DEPARTMENT OF INFORMATICS
LUDWIG MAXIMILIANS UNIVERSITY MUNICH


**Bachelor's thesis**

# Quantum process tomography of spin glasses via time-delayed measurements

Daniil Teplitskiy

DEPARTMENT OF INFORMATICS
LUDWIG MAXIMILIANS UNIVERSITY MUNICH


**Bachelor's thesis**

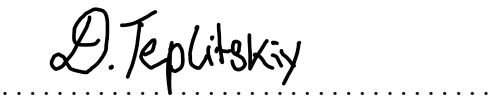
# Quantum process tomography of spin glasses via time-delayed measurements

Daniil Teplitskiy

| | |
|---|---|
| Examiner: | Prof. Christian B. Mendl |
| Advisor: | Irene López Gutiérrez |
| Submission Date: | 9. August 2021 |

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used. The implementation of this thesis can be found in the Github repository `https://github.com/MaestroDT/bachelorarbeit-quantum-tomography-of-spin-glasses-via-time-delayed-measurements` and access was given to the advisor.

Aachen, 09.08.2021



*(Signature of the candidate)*

**Abstract**

When characterizing quantum systems, quantum process tomography (QPT) is the standard primitive. But due to the high complexity of quantum systems and the curse of dimensionality, QPT becomes impractical when dealing with a large number of qubits. On the other hand, combining QPT and machine learning has shown great success in recent studies. In this thesis, the opportunity is explored of doing QPT in combination with machine learning and parametrized quantum circuits, regarding the reconstruction of Hamiltonians of spin glasses. This results in a rather simple and straightforward algorithm. For this, in the beginning, the necessary quantum circuit is derived. With this, the Hamiltonians of Ising spins are reconstructed. Finally, we switch to spin glasses, which doesn't differ much to Ising spins, and do the same here. From this, the systems are fully characterized by the obtained Hamiltonians afterwards. These approaches are done for system sizes of up to 12 qubits, whereas more qubits would be also possible. The results of the reconstructions are reaching high fidelity values using simulated data for the Ising model and spin glasses, showing and underlining the efficiency of the proposed algorithm.

# Contents

# 1 Introduction

In the modern world machine learning is on the rise in many different areas and is a useful tool in the ongoing research in these fields, i.e. quantum chemistry [Dra20] and quantum physics [Hus17]. It opens up new doors and opportunities to solve complex problems numerically, as it allows to learn an underlying structure and parameters which imitate the problem. Therefore, this is a reasonable approach to solve problems which might be to difficult or impractical to solve analytically. In this thesis, the opportunities are explored, which are given by machine learning and quantum circuits to do quantum process tomography (QPT) of spin glasses via time delayed measurements.

QPT has been already accomplished for various setups, be it for the reconstruction of quantum channels [TWA+20] with the means of unsupervised learning and tensor networks, the characterization of performance of an universal entangling gate between two superconducting qubits [BAH+10] or for the extraction of electrons and hole wavefunctions and their probabilities from electrical currents [BMR+19].

But first of all, what is QPT? It is the process of characterizing the dynamics of a quantum system via experimental measurements. To have such a method is very important as it is used, i.e. for benchmarking quantum hardware, as this verifies if the quantum hardware is working correct and does that what it is supposed to do. The need comes due to that Quantum hardware is getting bigger and bigger with having dozens of manipulatable qubits right now. It is expected that in near future the sizes will grow to hundreds of qubits [Pre18]. *IBM* already released their quantum systems (Q System One), *Falcon* (2019) with 20 qubits, and *Hummingbird* (2020) with 65 qubits and more are announced for the upcoming years. Thanks to the nature of QPT, it comes handy in applications where a system which consists of qubits or can be described with such needs to be characterised.

There are several different approaches to QPT. One method is the standard approach proposed by Isaac L. Chuang and M. A. Nielsen [NC10], where a $\chi$-matrix is derived that characterizes the dynamics $\varepsilon$ of the system and which will be further explained in section 2.1. An another approach would be ancilla-assisted process tomography [ABJ+03]. Here, the process $\varepsilon$ acting on a system A is characterized by prepraring a single state, $\sigma$ and then measuring $(\varepsilon \bigotimes I)\sigma$. For appropriate initial states and an ancilla system B with at least the same Hilbert space dimension as A, $\varepsilon$ can be characterized by performing the process on system A, leaving B isolated and performing tomography on the output of $(\varepsilon \bigotimes I)\sigma$. Unfortunately, QPT is covered with the curse of dimensionality as the preparation [NC10] and measurement sets [NC10, ABJ+03] scale exponentially with the number of qubits in the system. Because of this, QPT has only been implemented for quantum systems of a small size [BAH+10, OPG+04, RKS+06].

There are approaches to bypass these limitations, such as [TWA+20] where they use an efficient representation of a choi matrix in terms of a tensor network, combined with an unsupervised machine learning algorithm to learn the optimal tensor-network parameters. The algorithm consists of reducing the statistical divergence of the corresponding process

probabilities and of the underlying true probability distribution of the measurement data. Thereby, only the parameters of the tensors, of which there are significantly fewer than the choi matrix entries, need to be learned. This approach is a prime example of how machine learning can be used in connection with QPT, namely, in the first step, a good parametrized representation of the respective quantum system is chosen where there are as few parameters as possible. Followed up by a machine learning algorithm determining the optimal parameters to represent the system based on some distance measurement.

Accordingly, the algorithm of this thesis works the same way. First, the *Ising spins* or the *spin glasses* are getting represented by Hamiltonians according to sections 2.2.1 and 2.2.2, with parameters $J_i$ and $h_i$. Respectively the time evolution for a quantum system, $|\phi_{t+1}\rangle = e^{-iHt}|\phi_t\rangle$ is getting approximated by the trotter fromula [NC10] and can be interpreted as a quantum circuit consisting of parametrized rotation gates. Finally the machine learning algorithm is applied, which searches for the optimal parameters for $J_i$ and $h_i$, by minimizing the statistical divergence of the corresponding process probabilities and the underlying true probability distribution of the measurement data from $|\phi_{t+1}\rangle = e^{-iHt}|\phi_t\rangle$. How this is done will be explained in detail in the next chapters, starting with the foundations, followed by the technical setup, and application, as well as the results.

# 2 Foundation

Before the actual algorithm gets explained, it is necessary to explain the basics, that are needed to understand the approach of the thesis and upon what it is build.

## 2.1 Quantum process tomography

In theory $d^2$ pure quantum states $|\phi_1\rangle, \ldots, |\phi_{d^2}\rangle$ are chosen, where d stands for the dimensions of the quantum system, so that the corresponding density matrices $|\phi_1\rangle\langle\phi_1|, \ldots, |\phi_{d^2}\rangle\langle\phi_{d^2}|$ form a basis set for the space of $d \times d$ matrices. After that, the single quantum states would be prepared in quantum states and be subjected to the process which should be characterized. When all states were subjects to the process $\varepsilon(|\phi_i\rangle\langle\phi_i|)$, quantum state tomography (QST) is applied to the output of the respective processes. From a purist's point of view the system is now characterized, since the quantum operation $\varepsilon$ can now be determined by a linear extension to all states.

But how can a useful representation of $\varepsilon$ be determined in practice? Ultimately the goal is to determine the set $\{E_i\}$ of operators for $\varepsilon$

$$\varepsilon(\rho) = \sum_i E_i \rho E_i^\dagger \tag{2.1}$$

which characterizes the dynamics of the system. To achieve this, a convenient way is to use a *fixed* set of operators $\{\tilde{E}_i\}$, which form a basis on the state space

$$E_i = \sum_m e_{im} \tilde{E}_m \tag{2.2}$$

for some set $e_{im}$. Now equation (2.2) can be inserted into (2.1)

$$\begin{aligned} \varepsilon(\rho) &= \sum_i \sum_m e_{im} \tilde{E}_m \rho \sum_n e_{in}^* \tilde{E}_n^\dagger && (2.3)\\ &= \sum_{mn} \tilde{E}_m \rho \tilde{E}_n^\dagger \sum_i e_{im} e_{in}^* && (2.4)\\ &= \sum_{mn} \tilde{E}_m \rho \tilde{E}_n^\dagger \chi_{mn} && (2.5) \end{aligned}$$

where $\chi_{mn} = \sum_i e_{im} e_{in}^*$ are the $d^4$ entries of the positive hermitian $\chi$ matrix, which shows that $\varepsilon$ can be described by a complex number matrix, once the set of $E_i$ has been *fixed*. But still the question remains, how can these entries be obtained?

Let $\rho_j$, $1 \leq j \leq d^2$ be a fixed linear independent basis for $d \times d$ matrices. A convenient choice is to use the set of operators $|m\rangle\langle n|$. Experimentally, the output state $\varepsilon(|m\rangle\langle n|)$ may be obtained by preparing $|m\rangle$, $|n\rangle$, $|+\rangle = \frac{|m\rangle + |n\rangle}{\sqrt{2}}$ and $|-\rangle = \frac{|m\rangle + i|n\rangle}{\sqrt{2}}$ and performing a QST

on $\varepsilon(|m\rangle\langle m|)$, $\varepsilon(|n\rangle\langle n|)$, $\varepsilon(|+\rangle\langle +|)$ and $\varepsilon(|-\rangle\langle -|)$. Then these informations can be used to form linear combinations to get $\varepsilon(\rho_j)$

$$\varepsilon(|m\rangle\langle n|) = \varepsilon(|+\rangle\langle +|) + i\varepsilon(|-\rangle\langle -|) - \frac{1+i}{2}\varepsilon(|n\rangle\langle n|) - \frac{1+i}{2}\varepsilon(|m\rangle\langle m|). \tag{2.6}$$

Each $\varepsilon(\rho_j)$ can be expressed as a linear combination of the basis states

$$\varepsilon(\rho_j) = \sum_k \lambda_{jk}\rho_k \tag{2.7}$$

with the corresponding eigenvalues $\lambda_k$, which can be retrieved by standard linear algebraic algorithms. Furthermore

$$\tilde{E}_m \rho_j \tilde{E}_n^\dagger = \sum_k \beta_{jk}^{mn}\rho_k \tag{2.8}$$

gets defined, where $\beta_{jk}^{mn}$ are complex numbers which can also be determined by standard linear algebraic algorithms given $\tilde{E}_m$ and $\rho_j$. By combining (2.5), (2.7) and (2.8) it follows

$$\lambda_{jk} = \sum_{mn} \beta_{jk}^{mn}\chi_{mn}. \tag{2.9}$$

The only step left now, is to retrieve $\chi$. Therefore $\beta$ can be viewed as a $d^4 \times d^4$ matrix, with $mn$ indexing the columns and $jk$ indexing the rows. $\chi$ and $\lambda$ can be seen as vectors with $d^4$ entries. Now the generalized inverse $\kappa$ of beta is found by some standard linear algebraic algortihm and $\chi$ is determined:

$$\chi_{mn} = \sum_{jk} \kappa_{mn}^{jk}\lambda_{jk}. \tag{2.10}$$

At this point $\varepsilon$ is characterized, as $\chi$ is determined and can be used in (2.5). To get the respective operators $\{E_i\}$, the $\chi$ matrix gets diagonalized by a corresponding unitary matrix $U^\dagger$

$$\chi_{mn} = \sum_{xy} U_{mx} d_x \delta_{xy} U_{ny}^* \tag{2.11}$$

where $\delta_{xy}$ represents the kronecker delta and $d_x$ represents the eigenvalues of $\chi$. From this, the operators for $\varepsilon$ are determined by

$$E_i = \sqrt{d_i}\sum_j U_{ij}\tilde{E}_j. \tag{2.12}$$

The system is now fully characterized by the standard approach of QPT as described in [NC10].

Despite the beauty of this approach, the approach used in this thesis is slightly different. Instead of reconstructing a $\chi$ matrix for the time evolution of the spin glasses, the *Hamiltonian*, which characterizes the system is derived directly from the measurements. This allows for a simple yet effective machine learning algorithm in combination with quantum circuits, which will be described and analyzed throughout the thesis.

