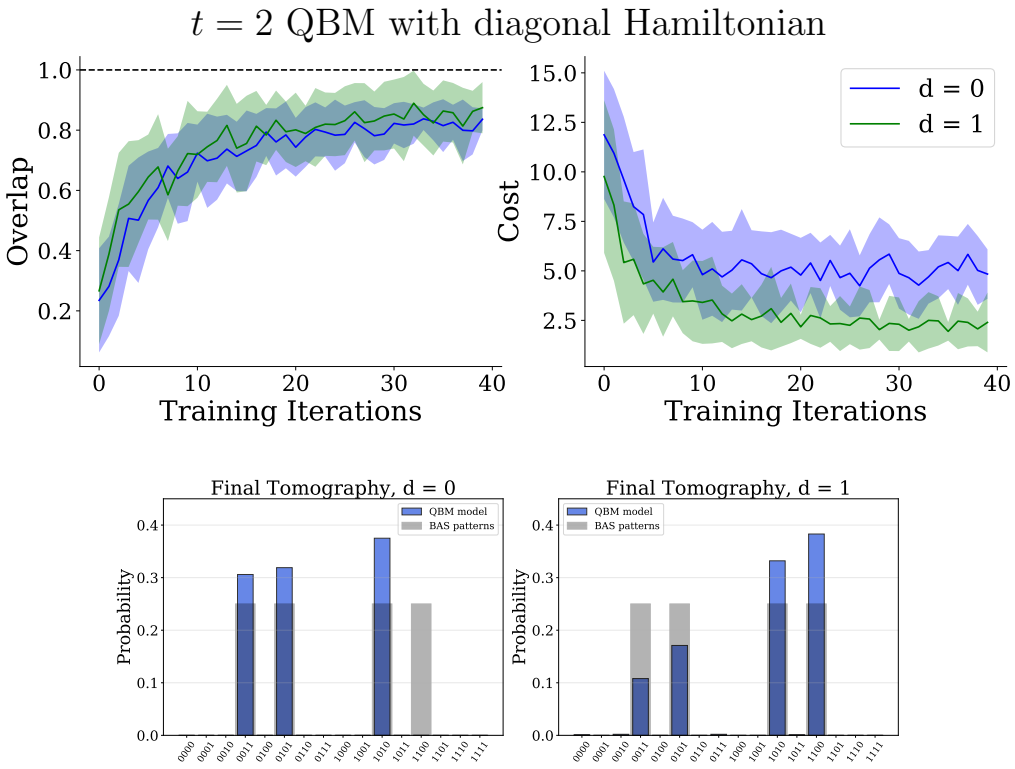

Figure 3.2.4: Training of a truncated $t = 2$ Quantum Boltzmann Machine with diagonal Hamiltonian on the BAS training set (see Sec. 2.5.3). Eigenstates of the diagonal Hamiltonian are found using the VQE algorithm (see Sec. 2.2). Compared are the training performance of the QBM when trained with the VQE at depth $d = 0$ and $d = 1$. The $d = 1$ state preparation contains entangling gates and thus allows a superposition of all four BAS patterns in three eigenstates. The amplitude of the superposition is not reliably controllable with the VQE.

algorithm (see Sec. 2.2) with 600 search iterations for each eigenstate. The wavefunction ansatz for the VQE is tested with depth $d = 0$ and $d = 1$. The models are trained with the gradients described in Eq. (3.13) that aim to minimize the negative log-likelihood (NLL). The gradients will tend towards making the four relevant eigenstates for the BAS patters approximately degenerate where the VQE for the $t = 2$ QBM finds the lowest three eigenstates. Each eigenstate of a diagonal Hamiltonian consists of exactly one basis state however the objective of the VQE is to find states of lowest energy. The state preparation with $d = 1$ implements entangling gates which allow superposition of degenerate eigenstates in the wavefunction like the example shown in Eq. (3.15).

Fig. 3.2.4 shows the average training progress for 20 repetitions as well as final tomographies of selected runs. Shaded areas indicate the standard deviation of the 20 repetitions. Indeed, the results show that the $t = 2$ diagonal QBM with $d = 1$ significantly outperforms the $d = 0$ model on the cost-function which is the negative

log-likelihood (NLL). As expected, the final tomographies provided show that the $d = 0$ model finds exactly three eigenstates while $d = 1$ allows a superposition of all four relevant eigenstates. The NLL is very responsive towards the correct distribution of BAS patterns which is why the contribution of all four BAS patterns instead of just three has significant effect on the NLL. The overlap during training is comparable between the models because of imperfect eigenstate search, especially in the more complicated $d = 1$ model. The final tomography of the $d = 1$ model has considerably higher overlap of over 95% compared to a definitive maximum of $\sqrt{\frac{3}{4}} \approx 86\%$ for $d = 0$.

A more involved argument holds for a $t = 0$ QBM with non-diagonal Hamiltonian. The groundstate of a non-diagonal Hamiltonian can be a general entangled superposition of basis states. As shown in Fig. 3.2.3, using exact diagonalization (see Sec. 2.3), a single groundstate of a non-diagonal Hamiltonian can be sufficiently trained to learn good approximations random training sets. Given a QBM Hamiltonian H (see Eq. (3.9)) of the form $H = H_{diag} + \sum_i h_i^x \sigma_i^x$, let $|\psi_0\rangle$ be the groundstate of H which contains the BAS patters with equal probability such that $|\langle \mathbf{s}|\psi_0\rangle|^2 = \frac{1}{4}$ for any BAS pattern $\mathbf{s} \in \{1100, 0011, 1010, 0101\}$. The energy functional $\mathcal{E}$ in that groundstate $|\psi_0\rangle$ reads

$$\begin{aligned} \langle\psi_0|\hat{H}|\psi_0\rangle &= \langle\psi_0|(\hat{H}_{diag} + \sum_{i=0}^{3} h_i^x \hat{\sigma}_i^x)|\psi_0\rangle \\ &= E_{diag} + \sum_{i=0}^{3} \langle\psi_0|h_i^x \hat{\sigma}_i^x|\psi_0\rangle \end{aligned} \tag{3.16}$$

where $H_{diag}|\psi_0\rangle = E_{diag}|\psi_0\rangle$ because H_{diag} is diagonal. The $\hat{\sigma}_i^x$ Pauli operator induces a flip on qubit i. This means for example that $\hat{\sigma}_2^x|1100\rangle = |1110\rangle$. Because all BAS patterns are at least two flips distant to each other, e.g. $\hat{\sigma}_1^x\hat{\sigma}_2^x|1100\rangle = |1010\rangle$, the non-diagonal contribution to the energy functional $\mathcal{E}$ for the VQE algorithm vanishes in $|\psi_0\rangle$. At that point, the same argument as for the diagonal Hamiltonian and Eq. (3.15) is valid where there is no reliable training signal of the model to evaluate the training progress because different superpositions of the BAS patters minimize the variational functional $\mathcal{E}$. The argument is valid for any trial state that contains a strong contribution of the samples that correspond to the BAS patterns. During training of a non-diagonal $t = 0$ QBM groundstate, beginning effects of this will already contribute and cause unreliable an training signal. Concluding, we expect a $t = 0$ QBM with a Hamiltonian like in Eq. (3.9) where the groundstate is found using the VQE algorithm to have severe difficulties converging.
This effect arises specifically from the interplay of the BAS training set in this binary representation and the chosen Hamiltonian ansatz. Because data scientists generally have to to work with the data that is provided, it becomes clear that the QBM Hamiltonian is not a suitable Hamiltonian ansatz for this application. For a Hamiltonian with a $\hat{\sigma}^x\hat{\sigma}^x$-term, this precise issue should not arise as the BAS patters

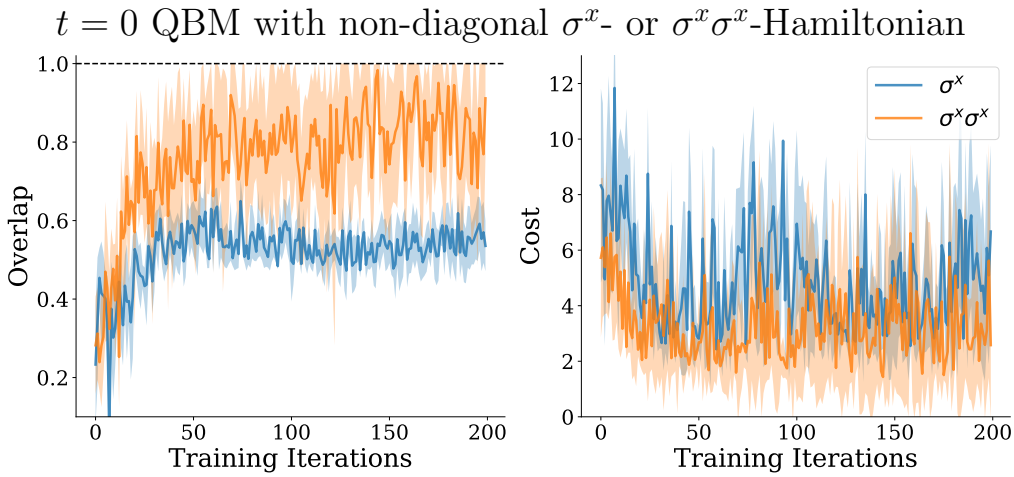

Figure 3.2.5: Training progress of truncated $t = 0$ Quantum Boltzmann Machines with non-diagonal Hamiltonians on the BAS training set (see Sec. 2.5.3). Compared are the QBM Hamiltonian with σ^x-term like in Eq. (3.9) and a Hamiltonian which instead contains a $\sigma^x\sigma^x$ term like in Eq. (3.17). The groundstates of the Hamiltonians are found using the Variational Quantum Eigensolver algorithm (Sec. 2.2). The groundstate training is degenerate for the σ^x Hamiltonian which causes the training to be unreliable and the progress to stagnate. Conversely, the $\sigma^x\sigma^x$-term lifts the degeneracy and allows for good training results.

are coupled through two spin flips.

To test our hypothesis, we train two $t = 0$ QBMs with different non-diagonal Hamiltonians. One Hamiltonian is the QBM Hamiltonian in Eq. (3.9) with σ_i^x-term and the other is the Hamiltonian

$$H = \sum_{ij} \sigma_i^z J_{ij}^z \sigma_j^z + \sum_i h_i^z \sigma_i^z + \sum_{ij} \sigma_i^x J_{ij}^x \sigma_j^x. \tag{3.17}$$

The groundstates of the respective Hamiltonians are calculated with the VQE eigensolver and 1500 search iterations. The large number of search iterations per groundstate are to ensure that the results are not artifacts of unfinished groundstate preparation. The VQE algorithm operates at a circuit depth of $d = 2$ which is considered deep enough for significant differentiation between the chosen models. The *cma-es* runs at $\lambda = 8, \mu = 3$.
Indeed, Fig. 3.2.5 shows clearly that the σ^x Hamiltonian for a $t = 0$ QBM is not appropriate when performing the algorithm on a gate-based quantum device and with the VQE. Conversely, the $\sigma^x\sigma^x$ Hamiltonian is able to more reliably find the groundstate which contains the correct superposition of BAS patterns. Note, that we only show the results of one training run. The shaded areas are given by the standard deviation of the of the λ function evaluations in the *cma-es* optimizer while the line denotes their mean. Frequently, the $\sigma^x\sigma^x$ model achieves approximately 100% overlap. Studying the new model more closely, we find that there exist other

conflicting solutions for this Hamiltonian but one of them is the exact BAS groundstate. It is not clear whether this is by caused by too shallow state preparation with $d = 2$ or more fundamental reasons.

This section shows that there is an important choice to be made when it comes to implementing different Quantum Machine Learning algorithms on different quantum hardware. For gate-based quantum computers, the more natural choice for a generative Machine Learning algorithm likely is circuit-based like the DVG in Sec. 3.1. For quantum hardware that more naturally implements groundstates or thermal ensembles of quantum Hamiltonians, such as a quantum annealer or analog quantum simulators (see Sec. 1.1.2), the algorithm of choice can certainly be Hamiltonian-based. The problem shown here arises specifically for the BAS patters, the QBM Hamiltonian with transverse field and the VQE algorithm. It can be partly solved by implementing the Hamiltonian in Eq. (3.17) which couples the BAS patterns. We show that algorithmic artifacts can arise from conflicting hardware and implementation techniques.

4 Quantum-assisted Generative Algorithms

To move towards implementation of Quantum Machine Learning algorithms for practical applications, it is required to scale the size of the models considerably. For current and near-term quantum computing devices, this may not be possible using just their quantum resources [1][21]. Instead, in this chapter we focus on implementing a qubit layer in the deepest layer of a generative Neural Network. This significantly reduces the number of qubits required to scale the model size while keeping the benefits of implementing qubits for generative Machine Learning tasks. The *quantum-assisted* generative Machine Learning algorithm discussed in this chapter is the *Quantum-assisted Generator* (QaG). It implements a Quantum Boltzmann Machine (QBM) in the deepest layer of a network, encoding the model's probability distribution and sampling from it. Additionally, this chapter provides first promising evidence that quantum phenomena may enhance performance of quantum-assisted generative neural networks. Finally, a hybrid training approach for quantum-assisted neural networks is proposed which promises scalability of the network size towards practical application on relevant data sets. It allows to train model which are not fully efficiently trainable with gradient descent, through simultaneous training iterations of a black-box optimization and gradient descent with backpropagation.

4.1 The Quantum-assisted Generator

The quantum-assisted Machine Learning algorithm studied in this section is the *Quantum-assisted Generator* (QaG). It is a generative algorithm with the goal to parametrize a target data distribution and by that extract essential features in the data. The QaG implements a Quantum Boltzmann Machine (QBM) (see Sec. 3.2) in the deepest layer of a neural network with subsequent classical layers (see Sec. 1.2.2 for Neural Networks). The QBM encodes the probability distribution of the generative model which can be sampled efficiently be measuring the qubit states. This provides a larger generative model with the sampling benefits of a qubit layer observed in Sec. 3.1.3 and reduces the number of qubits required to put quantum-assisted Machine Learning models to practical use. Practical code implementations of this algorithm can be found on github [51][52].

4.1.1 Model Description

A visual representation of the network architecture of the Quantum-assisted Generator can be seen in Fig. 4.1.1. In the following, we will use a short notation for the network architecture of a QaG, e.g. 2q-4c for a QaG with 2 qubits and 4 classical

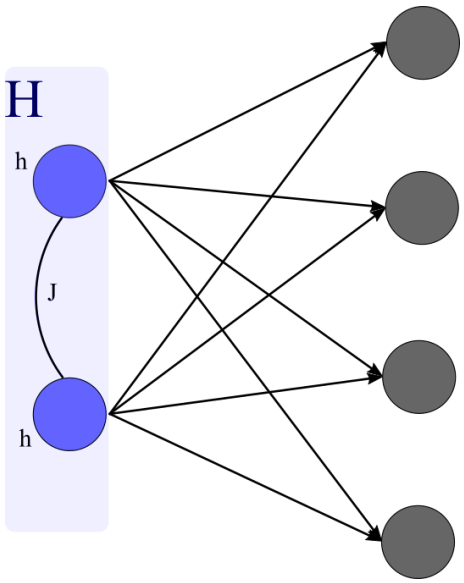

Figure 4.1.1: Schematic network architecture of a Quantum-assisted Generator with 2 qubits and 4 classical nodes. The qubit layer implements a Quantum Boltzmann Machine (QBM) which encodes the model's probability distribution. Samples of the QBM are upsampled by a subsequent classical network.

output nodes.
The model distribution P_{QaG} for the QaG is

$$P_{QaG}(\mathbf{v}) = \sum_{\mathbf{s}} P(\mathbf{v}|\mathbf{s}) P_\rho(\mathbf{s}) \tag{4.1}$$

where $P_\rho(\mathbf{s})$ is the probability distribution of the Quantum Boltzmann Machine and $P(\mathbf{v}|\mathbf{s})$ is the conditional probability of an output $\mathbf{v}$ for a given quantum sample $\mathbf{s}$. $P(\mathbf{v}|\mathbf{s})$ is defined by the classical network parameters, namely the weights between the layers and the biases of the classical visible nodes, and the activation function. The activation function for the classical layer is the sigmoid function

$$\text{sig}(y) = \frac{1}{1+e^{-y}} \tag{4.2}$$

which outputs values $\text{sig}(y) \in (0,1)$. Here, those outputs are interpreted as activation probabilities, i.e. the probability p_i for each node to output 1.
Given a quantum sample $\mathbf{s}$, the activation probability p_i for each output node is

$$p_i = \text{sig}(\sum_j W_{ij} s_j + b_i). \tag{4.3}$$

The probability $P_{QaG}(\mathbf{v})$ for a binary activation $\mathbf{v}$ is then calculated by multiplying the individual activation probabilities p_i or $(1-p_i)$ accordingly. The probability for

an example activation $\mathbf{v} = 0000$ thus equals

$$P_{QaG}(\mathbf{v} = 0000) = \prod_{i=0}^{3} \Big(1 - p_i\Big) \tag{4.4}$$

and for activation $\mathbf{v} = 0011$

$$P_{QaG}(\mathbf{v} = 0011) = (1 - p_0) \cdot (1 - p_1) \cdot p_2 \cdot p_3 \tag{4.5}$$

Generally, it is unfeasible to calculate $P_\rho(\mathbf{s})$ and thus $P_{QaG}(\mathbf{v})$ analytically. Common practice for evaluating the distribution of a generative model is to sample from it and approximate the true distribution with the distribution of the generated samples. Here, we draw samples from the QBM and from those calculate $P_{QaG}(\mathbf{v})$ analytically.