## 2.2 Hamiltonian

In this thesis, the Hamiltonian is used to characterize the dynamics of a system as it has fewer parameters to reconstruct which characterize it than the $\chi$-matrix from section 2.1. This will get clear in 2.2.1 and 2.2.2. Traditionally the time-independent Hamiltonian represents the total amount of energy in a (closed) system and characterizes, therefore the dynamics of it [Sch07].

The Schrödinger linear differential equation

$$i\hbar\frac{\partial}{\partial t}\left|\phi(t)\right\rangle = H\left|\phi(t)\right\rangle \tag{2.13}$$

with $\hbar$ standing for the reduced planck constant, describes how a quantum state $|\phi\rangle$ changes over time governed by a Hamiltonian $H$. The argument $t$ stands for the time dependence of $|\phi\rangle$. The solution for Eq. (2.13) is given by

$$\left|\phi(t)\right\rangle = U_t\left|\phi(0)\right\rangle \text{ with } U_t = e^{-iHt} \tag{2.14}$$

as long as H is time-independent with $U_t$ as the unitary time evolution operator [NC10].

### 2.2.1 For the Ising Model

The (transverse-field) *Ising model* is a mathematical model which was originally invented to describe ferromagnetism [Isi25]. With this model, it is possible to identify phase transitions of matter, as a simplified model of reality [Gal99]. It is defined over a d-dimensional lattice, consisting of $n$ Ising spins ($\uparrow$, $\downarrow$), which can also be represented as qubits, i.e. $|0\rangle$ as $\uparrow$ and $|1\rangle$ as $\downarrow$. In this thesis, the 2-dimensional Ising model is investigated. Here, the adjacent sites interact with each other with a consistent *coupling strength* $J$. Also, each site has a consistent external magnetic field $h$ interacting with it, which acts orthogonal to the plane spanned by the lattice, as can be seen in figure 2.1. The sites are enumerated from the top left counting through the rows to bottom right.

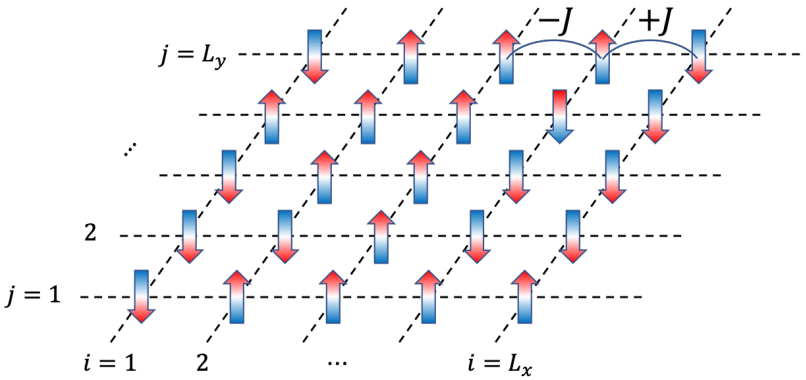



Figure 2.1: Lattice with Ising spins (arrow up stands for spin up and arrow down for spin down) on each site (external magnetic field not shown)[PG20].

The Hamiltonian function for such a system with periodic boundary condition and a

configuration $\sigma$ is defined as:

$$H(\sigma) = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - h \sum_{i}^{n} \sigma_i \tag{2.15}$$

where $\sigma_i \in \{+1, -1\}$ represents spin-up or -down [Isi25]. Here $H$ represents the total amount of energy in the configuration of the system.

For our purposes, as it will be worked with quantum states, we want to express this in a quantum mechanical way using the respective Pauli matrices for the Ising spins. That's the case as each of the Pauli matrices corresponds to an observable describing the spin of a spin-$\frac{1}{2}$ particle in each of the three dimensions. The resulting Hamiltonian is

$$H = -J \sum_{\langle i,j \rangle} \sigma_i^Z \sigma_j^Z - h \sum_{i}^{n} \sigma_i^X \tag{2.16}$$

where $\sigma_l^Z$ is the Pauli-Z matrix and $\sigma_l^X$ is the Pauli-X matrix acting on site $l$ (i.e. $\sigma_l^X = \underbrace{I \otimes I \otimes \ldots \otimes \sigma^X \otimes \ldots \otimes I}_{n-times}$, where $\sigma^X$ acts on the l-th qubit for an n-qubits big system and $I$ stands for the $2 \times 2$ Identity matrix) [SIC13]. In this case, the Hamiltonian represents a superposition of configurations with the eigenstates of $H$, referred to as energy eigenstates and the corresponding eigenvalues as energies.

As can be seen here, there are only two parameters characterizing the system, which need to be learned. This is a desired property, as in the standard QPT, there were too many, which made the process unfeasible for large systems.

### 2.2.2 For spin glasses

*Spin glasses* doesn't differ that much from the *Ising model*. They are a part of the condensed matter physics and are magnetic states characterized by randomness. In our case the $n$ spins are positioned in a lattice but not necessarily on the grid lines spanned by the lattice, see figure 2.2. According to the Edward-Anderson model [EA75] the Hamiltonian for a transverse-field is now defined as

$$H = -\sum_{\langle i,j \rangle} J_{i,j} \sigma_i^Z \sigma_j^Z - \sum_{i}^{n} h_i \sigma_i^X \tag{2.17}$$

with $J_{i,j}$ as the coupling strength of the adjacent sites $\langle i, j \rangle$ and $h_i$ as the magnetic field acting on spin at site $i$. Different from the *Ising model*, the J's and h's aren't necessarily the same and can differ. This has the effect, that there are now more parameters which are needed to describe a system but on the other hand, more complex systems can now be characterized. Still, the additional parameters aren't that many compared to QPT.

## 2.3 Trotterized circuit

The trick for the algorithm to work now lies in the approximation of eq. 2.14. For many systems the Hamiltonian can be written as a sum over many local interactions where $H_i$

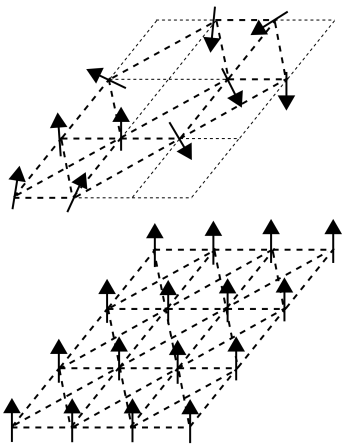


Figure 2.2: Random spin structure of a spin glass (top) and the ordered of a ferromagnet (bottom). The arrows represent the direction of the spins, the thick dashed lines, the interactions between neighboring sites and the thin dashed lines, the lattice structure. The external magnetic field isn't shown in this picture [zur10].

acting on a constant number of systems

$$H = \sum_k H_k \tag{2.18}$$

following from that, it is possible to rewrite $e^{iHt}$ in form of a multiplication of $e^{iH_k t}$ acting on small subsystems, which are easier to compute as $e^{iHt}$ and can be easier approximated via quantum circuits [NC10].

Generally, $e^{i(A+B)t} \neq e^{iAt}e^{iBt}$, where $A$ and $B$ are matrices, as it is only applicable if $A$ and $B$ commute. For general cases the *trotter formula* needs to be used. Let $A$ and $B$ be hermitian operators, then the trotter formula holds for any real $t$ [NC10]

$$e^{i(A+B)t} = \lim_{x\to\infty}(e^{iAt/x}e^{iBt/x})^x. \tag{2.19}$$

With further approximations we can rewrite eq. (2.19) as

$$e^{i(A+B)\Delta t} = e^{iA\Delta t}e^{iB\Delta t} \tag{2.20}$$

where when $\Delta t$ is getting smaller, the approximation gets more accurate.

Applying eq. (2.20) to the evolution operator of(2.14) with the Hamiltonian for the *Ising model* (2.16), results in

$$\begin{aligned} e^{-iH\Delta t} &= e^{-i(-J\sum_{\langle l,j\rangle}\sigma_l^Z\sigma_j^Z - h\sum_l^n \sigma_l^X)\Delta t} && (2.21)\\ &\underset{(2.20)}{\approx} e^{-i(-J\sum_{\langle l,j\rangle}\sigma_l^Z\sigma_j^Z)\Delta t}e^{-i(-h\sum_l^n\sigma_l^X)\Delta t} && (2.22)\\ &= \prod_{\langle l,j\rangle} e^{i(J\sigma_l^Z\sigma_j^Z)\Delta t}\prod_l^n e^{i(h\sigma_l^X)\Delta t}. && (2.23) \end{aligned}$$

Eq. (2.23) can be written, because $\sigma^Z$ and $\sigma^X$ commute with itself.

Further calculations for $e^{i(h\sigma_l^X)\Delta t}$ results in

$$
\begin{aligned}
e^{i(h\sigma_l^X)\Delta t} &= \cos(-h\Delta t)I - i\sin(-h\Delta t)\sigma_l^X && (2.24)\\
&= \cos(-h\Delta t)\underbrace{(I\otimes I\otimes\ldots\otimes I)}_{n-times} - i\sin(-h\Delta t)\underbrace{(I\otimes\ldots\sigma^X\ldots\otimes I)}_{n-times} && (2.25)\\
&= \underbrace{(I\otimes\ldots\otimes I)}_{\alpha-times}(\cos(-h\Delta t)I - i\sin(-h\Delta t)\sigma^X)\underbrace{(I\otimes\ldots\otimes I)}_{(n-\alpha-1)-times} && (2.26)\\
&= \underbrace{(I\otimes\ldots\otimes I)}_{\alpha-times} R_x(-2h\Delta t)\underbrace{(I\otimes\ldots\otimes I)}_{(n-\alpha-1)-times} && (2.27)\\
&= R_x(-2h\Delta t)_l && (2.28)\\
\text{with}\quad & R_x(\theta)_l = \cos(\theta/2)I - i\sin(\theta/2)\sigma_l^X = e^{-i\theta\sigma^X/2}. && (2.29)
\end{aligned}
$$

Therefore, eq. (2.23) can further be represented with rotation [NC10] and Entanglement-ZZ gates [KC01]

$$
\begin{aligned}
e^{-iH\Delta t} &= \prod_{\langle l,j\rangle} EntZZ(-2J\Delta t)_{l,j}\prod_{l}^{n} R_x(-2h\Delta t)_l && (2.30)\\
\text{with}\quad & EntZZ(\theta)_{l,j} = e^{-i\theta\sigma_l^Z\sigma_j^Z/2}. && (2.31)
\end{aligned}
$$

Now, everything is there to represent the time evolution of a system with the means of quantum circuits. The quantum circuit in figure 2.3 is equivalent to eq. (2.14) with eq. (2.30) used as the evolution operator, thus representing the Ising model.

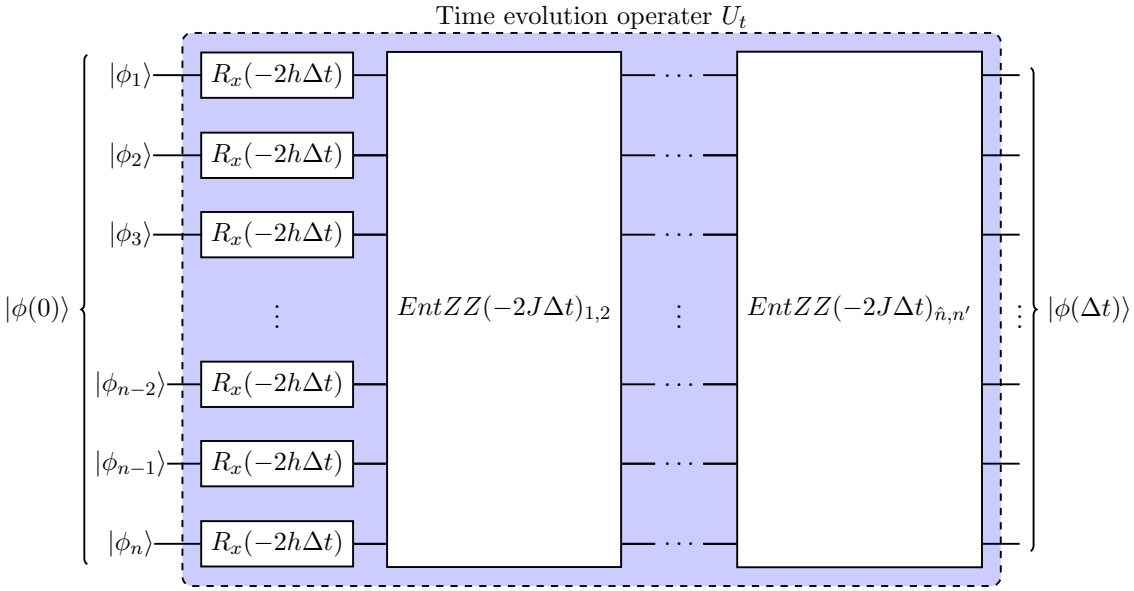



Figure 2.3: Quantum circuit representation of eq. (2.14) for the Ising model. The rotation gates are acting on each qubit seperately, while the entanglementZZ gates act only qubit $\hat{n}$ and n', the other qubits are left untouched.

As can be seen, the circuit consists of parametrized gates where all the $R_x$ gates have the same parameter as well as the $EntZZ$ gates. The goal is to learn these parameters for a

specific system via time-delayed measurements of the system. After they are learned, the system is fully characterized as the Hamiltonian of the system is known. Analogous to the Ising model, the formula for the time evolution operator of spin glasses is

$$e^{-iH\Delta t} = \prod_{\langle l,j\rangle} EntZZ(-2J_{\langle l,j\rangle}\Delta t)_{l,j} \prod_{l}^{n} R_x(-2h_l\Delta t)_l. \tag{2.32}$$

The resulting quantum circuit for spin glasses can be seen in fig. 2.4. In comparison to the

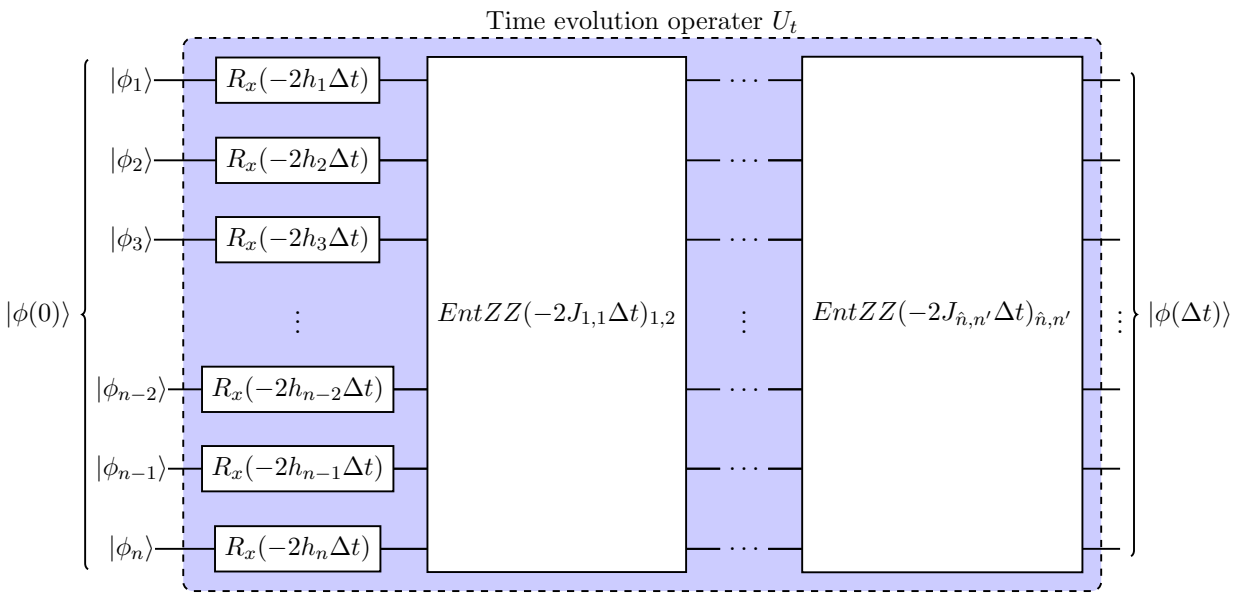



Figure 2.4: Quantum circuit representation of eq. (2.14) for spin glasses. The rotation gates are acting on each qubit seperately, while the entanglementZZ gates act only qubit $\hat{n}$ and n', the other qubits are left untouched.

Ising model, more parameters need to be learned but at most only $n + 2n = 3n$ (with the periodic boundary condition), as each qubit has a $R_x$ gate assigned and in a two dimensional lattice, with width and height bigger than one, are only $2n$ interactions, meaning each qubit has in theory two outgoing interactions. That's the case as an interaction between two qubits is bidirectional and therefore counted only once. It can be imagined that each qubit has an outgoing connection to the top and right. For lattices where the width or height equals two or less, there are $2n - width$ if $height \leq 2$ else $2n - height$ parameters, as in this cases, due to the periodic boundary condition, there are 'duplicate' interactions in the lattice which need to be subtracted.

The only bottleneck left are the high dimensional matrices and state vectors, which scale exponentially with the number of qubits, on which mathematical operations will be executed during the learning process.

# 3 Architecture and Technical setup

After the foundation is set, it can be proceeded with the architecture and technical setup. In this chapter, it will be explained how the algorithm is set up and works. I.e., how all the single parts like the Hamiltonian or the circuits are initialized and look like. The algorithm is written in the programming language Julia [BEKS17], for the machine learning part Flux [ISF+18] is used, as it enables an efficient and easy to use interface, and for the quantum circuits, Qaintessent and Qaintellect, a framework from the quantum computing group of Informatics 5 of the TU Munich is used, because it allows to create parametrized quantum circuits and let these to be trained efficiently via Flux. Julia is used as it is high-performant and delivers a good approach to numerical computing. Furthermore, it has good libraries like Flux and Qaintum, for the purposes of this thesis. As this thesis relies on fast and efficient computation regarding the curse of dimensionality, Julia is a good choice. Julia is also currently in the center of research [FIM+21, CCG21, GSW20] in computer and computational sciences, as it has a broadly diversified ecosystem for many different purposes. For example, in this thesis it is worked with mostly with sparse arrays and matrices, as Julia has a very nice and efficient way to execute computations on these and saves memory on the same time. As the matrices and arrays will become big, this comes handy.

Below is a table 3.1 of the tools which were used KrylovKit was used for the efficient

| Tool | Version | Reference |
|---|---|---|
| Julia | 1.6.0 | `https://julialang.org` |
| Flux | 0.12.1 | `https://fluxml.ai` |
| Qaintellect | 0.1.0 | `https://github.com/Qaintum/Qaintellect.jl#master` |
| Qaintessent | 0.1.1 | `https://github.com/Qaintum/Qaintessent.jl#master` |
| KrylovKit | 0.5.3 | `https://github.com/Jutho/KrylovKit.jl` |
| BenchmarkTools | 1.1.0 | `https://github.com/JuliaCI/BenchmarkTools.jl` |

Table 3.1: Tools used in this thesis.

exponentiation of matrices.

## 3.1 Hamiltonian generation

As there is no experimental data available from an experiment which can be used for learning the underlying Hamiltonian, data needs to be simulated which will be used instead. Therefore, first of all, a Hamiltonian needs to be generated, which describes the underlying system and from which the time-delayed measurements of quantum states will be taken. These measurements will then be used in the machine learning process as input/output data on which the training quantum circuit will be optimized and from which the true Hamiltonian can be recreated.

In this case, first of all, a lattice needs to be created, where the qubits (*Ising spins*) are positioned on, and the corresponding adjacency matrix. This is done via the function in figure 3.1. $Nx$ stands for the width and $Ny$ for the height of the lattice. Here the periodic boundary condition is assumed. This matrix is needed as only the neighboring spins interact with each other. With this, the different Hamiltonians for the Ising model and spin glasses

```
1  function lattice_adjacency_map(Nx::Integer, Ny::Integer)
2      L = Nx * Ny
3      adjacency = zeros(Int, (L, L))
4      for j in 0:Ny-1
5          for i in 0:Nx-1
6              # assuming periodic boundary conditions
7              i_next = (i+1) % Nx
8              j_next = (j+1) % Ny
9              # nearest neighbors
10             if Nx > 1 adjacency[j*Nx + i + 1, j*Nx + i_next + 1] = 1; end
11             if Ny > 1 adjacency[j*Nx + i + 1, j_next*Nx + i + 1] = 1; end
12         end
13     end
14     adjacency = adjacency + transpose(adjacency)
15     # only 0 or 1 entries
16     return (adjacency .!= 0)
17 end
```

Figure 3.1: Code to create the adjacency matrix for a lattice.

can be created.

### 3.1.1 For the Ising model

The Ising model Hamiltonian gets created with the function from figure 3.2, where J and h are the characterizing parameters. Adjacency is the adjacency matrix for the given lattice and sigma are the pauli matrices. First, a sparse zero Hamiltonian matrix is created and then the interaction terms of the neighboring spins, as well as the interaction of the spins with the external magnetic field, get substracted from the zero matrix. Line 17 in figure 3.2 represents $(I \bigotimes I \bigotimes \ldots \bigotimes \sigma^Z \bigotimes I \bigotimes \ldots \bigotimes \sigma^Z \bigotimes \ldots \bigotimes I)$ where the Pauli-Z matrices act on site $i$ and $j$ and Line 24 equals to $(I \bigotimes I \bigotimes \ldots \bigotimes \sigma^X \bigotimes I \bigotimes \ldots \bigotimes I)$ where the Pauli-X matrix acts on site i.

### 3.1.2 For spin glasses

The spin glass Hamiltonian gets created analogously. Therefore, we need to create first the parameters $J$ and $h$, where $J$ is a matrix and $h$ a vector. That's the case as in the spin glass model, the interaction strengths between the neighboring spins can differ as well as between the spins and the external transverse field. The matrix for $J$ gets created equally to the the adjacency matrix, but instead of setting 1, the strength for the pair is set. The vector for $h$ symbolizes the interaction with the magnetic field at each spin site, counting row by row in the lattice from top left through the rows to bottom right.

```
using SparseArrays

function construct_hamiltonian_ising(J::Real, h::Real, adjacency::
    AbstractMatrix)
    L = size(adjacency, 1)
    @assert(size(adjacency) == (L, L))
    H = spzeros(Float64, 2^L, 2^L)

    sigma = (sparse([0. 1.; 1. 0.]),
             sparse([0. -im; im 0.]),
             sparse([1. 0.; 0. -1.]))

    # interaction terms
    for i in 1:L
        for j in i+1:L
            # considering only entries in J for i ¡ j
            if adjacency[i, j] != 0
                H -= J * kron(sparse_identity(2^(L-j)), sigma[3],
                    sparse_identity(2^(j-i-1)), sigma[3], sparse_identity(2^(
                    i-1)))
            end
        end
    end

    # external field
    for i in 1:L
        H -= h * kron(sparse_identity(2^(L-i)), sigma[1], sparse_identity
            (2^(i-1)))
    end

    return H
end
```

Figure 3.2: Code for the Ising model Hamiltonian.

When this is done, the Hamiltonian for a spin glass can be created. It is created identically to the Ising model Hamiltonian but instead of a constant $J$ and $h$, a matrix and vector are now used to retrieve the needed parameters.

## 3.2 Data generation

Now that the Hamiltonians for the Ising model, as well as the spin glasses, can be created, the data generation can be approached. It is wanted that tuple pairs of (*before,after*) are created where *before* is the quantum state before the time evolution and *after* the state after the evolution eq. (2.14). Based on these tuples, a standard supervised machine learning algorithm is run, where the before state is applied to the quantum circuit and the resulting output state is compared with the true *after* state. Regarding the difference of these, the parameters of the circuit are modified.