4.1.2 Training

The aim in training a Quantum-assisted Generator is to tune the Hamiltonian parameters $\{J, h_z, h_x\}$ of the Quantum Boltzmann Machine in the qubit layer and the classical network parameters $\mathbf{W}$, $\mathbf{b}$ such that the model distribution P_{QaG} best approximates a target data distribution P_D. This is achieved by minimizing a cost-function which provides a measure of distance between the model distribution and a training set which represents the target distribution. Like for the models described in Chap. 3, the cost-function for the QaG is the *negative log-likelyhood* (NLL)

$$\mathcal{L} = -\sum_{D} P_D(\mathbf{v}) \log \mathbf{max}(P_{QaG}(\mathbf{v}), \epsilon) \tag{4.6}$$

The singularity of the logarithm at $P_{QaG}(\mathbf{v}) = 0$ is regularized with $\epsilon = 10^{-8}$.

Currently, a lot of work is being done to find good gradients of cost-functions with regards to the parameters of a Quantum Boltzmann Machine with non-diagonal Hamiltonian [6]. Those are required to train a QaG entirely with gradient descent and backpropagation. Here, we circumvent this ongoing research by optimizing all parameters with the *cma-es* optimizer (see Sec. 2.4). In Sec. 3.2.3 we have shown that a QBM with 14 parameters can be trained appropiately by the optimizer. We assume that the optimizer is also able to train the QaG on a considerably more complex cost-landscape. For the number of parameters in this section, i.e. 15 in a 2q-4c QaG with diagonal Hamiltonian up to 33 in a 3q-6c QaG with non-diagonal Hamiltonian, this is likely to be true as *cma-es* is a popular optimization algorithm in industrial research application cases.

The QaG as discussed here cannot be trained on continuous-valued data. Appendix B shows a potential implementation of a quantum-assisted generative model which can be trained on continuous data but is not further studied in this work.

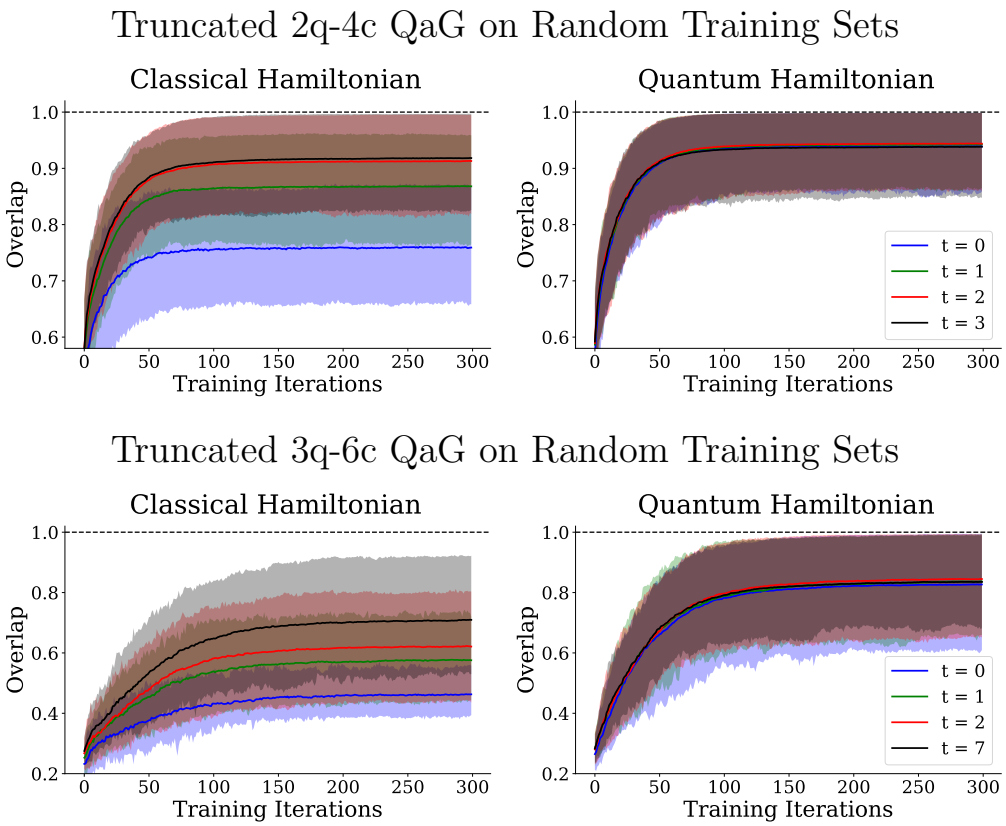

Figure 4.1.2: Training performance of truncated 2q-4c (top) and 3q-6c (bottom) QaGs for different truncation parameters t. The diagonal Hamiltonian (left) is considered classical while the non-diagonal Hamiltonian (right) is considered quantum. The models are trained on random training sets. Plotted are the means and the 5%-95% percentiles as shaded regions. The classical model generally requires all eigenstates for full performance while the quantum model can already be trained to good approximation using just the groundstate ($t = 0$). A quantitative difference in performance between the full classical QaG and quantum QaG arises for the 3q-6c model.

4.1.3 Error in Truncating the Boltzmann Density Matrix

In this section, we study the truncation error of the Quantum-assisted Generator. This is done by training QaGs with different truncation parameters t for the Boltzmann density matrix in Eq. (3.11) on different data and evaluate the loss in performance. In Sec. 3.2.4 we show the equivalent error in the Quantum Boltzmann Machine. For the QaG, we expect the error one makes by truncating the model to be smaller than for the QBM because the qubit layer no longer encodes the probability distribution of the data itself but instead the correlations between the classical output nodes.

For the simulation setup, we compare a QaG with diagonal Hamiltonian and a QaG with non-diagonal Hamiltonian with truncation parameters $t = 0, 1, 2, 2^n$-1.

The diagonal Hamiltonian is considered to be classical and the non-diagonal Hamiltonian to be quantum. Training is performed on two different types of training data sets: random binary training sets and binomial training sets (see Sec. 2.5). These training sets are chosen to analyze the truncation error for distributions which have no underlying structure and distributions which can be parametrized by few parameters. This showcases the fact that individual quantum samples in the QaG encode correlations between the classical output nodes and perhaps entire modes in the model distribution.
Training is performed on 100 generated training sets for 300 training iterations each. The diagonal and non-diagonal Hamiltonian model are always trained on the same data during each repetition.

Fig. 4.1.2 shows the results of a QaG with 2 qubits and 4 classical output nodes (2q-4c) and a 3q-6c QaG which were trained on the random training sets. Plotted are the average overlap for each truncation parameter t as well as the 5%-95% percentiles in the shaded regions. It is apparent that every additional eigenstate is required in the diagonal QaG model for good approximation of the random training data. Even though the qubit layer now encode the correlations between the classical nodes, the random training sets cannot be easily parametrized by few eigenstates in the diagonal model. In contrast to that, the QaG with non-diagonal quantum Hamiltonian shows no significant difference between different truncation paramters t.
Interestingly, in the case of the 3q-6c model, one can observe a quantitative performance difference between the full classical and quantum QaG model with $t = 7$. This is even the case when comparing just the quantum groundstate ($t = 0$) to the full classical model.

Fig. 4.1.3 shows the same simulation setup on binomial training sets. The QaG model with diagonal Hamiltonian and $t = 0$ achieves a low-quality approximation of a binomial. The qubit layer produces only one sample which the classical network of the QaG cannot shape into a binomial. The best approximation possible is a flat distribution with a slight focus on a peak in the middle of the histogram. In contrast to that, already $t = 1$ achieves close to 100% overlap and a very good approximation of a binomial. This provides instructive evidence of how a highly structured distribution like the binomial training set can be parametrized with less degrees of freedom. The truncation parameter t in the diagonal model could thus be seen as an external parameter for the quality of approximation of the generative model. By varying t, one can observe a hierarchy of features that the model learns to achieve the best possible training performance.

In analog to the random training set, the non-diagonal QaG performs equally well with any truncation paramter t. The results indicate that a mixed quantum state may strictly not be required to utilize the Quantum-assisted Generator algorithms to its fullest potential. This is a significant result for implementing an equivalent

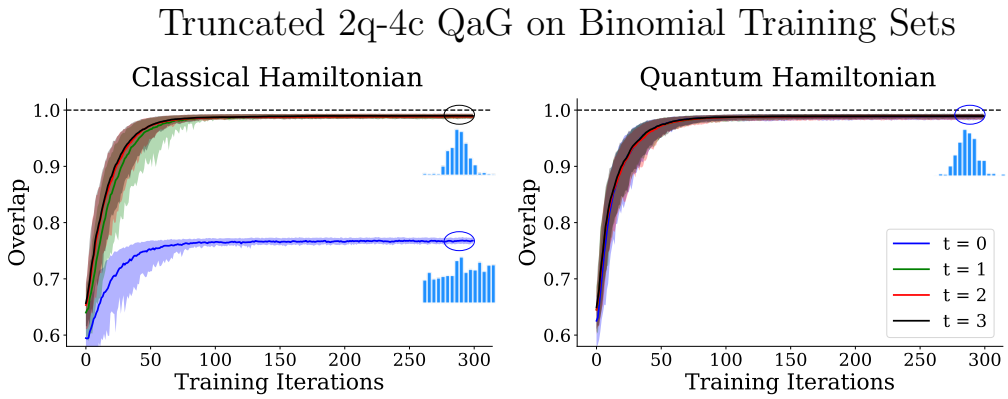

Figure 4.1.3: Training performance truncated of 2q-4c QaGs for different truncation parameters t. The diagonal (left) Hamiltonian is considered classical while the non-diagonal Hamiltonian (right) is considered quantum. The models are trained on binomial training sets (see Sec. 2.5.2) Plotted are the means of 200 generated training sets and the 5%-95% percentiles as shaded regions. $t = 1$ is already sufficient for the classical QaG to parametrize the binomial data which means that only two distinct quantum samples encode most of the essential correlations of between the classical nodes in the output layer. The quantum model performs equally well with any truncation t.

Hamiltonian-based generative algorithm on quantum devices which strictly operate on pure states such as gate-based quantum computers.

4.1.4 Benchmarking a Potential Quantum Advantage

Motivated by the findings in Sec. 4.1.3 where the Quantuma-assisted Generator model with non-diagonal Hamiltian shows a qualitative performance difference, this section provides quantitative benchmarking simulations between the diagonal and non-diagonal Hamiltonian QaG. The diagonal model is taken as classical reference while the non-diagonal model is considered quantum.

The benchmarking setup is to train a QaG with diagonal Hamiltonian and a QaG with non-diagonal quantum Hamiltonian on random training sets (see Sec. 2.5) and compare their relative training performance. The classical model is implemented with the full Boltzmann density matrix in Eq. (3.11) while for the quantum model, we benchmark the full Boltzmann distribution and also a single quantum ground-state with truncation parameter $t = 0$. The QaG network architecture is 3q-6c with $n = 3$ qubits and 6 classical output nodes. The random training sets are of different sizes, i.e. with a different number of random training samples. Those range from 2 samples per training set up to 1024. As the 3q-6c QaG has $2^6 = 64$ possible output activations, a training set with 1024 samples is relatively flat while training sets in the intermediate range may contain many samples repeatedly and other samples not at all.

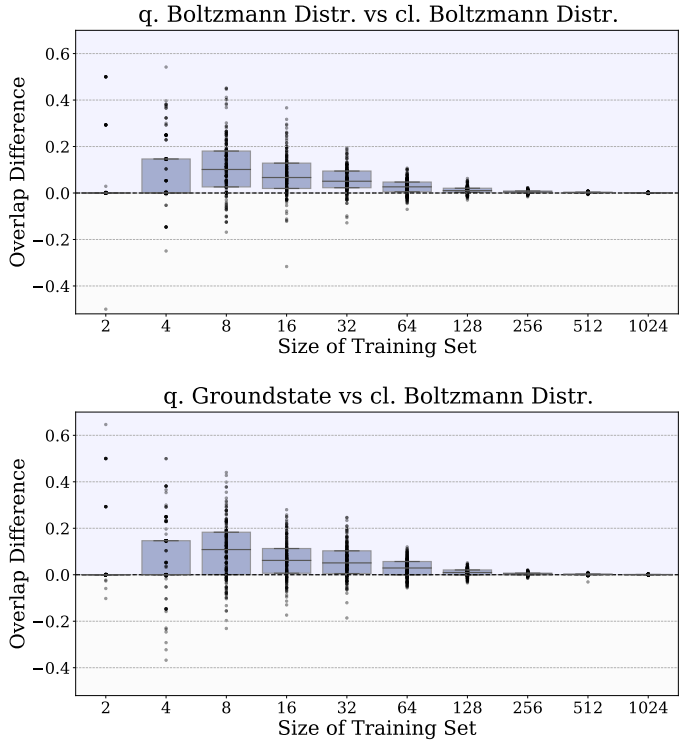

Figure 4.1.4: Benchmarking results of a 3q-6c Quantum-assisted Generator. A QaG with non-diagonal quantum Hamiltonian and a QaG with diagonal classical Hamiltonian are trained on random training sets of increasing size, i.e. increasing number of samples per training set. Both models are trained for 5.000 iterations and their final overlap with the respective training set is calculated into an overlap difference. A positive overlap difference (blue shaded area) indicates a better training performance of the quantum model. The simulations are repeated on 200 random training set per training set size and the 25%-75% percentiles are shown as boxes. The quantum QaG model generally outperforms the classical model. This is especially true for intermediate training set sizes. Surprisingly, the quantum advantage is similarly maintained when training just a quantum grountstate and comparing to the full Boltzmann distributed classical model.

The classical and quantum model are trained for 5.000 training iterations on a given training set to guarantee a final result. The resulting final overlap of the model distribution with the training set is calculated into an overlap difference between the classical and the quantum model. A positive overlap difference implies that the quantum model has a larger overlap and thus achieved a better approximation of the target distribution. This procedure is performed for growing training sets and 200 repetitions per training set size.

Fig. 4.1.4 shows the overlap differences in each run marked as grey points as well as the 25%-75% percentiles of the 200 repetitions inside the boxes. The blue-shaded areas indicates a positive overlap difference and thus a better training result of the quantum model.

Indeed, we observe a quantitative quantum advantage of the QaG with non-diagonal quantum Hamiltonian as compared to a QaG with diagonal Hamiltonian. This is especially true for training set sizes in the intermediate range of $4-64$ which one might consider to be the most random as there are many samples but the distribution is still far from evenly distributed. Interestingly, a very similar quantum advantage is maintained when comparing just a trained quantum groundstate to the full classical QaG distribution. For training set size in the range of 8-16, which are equivalent measurements to those in Fig. 4.1.2, we are able to replicate an overlap difference of roughly 10%.

To conclude, we have measured a quantum advantage in training a Quantum-assisted Generator with non-diagonal Hamiltonian. Given that the model contains 3 additional parameters in the qubit layer, one may expect that some of the quantum advantage due to that. Very interestingly, even the quantum groundstate outperforms the QaG with diagonal Hamiltonian by a very similar amount. This indicates that indeed quantum phenomena are the leading cause of these performance enhancing effects. It seems like local minima in the training of the quantum-assisted model are more effectively being avoided by the model which contains superposition and entanglement in its eigenstates.
It has to be noted, that the QaGs in this section are trained using the *cma-es* optimizer to minimize the negative log-likelihood. We have not performed this benchmarking setup with other optimizers. Given that the results are consistent between the thermal ensemble implementation of the QBM and a single quantum groundstate, which are very different models as one is a mixed state and one is a pure state, we expect a certain quantum advantage to be maintained.

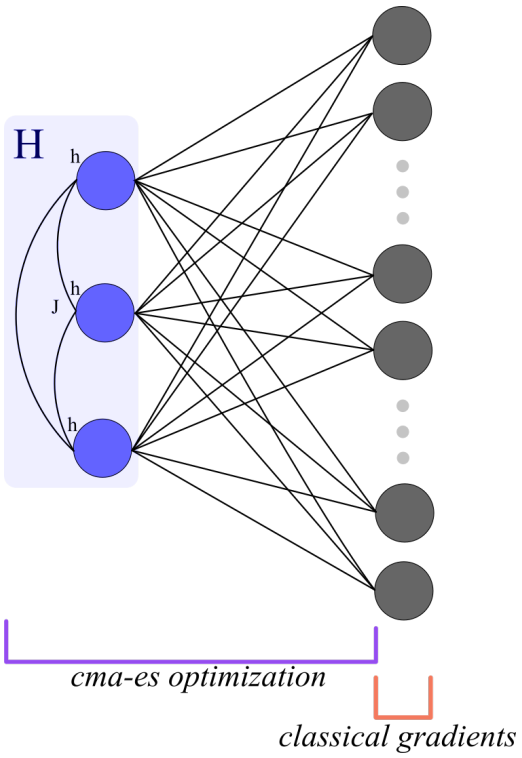

Figure 4.2.1: Schematic representation of the hybrid training of a Quantum-assisted Generator. Here, only the visible biases of the QaG are trained by conventional gradient descent while the rest of the model parameters are trained by the *cma-es* optimization algorithm.

4.2 Hybrid Training of Quantum-assisted Neural Networks

Convential Artificial Neural Networks (ANNs) are known to be highly scalable in the number of layers and nodes. To be enhance modern ANNs through quantum layers, one needs to be able to add layers to their quantum-assisted architectures and be able train them. In this section, we present a hybrid training approach for Quantum-assisted Neural Networks which satisfies these requirements. The relevant code for the hybrid training can be viewed on github [51][52].

4.2.1 Description of the Hybrid Training Method

The term 'hybrid training' used in this work refers to the training of Quantum-assisted Neural Networks with an interplay of the *cma-es* optimization algorithm (Sec. 2.4) and gradient descent. Generally, *cma-es* is used for all parameters of a model which cannot be efficiently trained with gradients. Sometimes one may prefer to train some additional parameters with the optimizer in order to provide it with more flexibility. Classical gradients are used everywhere else.