In figure 3.3 the code which was used to generate the data tuples can be seen. The pa-

```
using KrylovKit

function calculateNTimeSteps(H::AbstractMatrix, phi_init, dt::Real,
    ntimesteps::Int)
    @assert(size(H,1)==length(phi_init))
    phi_out = [phi_init]

    for i = 1:ntimesteps
        phi_mom, info = exponentiate(H, -im * dt, phi_out[i], ishermitian =
            false)
        push!(phi_out, phi_mom)
    end

    return phi_out
end

function dataGeneration(H::AbstractMatrix, epochs::Number,dt::Real,
    timesteps::Real, nqubits::Number; showConsoleOutput = true)
    res = []

    for i in 1:epochs
        phi0 = rand(ComplexF64,2^nqubits)
        phi0 /= norm(phi0)
        phiref = calculateNTimeSteps(H,phi0,dt,timesteps)

        for x in 1:timesteps
            push!(res, (phi0[x],phiref[x+1]))
        end
        if showConsoleOutput
            @show(i)
        end
    end

    return res
end
```

Figure 3.3: Code for the data generation.

rameter *epochs* signalizes how many initial states should be produced and applied to the evolution process. *phiref* is an array of several sequential evolutions beginning at an initial state. This array is produced in the function *calculateNTimeSteps*, where the method *exponentiate* is used from the package KrylovKit. *exponentiate* performs an exponentiation of a matrix by approximating the true result of the exponentiation followed by a multiplication of a vector by means of krylov subspaces [Sim15] , $\phi_1 = e^{-i \cdot dt \cdot H} \phi_0$. It could also be accomplished by using the power series method for the exponentiation $e^X = \sum_{k=0}^{\infty} X^k / k!$, where X is a matrix, and then multiply the vector. Nevertheless, the krylov method is more efficient and performs faster than the power series method as it tries to avoid matrix-matrix operations. After n-timesteps are calculated starting at the initial state, these are returned as an array. This is done as $\Delta t$ (in the code present as $dt$) is usually very small so that

the to be trained circuit is an accurate representation of the time evolution operator (2.14) by means of the trotter formula (2.20). Due to this, the evolution of only one intermediate time step for a state will not change significantly. That's why several consecutive time steps of an initial state can be computed to simulate an overall bigger time step as shown in eq. (3.2).

$$|\phi(l \cdot \Delta t)\rangle = e^{-i \cdot H \cdot l \cdot \Delta t} |\phi(0)\rangle \tag{3.1}$$

$$= \left( \prod_{k}^{l} e^{-i \cdot H \cdot \Delta t} \right) |\phi(0)\rangle \tag{3.2}$$

Due to eq. (3.2) the final resulting state differs significantly enough to the initial state so that when two different evolution operators with a small $\Delta t$ are acting on the same state, a sufficient difference will be seen in the resulting states if eq. (3.2) is applied. This can be used to generate the needed amount of training data, because the intermediate time steps can be paired up to training tuples. Another way to generate more training data would be to replicate the to the time available data as much as needed. Theoretically, one time step of one initial state should be enough to train the circuit. Still, for the purposes of the thesis several time steps of an initial state are taken as it diversifies the training data, which is something machine learning benefits from.

After the time steps are computed, they are formed to tuples by taking a state and the intermediate state in the array and forming so the tuple. These tuples are then added to the result array. The result array is then used as the training data where the probability distributions of the *after* states are taken later in the training process, section 3.3.2. This is done to simulate real world experiments, as the true *after* state wouldn't be known of an unknown system which should be learned. The probability distribution of the *before* state doesn't have to be taken because when the experiment is performed, the *before* state will be artificially prepared, which then will be known.

## 3.3 Setup with Flux

The core of this thesis lies in the machine learning algorithm. For this, the framework *Flux* is used. In the following, it will be described how the environment is setup.

### 3.3.1 Training circuit

In the beginning, we define the quantum circuit, which should be trained in the process. For this purpose, we use the package *Qaintellect*. With this, it is allowed to easy define circuits, which can then be handed over to Flux. Flux then recognizes the parameters of the parametrized gates and will optimize these in the training process so that the whole circuit equals, in the end, the true evolution operator (2.14). By considering two approaches, the *Ising model* and *spin glasses*, two circuits, which don't differ that much, must be constructed.

#### 3.3.1.1 For the Ising model

The construction of the circuit according to figure 2.3 can be seen in figure 3.4. First, the rotation gates are constructed as many as there are qubits. Second, the entanglementZZ gates are constructed according to the adjacency matrix. Because the rotation gate and

```
using Qaintellect

function trotterized_circuit_timestep_ising(J::Real, h::Real, adjacency::
    AbstractMatrix, dt::Real)
    L = size(adjacency, 1)
    meas = [MeasurementOperator([1 0; 0 -1], (L,))]

    # local (single unitary) gates
    rg = RxGate(-2*h*dt)
    Uh = [circuit_gate(i, rg) for i in 1:L]

    # interaction gates
    intgates = CircuitGate[]
    eg = EntanglementZZGate(-2*J*dt)
    for i in 1:L
        for j in i+1:L
            if adjacency[i, j] != 0
                push!(intgates, circuit_gate(i, j, eg))
            end
        end
    end

    return Circuit{L}(vcat(Uh,intgates),meas)
end
```

Figure 3.4: Code for the Ising model circuit creation.

the entanglementZZ gate are only instantiated once, it gets recognized from Flux as two parameters as it is intended. Usually, when the circuit is created to be trained, $J$ and $h$ aren't known. In that case, an arbitrary value is passed. In the case of this thesis, 0. is handed over when the circuit is created for training purposes.

#### 3.3.1.2 For spin glasses

In the case of spin glasses, there isn't much of a difference. Instead of only two values, $J$ and $h$ are handed over, a matrix for the $J$'s and a vector for the $h$'s are passed. Also, instead of instantiating the rotation and entanglementZZ gates only once, they get instantiated anew in each iteration of the list comprehension, as well as in the if-clause in the double for loop with the values from $J$-matrix and $h$-vector. This way, Flux recognizes them as separate parameters. Here also, a 0. matrix and vector are handed over when the circuit is constructed for training purposes.

### 3.3.2 Loss function

While training, it is wanted that each iteration of the training process is evaluated regarding how close it is to the optimum the wanted result. This quality measurement is assured by the loss function. The loss function maps an event/iteration to a real number, which signalizes how good the machine learning algorithm represents the underlying true system. Usually an optimization seeks to minimize the loss function, this will also be the case in this thesis.

Here, the *Kullback–Leibler divergence* (3.3) [KL51], with discrete probability distributions $P$ and $Q$ defined over the same probability space $\chi$

$$D_{kl}(P||Q) = \sum_{x \in X} P(x) \log \left( \frac{P(x)}{Q(x)} \right) \tag{3.3}$$

will be used as the loss function, because when working with quantum states, the quantum states will be measured in real world examples resulting in a probability distribution. As the *Kullback–Leibler divergence* was designed to measure the entropy of probability distributions, it fits perfectly for this use case. The probability distribution of the predicted quantum state from the machine learning algorithm will be compared to the true underlying distribution of the *after* state from the training data. And then, according to this, the circuit will be optimized [BLSF19] to resemble the true evolution operator from which the Hamiltonian can be extracted by the parameters of $J$ and $h$.

The code which was used to create the loss functions can be seen in figure 3.5. There were functions used which were defined in the previous sections where $Nx$ and $Ny$ are placeholders for the width and height of the lattice and need to be initialized once the algorithm should be executed. The defined *lossSpec* function takes, in this case, four parameters, the *before*

```
1  using Flux
2  using SparseArrays
3  using Qaintellect
4
5  function lossSpec(before, cgc, after, loss)
6      phi1 = apply(before, cgc.moments)
7      loss(phi1,after)
8  end
9
10 adj = lattice_adjacency_map(Nx, Ny)
11 hrecrspn = spzeros(Float64, Nx * Ny)
12 jrecrspn = spzeros(Float64, Nx * Ny, Nx * Ny)
13 ctrain = trotterized_circuit_timestep_spinGlass(jrecrspn,hrecrspn,adj,dt)
14
15 hadamardsFront = kron([sparse_matrix(HadamardGate()) for i in 1:L]...)
16 kldivHHH(phi1,ref) = Flux.Losses.kldivergence(abs2.(hadamardsFront*phi1),
       abs2.(hadamardsFront*ref))
17 kldiv(phi1,ref) = Flux.Losses.kldivergence(abs2.(phi1), abs2.(ref))
18
19 lossFirstround(before, after) = lossSpec(before, ctrain, after,kldivHHH)
20 lossSecondround(before, after) = lossSpec(before, ctrain, after,kldiv)
```

Figure 3.5: Code for the loss definition.

and *after* of the training tuple, the training circuit and the wanted loss function. In this function, the to be optimized training circuit is applied to the initial *before* state in line 6. Later, based on this application, the resulting state and the application of the chosen loss function comparing to the true *after* state in line 7, the gradients will be computed, which are necessary for the correct optimization of the circuit. As it will be worked with complex vectors for the states, the gradients will be taken of complex values for which the wirtinger formalism will be used [GM21].

It will be used two different *lossSpec* functions as the training process will have two phases in the proposed algorithm. In the first phase, the model is trained according to a $\sigma^X$-basis measurement of the Ising-spins/ spin-glasses . This can be easily accomplished by simply putting a layer consisting of hadamard gates acting on each single qubit in the end of the circuit. In the second phase, the setup will be trained according to a measurement in the $\sigma^Z$-basis. Here, the resulting state is measured without the hadamard layer. The two different *lossSpec* functions can be seen in line 19 and 20, where the first function represents the $\sigma^X$-measurement and the second one the $\sigma^Z$-measurement. The according loss functions are in line 16 and 17. Here, the probabilities of the *after* states are taken.

#### 3.3.2.1 X/Z-Measurement

But why are these two different measurements needed? Wouldn't only the $\sigma^Z$- or the $\sigma^X$-measurement be sufficient? Interestingly, these two are needed to be used together and cannot be replaced by only one or the other measurement. When only the $\sigma^Z$-measurement is executed, the interaction terms of the Hamiltonian aren't recreated as they don't have an effect on the probability distribution of the evolved state in the $\sigma^Z$-basis.

This can be seen in the following. First, the time evolution operator is represented as a matrix for a system of size $n$:

$$e^{-iH\Delta t} = \prod_{\langle l,j\rangle} EntZZ(-2J_{\langle l,j\rangle}\Delta t)_{l,j} \prod_{l}^{n} R_x(-2h_l\Delta t)_l \tag{3.4}$$

$$= \underbrace{\begin{pmatrix} z_1 & 0 & 0 & \cdots & 0 \\ 0 & z_2 & 0 & \cdots & 0 \\ 0 & 0 & z_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & z_{2^n} \end{pmatrix}}_{\text{unitary}} \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,2^n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,2^n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,2^n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{2^n,1} & x_{2^n,2} & x_{2^n,3} & \cdots & x_{2^n,2^n} \end{pmatrix} \tag{3.5}$$

$$= \begin{pmatrix} z_1x_{1,1} & z_1x_{1,2} & z_1x_{1,3} & \cdots & z_1x_{1,2^n} \\ z_2x_{2,1} & z_2x_{2,2} & z_2x_{2,3} & \cdots & z_2x_{2,2^n} \\ z_3x_{3,1} & z_3x_{3,2} & z_3x_{3,3} & \cdots & z_3x_{3,2^n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{2^n}x_{2^n,1} & z_{2^n}x_{2^n,2} & z_{2^n}x_{2^n,3} & \cdots & z_{2^n}x_{2^n,2^n} \end{pmatrix} \tag{3.6}$$

Where $z_i$ are the entries of the resulting matrix from the multiplication of the entanglement-ZZ gates. This multiplication results in a diagonal matrix as the entanglement-ZZ gates are diagonal themselves. The $x_{i,j}$ variables stand for the entries of the resulting matrix from the multiplication of the rotation gates. Now when performing a time evolution on an arbitrary

state

$$\Rightarrow \begin{pmatrix} z_1x_{1,1} & z_1x_{1,2} & z_1x_{1,3} & \cdots & z_1x_{1,2^n} \\ z_2x_{2,1} & z_2x_{2,2} & z_2x_{2,3} & \cdots & z_2x_{2,2^n} \\ z_3x_{3,1} & z_3x_{3,2} & z_3x_{3,3} & \cdots & z_3x_{3,2^n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{2^n}x_{2^n,1} & z_{2^n}x_{2^n,2} & z_{2^n}x_{2^n,3} & \cdots & z_{2^n}x_{2^n,2^n} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_{2^n} \end{pmatrix} \tag{3.7}$$

$$= \begin{pmatrix} z_1x_{1,1}\phi_1 + z_1x_{1,2}\phi_2 + z_1x_{1,3}\phi_3 + \cdots + z_1x_{1,2^n}\phi_{2^n} \\ z_2x_{2,1}\phi_1 + z_2x_{2,2}\phi_2 + z_2x_{2,3}\phi_3 + \cdots + z_2x_{2,2^n}\phi_{2^n} \\ z_3x_{3,1}\phi_1 + z_3x_{3,2}\phi_2 + z_3x_{3,3}\phi_3 + \cdots + z_3x_{3,2^n}\phi_{2^n} \\ \vdots \\ z_{2^n}x_{2^n,1}\phi_1 + z_{2^n}x_{2^n,2}\phi_2 + z_{2^n}x_{2^n,3}\phi_3 + \cdots + z_{2^n}x_{2^n,2^n}\phi_{2^n} \end{pmatrix} \tag{3.8}$$

Determining the probability distribution of the resulting state results in

$$\Rightarrow \begin{pmatrix} |z_1x_{1,1}\phi_1 + z_1x_{1,2}\phi_2 + z_1x_{1,3}\phi_3 + \cdots + z_1x_{1,2^n}\phi_{2^n}|^2 \\ |z_2x_{2,1}\phi_1 + z_2x_{2,2}\phi_2 + z_2x_{2,3}\phi_3 + \cdots + z_2x_{2,2^n}\phi_{2^n}|^2 \\ |z_3x_{3,1}\phi_1 + z_3x_{3,2}\phi_2 + z_3x_{3,3}\phi_3 + \cdots + z_3x_{3,2^n}\phi_{2^n}|^2 \\ \vdots \\ |z_{2^n}x_{2^n,1}\phi_1 + z_{2^n}x_{2^n,2}\phi_2 + z_{2^n}x_{2^n,3}\phi_3 + \cdots + z_{2^n}x_{2^n,2^n}\phi_{2^n}|^2 \end{pmatrix} \tag{3.9}$$

$$= \begin{pmatrix} x_{1,1}\overline{x_{1,1}}\phi_1\overline{\phi_1} + x_{1,2}\overline{x_{1,1}}\phi_2\overline{\phi_1} + \cdots + x_{1,2^n}\overline{x_{1,2^n}}\phi_{2^n}\overline{\phi_{2^n}} \\ x_{2,1}\overline{x_{2,1}}\phi_1\overline{\phi_1} + x_{2,2}\overline{x_{2,1}}\phi_2\overline{\phi_1} + \cdots + x_{2,2^n}\overline{x_{2,2^n}}\phi_{2^n}\overline{\phi_{2^n}} \\ x_{3,1}\overline{x_{3,1}}\phi_1\overline{\phi_1} + x_{3,2}\overline{x_{3,1}}\phi_2\overline{\phi_1} + \cdots + x_{3,2^n}\overline{x_{3,2^n}}\phi_{2^n}\overline{\phi_{2^n}} \\ \vdots \\ x_{2^n,1}\overline{x_{2^n,1}}\phi_1\overline{\phi_1} + x_{2^n,2}\overline{x_{2^n,2}}\phi_2\overline{\phi_1} + \cdots + x_{2^n,2^n}\overline{x_{2^n,2^n}}\phi_{2^n}\overline{\phi_{2^n}} \end{pmatrix} \tag{3.10}$$

where $\overline{z}$ stands for the complex conjugate of $z$. The $z_i$ cancels themselves out as the matrix is diagonal and unitary, therefore $z_i\overline{z_i} = 1$, resulting in the EntZZ gates having no impact on the probability distribution in the $\sigma^Z$-measurement.

On the other hand, when only performing a measurement in the $\sigma^X$-basis, it could be observed that the parameters of the rotation gates are reconstructed wrong by a factor $\backsim \frac{1}{2}$. This could be due to the fact that when measuring in the $\sigma^X$-basis, the rotation gates commute with the measurement operators and therefore having no effect on the expectation value and measurement results:

$$\begin{aligned} \langle(\sigma^X \otimes \cdots \otimes \sigma^X)\rangle &= \langle\phi| e^{i\sigma_i^X \Delta t}(\sigma^X \otimes \cdots \otimes \sigma^X)e^{-i\sigma_i^X \Delta t} |\phi\rangle & (3.11) \\ &= \langle\phi| (\sigma^X \otimes \cdots \otimes \sigma^X)e^{i\sigma_i^X \Delta t}e^{-i\sigma_i^X \Delta t} |\phi\rangle & (3.12) \\ &= \langle\phi| (\sigma^X \otimes \cdots \otimes \sigma^X) |\phi\rangle & (3.13) \end{aligned}$$

Still, the interaction terms will be reconstructed as the hadamard front breaks the above described symmetry (eq. (3.10)). Due to these properties, in the first phase the $\sigma^X$-measurement is performed to reconstruct the parameters of the interaction gates and after that, in the second phase, the $\sigma^Z$-measurement is performed to reconstruct the parameters of the rotation gates, leaving the interaction gates untouched.

| $\Delta t$ | System size | Learning rate |
|---|---|---|
| $0.01, 0.1$ | $\leq 12$ Qubits | 0.001 |
| $10^{-4}, 10^{-3}$ | $< 12$ Qubits | 0.0001 |
| $10^{-3}$ | 12 Qubits | 0.0001 |

Table 3.2: Used learning rates in dependence of the $\Delta t$ and the system size.

### 3.3.3 Optimizer

As already mentioned, an optimization seeks to reduce the loss function. In *Flux*, there are several ways to do it. To achieve this, optimizers are used in this thesis, which will alter the parameters. It optimizes the parameters in dependence of the loss function and the associated gradients. There are several optimizers to choose from. Here *ADAM* [KB17] was used, as it is an advanced optimizer and performs probably the best on average. It implements adaptive learning rates and momentum, which allows the algorithm to converge faster to the optimum. Adaptive learning rates help insofar that the algorithm begins with big steps and ends with small steps, allowing faster progress. The momentum supports finding the global optimum and not only some local, as it helps to push the process into the right direction, overcoming the local optimum valleys. There would have been also other options, such as *RMSProp* [SLHS21], but due to the fact that ADAM combines ideas from different optimizers, among others, also the RMSProp, ADAM was chosen.

For the initialization of the ADAM optimizer, the learning rate and the tuple of the first and second decay needs to be set. As the optimal learning rate depends on parameters such as the $\Delta t$, the $J/h$ parameters and the system size. That is the case because when $dt$ gets small, the evolution doesn't change the initial state that much, resulting in smaller gradients on average, as the initial and resulting states are similar in the case of the training circuit as well as in the case of the underlying true evolution. Because the initial states are the same in both cases, the resulting states should be similar too. For $J$ and $h$, it is analogous as they scale $dt$ too. The size of the system plays a role because when the system is bigger, there are more dimensions of which the gradients are taken. Due to that, even if the gradients of each dimension are small and are signalizing a small error in the accuracy for the according parameter, in sum, they may be big, which could lead to a too big modification of the parameter if the learning rate isn't chosen well. Therefore, if these parameters scale, the learning rate has to be scaled too. But due to the fact that $J$ and $h$ are mostly unknown, determining the optimal learning rate can be non-trivial.

The default learning rates used, in dependence of $J \in [0.6, 2.2]$, $h \in [-10.8, 15.5]$, $\Delta t$ and the system size can be taken from the table 3.2.

#### 3.3.3.1 Decay

For a smoother and more automated optimization, a decay of the learning rate can be used in Flux. This will periodically decrease the set learning rate. It is slightly different to the adaptive learning rate of ADAM as in ADAM, the learning rate itself doesn't get changed, but scaled with a specific factor. Here, the learning rate itself would be changed, allowing an even faster convergence if configured right.

Nevertheless, this wasn't used in the thesis, as the configuration of the decay for a general case of a system is difficult because the optimal learning rate depends on many factors.

Therefore, the space where the optimal learning rate lies would be big. In theory, for a general case, a decay could be implemented, but therefore in each iteration of the decay, the training algorithm would have to learn for a sufficient amount of time, to get the most out of each iteration. If it gets trained for a too short amount of time the reconstruction isn't good enough and lowering the learning rate further could result that it gets stuck in a local optimum. As in this thesis it was wanted to create an algorithm which is efficient and fast, it has come to pass that the decay wasn't used as it needs more time than just to specify the learning rate in the beginning and observe the evolution of the loss and if needed to restart the process with a modified learning rate. Also looking forward to bigger systems, the amount of time per decay iteration needed, implied by the time needed to execute the mathematical operations on high dimensional data for sufficient amount of data, exceeds too much that it wouldn't be beneficial in the case of this thesis to work with as time plays a role.

Still, if needed and wanted, the decay can be easily added as Flux offers an easy way to implement different decay options.

### 3.3.4 Callback function while training

Another useful feature Flux offers are callbacks. This feature allows to observe the training process by executing a routine after every training iteration. With this, i.e. it is possible to show the current loss or compute other values of interest. Usually, callbacks aren't necessary for training, they just allow the user to have more insight in what is going on. For this thesis, it was very beneficial to have this feature, as it allows to collect data on how well the algorithm performs. Here, it was used to collect the loss of each iteration and the difference between the true $J$ and the current trained $J$, as well as the difference between the true $h$ and the current trained $h$. In figure 3.6, the used callback function can be seen. *plotDataLs*, *plotDataJErr*, *plotDataHErr* are the arrays for the loss, as well as for the corresponding error where the data is accumulated. *data* is the data batch which is used for testing, $L$ is the size of the lattice of the system and *paras* are the parameters of the circuit which are getting optimized.

Later, the collected data will be analyzed and plotted to see how the loss function correlates to the error of the reconstruction.

### 3.3.5 Training

After all the necessary functions are defined, the core training function can be defined. For the training function, the previously defined functions and variables will be used to generate the needed input. The training function can be seen in figure 3.7. In the end, the optimized parameters are returned. From these, the wanted $J$ and $h$ can be extracted how it was already done in the callback function (fig. 3.6). The training is done in several rounds as a further prevention of getting stuck in a local optimum. By adding in each round new data, which represents an initial state and its *ntimesteps*, the training space gets bigger. This method shows the wanted prevention and results even in a higher accuracy of the reconstruction. Flux takes over the whole training process, from calculating the gradients to optimizing the parameters with predefined settings.