The functionality of the hybrid training is as follows:
cma-es is based on mutating the current best parameters, recombining them and taking the best performing ones as parents for the next generation of mutated parameters. It proposes a population of λ different sets of mutated parameters in its evolutionary strategy. These define λ different networks with a specific cost associated with their outputs. Those networks only differ in the parameters that are tuned by the optimizer but are otherwise identical. For each of the networks, we calculate hypothetical gradients for the parameters which are not tuned by the optimizer but they are not yet applied to the respective parameters. The best μ out of λ parameter sets are carried over to the next generation of the *cma-es* algorithm and of these μ best-performing networks, the calculated classical gradients are averaged and applied to the corresponding network parameters.

With the hybrid training proposed in this section, the optimization algorithm and gradient descent are applied simultaneously on different parts of the network. It is also possible to perform the training in repeated cycles where first a *cma-es* iteration is performed followed by one iteration of gradient descent. This cycle-method has not been found to work in practice. A likely explanation is that parameter training is a very high-dimensional and complex problem where all parameters need to be tuned at once to correctly navigate the space. Some minima might only be accessible this way.

4.2.2 Improved Convergence Results

In this section, we apply the hybrid training described in Sec. 4.2.1 to the Quantum-assisted Generator network in Sec. 4.1. For a QaG with 3 qubits and 100 classical visible nodes (3q-100c), there are 409 model parameters to be trained. In a hybrid training approach for this QaG, the 100 visible biases are trained with classical gradients while the remaining 309 weights and Hamiltonian parameters are trained by the optimizer. A schematic representation of the network architecture and the hybrid training can be seen in Fig. 4.2.1.

The training sets in this section for testing the hybrid training consist of binary 10x10 samples of the MNIST training data set (see Sec. 2.5.4). This data set is one of the standard data sets for benchmarking Artificial Neural Networks and Machine Learning algorithms in general. Here, the training sets are constructed of just two MNIST samples which are selected at random in 100 different training repetitions. We compare the convergence rate of the hybrid-trained QaG with a purely *cma-es* optimized QaG.

Fig. 4.2.2 shows the training progress of the two models on the 100 different training sets and 1.000 training iterations. The simulations give a mean final overlap of 45% for the hybrid-trained model and 13% for the *cma-es* optimized model. Fig.

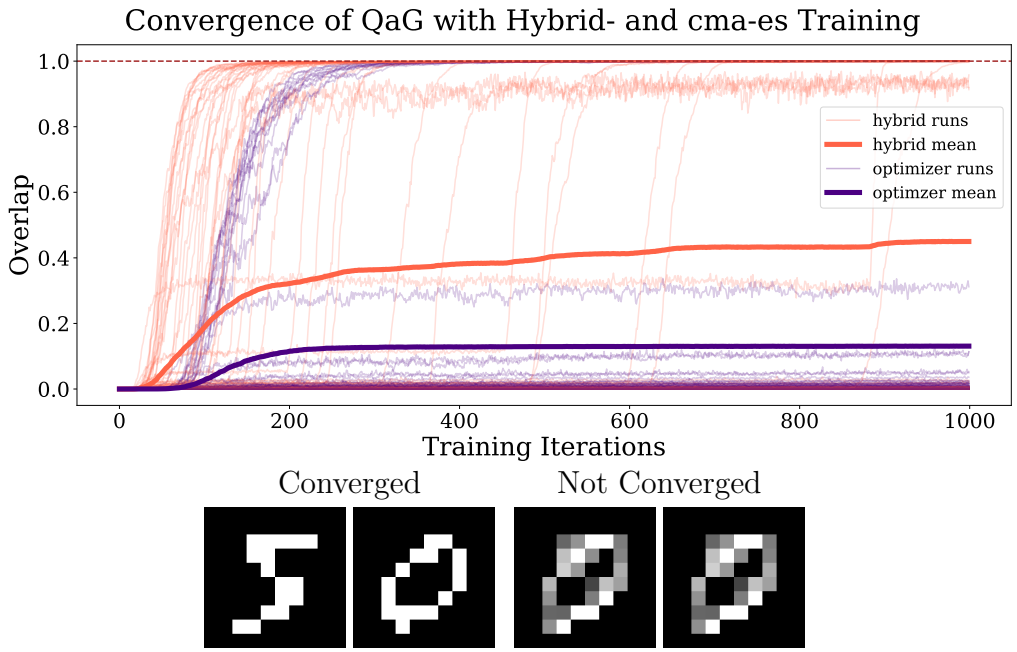

Figure 4.2.2: Training performance comparison between a 3q-100c hybrid-trained Quantum-assisted Generator (orange) and a 3q-100c QaG which is purely trained by the *cma-es* optimization algorithm (purple). Training is performed 100 times on two newly selected 10x10 MNIST training data samples for 1.000 training iterations each. The hybrid-trained model shows a mean final overlap of 45% with the respective training set while the *cma-es* optimized model achieves only 13%. Shown are typical samples of a converged model with near 100% and a model which did not converge and has close an overlap of $< 10\%$ with the correct training samples.

4.2.2 also provides typical samples of a QaG that did not converge successfully. The algorithm typically learns an average representation of the training set if it cannot learn the distinct samples. Those average solutions have an overlap of barely over 0% with the correct data unless the two MNIST data samples contain the same digit in a similar style. The final overlap of the QaGs thus mostly corresponds to whether the QaG was able to learn two distinct MNIST samples or only their average.

This experiment provides indication that black-box optimization of entire deep networks in unlikely to be effective. Conversely, hybrid training has the potential to provide Quantum-assisted Neural Networks with one of the most important factors in today's context of data science, namely *scalability*.

5 Input-Dependent Quantum-Assisted Artificial Neural Networks

This chapter introduces three input-dependent Quantum-assisted Artificial Neural Networks, in particular two Hamiltonian-based Autoencoders and one Gate-based Autoencoder. Typical goals of Autoencoders are dimensionality reduction Autoencoder is dimensionality reduction. The general network architecture consists of an encoder network, a bottleneck layer that displays the lowest-dimensional representation of the data in so-called latent space, and a decoder network. The encoder network performs a forward pass of the data where consecutive layer generally have a lower number of nodes. This is called *downsampling*. The decoder network scales the activation back up to the size of the original input. This is called *upsampling*. Autoencoders are trained to minimize the reconstruction error between the output and the input. If the high-quality reconstruction is successful, then the model must have learned an efficient encoding of the data as it was passed through a much smaller-dimensional layer. Typical bottleneck layer sizes in conventional Autoencoder range from two or three up to a few dozen.
An Autoencoder can be trained with different applications in mind like image compression and denoising [25], data generation [13], feature extraction [63], and more. For this work, we focus on the ability to strictly achieve good reconstruction of the training data. In our networks, a qubit layer encodes the lowest-dimensional representation of the data and is thus located in the middle bottleneck layer.

5.1 The Hamiltonian-based Autoencoder

The *Hamiltonian-based Autoencoder* (HAE) proposed and studied in this work is a Quantum Machine Learning algorithm and, more specifically, a Quantum-assisted Neural Network. As an Autoencoder, the most fundamental property of the algorithm is to calculate a low-dimensional representation of input data in the middle bottleneck layer of the network. This bottleneck layer consists of a Quantum Boltzmann Machine (see Sec. 3.2 for the QBM) which is implemented on qubits. The purpose of the QBM is to provide a distribution of quantum samples for a specific input and thus a distribution of model outputs. This probabilistic modeling in the so-called latent space of the HAE may aid the training of the Autoencoder or enhance the application of a trained model. The relevant code for the HAE in this work can be found on github [51][52].

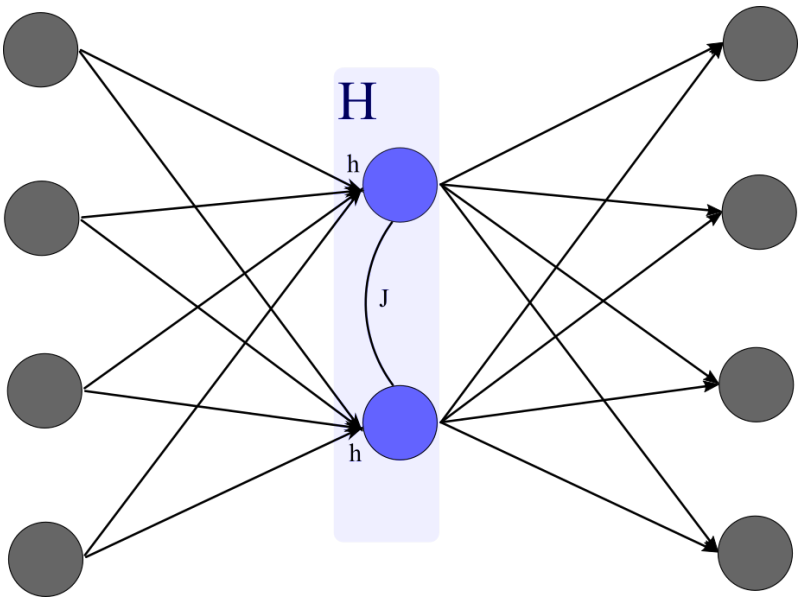

Figure 5.1.1: Network architecture of the Hamiltonian-based Autoencoder (HAE). The goal of this model is to learn a lower dimensional representation of data. The HAE is a Quantum-assisted Autoencoder which contains qubits in the bottleneck layer of the network. The qubits implement an input-dependent Quantum Boltzmann Machine which produces samples according to the input of the network.

5.1.1 Model Description

The model architecture of the Hamiltonian-based Autoencoder can be seen in Fig. 5.1.1. It consists of an encoder network, a middle layer that is commonly called *bottleneck layer* or *latent space*, and a decoder network. The input layer takes an input $\mathbf{x}$ and the network produces outputs $\mathbf{v} = f(\mathbf{x})$ where f is the transformation of the Autoencoder on the input. The HAE is trained to reconstruct inputs such that $f(\mathbf{x}) = \mathbf{v} \approx \mathbf{x}$. Given that $\mathbf{x}$ is propagated through a smaller bottleneck layer, if high-quality reconstruction is successful, the model must have learned an efficient representation of the data in the bottleneck layer which contains the most essential structural information.

In this work, the technique for making the Quantum Boltzmann Machine in the qubit layer input-dependent is to add a local field offset $\tilde{\mathbf{h}}_z$ to the QBM Hamiltonian (Eq. (3.9)). The offset depends on the input $\mathbf{x}$ and the paramters of the encoder, so the weights $\mathbf{W}$ connecting the input layer and the qubit layer:

$$\tilde{h}_i^z = \sum_j W_{ij} x_j. \tag{5.1}$$

The input-dependent QBM Hamiltonian $\tilde{H}$ is then calculated via

$$\begin{aligned}\tilde{H} &= H + \sum_i \tilde{h}_i^z \sigma_i^z \\ &= \sum_{ij} \sigma_i^z J_{ij} \sigma_j^z + \sum_i (h_i^z + \tilde{h}_i^z)\sigma_i^z + \sum_i h_i^x \sigma_i^x\end{aligned} \tag{5.2}$$

where H is a foundation Hamiltonian that is not directly implemented in this algorithm. The parameters $\{J, h^z, h^x\}$ of the foundation Hamiltonian H and the weights $\mathbf{W}$ of the encoder that lead to the qubit layer determine an input-dependent QBM. These are the parameters which are trained to adjust the Autoencoder. The QBM can be sampled as explained in Sec. 3.2 to produce quantum samples $\mathbf{s}$. The decoder network upsamples the quantum samples $\mathbf{s}$ with the subsequent weights $\mathbf{W}^T$ into visible activations

$$\mathbf{v} = \text{sig}\left(\mathbf{W}^T\mathbf{s} + \mathbf{b}\right). \tag{5.3}$$

The activation function for the output layer is the sigmoid activation function (Eq. (4.2)). Note, that the weights $\mathbf{W}^T$ of decoder network are the transposed weights of the encoder. This *shared-weight* Autoencoder architecture is a choice we make and is also used in literature [32]. The visible activations v_i in the HAE are not to be understood as activation probabilities p_i like in the case of the Quantum-assisted Generator in Sec. 4.1. Rather, they are gray-scale values and the direct output of the model in response to either binary- or continuous-valued input.

The function of the decoder of the Hamiltonian-based Autoencoder is an analog to the Quantum-assisted Generator in Sec. 4.1. In fact, the different h^z define a family of QaGs which are called depending on the input to the network. Hence, training of the model implicitly consists of training a family of QaGs and a correct mapping of input to one or several of those QaGs.

The Hamiltonian-based Autoencoder is noticeably limited by the number of possible quantum samples $\mathbf{s}$. For a model with n qubits, there are 2^n possible states that could be measured. Consequently, the HAE cannot create more than 2^n different output activations because upsampling of the quantum samples is deterministic. Making use of the limited resources that the binary quantum samples offer compared to floating-point values implemented in conventional Autoencoders, an interesting application of this model is to finding a good pairing of different inputs where structurally similar inputs are grouped into the same quantum sample and thus the same output. This has very close resemblance to clustering algorithms and *vector quantization* which is a technique used audio and image compression [41][45]. In practice, not all 2^n quantum states will actively take part in the model as it requires a large number of network weights to separate different inputs into distinctly different $\tilde{h}^z$ offsets.

An additional potential application of the Hamiltonian-based Autoencoder is in generating samples from a trained latent space. Autoencoders that can be used to sample from latent space are called *Variational Autoencoders* [50]. In Chapters 3 & 4, we have studied the benefits of implementing qubits as in generative Quantum Machine Learning models and in qubit layers of generative Quantum-assisted Neural Networks. It is likely possible to implement a HAE such that one can efficiently sample the latent space of the model and from that generate valid samples that follow the training set distribution. This application is not further studied in this work but is a good candidate for further research.

5.1.2 Training

The training of the Hamiltonian-based Autoencoder in this section is performed with the *cma-es* optimization algorithm (see 2.4). The cost-function that is used as objective function for the optimization algorithm is the *mean square error* (MS error)

$$\mathcal{C} = \sum_{D} \frac{1}{2} \sum_{i} \frac{(x_i - v_i)^2}{N} = \sum_{D} \frac{1}{2} \sum_{i} \frac{(x_i - f(x)_i)^2}{N} \tag{5.4}$$

where x_i and v_i are the values of pixel i of the input $\mathbf{x}$ and the output $\mathbf{v} = f(\mathbf{x})$ of the HAE respectively. The MS error measures the average squared difference of the N output nodes v_i to the corresponding input node x_i. Hence, we will also refer to it as the *reconstruction error*. This pixel-wise squared difference is summed over the training set D. By using the MS error as cost-function for the *cma-es* algorithm, the network- and Hamiltonian parameters are tuned to optimize the reconstruction quality of the HAE.

A relevant consideration in training the HAE is the dimensionality of the qubit layer's Hilbertspace. For n qubits, it is 2^n which means that only 2^n different quantum samples $\mathbf{s}$ can be produced by the entire model, irrespective of the size of training set or number N of visible nodes. One of the challenges in training the HAE is therefore that it must learn to manage its resources and pair similar inputs together to the same output. The Hamiltonian local field offsets $\tilde{h}_i^z$ must thereby be learned such that the resulting offset Quantum Boltzmann Machines produce similar samples for inputs of similar shapes, ideally with a slightly different distribution of those samples.

A common challenge for Autoencoders is to converge to a solution of network parameters which does not create averaged outputs for different input samples. In this model, the goal in training is to converge to good values of $\mathbf{W}, \mathbf{b}$ such that the Hamiltonian offsets $\tilde{h}^z$ are distinctly different for significantly different inputs, but also that the limited number of quantum states are used to effectively pair similar inputs together.

During training, the HAE has to be evaluated for every training sample in every training iteration. This comprises of finding the eigenstates of the input-dependent QBM and constructing the truncated density matrix in Eq. 3.9 for every training sample. This is a challenge for practical application on industrial-sized training sets. The expected pay offs for implementing such a quantum-assisted Machine Learning algorithm need to warrant the computational cost, i.e. efficient grouping of inputs, enhanced training performance, probabilistic modeling or direct sampling of the latent space.

5.1.3 Bencharking a Quantum Advantage

Sec. 4.1.4 shows a quantum advantage in training for a Quantum-assisted Generator when implementing a Quantum Boltzmann Machine with non-diagonal Hamiltonian in the generating layer of the network. In this section, we test for a quantum advantage in the Hamiltonian-based Autoencoder. As in the case with the QaG, we define a quantum advantage as improved training performance of the HAE when implementing a QBM with non-diagonal Hamiltonian in the qubit layer as compared to a QBM with diagonal Hamiltonian. The model with diagonal Hamiltonian acts as classical reference whereas the model with non-diagonal Hamiltonian is considered quantum.
In analogy to Sec. 4.1.4, the diagonal and non-diagonal Autoencoder models are trained on random training sets (see Sec. 2.5 for random training sets). The number of iterations in training is chosen to be high enough for the training to have plateaued for a long time or until convergence with very small MS error. In some cases, the models may have been able to converge further after a long plateau but this is not considered for these measurements. The final mean square-error over the training set for the classical and quantum model is calculated into a *fitness difference* which is the negative difference of the MS error. A positive fitness difference indicates an advantage of the non-diagonal quantum model and a smaller average MS error.
Training is performed on different random training sets with either binary values $\in \{0, 1\}$ or continuous values $\in (0, 1)$. By varying the number of samples in the training set, we study the model's ability to learn distinct reconstructions of random samples and to find the best mapping of different inputs to the same output if it cannot achieve distinct reconstruction. For the benchmarking simulations, we study a HAE with 6 classical input- and output nodes and 2 qubits (6c-2q) and one with 6 classical nodes and 4 qubits (6c-4q). Note, that 2 qubits have $2^2 = 4$ possible states and therefore partly less than the number of different inputs during training which range from two to eight samples per training set. It is therefore expected for the 6c-2q model to have a larger MS error than the model with 4 qubits.

All network parameters are trained with the *cma-es* algorithm according to Sec. 5.1.2. The measurements of the 6c-2q model are repeated on 100 different training

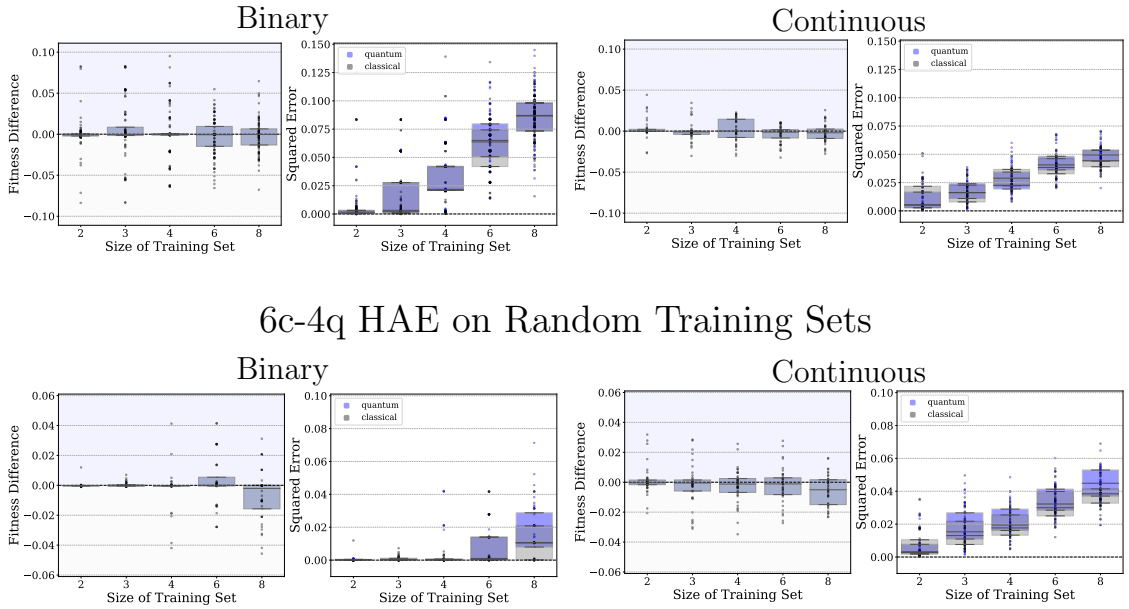

Figure 5.1.2: Benchmarking results for a potential quantum advantage in the Hamiltonian-based Autoencoder (HAE). A quantum advantage is defined as a smaller mean square error when comparing a HAE with non-diagonal Hamiltonian (blue) to a HAE with diagonal Hamiltonian (grey). The diagonal model is taken as classical reference while the non-diagonal is considered quantum. HAE models with 2 and 4 qubits in the bottleneck layer are trained on random binary and continuous training sets of different sizes, i.e. increasing number of training samples. Their error after training is calculated into a fitness difference. A positive fitness difference (blue shaded area) indicates a quantum advantage. 100 and 40 individual runs are marked as dots in the top and bottom plot respectively while the 25%-75% percentiles are shown as boxes. On binary data, the results are indecisive but on continuous data, there is indication for the quantum model to perform worse.

sets contrary to 40 different training sets for the 6c-4q model. This is due to limitations in simulation time.

Fig. 5.1.2 shows the results of the benchmarking simulations with individual runs marked as points and the boxes indicating the 25%-75% percentiles. The blue shaded area in the relative fitness difference plot indicates a quantum advantage but there is no clear systematics that indicate a such a quantum advantage. The absolute error of classical and quantum models over the training repetitions are also shown. As expected, both the diagonal and the non-diagonal HAE models perform better with more qubits. The average MS error of the 6c-4q model is smaller than that of the 6c-2q model, especially on binary training sets where reconstruction usually succeeds either perfectly or only as an average. This observation may partly be caused by the fact that the 6c-4q model has an additional 12 weights that can be used to

better encode the input data. Continuous data is generally harder to reconstruct exactly but there are also smoother average solutions which reduce the expected MS error for a larger number of training samples.

Contrary to the measured quantum advantage in Sec. 4.1.4, where the quantum effects in the non-diagonal Hamiltonian system of a Quantum-assisted Generator seem to enhance training performance, Fig. 5.1.2 shows no quantum advantage for the Hamiltonian-based Autoencoder. Instead, there is a slight trend that the HAE with diagonal Hamiltonian is outperforming the model with non-diagonal Hamiltonian on continuous data, especially for the 6c-4q model. For binary data, the results are indecisive. The simulations for the 6c-2q have more statistics over 100 repetitions and show no significant difference. For the task of strictly fitting the model parameters to a given training set without any focus on generalization or other performance metrics, the HAE with diagonal Hamiltonian seems more reliable than the non-diagonal model. How other characteristics than strict training for reconstruction of training data depend on quantum effects such as superposition and entanglement is not clear from these simulations and need to be tested further.

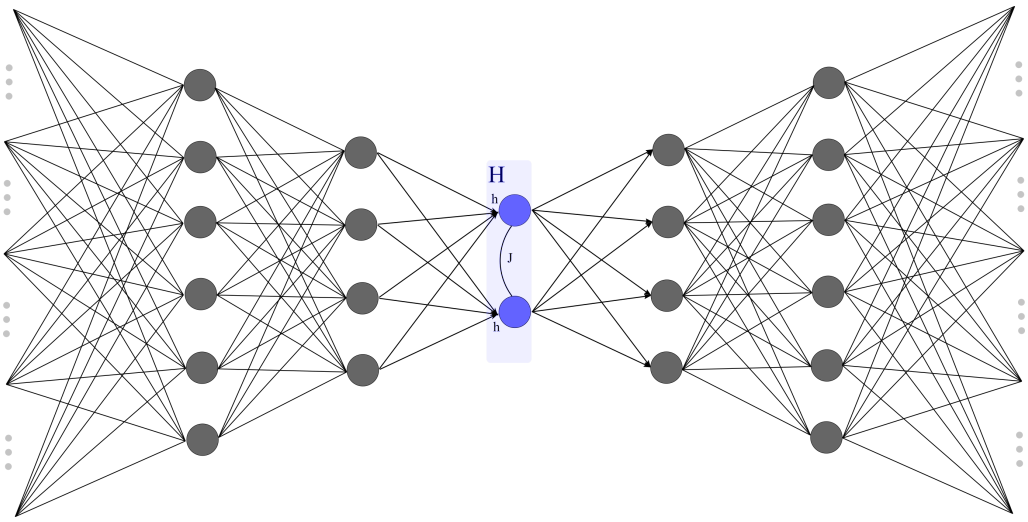

Figure 5.2.1: Network architecture of the Hybrid-trained Quantum-assisted Autoencoder (HTHAE). The goal of this model is to learn a lower dimensional representation of data. The HTHAE is a Quantum-assisted Autoencoder which contains qubits in the bottleneck layer of the network. The qubits implement an input-dependent Quantum Boltzmann Machine which produces samples according to the input of the network. This model employs a hybrid training method which consists of an interplay of black-box optimization, gradient descent and backpropagation. Thus, the HTHAE is scalable with additional classical network layers.

5.2 The Hybrid-trained Hamiltonian-based Autoencoder

The Hybrid-trained Hamiltonian-based Autoencoder (HTHAE) is a quantum-assisted Machine Learning algorithm and an extension of the Hamiltonian-based Autoencoder (HAE) archictecture discussed in Sec. 5.1 by including additional classical network layers which are trained using a hybrid training method. As an Autoencoder, the HTHAE is trained to reconstruct inputs and learn low-dimensional structure in the data. The difference to the HAE is the training and thus scalability of the model. Our implementation of the HTHAE can be viewed on github [51][52].

5.2.1 Model Description

The network architecture of the Hybrid-trained Hamiltonian-based Autoencoder can be seen in Fig. 5.2.1. It consists of an encoder network, a Quantum Boltzmann Machine in the bottleneck layer, and a decoder network. Inputs to the network are passed through the classical encoder network which consists of several layer, generally becoming smaller in size. The layers of the encoder network follow the notation (0) for the input layer, (1) for the first layer, and (ℓ) for layer ℓ. The corresponding biases are $\mathbf{b}^{(\ell)}$ while the weights connecting layer (ℓ) and $(\ell+1)$ are

called $\mathbf{W}^{(\ell)}$.
The encoder propagates the activation up to the qubit layer in layer (k). This qubit layer implements a Quantum Boltzmann Machine (QBM) (see Sec. 3.2). The QBM Hamiltonian is input-dependent with an added local field

$$\tilde{h}_i^z = \sum_j W_{ij}^{(k-1)} u_j^{(k-1)} \tag{5.5}$$

such that the QBM Hamiltonian reads

$$\begin{aligned} \tilde{H} &= H + \sum_i \tilde{h}_i^z \sigma_i^z \\ &= \sum_{ij} \sigma_i^z J_{ij} \sigma_j^z + \sum_i (h_i^z + \tilde{h}_i^z) \sigma_i^z + \sum_i h_i^x \sigma_i^x \end{aligned} \tag{5.6}$$

where H is a foundation Hamiltonian of the QBM to which input-dependent local fields are added.
The decoder network upsamples the quantum samples $\mathbf{s}$ of the input-dependent QBM up to the visible output activation $\mathbf{v}$ which has the same size as the original input.

In this work, network architectures may vary in the number of layers and nodes. For a hypothetical HTHAE consisting five layers with 100, 20, 4, 20, 100 nodes in the respective layers and the middle qubit layer implementing a QBM on 4 qubits, we will follow the notation 100c-20c-4q.

Because the model aims to reconstruct the input samples which may have continuous grey-scale values $\in [0, 1]$, we choose the *sigmoid* activation function (Eq. 4.2) for the final output layer

$$\mathbf{v} = \mathrm{sig}\left(\mathbf{W}^{(-1)}\mathbf{u}^{(-1)} + \mathbf{b}^{(0)}\right) \tag{5.7}$$

The activation function for all other intermediate layers is the *leaky Rectified Linear Unit* (leaky ReLU) $\zeta : \mathbb{R} \to \mathbb{R}$. The leaky ReLU is defined as

$$\zeta(y) = \begin{cases} y, & \text{if } y \geq 0 \\ 0.1\, y, & \text{if } y < 0 \end{cases} \tag{5.8}$$

and is a popular modification of the commonly used ReLU activation function. It is 'leaky' because, unlike the regular ReLU, it leaks small contributions of negative values of y instead of setting them equal to zero.

One significant feature of the HTHAE model is that decoder shared parameters with the encoder network. This means that the layers which are the same distance to the bottleneck layer have the same shape and parameters i.e. $\mathbf{W}^{(\ell)T} = \mathbf{W}^{(-\ell)}$

where in reference to their corresponding layers in the encoder, the layers of the decoder are called $(-\ell)$. The network is essentially 'mirrored' around the middle which is a necessary condition for the hybrid training of the model. Details can be seen in Sec. 5.2.2.

As mentioned in Sec. 4.1, the HAE and HTHAE define a mapping to a family of Quantum-assisted Generator networks (Sec. 4.1) which are called depending on the input to the network. Hence, training of the HTHAE implicitly consists of hybrid training a family of QaGs and a correct mapping of input to one or a distribution of QaGs.

5.2.2 Training

The Hybrid-trained Hamiltonian-based Autoencoder is trained by tuning all parameters $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}, \mathbf{h}^z, \mathbf{h}^x, \mathbf{J}\}$ of the encoder, the decoder and the Quantum Boltzmann Machine in the bottleneck layer with the aim to minimize the *mean square error* (MS error). The MS error

$$\mathcal{C} = \sum_D \frac{1}{2} \sum_i \frac{(x_i - v_i)^2}{N} \tag{5.9}$$

provides a reconstruction error between the input $\mathbf{x}$ and the output $\mathbf{v}$. The cost-function for the HTHAE is thus the same as for the HAE in Sec. 5.1. The difference between the models is how the cost-function is minimized. The parameters of the QBM as well as the parameters of the adjacent layers are trained with the *cma-es* optimization algorithm (Sec. 2.4). Additional layers in the classical network with the respective parameters are trained by gradient descent. This is motivated by Sec. 4.2 where we show the benefits of reducing the amount of parameters tuned by the optimizer and implementing a hybrid training method. The results imply that hybrid training with classical gradients wherever possible might not only be helpful for training success but is very likely required for larger quantum-assisted networks.

A schematic representation of the hybrid training approach can be seen in Fig. 5.2.2. The gradient of the cost function with respect to deeper network parameters can be calculated via *backpropagation* (see Sec. 1.2.2). This is done from layer (-0) to layer $(-k+1)$. As with a QBM with non-diagonal Hamiltonian, calculation of the cost function gradient with respect to the Hamiltonian parameters is in general unfeasible [6]. The error can thus not be propagated through the quantum layer to the previous classical layers. This is the exact reason why the network parameters are shared between the encoder and the decoder network as a first approximation. It is essentially required to effectively train the HTHAE with the hybrid training method. The weights of the encoder are consequently indirectly trained by the gradients that were calculated in the decoder up to the qubit layer. The gradient contribution of the encoder to the shared weights is systematically. In purely classical network simulations, which were implemented with an analogue architecture

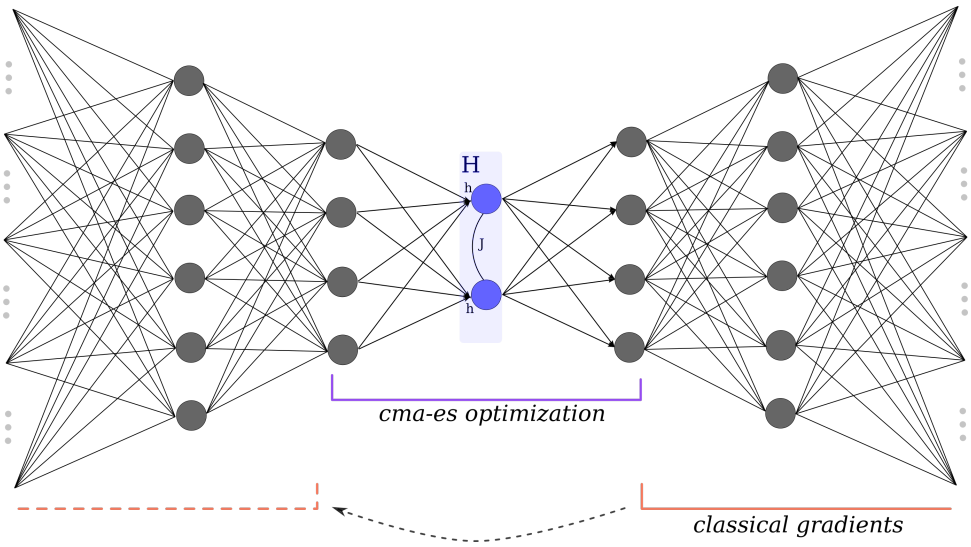

Figure 5.2.2: Schematic representation of the hybrid training of the HTHAE. The inner layers of the network, including the qubit layer that implements a Quantum Boltzmann Machine, are trained with the *cma-es* optimizer while the remaining parameters of the decoder are trained with classical gradient descent and backpropagation. The gradient of the decoder cannot be propagated back through the qubit layer such that the encoder which shared weights with the decoder only receives an implicit gradient update.

similar to the quantum case, it was observed that while the gradients of the encoder help convergence, they might not be strictly needed.

Even though the weights $\mathbf{W}^{(k)}$ that lead out of the qubit layer and the biases $\mathbf{b}^{(k-1)}$ that belong to the next adjacent layer could be trained with gradients, those additional parameters offer vital flexibility for the *cma-es* optimizer. There are only few Hamiltonian parameters relative to all model parameters and the quantum state strongly depends on the local field offset $\tilde{h}^z$ which directly depend on $\mathbf{W}^{(k)T}$. In practice, it has proven to be necessary for the training of a HTHAE to not train these parameters with gradient descent but instead with the optimizer.

The hybrid training of a HTHAE with several deep layers is fragile and requires various considerations:
First, it is important that the learning rate of the gradient descent and the step size of the *cma-es* optimizer are in some sort of balance for the model to converge. Commonly, learning rates in deep layers of an Artificial Neural Network (ANN) are the same or sometimes smaller than the learning rate of outer layers. The reason for that being that changes in deeper layers affect the activations in all subsequent layers and thus have a larger response on the outcome. In our case, the *cma-es* optimizer's step size is initiated to a significantly larger value than the learning rate for the outer layers. In practice, this ensures a more robust training because the gradient-free optimizer adapts more flexibly to changes in the optimization land-

scape. Also, the information flow in the hybrid training goes one way: The best optimizer evaluations determine the gradients used. Strong gradients will merely narrow down the space of solutions that are explorable by the optimizer.

A second important factor in training a HTHAE is the use of the leaky ReLU activation function in all intermediate layers of the model. The ReLU activation function has helped solving the so-called *vanishing gradient problem* in deep networks [47] which is very typical for the sigmoid activation function. It occurs when the parameter constellation of an ANN is such that the layer activations fall in a range where the derivative of the activation function is very small. Backpropagating a small error gradient further reduces the absolute value of the gradient for every layer until it practically vanishes. The piece-wise constant derivative of the leaky ReLU does not have this problem and offers a more reliable interplay with the *cma-es* optimization algorithm. The vanishing gradient problem in the HTHAE generally causes the model to converge to average solutions where each input results in the same output.
Additionally, the 'leaky' property of the leaky ReLU, i.e. setting negative inputs to small but non-zero values, makes it generally more robust in training. The hybrid training relies on a dynamic interplay between the different optimization procedures and benefits from nodes not outputting zeros after a certain breakpoint is reached.