Now all functions are defined which are needed to execute the algorithm. First of all, the parameters of the circuit which should be optimized need to be retrieved. This is done

```
using SparseArrays

function callback(data, loss, paras, trotterized, isSpinGlass, adj, L, dt,
    J, h, isZMeas, plotDataLs,plotDataJErr,plotDataHErr; showLoss = true)
    recreatedJ = spzeros(Float64, L, L)
    recreatedH = spzeros(Float64, L)
    tup = data[rand(1:length(data))]
    ls = loss(tup[1],tup[2])
    push!(plotDataLs, ls)

    # Recovers J and h
    if trotterized
        if isSpinGlass
            counter = 1
            recreatedH = [first(paras[i])/(-dt*2) for i in 1:L]

            #inverse operations of how the trotterized circuit for spinGlasses is
            #build from J and h.
            for i in 1:L
                for j in i+1:L
                    # considering only entries in J for i ¡ j
                    if adj[i, j] != 0
                        recreatedJ[i,j] = first(paras[L+counter])/(-dt*2)
                        recreatedJ[j,i] = recreatedJ[i,j]
                        counter += 1
                    end
                end
            end
        else
            recreatedJ = first(paras[2])/(-dt*2)
            recreatedH = first(paras[1])/(-dt*2)
        end
    end

    # Calculates the mean error of the recreated J and h (array)
    meanJ = mean(abs.(J - recreatedJ))
    meanH = mean(abs.(h - recreatedH))

    if showLoss
        @show(ls)
    end

    # Depending on the measurement the J or h error is getting appended, because while
    # ZMeasurement h is getting reconstructed
    # and while XMeasurement J is getting reconstructed.
    if isZMeas
        push!(plotDataHErr,meanH)
    else
        push!(plotDataJErr,meanJ)
    end
end
```

Figure 3.6: Code for the callback definition.

```
using Flux

function trainCircuit(rounds, databatch, ntimesteps, loss, paras, opt,
    multiple, evalcb)

    for r in 1:rounds
        # In each round new initial state and all its timesteps are added
        data = databatch[1:(r*ntimesteps)]

        # training
        Flux.train!(loss, paras, ncycle(shuffle(data), multiple), opt, cb =
            evalcb)
    end

    return paras
end
```

Figure 3.7: Code for the training function.

via the following code, including also the initialization of the optimizer with a previously defined learning rate, as well as the data generation through the function from fig. 3.3:

```
using Flux

ctrain = trotterized_circuit_timestep_spinGlass(jrecrspn,hrecrspn,adj,dt)
paras = Flux.params(ctrain)
opt = ADAM(learningRate,(0.9, 0.999))
databatch = dataGeneration(H,ntrials,dt,ntimesteps,L, showConsoleOutput =
    true)
```

where *jrecrspn* and *hrecrspn* are arbitrary values like it is described in section 3.3.1.2. *adj* is the adjacency map for a lattice (also defined previously) and *dt* is the length of one time step. $H$ is in this case, a spin glass Hamiltonian, *ntrials* is the number of wanted initial states, *ntimesteps* the wanted amount of time steps and $L = N_x \cdot N_y$, the number of qubits in the system. Following on that the two callback functions for the two different phases need to be initialized, where *trotterized*/*isSpinGlass* are booleans and $J/h$ are the true $J/h$:

```
plotDataLs = []
plotDataJErr = []
plotDataHErr = []

evalcbFirst() = callback(databatch, lossFirstround, paras, trotterized,
    isSpinGlass, adj, L, dt, J, h, false, plotDataLs,plotDataJErr,
    plotDataHErr, showLoss = true)
evalcbSecond() = callback(databatch, lossSecondround, paras, trotterized,
    isSpinGlass, adj, L, dt, J, h, true, plotDataLs,plotDataJErr,
    plotDataHErr, showLoss = true)
```

Now that everything is set up, the training function can be executed. As already written, this will happen in two phases, first, the training on the $\sigma^X$-measurement data is executed and then on the $\sigma^Z$-measurement data.

```
paras = trainCircuit(ntrials, databatch, ntimesteps, lossFirstround, paras,
    opt, multiple, evalcbFirst)
paras = trainCircuit(ntrials, databatch, ntimesteps, lossSecondround, paras
   , opt, multiple, evalcbSecond)
```

*multiple* is a parameter which is responsible for how often the initial training data batch, which is used in the current iteration, should be replicated so that the resulting data batch has more entries. This can be done so that the training process has enough data for learning. For the algorithm to run, the needed parameters need to be initialized explicitly, as the above code only use placeholders to describe a more general case. From the optimized parameters of the circuit, $J$ and $h$ can be retrieved now as described earlier.

# 4 Application and Results

At this point, the algorithm is set up and can be executed. This will be done for various system sizes, time step lengths and learning rates. Also, it will be run several times to generate mean data and according to this, the reconstruction fidelity in dependence of the loss will be measured to get empirical evidence for the correlation of the loss and the fidelity. Also, different circuit types and loss functions will be tested on how well they perform and if they represent a good alternative. But first and foremost, the hardware and system properties will be listed as the performance and efficiency depends strongly on it. They can be seen in table 4.1.

| Component | Model |
|---|---|
| Processor | Intel(R) Core(TM) i5-7300U CPU @ 2.60 GHz |
| RAM | 8 GB 1866 MHz LPDDR3 |
| System | 64-Bit-Operating system, x64-based processor |
| Operating system | Windows 10 v.20H2 |

Table 4.1: The properties of the hardware and the system where the algorithm was run.

## 4.1 Runtime

First, the algorithm (sec. 3.3.5) is executed on single runs. With single runs, executions are meant where the algorithm is only run once for a specific configuration (setting) of hyperparameters. This is specified as later, several runs for a specific setting of hyperparameters will be executed to generate mean data. Several single runs for the code in section 3.3.5 with different configurations are done and the elapsed time is measured. This shall give an intuition in what dimensions the execution time of this algorithm lies. This is interesting insofar because QPT and the variations of it are most of the time not feasible due to sheer amount of time, which is needed. This comes from the exponential scaling of the dimensions when increasing the system size, which has the consequence that the computations need significantly longer. That's why it's from interest to get an intuition about the run time for the proposed algorithm. The measurements are done with *BenchmarkTools*, where each run is executed 10 times to retrieve a mean time.

This is done for the Ising model, as well as for spin-glasses. The general configurations of each run can be seen in table 4.2, the model specific configurations, such as $J$ and $h$, will be discussed in the associated sections. *ntrials* is everywhere set to 1. As already discussed in sec. 3.3.3 it is a non trivial task to choose a learning rate as it depends from many factors. The learning rates in tab. 4.2 were chosen by trial and error. Surely there are better choices for individual cases, but here it was sufficient enough. When working with real world data and unknown $J$ and $h$, the training process must be supervised to check whether the loss is getting smaller. Normally, $h$ should be known as this parameter can be controlled/measured

| Run ID | Nx | Ny | $\Delta t$ | learning rate | multiple | ntimesteps |
|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 0.1 | 0.001 | 81 | 10 |
| 2 | 2 | 2 | $10^{-2}$ | 0.001 | 41 | 100 |
| 3 | 2 | 2 | $10^{-3}$ | 0.0001 | 1 | 1000 |
| 4 | 2 | 2 | $10^{-4}$ | 0.0001 | 1 | 10000 |
| 5 | 3 | 3 | 0.1 | 0.001 | 1 | 10 |
| 6 | 3 | 3 | $10^{-2}$ | 0.001 | 1 | 100 |
| 7 | 3 | 3 | $10^{-3}$ | 0.0001 | 1 | 1000 |
| 8 | 3 | 3 | $10^{-4}$ | 0.0001 | 1 | 10000 |
| 9 | 4 | 3 | 0.1 | 0.001 | 1 | 10 |
| 10 | 4 | 3 | $10^{-2}$ | 0.001 | 1 | 100 |
| 11 | 4 | 3 | $10^{-3}$ | 0.0001 | 1 | 1000 |

Table 4.2: The configuration settings for single runs.

by the operator and $J$ should be a number around 1 with some variance. Due to this, a good learning rate can be guessed and then be tuned during the process. But as here it was worked with simulations, the learning rates could be guessed good as $J$ and $h$ were known.

It wasn't possible to go bigger in the system size as already, for a lattice size of $4 \times 4$, Julia threw an *OutOfMemory* exception due to the curse of dimensionality.

For each execution it was configured that the overall time step should equal to 1, that's why, for example, in run 7 tab. 4.2, 1000 time steps of an initial state are taken according to eq. (3.2) as the time step is set to $10^{-3}$. More could also have been done, but due to the exponential scaling, it is only feasible for small systems. Nevertheless, an overall time step of 1 was chosen. It could also have been chosen less time steps, or even only one with a good amount of replications (i.e. *multiple* $= 200$) as tests showed that this was also sufficient enough. This is beneficial because when the algorithm is run with real world data, it is advantageous to do as little measurements as possible on the environment. It is also beneficial to have as little as possible initial states (represented by *ntrials*), which should be prepared as in these, both cases, it would be more complex to gather data. One initial state is sufficient enough. But again, as it was worked with simulated data, this wasn't an obstacle.

### 4.1.1 For the Ising model

For the Ising model in each run, the algorithm is run with random $J \in [0.6, 2.2]$ and $h \in [-10.8, 15.5]$. These intervals are chosen as the $J$ parameter is, on average, around 1 in real systems and $h$ can be arbitrary big, as this parameter is determined by the external magnetic field. That's why the interval for $J$ is chosen generously around 1 and for $h$ larger, as this is arbitrary. The mean time for each run can be taken from the table 4.3, where the time is rounded to first integer, as the time varies slightly from run to run and only the mean is considered. It can clearly be seen that the time rises with increasing $\Delta t$, *ntimesteps* or system size. Concluding that the elapsed time depends on the system size, as well as the batch size, as $\Delta t$ and *ntimesteps* determine, in this case, the data batch size. This was expected, as with increasing system sizes, the dimensions increase exponentially. Also, with increasing batch sizes, it is clear that the algorithm has to work through more data, increasing the time.

| Run ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 4 | 24 | 6 | 59 | 1 | 11 | 95 | 1171 | 21 | 94 | 837 |

Table 4.3: Ising: Mean elapsed time for each run of the algorithm, determined according to 10 samples.

It needs to be said that these times doesn't give evidence for how much time elapses until a good recreation was made, but more that even with the curse of dimensionality and data collection, this algorithm is feasible in a good amount of time for system sizes below or equal to 12 qubits. The case why run 3 needed less time than run 2 is because the data batch was smaller.

As these times (table 4.3) were measured with an algorithm where data (accuracy measurements) was gathered and much extra computation was going on to have an insight on the performance. It would be interesting to see if the time needed for the core algorithm, which only learns the parameters, is lower. The results of this test can be seen in tab. 4.4. It can be seen that there is a tendency that the times are getting smaller and even for some

| Run ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 3 | 10 | 3 | 26 | 1 | 5 | 47 | 446 | 7 | 64 | 600 |

Table 4.4: Ising: Mean elapsed time for each run, determined according to 10 samples, with only the core functions used. Callback and other computations are excluded.

runs, i.e. run 2;4;7;8;10;11, they are significantly lower and even halved compared to tab. 4.3. This was expected as the computation of the callback in each iteration of the training process can take up a good amount of computational power, which slows down the whole process, showing that the machine learning algorithm itself is even faster.

Now it would be interesting to see how long it would take to have an accurate recreation of the parameters. Therefore, the algorithm is modified that it will be interrupted when it reaches a defined accuracy. This is done via the following lines of code, which replace the lines 45 to 49 in fig. 3.6.

```
1 if isZMeas
2         push!(plotDataHErr,meanH)
3         meanH < 0.005 && Flux.stop()
4     else
5         push!(plotDataJErr,meanJ)
6         meanJ < 0.005 && Flux.stop()
7     end
```

In this example, the wanted accuracy is that the recreated parameters don't differ more than 0.005 from the original ones. The measured time can be seen in tab. 4.5. The algorithm is run under the same conditions as for tab. 4.3 only with the above shown changes in the code where a threshold was set to 0.005.

As the elapsed times show, the needed time for the algorithm to have a recreation, which only differs 0.005 to the original parameters is less, then in 4.3. This shall give an intuition how long a recreation can take until it is somehow accurate. Unfortunately, in reality it isn't possible to measure the accuracy in reality this way, which means that the algorithm cannot be interrupted like it was above. The only thing that can be done is to use heuristics and

| Run ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 3 | 1 | 1 | 7 | 1 | 5 | 12 | 33 | 9 | 44 | 90 |

Table 4.5: Ising: Mean elapsed time for each run, determined according to 10 samples of the algorithm, until an accurate recreation of the parameters is accomplished with a threshold difference of 0.005 .

guess when the wanted accuracy (in dependence of the loss) is reached and interrupt the algorithm then. Still, the risk will remain that the wanted accuracy isn't reached. That's why, when using the algorithm, it is therefore better to train, rather more than less, to be on the safe side that the wanted accuracy is reached if not wanted otherwise.