Finally, to initialize the HTHAE, the Hamiltonian parameters and the network weights are set to small values e.g. chosen randomly in the interval $\in (-0.1, 0.1)$, as well as the biases to 0.1. This will result in mostly positive initial activations in all layers which receive strong gradients from the leaky ReLU and are close to the non-linear region around zero. The Hamiltonian parameters should be small enough such that initial $\tilde{h}^z$ offsets significantly impact the distribution of the Quantum Boltzmann Machine in the qubit layer.

5.2.3 Training Analysis

To understand the training procedure of the Hybrid-trained Hamiltonian-based Autoencoder, we train a 196c-120c-20c-3q model on the first three binary 14x14 pixel samples from the MNIST training data set (see Sec. 2.5 for details) and track several metrics. Besides the mean square error as cost-function, we observe the values of the local field offsets $\tilde{h}^z$ relative to the parameters of the foundation Hamiltonian H. Their proportion indicate how strongly the quantum sample depends on the local fields. Additionally, we measure the spectral energy gaps between the eigenstates of the offset Hamiltonian $\tilde{H}$ and the groundstate percentage of the Boltzmann distribution. As for three training samples there is three $\tilde{h}^z$ and thus three $\tilde{H}$ in each training iteration, we representatively only observe the measures connected to the first training sample.

The results in Fig. 5.2.3 for one training run of the HTHAE unveil remarkable

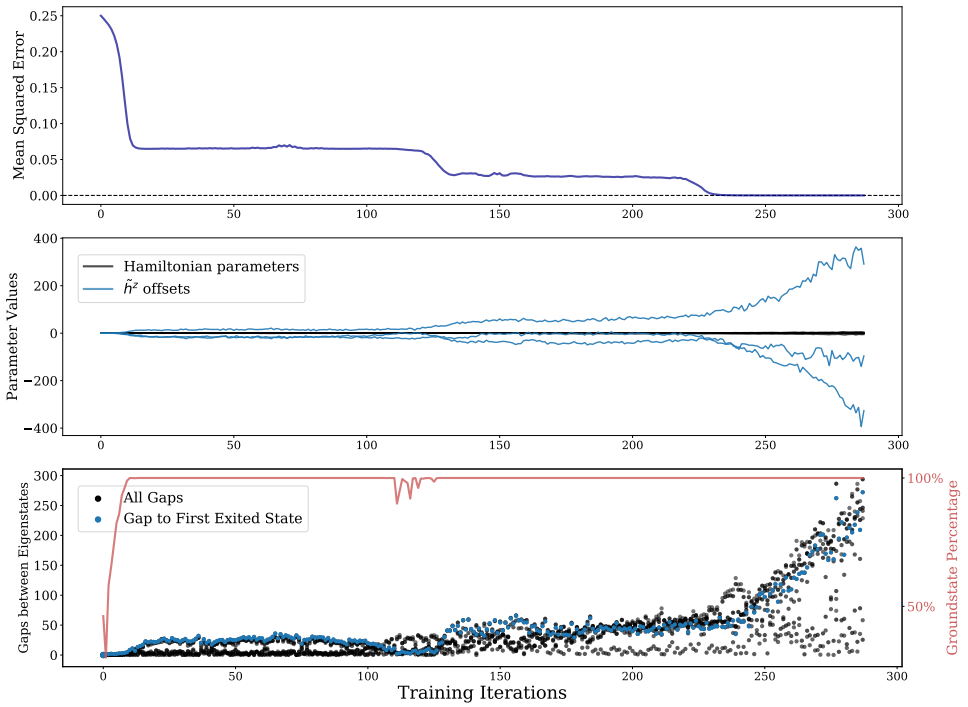

Figure 5.2.3: Detailed visualization of important quantities in a 196c-120c-20c-3q HTHAE during training on three binary 14x14 MNIST samples. Observed are the MS error, the size of the input dependent $\tilde{h}^z$ offsets relative to the parameters of the foundation Hamiltonian H, the groundstate percentage of the Boltzmann ensemble and the gaps between the eigenstates in the Hamiltonian with added offset. The $\tilde{h}^z$ offsets become overwhelmingly dominant compared to the other Hamiltonian parameters while the model is converging. The system also is mostly in the groundstate with the distance to the first exited state increasing drastically at the end. These results indicate that the HTHAE tends towards a deterministic classical mapping between an input and a quantum sample because the $\tilde{h}^z$ offsets fully determine the produced quantum sample.

insight into how the model operates. The MS error shows a first plateau at 0.065 which exactly corresponds to the pixel-wise average of all three training samples. The second plateau at 0.0264 is the average between the first two training samples where the third is distinctly learned. The plateauing behavior is a very typical response in the training of a HTHAE. The classical gradients quickly optimize the samples generated by the untrained QBM, i.e. it minimized the MS error on average for all possible QBM samples. A specific constellation and magnitude of $\tilde{h}^z = \mathbf{W}^T\mathbf{u}$ offset will dissolve the average of certain samples and improve the MS error. When the model has successfully converged to distinct outputs for each input, the offset parameters rapidly grow and fully determine the quantum samples through $s_i = \frac{\text{sign}(\tilde{h}_i^z)+1}{2}$. This results in a product state of the qubits where if h_i^z is large and positive, qubit i will always be 1, and vice-versa. The effect of the model's tendency to converge to these classical solutions is of course over-exemplified for

only three training samples where the model can overfit drastically.

The groundstate percentage shows that the model is predominantly in the groundstate with some fluctuations before converging to a better solution. Certainly, the degrees of freedom provided by additional eigenstates increase the ability to surpass training plateaus but they seem mostly unused in this case. This measurement is performed with a non-diagonal Hamiltonian in the quantum layer but interestingly, the results are qualitatively equivalent for a model with diagonal Hamiltonian. Intuitively, one might expect a wider Boltzmann distribution over several eigenstates because individual eigenstates only contain exactly one sample. Instead, the model seems to mostly depend on the $\tilde{h}^z$ parameters to deterministically set the quantum samples. The findings are coherent with the 7 gaps between the 8 eigenstates in the offset Hamiltonian. The groundstate more strongly seperates itself for every better solution that the model finds. This indicates that the training of the HTHAE will tend towards a fully classical model with a deterministic mapping between input to- and output of the qubit layer. For larger and more complicated training sets, this may not be the general expected behavior of the model and is subject to further study.

An interesting variation of the HTHAE that might change the behavior observed in this section is to implement the qubit layer as a QBM with $\sigma_i^x \sigma_j^x$ term. This is motivated by Sec. 3.2.5 where a QBM implemented on the qiskit *Qasm simulator* shows considerably more reliable and quantum-like behavior with the coupling term in transverse direction. It is possible that such a Hamiltonian would for a HTHAE model to learn representations of the data that are more distributed and not fully determined by the offset in computational basis.

5.2.4 Learning the Full MNIST Data Set

In this section, the Hybrid-trained Hamiltonian-based Autoencoder is trained with the full 28x28 binary MNIST data set. The MNIST data set contains 60.000 training samples and is a standard training set for testing and benchmarking Machine Learning algorithms. See Sec. 2.5 for details on the MNIST data set. The architecture is 785c-1000c-50c-6q with 6 qubits implementing a Quantum Boltzmann Machine in the bottleneck layer. The model is trained with a batch size of 5 for 24.000 iterations. This means that the model is performing stochastic training with 5 MNIST samples in the training set for a given iteration. The choice for the batch size is generally a trade-off between computational time per iteration, memory requirements and training performance. For this simulation, a batch size of 5 is remarkably small and can certainly be increased. Note, that the input-dependent QBM needs to be evaluated for every training sample in every iteration.
During training, the learning rate of the classical gradients was manually adapted from $\eta = 10^{-2}$ to $\eta = 10^{-4}$.

With 6 qubits in the latent space, the model can output $2^6 = 64$ different samples. Clearly, not all 60.000 training samples can be learned accurately to minimize the mean square error. For ten different hand written digits '0' - '9', there can be an average of six to seven variations of each digit. Some handwritten digits vary more in their style than others, e.g. the digit '4' which is commonly written in two topologically different ways, and some are more complex than other, e.g. compare the simple digit '1' to an '8'. This will significantly impact how many different quantum samples are assigned to a specific digit. The model has to learn to minimize the reconstruction error on average and distribute its resources accordingly. Realistically, not all 64 quantum samples will result in a meaningful output which may be an artifact of incomplete training but also partly because of the shared-weights of the Autoencoder are less flexible to address each different quantum sample and also reconstruct each sample to a good output.

Fig. 5.2.4 shows the final results of the trained Hybrid-trained Hamiltonian-based Autoencoder when inputting the first 24 images of the MNIST test data set. The test data samples are unknown to the model and have never been input before. The outputs are the results of upsampling the most likely quantum sample in the respective input-dependent QBM.
The average mean square error for all training and test data respectively are

$$\begin{aligned} \text{MS-Error(training)} &= 0.046 \pm 0.019 \\ \text{MS-Error(test)} &= 0.047 \pm 0.018 \end{aligned} \tag{5.10}$$

Remarkably, the model performs equally well on the training and the test set. It has very few degrees of freedom in the qubit bottleneck layer compared to the total number of 60.000 training samples which is why all learned features need to be as generally applicable to MNIST data samples as possible. It is also apparent in Fig. 5.2.4 that the HTHAE is able to learn significant features of the training data which generalize to new data. One can see the compromise the model makes for different inputs. The digits '7' and '9' often output the same group of samples, whereas '0' and '1' are very distinct compared to other digits.
In this case, it may not be very surprising that the model generalizes well to unseen data. Common practices to prevent *overfitting* (see Sec. 1.2.2) make a model weaker so that it can't perfectly fit each input training sample. With 6 qubits and 64 possible outcomes, the HTHAE is put in a situation where it must learn the broadest generalization of numbers in order to reduce the cost-function.

A more intuitive score on the test data is provided in Table 5.1 which shows manually-counted subjective judgments of whether or not a test sample input was successfully reconstructed. The judgement not only takes into account the quality of the most likely samples, but also their confidence and the quality of the second most likely sample. As the score is subjective and only takes into account the first 100 test samples of the MNIST data set, it is not to be understood as objective

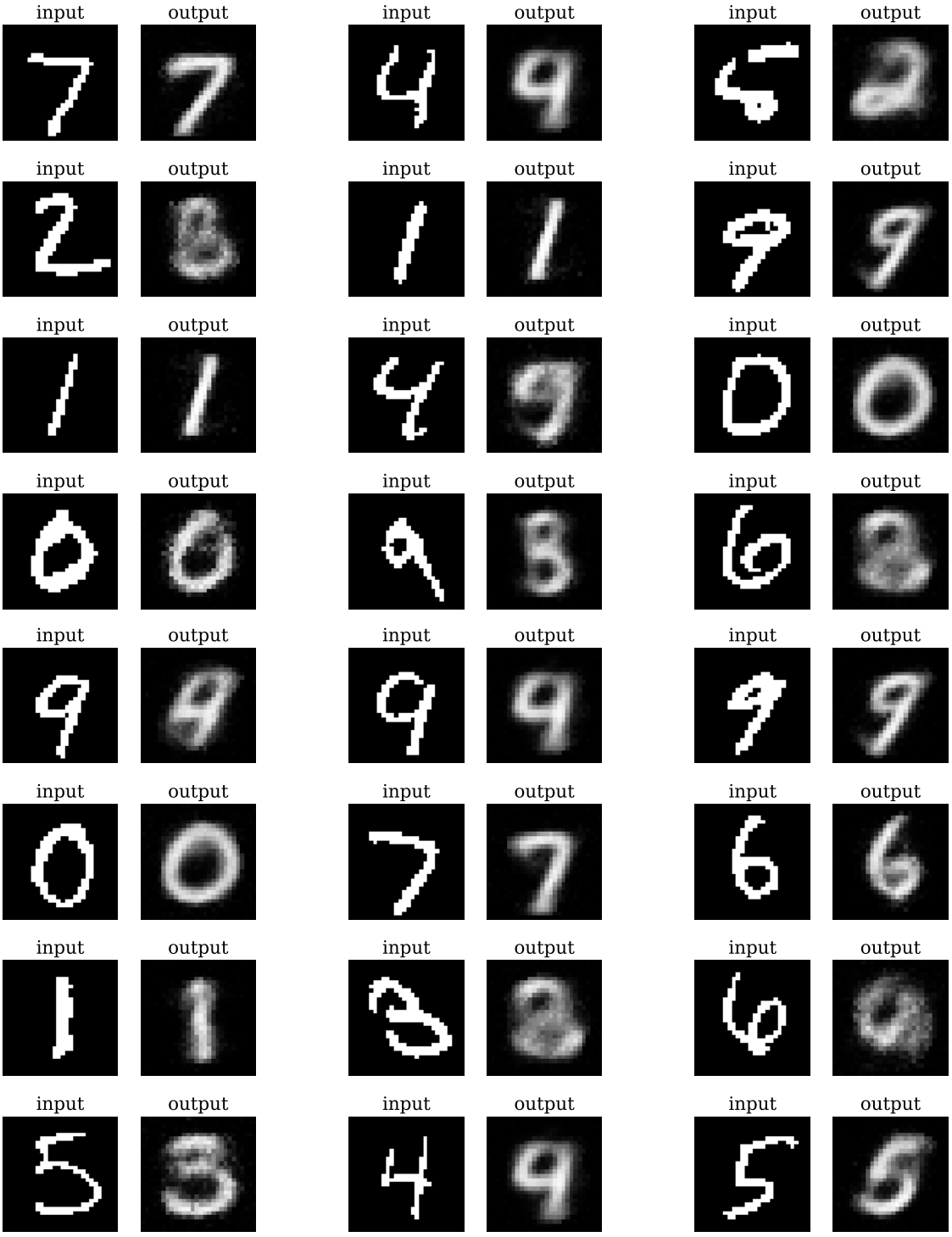

Figure 5.2.4: Output of a trained HTHAE model for 24 previously unseen samples of the MNIST test data set. The model was trained for 24.000 iterations on the full MNIST training data set and a batch size of 5. Outputs are the result of upsampling the most common quantum sample in response to a given input.

quantitative score. An interested reader may take a closer look at Fig. 5.2.4 and judge the first 24 test samples by themselves.

Correct	Unclear / Mix	Wrong
59%	15%	26%

Table 5.1: Manually-counted subjective score of the trained HTHAE when inputting the first 100 samples of the MNIST test data set. Those test samples are previously unknown to the model. It was trained on the full MNIST training data set with a batch size of 5 and 24.000 training iterations. The score takes into account the quality of the most common output of the Autoencoder for a given input, its confidence, and the quality of the second most common output.

Many times, the confidence of the most likely sample is above 90% or exactly 100%. This indicates that the model tends to learn deterministic mappings between input and binary quantum samples $\mathbf{s}$, as shown by the results in Sec. 5.2.3. Still, that is not the case for all samples and one might learn more about the data set by studying the non-determinative distribution in the latent space. This is a feature that is qualitatively different to conventional implementations of classical Autoencoders and warrants further study.

5.2.5 Benchmarking a Potential Quantum Advantage

In this section, we test a potential quantum advantage in the Hybrid-trained Hamiltonian-based Autoencoder with 100c-60c-20c-3q network architecture. A HTHAE model with diagonal Hamiltonian acts as classical reference for a HTHAE with non-diagonal Hamiltonian which is considered quantum. For the benchmarking, both models are trained on different training data sets, random training sets and a 10x10 MNIST training sets. For both types of training sets, we train the models on binary and continuous-valued versions of the data in order to test for certain strengths and weaknesses. After training, the final mean square error on the training sets is evaluated and calculated in to a fitness difference between the classical and quantum model. The fitness difference is the negative difference between the MS errors of the models. A positive difference indicates a quantum advantage where the quantum model with non-diagonal Hamiltonian was able to converge to a solution with overall lower MS error. The number of training iterations is 800 where in practice the models have shown to have either converged to a good solution of plateaued for a long time.

The measurements are performed on training sets of different sizes, i.e. different number of samples in each training set. The size of the training set is an important external parameters in the benchmarking because of two reasons. First, it shows how many samples a model can distinctly learn such that each input sample results in a different reconstructed output. Second, quantifies how well a model utilizes its resources to average similar inputs the the same output if it is not able to distinctly

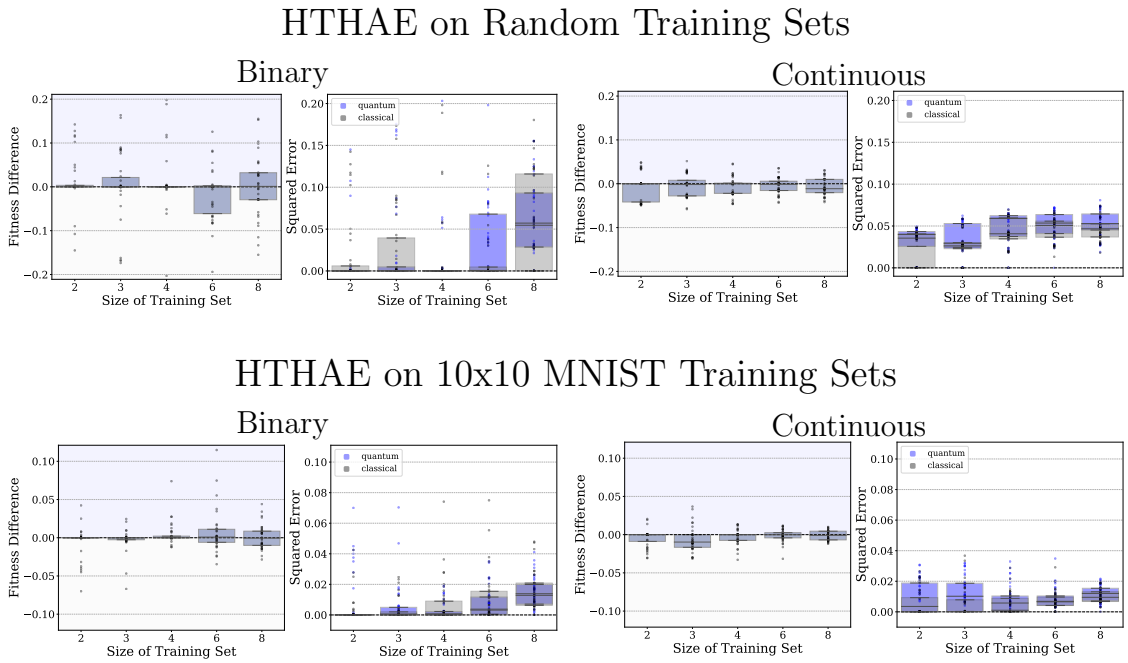

Figure 5.2.5: Benchmarking results for a potential quantum advantage in a 100c-60c-20c-3q HTHAE. A quantum advantage is defined as a smaller mean square error when comparing a HTHAE with non-diagonal Hamiltonian (blue) to a HTHAE with diagonal Hamiltonian (grey). The diagonal model is taken as classical reference while the non-diagonal is considered quantum. Quantum and classical models are trained of random training sets and 10x10 MNIST training sets of different sizes, i.e. increasing number of training samples. Their error after training is calculated into a fitness difference. A positive fitness difference (blue shaded area) indicates a quantum advantage. 40 individual runs are marked as dots while the 25%-75% percentiles are shown as boxes.

reconstruct them. This is especially relevant for the MNIST data set which has systematic structure and may contain more than one instance of the same digit. Because the qubit layer and the input-dependent Quantum Boltzmann Machine need to be evaluated for every training sample in every training iteration, the sizes of training sets in this section are limited by computational time. Consequently, the number of training samples do not exceed the possible number of quantum samples, though in practice, not all quantum samples will contribute in the trained model.

The benchmarking results for the HTHAE in Fig. 5.2.5 show that the classical reference model with diagonal Hamiltonian generally outperforms the quantum model with non-diagonal Hamiltonian on continuous-valued data. For binary data, the results are indecisive. Interestingly, the results in Sec. 5.1.3 for the Hamiltonian-based Autoencoder are qualitatively the same. This indicates that the results are not directly caused by the hybrid training approach of the HTHAE.
It is not clear why the quantum model performs worse on continuous data than the classical equivalent. One of the key properties of the HTHAE network is the $\tilde{h}^z$ offset

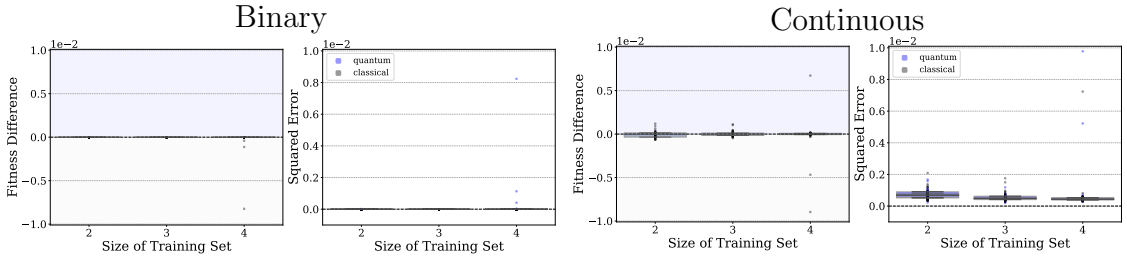

Figure 5.2.6: Benchmarking results for a potential quantum advantage in a 784c-500c-50c-3q HTHAE. A HTHAE with diagonal Hamiltonian and one HTHAE with non-diagonal Hamiltonian are trained on binary (left) and continuous (right) 28x28 MNIST training sets of different sizes, i.e. increasing number of training samples. Both models always learn very good reconstructions of the samples which have spatially well resolved features.

to the local fields in the Hamiltonian. The offset is strictly in the computational z-basis and therefore classical. It is possible to assume that these classical constraints provided by the local field offsets don't naturally promote quantum effects such as superposition and entanglement in the eigenstates of the Hamiltonian. Also, for the task of learning reconstruction through overfitting, which is the case in this section with such small training sets, superposition might even prove to be a drawback. It is not directly beneficial for the minimization of the cost to produce more than one sample given that one good sample is optimal for the MS error. Conversely, if instead the offset is in transverse direction with $\tilde{h}^x$, the model in practice has shown to never converge to good solutions because the decoder network would need to upsample each quantum sample that is part of the superposition into the same reconstructed output.
A promising alternative to the transverse field offset $\tilde{h}^x$ is to implement a different non-diagonal Hamiltonian for the QBM, e.g. one with $\sigma_i^x \sigma_j^x$ term. As shown in Sec. 3.2.5, it might improve the training performance of the QBM.

These observations warrant interesting experimental setups which are not shown in this work. Before all, testing generalization of the models by measuring the final square error not on the training set but on previously unseen test sets. This is especially possible on the MNIST data set.
Additional factors dismissed in this work are for example robustness to noise and input of systematically different data. One would generally like for the model not to create sharp output with 100% confidence if the input is very noise or follows are completely different structure than the training data. For practical application in moder data analyis, those are important properties and need to be considered.

Note, that equivalent benchmarking protocol is also performed on full-size 28x28 MNIST training sets. The results are shown in Fig. 5.2.6. Both classical and quantum model are able to learn the large training samples significantly better and fully

converge almost always. This indicates that the benchmarking of a potential quantum advantage as presented here is insufficient for training data where structural features are considerably better resolved. For better a better comparison between the classical and the quantum model, the number of training samples per training set needs to be increased to 6-8 with an interesting outlook to even more training samples.

5.3 The Gate-based Autoencoder

The *Gate-based Autoencoder* (GAE) is a quantum-assisted Autoencoder algorithm that implements qubits in the bottleneck layer of the network. Unlike the Hamiltonian-based Autoencoders described in this chapter, the behaviour of the qubit layer of the GAE is governed by quantum circuits consisting of parametrized gates. Inputs to the GAE are propagated towards the qubit layer and non-linearly calculated into single-qubit rotation angles. The rotated qubit state is thus fully parametrized by the encoder network of the GAE and can be measured to create quantum samples. The quantum samples are then upsampled to output a high-quality reconstruction of the original input. The code implementation of the GAE in this section can be found on github [51][52].

5.3.1 Model Description

The model architecture of the GAE can be seen in Fig. 5.3.1. It is structurally very similar to the Hybrid-trained Hamiltonian-based Autoencoder in Sec. 5.2 and consists of an encoder network, a qubit layer in the middle bottleneck layer, and a decoder network. The essential difference to the Hamiltonian-based quantum-assisted Machine Learning algorithms in this work lies in the qubit layer and the input-dependency of the qubit states. The qubit states are initialized in the $|0\rangle^{\otimes n}$ state and then rotated by parametrized single-qubit quantum gates. The parameterized quantum circuit ansatz employed in this algorithm is very natural for gate-based quantum computers. With the complications shown in Sec. 3.2.5 with implementing Hamiltonian-based Quantum Machine Learning algorithms on gate-based quantum devices, this may be the algorithm of choice when implementing a quantum-assisted Autoencoder. This is especially true when considering that one does not have to indirectly find Hamiltonian eigenstates by using the Variational Quantum Eigensolver (Sec. 2.2) but can instead directly implement the relevant operations on the qubits themselves.

The rotation angles $\boldsymbol{\theta}$ for the single-qubit rotations in the qubit layer are calculated as a non-linear transformation on the output of the encoder network which is given by

$$\boldsymbol{\theta} = \text{sig}(\mathbf{W}\mathbf{u} + \mathbf{b}) \cdot \pi \tag{5.11}$$

where $\mathbf{u}$ are the activations of the previous layer given an input $\mathbf{x}$ to the model, $\mathbf{W}$ are the weights connecting the previous layer and the qubit layer and $\mathbf{b}$ biases for each qubit. The activation function is the sigmoid activation function sig which has values in the range of $(0, 1)$. Those values are transformed to angles in a range of $\boldsymbol{\theta} \in (0, \pi)$ to connect the tails of the sigmoid function to the qubit states of 0 and 1. A change in the network parameters before the qubit layer therefore results in a change of the θ rotation angle. The parametrized quantum circuit on the initial qubit state essentially serves the purpose of separating the different input data in

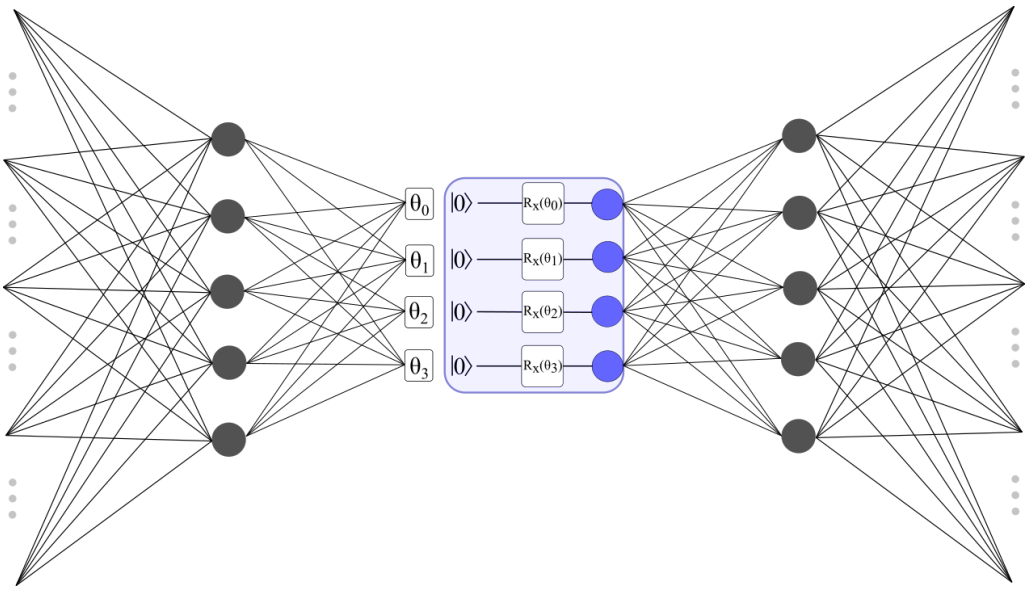

Figure 5.3.1: Network architecture of the Gate-based Autoencoder (GAE). The GAE is a quantum-assisted Autoencoder which implements qubits in the bottleneck layer of the network. Given an input to the network, the output of the encoder is calculated into single-qubit rotations. The input-dependent qubit state is then measured and upsampled by the decoder.

the exponentially growing Hilbertspace. Hopefully, the model can learn to keep data with similar features closer together than distinctly different data.
The angle range of $\boldsymbol{\theta} \in (0, \pi)$ is somewhat arbitrarily chosen an can be changed. Note, that for an example range of $\boldsymbol{\theta} \in (0, 2\pi)$, $s_i = 0$ is located at both tails and $s_i = 1$ and the point of strongest gradient of the sigmoid function. This is an asymmetry we do not currently want to employ.

The matrix exponential which describes the single-qubit gates applied on the initial qubit state is

$$|\psi\rangle = \prod_i e^{-i\theta_i \hat{\sigma}_i^x} |0\rangle_i. \tag{5.12}$$

The rotations are X-rotations, as indicated by the $\hat{\sigma}^x$ Pauli operator. There is no interaction between the qubits and the resulting wavefunction is a strict product state.
It is possible to employ a more complicated parametrized state preparation with entangling operations similar to the state preparation of the Direct Variational Generator in Sec. 3.1. Entanglement between individual qubits might enhance the performance of the GAE but this is not implemented in this work.

The encoder and the decoder networks which perform the downsampling of an input $\mathbf{x}$ toward the qubit layer and the subsequent upsampling of quantum samples to

produce an output $\mathbf{v}$ respectively are equivalent to the Hamiltonian-based Autoencoders described in this work. For detailed explanation of the network architecture, see Secs. 5.1 & 5.2.

5.3.2 Training

The training of the Gate-based Autoencoder consists tuning the parameters of the GAE in order to minimize the reconstruction error between an input $\mathbf{x}$ and the output $\mathbf{v}$of the model. The reconstruction error is measured with the mean square error cost-function (see. Eq (5.9)).
One has the choice of the number of layers in the encoder and decoder networks. The GAE studied in this work consists of five layer, two layers in the encoder, one qubit layer, and two layers in the decoder. Following the results in Sec. 4.2, we perform the hybrid training method on the model. Consequently, we employ shared-weights in the encoder and deconder networks. The parameters optimized by the *cma-es* opimizer are the biases of the qubits and the weights of the adjacent layers (see Eq. (5.11)). The remaining outer network parameters are trained with gradients descent. See Sec. 5.2.2 for more details on the hybrid training of deep quantum-assisted Autoencoders.

5.3.3 First Training Results

To provide prove of principle of the Gate-based Autoencoder, we train a GAE with a network architecture of 784c-120c-3q and 3 qubits on a subset of the 28x28 continuous-valued MNIST data set (see Sec. 2.5 for information on the MNIST data set). Selected for the training set are the first four '4's, four '0's and four '3' of the MNIST training data. The number of training samples is therefore larger than the $2^3 = 8$ states available in the qubit layer. It is to show that this GAE can learn an efficient representation of the training set by pairing different training samples to the same output.

Fig. 5.3.2 shows the final results of the trained GAE model after 2.000 training iterations when inputting the 12 training samples. The GAE is visibly capable of learning good reconstructions of the training samples. 6 out of 8 possible quantum samples are used to perform the reconstruction. Interestingly, the model has learned that samples with the digit '4' vary the most and therefore require additional degrees of freedom.
To show that the model has not overfitted on the training data, we test the model on the first examples of the respective digits in the MNIST test data which are previously unseen. Fig. 5.3.3 shows the performance the test samples. The model seems to generalize quite well but it is apparent that it has not been trained on enough variations of the digits. Especially the fourth '0', which is very different to the other examples, results in a completely untrained sample.
Given that the GAE as explained in this section implements a product state ansatz

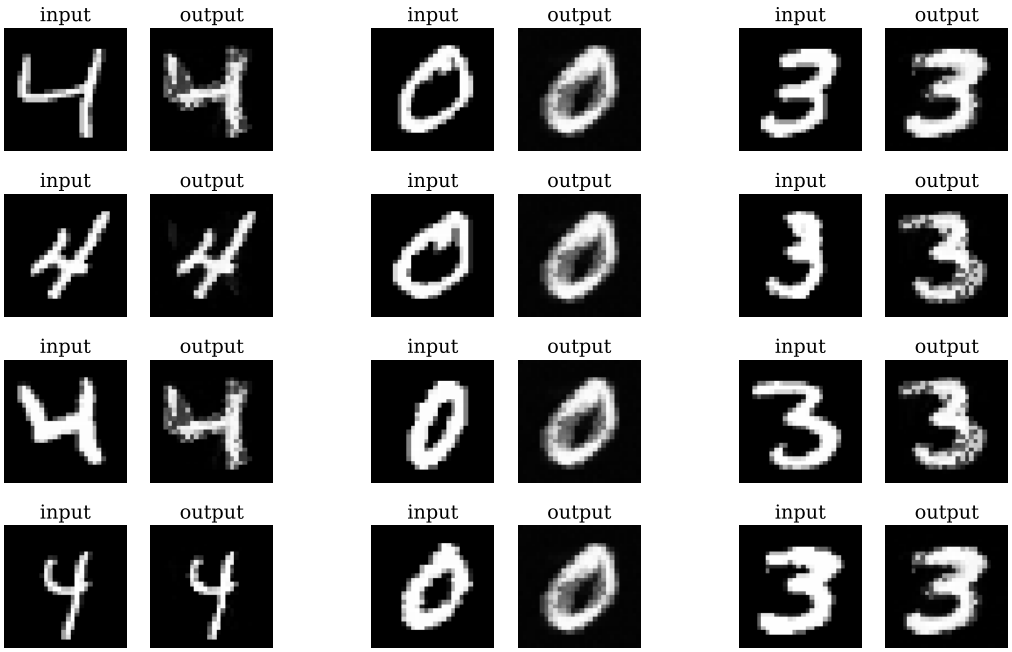

Figure 5.3.2: Demonstration of a 784c-120c-3q GAE after successful training on 12 selected 28x28 MNIST data samples. Inputs are the same 12 MNIST images that were used to train the model. Outputs are the result of upsampling the most common quantum sample in response to a given input.