### 4.1.2 For spin glasses

The same is done for spin glasses. Also, as for the Ising model, the $J$-matrix entries are element of $[0.6, 2.2]$ and the $h$-vector entries are element of $[-10.8, 15.5]$. Here the intervals were chosen as in sec. 4.1.1. Running the algorithm for each configuration of tab. 4.2 for spin glasses results in the times of tab. 4.6. As it can be seen, the times of the runs for

| Run ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 3 | 14 | 5 | 39 | 1 | 7 | 62 | 908 | 20 | 170 | 1071 |

Table 4.6: Spin glass: Mean elapsed time for each run of the algorithm until an accurate recreation of the parameters is accomplished.

the Ising model, as well as spin glasses, are similar, although a tendency cannot be clearly seen as for the runs 2;3;4;6;7;8;9 the times are faster for spin glasses, but for the runs 10 and 11, they are slower. This doesn't have to mean anything and can be only a coincidence, as the needed time for any run has a high variance up to several seconds. Again, these measurements aren't done to get an exact time a run needs for a configuration, but to see how long, approximately a run can take. Also, the elapsed time depends on many factors, how many programs are running in parallel, on I/O actions, is chrome running and many more. That's why these times only should be used to get an intuition instead of exact results.

But what we can conclude is the same as for the Ising model. The elapsed time depends strongly on the system size, as well as from the data batch that needs to be learned. But as the times above (tab. 4.6) were measured with much computation and data gathering going on, the times, as in the Ising model case, were measured again with only the core functions running, which are needed to learn the parameters. Tab. 4.7 represents the results of the measurements for the same configurations as above.

| Run ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 2 | 10 | 3 | 27 | 1 | 5 | 47 | 446 | 7 | 64 | 609 |

Table 4.7: Spin glass: Mean elapsed time for each run, determined according to 10 samples, with only the core functions used. Callback and other computations are excluded.

What now can be seen is that the times from tab. 4.4 and 4.7 are pretty similar, if not

identical. This leads to the conclusion that for the machine learning algorithm in these cases, the number of parameters which need to be learned doesn't matter and only the data batch size matters and defines the run time. Surely, it can also be the case that there isn't a difference seen due to the relatively 'small' amount of parameters which need to be learned for spin glasses and with significantly more parameters, there maybe could be a difference visible as more computations would go on in the background. Still, it is an interesting observation, as that means the spin glass model, can be assumed every time, at least for system sizes up to twelve qubits, when something needs to be learned as this is more general than the Ising model and can therefore represent the Ising model as well. Also, there wouldn't be a performance benefit to restrict the model to the Ising model, as the spin glass reconstruction claims the same amount of time for the same data batch size. The only benefit by choosing the Ising model there is, is that there is a performance boost when data is gathered and computations are executed on them, i.e. in the callback function in each iteration of the training process where the reconstructed $J$ and $h$ are retrieved to compute the error between the reconstructed and true parameters. As this only happens when working with simulations, it can be advised to use the spin glass model when working with real world data.

Last but not least the same test from tab. 4.5 is run for spin glasses to find out how long they need until an accurate recreation is present. The test is executed for the same configurations only this time with the spin glass model. The times measured can be seen in tab. 4.8.

| Run ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 3 | 4 | 2 | 9 | 1 | 7 | 18 | 37 | 9 | 74 | 153 |

Table 4.8: Spin glass: Mean elapsed time for each run, determined according to 10 samples of the algorithm, until an accurate recreation of the parameters is accomplished with a threshold difference of 0.005 .

What can be seen is that the times from tab. 4.8 are lower than from tab. 4.6 but still higher than the time from tab. 4.5, which was expected as more parameters need to be learned. Unfortunately, the algorithm cannot benefit directly from this observation, as such accuracy measurements cannot be made in reality as already described in sec. 4.1.1.

## 4.2 Reconstruction fidelity

Now, as the runtime of the algorithm was discussed, the fidelity of the reconstruction can be approached. For this, the algorithm was run 34 times for each configuration on the spin glass model with various configurations. It was done for spin glasses with $J$ and $h$ from 4.1.2, as the Ising model can also be represented by the spin glass model, which is the more general case.

In fig. 4.1 and 4.2 the $J/h$ error of the reconstruction between the reconstructed and the true parameters can be seen in dependence of $\Delta t$ and the loss. The error was computed by subtracting the true and recreated parameters and taking the mean of the absolute values ($mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$ for the $J$ parameters and analogues for $h$ parameters where the absolute values are taken of the entries from the resulting matrix/vector). The runs range from system sizes of one qubit to twelve qubits and different $\Delta t$. The idea

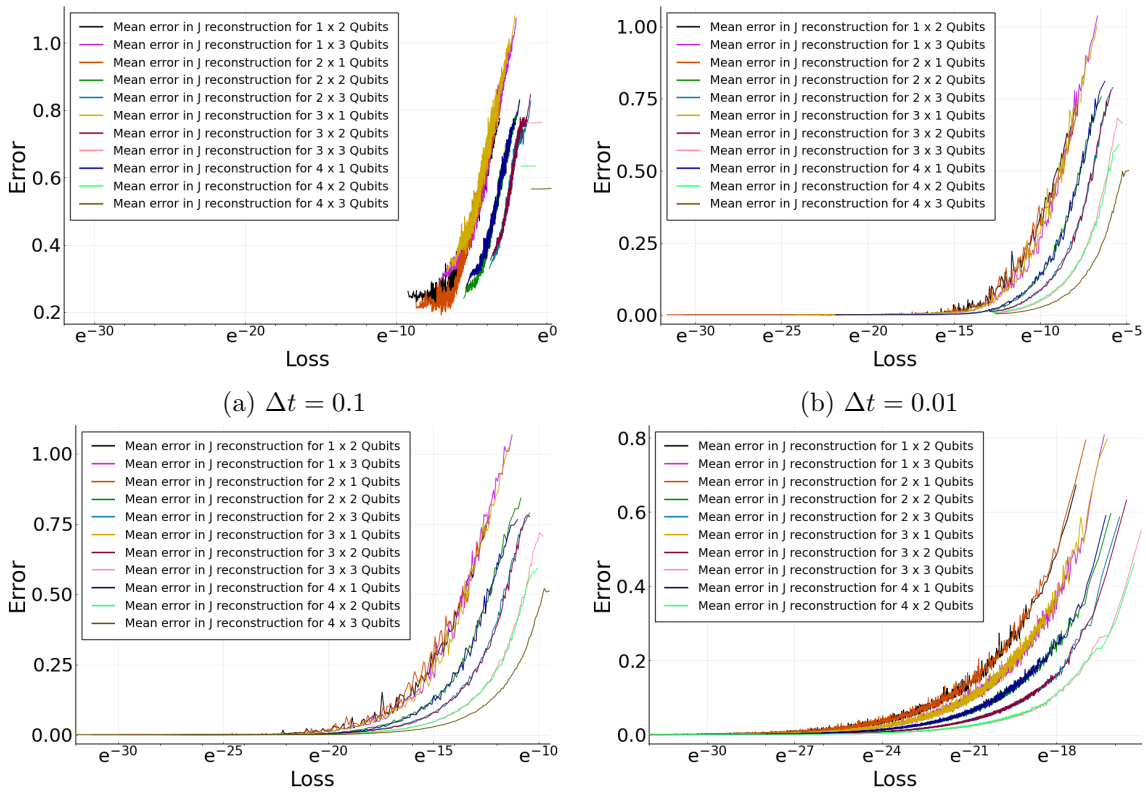
behind this test was to see if it is possible to judge on the basis of the loss if the wanted reconstruction accuracy is reached. This is done as the loss isn't a direct indicator if the reconstruction of the parameters, $J$ and $h$ are sufficient. It only shows if the probability distributions of the resulting states are similar, but not if the underlying Hamiltonians are similar. The configurations themselves aren't from interest, as it is only wanted to see if there is a correlation between loss and accuracy. If there is a correlation, this should also be seen for badly chosen parameters, as the loss would signalize that the wanted accuracy isn't reached.

As expected, the error for a 'high' $\Delta t$ is high as the approximation by the eq. (2.20) circuit is bad. This can be seen in fig. 4.1a as well as in 4.2a, where the error for the reconstruction of the J parameters is relatively low, but for the h high. The $J$-curve curves doesn't surpass the 0.2 mark and the $h$-error don't surpass the 1 mark on the y-axis. It can be seen that for the other $\Delta t$, the approximation seems to be sufficient, as here the errors drop nearly to zero. Concluding that a $\Delta t$ of 0.01 or lower, but preferably lower, should be sufficient enough for a good approximation with the current $J$ and $h$. If the true $J$ and $h$ should be higher, the $\Delta t$ has to get lower, as the approximation depends from both.

The reconstruction for one qubit (only the $h$ parameter gets reconstructed) shows high fluctuations (fig. 4.3a). With increasing system size, the error curve is shifted to the right, meaning that a sufficient reconstruction is reached earlier at a higher loss, but also that the run of the algorithm for this configuration has, in general, higher losses. Also, the curve is more stable with less fluctuations, which can be seen in figures 4.1 and 4.2. Actually, all other reconstructions, except for the one qubit system, have low fluctuations and produce a rather stable curve. The possible reason for this will be later discussed in this section.

It can be seen that the loss generally doesn't state if the reconstruction is sufficient enough, as the loss in the diagrams of fig. 4.1 and 4.2 are all really low and are getting averagely lower with lower $\Delta t$. In addition, the accuracy of the reconstruction also depends on the system size. That's why it isn't possible to judge, according to the loss itself, if the reconstruction is good for a general case as there are many parameters on which this depends. Due to this the error of $J$ and $h$ differs for each configuration at the same loss value. But if a specific case is looked at, it is possible to judge based on the loss if the reconstruction is good. The specific values can be taken from the figures 4.1, 4.2, 4.3 and 4.4. What generally can be said is that when the loss gets lower during the training process, the reconstruction gets better. If the loss dropped enough during the process, it can be assumed that the reconstruction accuracy is high, concluding that it is also generally possible to judge based on the loss development during the training process. Here, the loss dropped, on average, by a factor of $e^{-10}$ until a good reconstruction was given.

Another observation, which was expected, is that the error curves for $J$ and $h$ are nearly identical for systems of the type $(Nx \times Ny)$ or $(Ny \times Nx)$ with (width $\times$ height), as these are the same systems only rotated by 90° on the plane where the lattice lies. That's the case as the coupling strengths and the external magnetic field interaction, which is positioned orthogonal to the plane the lattice spans don't depend on the angle. The $h$-error curve for the systems $(2 \times 1)$-$(1 \times 2)$ and $(3 \times 1)$-$(1 \times 3)$ deviate from this somewhat in the figures 4.2c and 4.2d, but for the remaining systems this is the case. In Figure 4.1, this correlation can be seen at best. Moreover, it can be seen that in this test this also holds for systems of the same size, which can be seen in fig. 4.1 and 4.2 for the systems $(2 \times 2)$ and $(4 \times 1)$. This is an interesting observation as this is not evident because the amount and type of interaction terms of the systems are different, but still, the error curves are pretty similar.