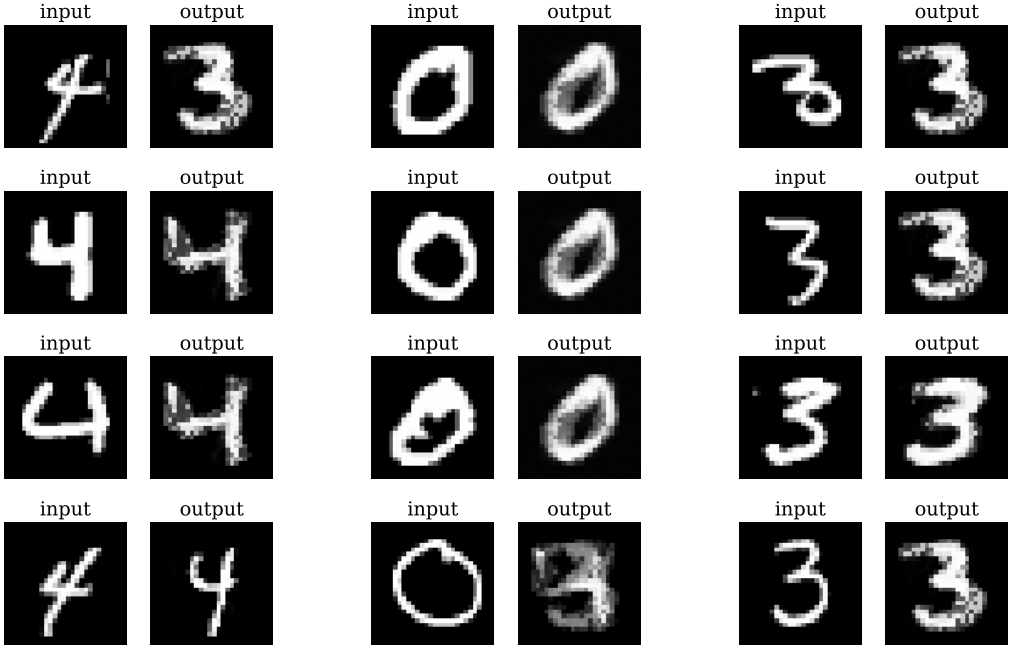

Figure 5.3.3: Demonstration of a 784c-120c-3q GAE after successful training on 12 selected 28x28 MNIST data samples. Inputs are 12 previously unseen MNIST data samples. Outputs are the result of upsampling the most common quantum sample in response to a given input.

for the qubit layer, the results are surprisingly good. Sec. 5.2.3 shows that also the Hamiltonian-based Autoencoder tends towards a classical mapping between input and output where interactions in the quantum system are no longer relevant. It is remarkable that interactions are also not strictly needed in training the GAE and to achieve first good results on MNIST data.

5.3.4 Learning the Full MNIST Data Set

In order to better evaluate the learning and generalization capabilities of the Gate-based Autoencoder, the model is trained on the full-size 28x28 binary MNIST data set. The quantum-assisted network has an architecture of 784c-1000c-50c-6q with 6 qubits in the bottleneck layer. The GAE is trained for 24.000 training iterations with a batch size of 5. These are the same hyperparameters as for the Hybrid-trained Hamiltonian-based Autoencoder in Sec. 5.2.4 to allow for a better comparison between the algorithms. The learning rate for gradient descent on the outer classical layers is kept constant at a $\eta = 10^{-3}$.

Fig. 5.3.4 shows the final performance of the Gate-based Autoencoder on the first 12 samples of the previously unseen MNIST test data set. It is apparent that the model does not succeed in learning the general structure of MNIST data samples that would allow it to perform high-quality reconstruction on test samples. In fact, the GAE performs equally on the training set. The mean square error on the entire training and test data respectively are

$$\begin{aligned} \text{MS-Error(training)} &= 0.083 \pm 0.024 \\ \text{MS-Error(test)} &= 0.081 \pm 0.024 \end{aligned} \tag{5.13}$$

Compared to the results in Eq. (5.10) for the HTHAE, the GAE performs considerably worse. One explanation that suggests itself is that the qubit layer contains no interaction between the qubits. The ansatz for the GAE as explained in this section is to only apply parametrized single-qubit rotations.
A natural extension for the Gate-based Autoencoder is to implement single-qubit rotations together with entangling operations. It is to be studied if interactions improve the performance of the model significantly and if it it is beneficial to implement a more involved state preparation ansatz like for the DVG in Sec. 3.1 with consecutive cycles of single-qubit rotations and entangling gates. To test if interactions between the qubits or entanglement are the leading cause for improved performance in the quantum-assisted Autoencoders proposed in this work, the same experimental setup could be performed with the HTHAE with diagonal Hamiltonian. This model implements interaction between qubits but no entanglement as the Quantum Boltzmann Machine with diagonal Hamiltonian is equivalent to a classical Boltzmann Machine [6].

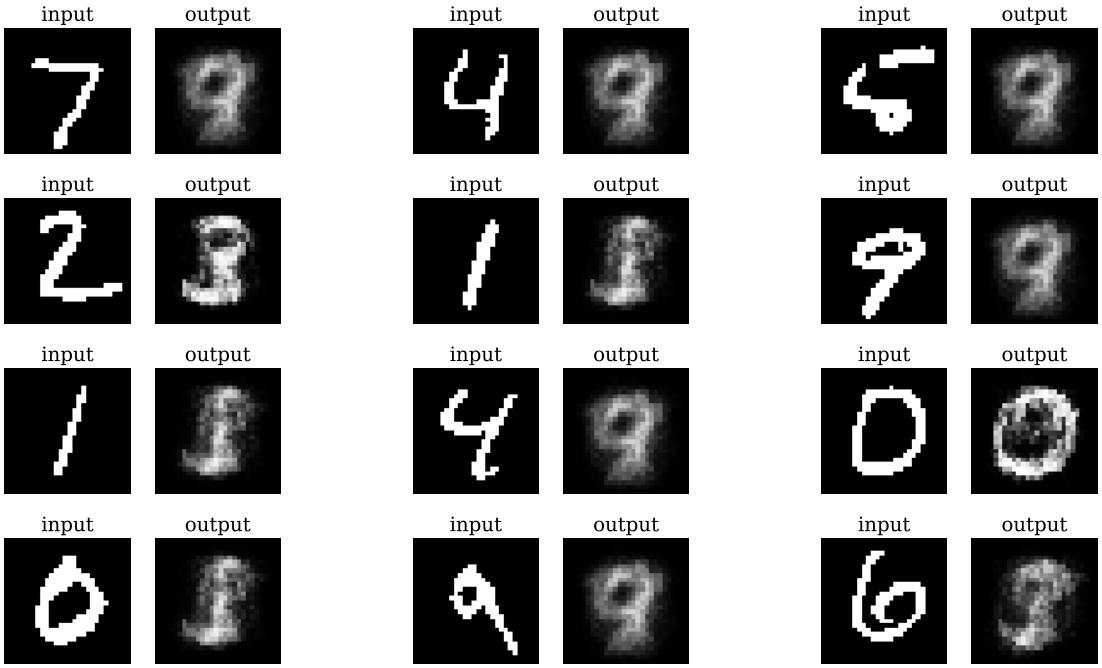

Figure 5.3.4: Demonstration of the GAE results after 24.000 training iterations on the full MNIST data set and a batch size of 5. Inputs are the first 12 images of the MNIST test data set which are is previously unseen to the model. Outputs are the result of upsampling the most common quantum sample in response to a given input.

6 Conclusion & Outlook

The aim of this work was to explore practical implementations of Quantum and Quantum-assisted Machine Learning algorithms and benchmark potential benefits of utilizing quantum phenomena in Quantum-assisted Neural Networks. It provided several examples of generative single qubit-layer algorithms as well as quantum-assisted generative and Autoencoder algorithms with qubit layers in essential locations of the networks.

Two known generative Quantum Machine Learning algorithms, the Direct Variational Generator (DVG) [9] and the Quantum Boltzmann Machine (QBM) [6] were discussed. The purpose of the DVG was to demonstrate the sampling advantages of generative algorithms that utilize qubits. The qubit wavefunction encoding the target probability distribution can be sampled efficiently by measuring the qubit state. In contrast to classical sampling where the sampling is either global but inefficient or efficient but local, the quantum sample is always a global sample of the true encoded distribution. We demonstrated the sampling advantage in the DVG compared to a Restricted Boltzmann Machine by training both models on the Bars-and-Stripes (BAS) training set and interpreting the sequence of the generated samples as a random walk. By deriving expected average behavior of that BAS random walk, we introduced a quantitative measure that not only considers the quality of the learned distribution but also the randomness of the sampling technique.

The Quantum Boltzmann Machine is the quantum extension of the classical Boltzmann Machine. It is based on a spin-Hamiltonian which can be chosen to be diagonal or non-diagonal. We showed that just the groundstate of a non-diagonal Hamiltonian can be trained to approximate random data distributions. This indicates that a full QBM framework, which includes a hierarchy of thermally exited states in the density matrix, might not be necessary in Hamiltonian-based models as long as quantum fluctuations are included. This indicates that in this situation quantum fluctuations and thermal fluctuations act similarly. When implementing a QBM on gate-based quantum computers with the Variational Quantum Eigensolver (VQE), we found that a Hamiltonian with σ_i^x transverse fields is not generally suited for the training of a quantum groundstate. The model may run into a degenerate solution space which sends an unreliable training signal. A Hamiltonian which includes an interaction term like $\sigma_i^x \sigma_j^x$ could resolve the problem in the example provided in this work though there are still conflicting solutions for that case. These results imply the necessity to consider the complexity of the achievable groundstate in order to attain satisfactory results and which quantum algorithm to implement on a given

quantum device.

The proposed Quantum-assisted Generator (QaG) allows for taking advantage of the efficient sampling properties of qubit systems in larger overall networks with additional classical layers while reducing the number of qubits required to put near-term quantum computing devices to practical use. The qubit layer implements a truncated QBM with a parametrized number of eigenstates. We find that for a diagonal QaG, the truncation parameter t may serve as an approximation parameter in learning a target distribution. Conversely, the non-diagonal QaG is agnostic to the truncation which implies that this Hamiltonian-based algorithm can be implemented on quantum devices that create mixed states or pure states.
In a systematic benchmarking of the Quantum-assisted Generator, we observed performance enhancing effects of implementing a quantum Hamiltonian as compared to a classical Hamiltonian. The quantum model on average converged to significantly better approximations of the training data compared to the classical model. Interestingly, the quantum advantage was maintained when just perfoming ground-state training on a quantum Hamiltonian. This indicates that quantum systems may more easily navigate complex cost-function landscapes in quantum-assisted algorithms. Further benchmarking with different training methods and models is required to evaluate whether quantum effects generally enhance the performance of generative Machine Learning algorithms.
To move towards practical application of a qubit quantum layer in a large generative network, we developed a hybrid-training method to train quantum-assisted neural network which cannot be efficiently trained with the conventional methods of gradient descent and backpropagation. The hybrid-training consists of a dynamic interplay of the gradient-free *cma-es* optimization algorithm on the parameters of the qubit layer and gradient descent on the remaining classical network parameters. We showed a significant improvement in convergence rate when using the hybrid-training method compared to a pure *cma-es* optimization approach.

The Quantum-assisted Autoencoder networks studied in this work utilize a qubit layer as the bottleneck layer of a Autoencoder architecture. The first Autoencoder presented was the Hamiltonian-based Autoencoder (HAE). As for the QaG, the qubit layer follows a QBM framework on a spin-Hamiltonian. The ansatz for making the qubit layer input dependent is to calculate the output of the encoder network of the Autoencoder into local field offsets in computational basis. A HAE with n qubits has 2^n possible outputs irrespective of the number of input samples and thus an interesting application in clustering arises where the HAE learns effective grouping of inputs to the network into the same output. We could not confirm an equivalent quantum advantage in the HAE as in the QaG. In fact, for continuous-valued training samples, the classical model with diagonal Hamiltonian outperformed the quantum model and generally achieved a better training performance.
The Hybrid-trained Hamiltonian-based Autoencoder is a scalable extension of the HAE with additional classical network layers. We showed that a HTHAE with

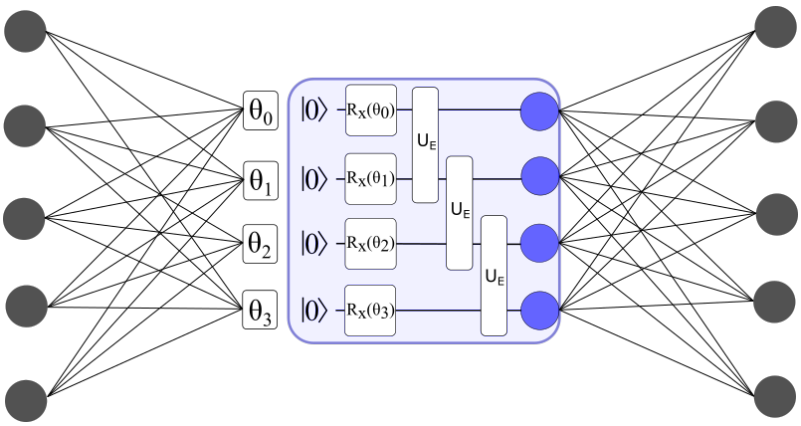

Figure 6.1: Schematic network architecture of a proposed Gate-based Autoencoder which implements interactions in the qubit layer. In the qubit layer, this algorithm utilizes input-dependent single-qubit rotations and entangling operations U_E between qubits to parametrize the quantum state in response to an input to the Autoencoder. The entangling operations here are depicted as two-qubit gates which is an arbitrary choice. Depending on the quantum device, they can be multi-qubit or global entangling operations.

non-diagonal Hamiltonian and only 6 qubits could be trained on the full MNIST training data set with promising results. The algorithm evidently makes use of grouping similar inputs into the same output and thus has good generalization capabilities. We also found that with the local field offset approach, the HTHAE has the tendency to learn deterministic classical mappings between inputs and quantum samples. Additionally, we showed the same lack of quantum advantage in the HTHAE as in the HAE. The results were indecisive for binary training data while the classical model outperformed on continuous data. It is plausible that these results are partly caused by the tendency of the HTHAE to learn classical mappings where the quantum model has additional unnecessary parameters that need to be tuned. An essential piece of understanding that has not been investigated is how the quantum model and the classical model group different inputs and how well they generalize. It is reasonable to assume that the classical model performs very well in the task of overfitting the training data and in that better than the quantum model. It has to be determined whether the quantum model, which here seems to be at a disadvantage in overfitting the training data, is better suited to learn efficient groupings of data and learn broader features.

A final quantum-assisted Autoencoder proposed in this work is the Gate-based Autoencoder (GAE). The GAE is an Autoencoder algorithm which applies input-dependent single-qubit gates in the qubit layer in order to separate inputs in the qubit Hilbertspace. It was trained with the hybrid-training method developed in this work which highlights the flexibility of the training approach. In this work, we used a product state ansatz for the quantum circuit as there were no interactions between the qubits. Even so, the GAE performed well on first training simulations on few

MNIST data samples. When training the model on the full MNIST training data set, we observe considerably inferior training results compared to the HTHAE. A straight-forward extension of the model is to include interactions between qubits in the quantum circuit of the Gate-based Autoencoder. A potential modified model is depicted in Fig. 6.1 which could be used in future work. In particular, the question whether classical interactions are sufficient or if quantum entanglement is necessary to effectively train the model on given data could be addressed by investigating different couplings U_E between the qubits. We think that valuable insights could from arise this which would help clarifying the role of true quantum effects introduced by quantum layers for future Quantum Machine Learning applications.

7 Bibliography

[1] S. Aaronson. *Read the fine print.* Nature 11, 291-293, 2015.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Héctor Abraham, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, et al. Qiskit: An open-source framework for quantum computing, 2019.

[4] A.Khoshama and W.Vinci. *Quantum Variational Autoencoder.* arXiv:1802.05779, 2019.

[5] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90:015002, Jan 2018.

[6] M.H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko. Quantum boltzmann machine, 2018.

[7] I. Arrazola, J. Pedernales, and L. Lamata et al. Digital-analog quantum simulation of spin models in trapped ions, 2016.

[8] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[9] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, and A. Perdomo-Ortiz. *A generative modeling approach for benchmarking and training shallow quantum circuits.* arXiv:1801.07686v3, 2019.

[10] M. Benedetti, J. Realpe-Gomez, and A. Perdomo. *Quantum-assisted Helmholtz machines: A quantum-classical deep learning framework for industrial datasets in near-term devices.* Quantum Science and Technology 3, 034007, 2018.

[11] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.

[12] Rupak Biswas, Zhang Jiang, Kostya Kechezhi, Sergey Knysh, Salvatore Mandrà, Bryan O'Gorman, Alejandro Perdomo-Ortiz, Andre Petukhov, John Realpe-Gómez, Eleanor Rieffel, Davide Venturelli, Fedir Vasko, and Zhihui

Wang. A nasa perspective on quantum computing: Opportunities and challenges, 2017.

[13] T. Blaschke, M. Olivecrona, O. Engkvist, J. Bajorath, and H. Chen. Application of generative autoencoder in de novo molecular design, 2017.

[14] Sergio Boixo, Troels F. Rønnow, Sergei V. Isakov, Zhihui Wang, David Wecker, Daniel A. Lidar, John M. Martinis, and Matthias Troyer. Evidence for quantum annealing with more than one hundred qubits. *Nature Physics*, 10(3):218–224, 2014.

[15] H.-J. Briegel, T. Calarco, D. Jaksch, J. I. Cirac, and P. Zoller. Quantum computing with neutral atoms. *Journal of Modern Optics*, 47(2-3):415–451, 2000.

[16] Iulia Buluta and Franco Nori. Quantum simulators. *Science*, 326(5949):108–111, 2009.

[17] Juan Carrasquilla, Giacomo Torlai, Roger G. Melko, and Leandro Aolita. Reconstructing quantum states with generative models. *Nature Machine Intelligence*, 1(3):155–161, 2019.

[18] E. Chen. restricted-boltzmann-machines. GitHub repository, https://github.com/echen/restricted-boltzmann-machines.

[19] W. Cottrell, B. Freivogel, D. M.Hofman, and S.F. Lokhande. *How to Build the Thermofield Double State*. arXiv:1811.11528, 2018.

[20] Dong-Ling Deng, Xiaopeng Li, and S. Das Sarma. Quantum entanglement in neural network states. *Phys. Rev. X*, 7:021021, May 2017.

[21] Alejandro Perdomo-Ortiz et al. Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers, 2018.

[22] P.J.J. O'Malley et al. Scalable quantum simulation of molecular energies, 2016.

[23] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.

[24] C. Figgatt, A. Ostrander, N. M. Linke, K. A. Landsman, D. Zhu, D. Maslov, and C. Monroe. Parallel entangling operations on a universal ion-trap quantum computer. *Nature*, 572(7769):368–372, 2019.

[25] L. Gondara. Medical image denoising using convolutional denoising autoencoders, 2016.

[26] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[27] D.J. Griffith and D.F. Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Presss Third Edition, 2018.

[28] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[29] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, feb 2019.

[30] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), Oct 2009.

[31] G. Hinton and T. Sejnowski. Learning and relearning in boltzmann machines. *Parallel Distributed Processing*, 1, 01 1986.

[32] G. E. Hinton* and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 7 2006.

[33] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets, 2017.

[34] D. Kielpinski, C. Monroe, and D. J. Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417(6890):709–711, 2002.

[35] B. Lamberta, Y. Katariya, M. Daoust, and Diego. Deep convolutional generative adversarial network. GitHub, Feb 2020.

[36] B. P. Lanyon, C. Hempel, D. Nigg, M. Müller, R. Gerritsma, F. Zähringer, P. Schindler, J. T. Barreiro, M. Rambach, G. Kirchmair, M. Hennrich, P. Zoller, R. Blatt, and C. F. Roos. Universal digital quantum simulation with trapped ions. *Science*, 334(6052):57–61, 2011.

[37] Ryan LaRose. Overview and Comparison of Gate Level Quantum Software Platforms. *Quantum*, 3:130, March 2019.

[38] J. Larsen and L. K. Hansen. Generalization performance of regularized neural network models. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 42–51, Sep. 1994.

[39] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist databas of handwritten digits.

[40] W. Lenz. Beiträge zum verständnis der magnetischen eigenschaften in festen körpern, 1920.

[41] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, 73(11):1551–1588, Nov 1985.

[42] M.Benedetti, J.Realpe-Gomez, and A.Perdomo-Ortiz. *Quantum-assisted Helmholtz machines: A quantum-classical deep learning framework for industrial datasets in near-term devices*. arXiv:1708.09784, 2018.

[43] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, May 2019.

[44] F. Mosteller. *Probability and Statistics*. Addison-Wesley Pub Co, First Edition, 1961.

[45] N. M. Nasrabadi and R. A. King. Image coding using vector quantization: a review. *IEEE Transactions on Communications*, 36(8):957–971, Aug 1988.

[46] J.v. Neumann. *Mathematische Grundlagen der Quantenmechanik*. Springer, 2011.

[47] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[48] S. Brierley O. Higgott, D. Wang. Variational quantum computation of exited states, 2019.

[49] Román Orús, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4:100028, 2019.

[50] J. Rocca. Understanding variational autoencoders (vae). 9 2019.

[51] M. Rudolph, F. Jendrzejewski, and S. Schmitt. KIP Quantum Machine Learning. GitHub repository, https://git.kip.uni-heidelberg.de/fnj/kipqml, 2020.

[52] M. Rudolph, F. Jendrzejewski, and S. Schmitt. Near-Term Quantum Machine Learning. GitHub repository, https://github.com/HRI-EU/NTQML, 2020.

[53] D. E. Rumelhart and J. L. McClelland. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, pages 194–281. MITP, 1987.

[54] Matthias Rupp, Raghunathan Ramakrishnan, and O. Anatole von Lilienfeld. Machine learning for quantum mechanical properties of atoms in molecules. *The Journal of Physical Chemistry Letters*, 6(16):3309–3313, Aug 2015.

[55] Sentewolf. Concept_of_directional_optimization_in_cma-es_algorithm, Oct 2008.

[56] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct 1997.

[57] Anders Sørensen and Klaus Mølmer. Spin-spin interaction and spin squeezing in an optical lattice. *Phys. Rev. Lett.*, 83:2274–2277, Sep 1999.

[58] J.L. Ticknor. A bayesian regularized artificial neural network for stock market forecasting. 40:5501–5506, 2013.

[59] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14(5):447–450, 2018.

[60] Hayato Ushijima-Mwesigwa, Christian F. A. Negre, and Susan M. Mniszewski. Graph partitioning using quantum annealing on the d-wave system. In *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*, PMES'17, page 22–29, New York, NY, USA, 2017. Association for Computing Machinery.

[61] P. Weinberg and M. Bukov. Quspin: a python package for dynamics and exact diagonalisation of quantum many body systems part i: spin chains, 2017.

[62] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and D. G. Cory. Hamiltonian learning and certification using quantum resources. *Phys. Rev. Lett.*, 112:190501, May 2014.

[63] Jaime Zabalza, Tong Qiao, Jiangbin Zheng, Huimin Zhao, Chunmei Qing, Zhijing Yang, and Stephen Marshall. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185, 12 2015.

[64] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. 2015.

Appendices

A Benchmarking a Quantum Advantage on a QBM

In this section, we test for a potential quantum advantage in training of a Quantum Boltzmann Machine. Here, a quantum advantage is defined as a superior training performance of a QBM with non-diagonal Hamiltonian as compared to a QBM with diagonal Hamiltonian. The diagonal model acts as classical reference while the non-diagonal model is considered quantum. Both models are given the same random binary training sets and are trained for 500 training epochs. At the end of training, the overlap of each model with the training set is calculated and compared. The experiment is repeated for a different number of random binary training samples in the training set and averaged over 40 repetitions.

Fig. A.1 shows that after 500 iterations and 40 repetitions, there is never a run where either QBM model converges to a solution that is more than $3 \cdot 10^{-6}$ different in overlap than the other model. There is no quantum advantage found in this setup and no difference at all between the models. Instead, both models always converge with high accuracy which implies that this task is too simple for potential quantum effects to make a difference. As the training data was random with no structure and arbitrary shape, it is unclear which kinds of data are challenging for a QBM. In this setup, there is no pre-processing of the classical data and no post-processing of the quantum samples which are essentially classical binary samples. A natural progression from this experiment is to introduce classical layers that perform non-linear transformations on the QBM samples. This will scale up the dimension of the output while keeping the number of qubits constant and hence increase the difficulty of the learning task. Not only is this more efficient in the number of qubits used but it offers further possibilities to study quantum advantages in generative networks. For that, we refer to Sec. 4.1.

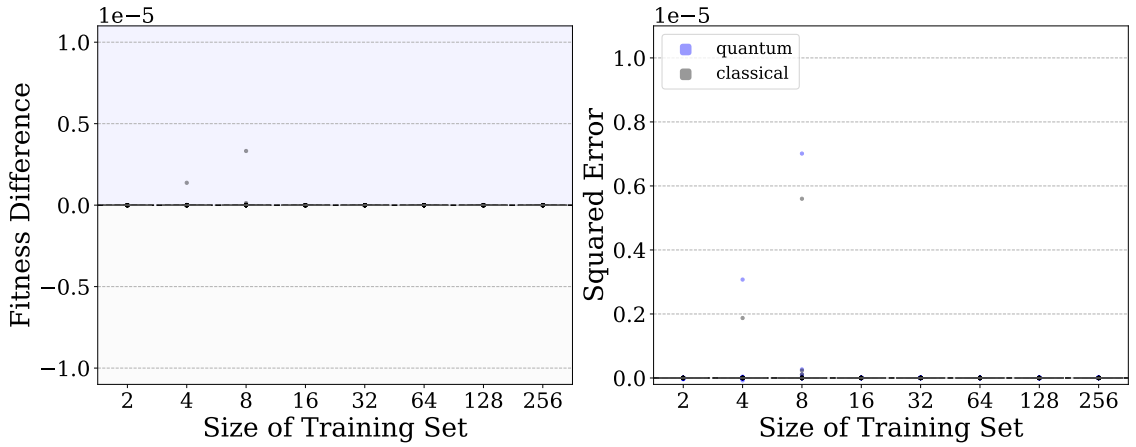

Figure A.1: Benchmarking results for a potential quantum advantage in a 4 qubit Quantum Boltzmann Machine (QBM). A quantum advantage is defined as a smaller mean square error when comparing a QBM with non-diagonal Hamiltonian (blue) to a QBM with diagonal Hamiltonian (grey). The diagonal model is taken as classical reference while the non-diagonal is considered quantum. All models are trained on random binary training sets of different sizes, i.e. increasing number of training samples. Their final overlap after training is calculated into an overlap difference. A positive fitness difference (blue shaded area) indicates a quantum advantage. 200 individual runs are marked as dots while the 25%-75% percentiles are shown as boxes. There is no indication for a quantum advantage. In fact, both model converge always with very high accuracy. The task of learning arbitrary distributions seems like too simple task to compare quantum and classical model.

B Quantum-assisted Generative Adversarial Network

The *Quantum-assisted Generative Adversarial Network* (QaGAN) presented here is a model consisting of a conventional *Generative Adversarial Network* (GAN)[26] archtitecture and a qubit system that produces the random noise for the GAN (see Fig. B.1).
Typically, a GAN consists of an Generator network that upsamples white noise into an output and a Discriminator that is trained to label the generated samples as *fake* and the training data as *real*. The two networks take part in an adversarial min-max game where the Generator is trained to fool the Discriminator with its generated data and the Discriminator is trained to detect the generated samples. During training, the Discriminator becomes better at detecting real training samples such that the Generator consequently has to generate samples which are more similar to the training data samples.
The random noise that is input to the Generator to create a sample is usually white noise of random continuous pixel values. This is efficient to create classically and ensures that the Generator needs to learn important systematics of the training data because it has to be robust against practically infinite different inputs.

By replacing the random noise input to the Generator with quantum samples generated by generative Quantum or Quantum-assisted Machine learning algorithm (Secs. 3.1, 3.2 & 4.1), one sample the input to the GAN from complex parametrized distributions. This allows for two distinct training setups for a QaGAN:
First, the quantum system can be completely replaced by random binary noise during training. This binary noise with flat distribution can be utilized efficiently during the training iterations of the model. After the classical Generator is trained, the quantum model is initiated and trained to learn a specific probability distribution. In other words, the Generator learns to map binary noise to high quality output and the quantum layer then learns a distribution of those samples, for example if one prefers to generate more number 5s than 1s, which is something that one cannot usually control in a GAN.
Second, the distribution of the binary quantum samples can be adapted during training. This could make the training more reliable for certain expected results and may help solve common problems in training GANs such as *mode collapse* and generally unstable training. This work provides no demonstration of such benefits.

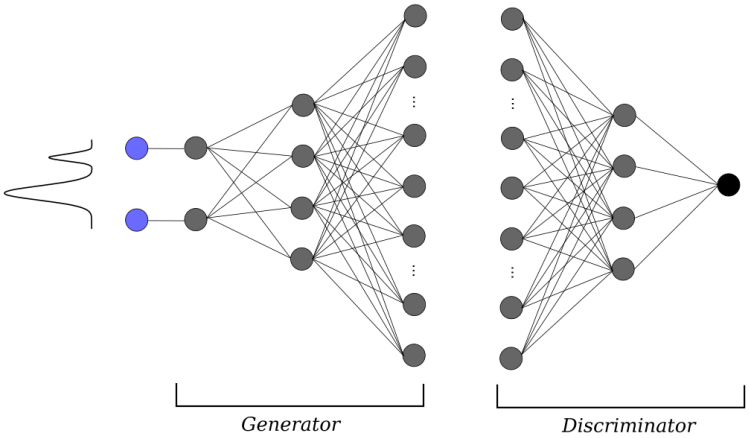

Figure B.1: Example model architecture of a Quantum-assisted Generative Adversarial Network (QaGAN). The noise that is fed into the generator network of a conventional GAN [26] is replaced by binary quantum samples produced by a qubit layer. This allows to reparametrize the distribution from which the input to the generator network are drawn during or after training.

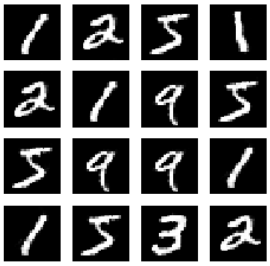

Figure B.2: Generated samples of a QaGAN that was trained on the first 8 MNIST samples after 2000 training epochs.

Fig. B.2 shows a prove of principle run of a QaGAN which consists of a slightly modified convolutional GAN Python template [35] in the *tensorflow* framework [2] and qubit layer which is initiated in a flat distribution of all possible superpositions. The training set for the QaG are the first eight samples of the 28x28 MNIST training data.

C Thermofield Double State

A relevant question is how to create a thermal state on a qubit quantum computer in order to not. One idea is to create a so-called Thermofield Double State [19]. One doubles the number of qubits in order to have an environment with which the target system interacts. In the end, the environment is traced out and a thermal state in the system remains.
The *thermofield double* (TFD) state in our case is the pure state

$$|\text{TFD}\rangle = \frac{1}{\sqrt{Z}} \sum_n e^{-\beta E_n/2} |n\rangle_S \otimes |n\rangle_E \tag{C.1}$$

where S and E are two systems (system-environment) with the property that if you trace out one, the remaining system is in the purified thermal state with eigenstates n, eigenenergies E_n and inverse temperature β.
The Hamiltonian implemented for the TDF is

$$H_{total} = H_S \otimes H_E = H_S + H_E + H_{int}. \tag{C.2}$$

H_S and H_E are the system- and environment Hamiltonian, both follow form of a Quantum Boltzmann Machine spin Hamiltonian (3.9) and importantly, $H_S \stackrel{!}{=} H_E$ must be equal. The interaction Hamiltonian H_{int} between the systems can be chosen arbitrarily but constant. For our system, we naturally choose

$$H_{int} = \sum_{i \in S, j \in E} J_{ij} \sigma_i^z \sigma_j^z. \tag{C.3}$$

Providing prove of prinziple that a Quantum Boltzmann Machine can be trained using a TFD state, we train a $n = 3$ qubit QBM on a binomial training set (see sec. 2.5). Fig. C.1 shows the final system+enviroment tomography before tracing out the envirmonment as well as the final traced system.

Fig. C.2 shows a comparison in performance of $n = 3$ Quantum Boltzmann Machines that are trained with the full Boltzmann density matrix ($t = 2^n - 1$ in eq. 3.11), the quantum groundstate and the TFD state. The results of the groundstate and the TFD state are idential which shows that the performance benefits of a thermal state (compare fig. 3.2.3) do not come from the distribution that the thermal state has but from the mixed ensemble nature of the models. In fact, these results indicate that every trained pure state in a truncated QBM framework will perform equally.

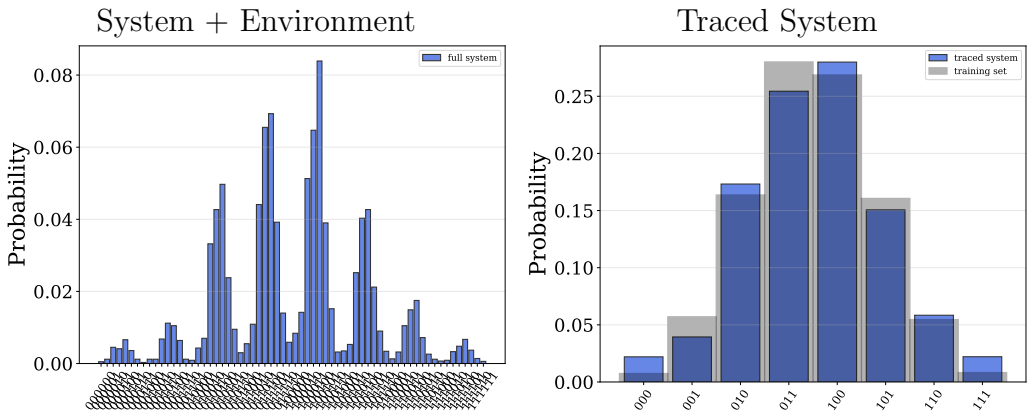

Figure C.1: Tomographies of a Quantum Boltzmann Machine where the thermal state is obtained by a Thermofield Double (TFD) state. The training set is a binomial training set with $n = 3$ qubits. Left: Tomography of the System + Environment before tracing the environment. Right: Tomography of the traced TDF state implements a purified thermal state.

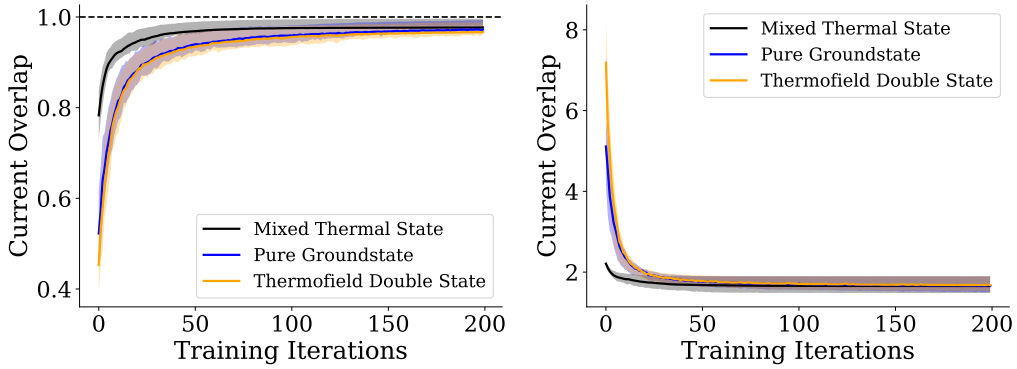

Figure C.2: Comparison in training performance between a $n = 3$ qubit Quantum Boltzmann Machine with full Boltzmann density matrix, just a quantum groundstate and the Thermofield Double state. The models are trained on random training sets. The TFD state is a purified thermal state and thus does not perform better than a quantum groundstate. In fact, they are virtually equal and both outperformed by the full Boltzmann thermal ensemble model. This indicates, that the TDF state has the same performance as a quantum groundstate and in fact any trained quantum pure state.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 13.03.2020 .....