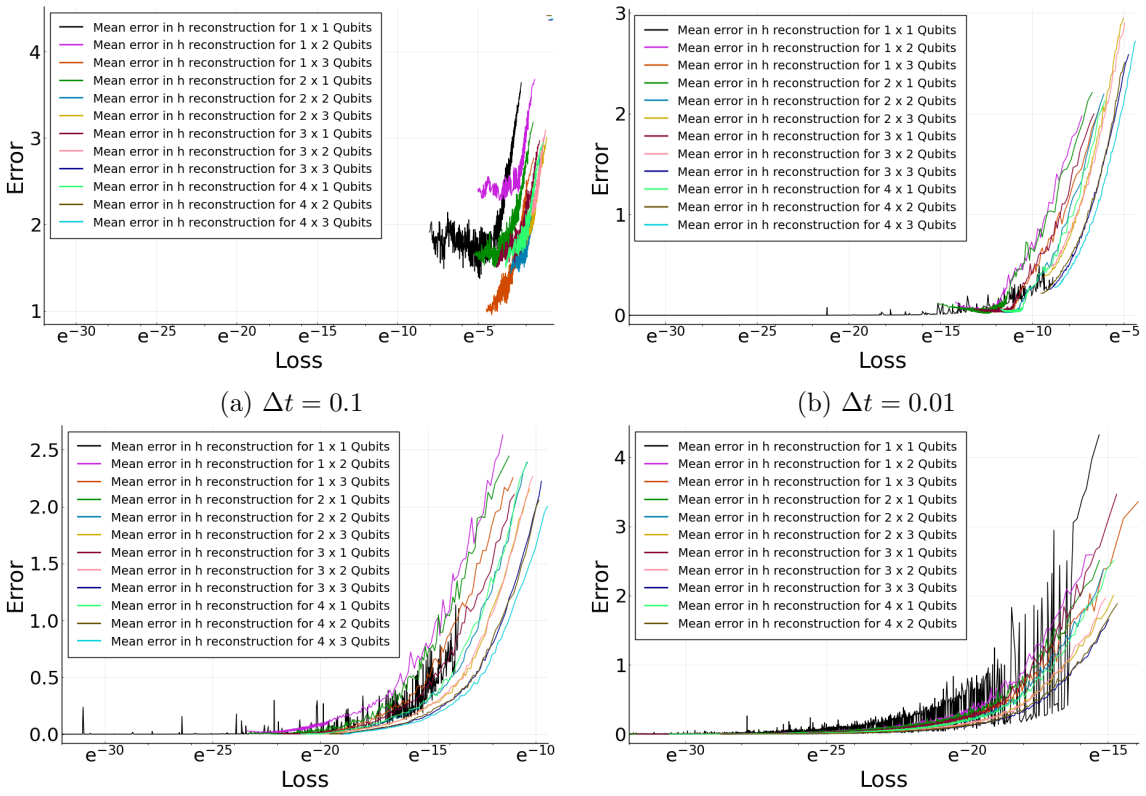

(a) $\Delta t = 0.1$

(b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$

(d) $\Delta t = 0.0001$

Figure 4.1: J error dependency of loss (KL-divergence) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting matrix. It can be seen that with decreasing loss, the error also decreases. A sufficient reconstruction (error $\approx 0$) can be found after a decrease by approximately a factor of $e^{-10}$ from the highest loss (loss at which the algorithm started). The reconstruction for 4.1a is the worst and doesn't reaches high accuracy. Also, with increasing system sizes, the curves are shifted to the right and the fluctuations in the curves decrease. Systems of the same size seem to have similar(/identical) error curves.




(a) $\Delta t = 0.1$

(b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$

(d) $\Delta t = 0.0001$

Figure 4.2: h error dependency of loss (KL-divergence) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|h_{\text{true}} - h_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting vector. A sufficient reconstruction (error $\approx 0$) can be found after a decrease by approximately a factor of $e^{-10}$ from the highest loss (loss at which the algorithm started). The reconstruction for 4.2a is the worst and doesn't reaches high accuracy. Also, with increasing system sizes, the curves are shifted to the right and the fluctuations in the curves decrease. Systems of the same size seem to have similar(/identical) error curves.

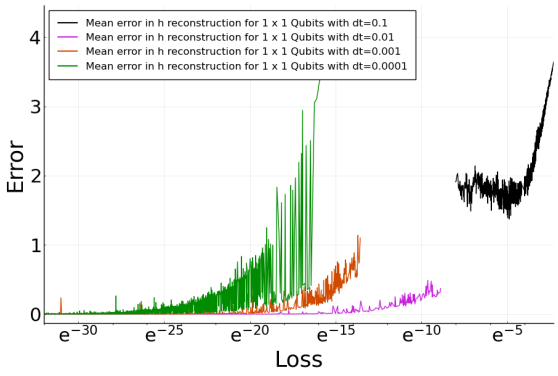
Another interesting thing is that this also could hold for systems of nearly the same size ($size_1/size_2 \approx 1$), as this holds for the systems ($4 \times 2$) and ($3 \times 3$) ($8/9 = 0.89$), but not for ($1 \times 3$) and ($2 \times 2$) ($3/4 = 0.75$). These are the reasons why, in the figures, there are often curves seen which appear in couples.

Further, in figures 4.3 and 4.4, the accuracy is compared for the same system sizes but different $\Delta t$. One can clearly see that with lower $\Delta t$, the loss is, on average, also lower. That's the case, as with lower $\Delta t$, the evolution doesn't change the state that much and due to this, the loss is also lower as it doesn't notice that much of a difference. Still, the behavior (alias the curves) for all cases is the same as all errors are getting lower with decreasing loss, only that the curves are shifted to the left for smaller $\Delta t$ and to right for bigger. This leads to the conclusion that for each system, the factor by how much the loss has to drop before a sufficient reconstruction is reached doesn't seem to depend on the set $\Delta t$, but is more or less constant around $e^{-10}$ (figures 4.3 and 4.4). The reconstruction for $\Delta t = 0.1$ is the worst, as already mentioned. And as already noted, with increasing system size, the curve is getting more stable with less fluctuations. This is most likely due to statistical effects as with growing system size the $J$ and $h$ parameters for spin glasses grow too, making the mean, which is taken in each iteration of the training, more sound as there are more samples considered, preventing a too big effect of outliers and fluctuations between taken data from different iterations and runs if there exists a correlation between loss and error (which does exist as it can be seen in the figures 4.1, 4.2, 4.3 and 4.4).

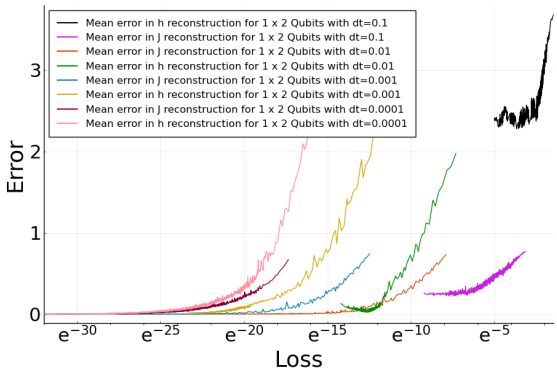
Another observation, which can be made from figures 4.3 and 4.4, is that the $J$ and $h$ error curves of the same $\Delta t$ are positioned in the same interval of the loss. Whereas the $J$-error curve has in general a lower slope than the $h$-error. Still, in the most cases, both curves reach a sufficient reconstruction at the same loss value. Concluding that, the h parameter has a higher mean error for the same loss values while reconstructing.

The best reconstruction accuracy for these configurations is given for $\Delta t = 0.001$ or $\Delta t = 0.0001$ as for $\Delta t = 0.01$, there is sometimes no optimal reconstruction accuracy (see i.e. fig. 4.4b or 4.4c). Interesting is also that for $\Delta t = 0.01$, in some cases, the reconstruction error starts to rise with decreasing loss, see fig. 4.3b, 4.3d, 4.3c, 4.4a, 4.3e, 4.4d or generally 4.1b and 4.2b.

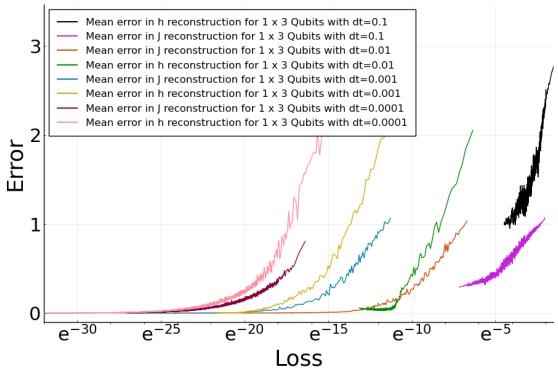
All in all, from the figures 4.1, 4.2, 4.3 and 4.4 can be concluded that there is a correlation between loss and the reconstruction accuracy of the $J$ and $h$ parameters. With decreasing loss, the error between the true parameters and the reconstructed parameters also decreases. After a sufficient decrease in the loss by an average factor of $e^{-10}$, an accurate reconstruction can be expected. Due to this, it is possible to judge, based on the development of the loss, if the reconstruction is sufficient. Also, it could be seen that the $J$-error curves (fig. 4.1) or the $h$-error curves (fig. 4.2) are almost the same for all configurations (figures 4.3 and 4.4) only shifted to right or left in the diagrams. From figures 4.1 and 4.2, it could also be seen that systems of the same or even similar system sizes produce nearly identical error curves. The best reconstruction could be seen for a $\Delta t = 0.001 \vee 0.0001$, whereas the fluctuations in the curves decreases with growing system sizes. While the run of the algorithm, the error in reconstruction for the $h$ parameter is larger than for the $J$ parameter for the same loss values. Still, at some point, the error for both parameters is the same. Therefore, the algorithm shouldn't be interrupted too early, even if the loss seems to be low enough.



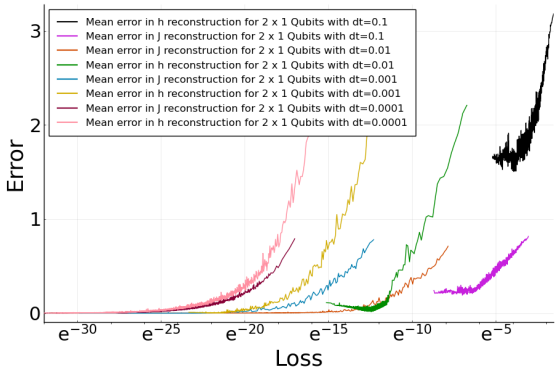

(a) $1 \times 1$ qubits



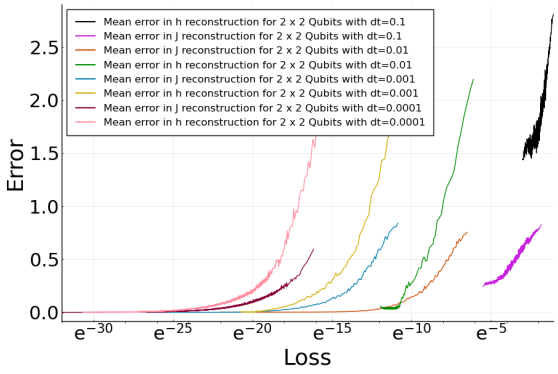

(b) $1 \times 2$ qubits



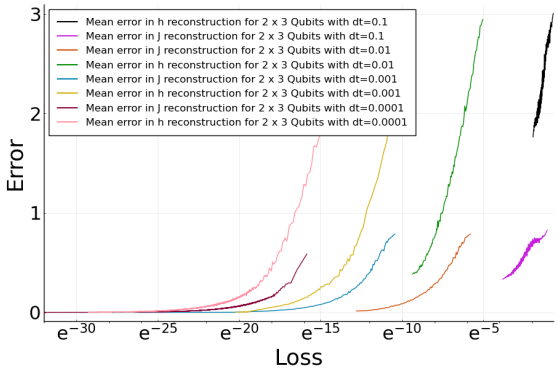

(c) $1 \times 3$ qubits




(d) $2 \times 1$ qubits

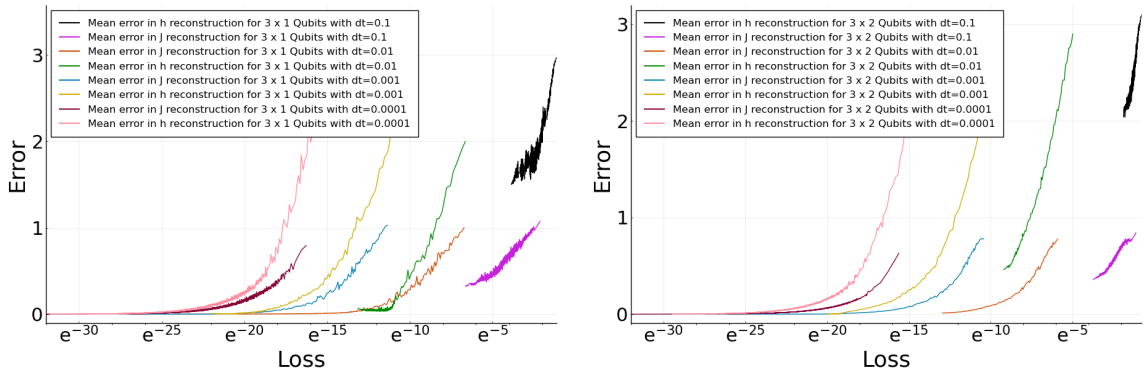



(e) $2 \times 2$ qubits




(f) $2 \times 3$ qubits

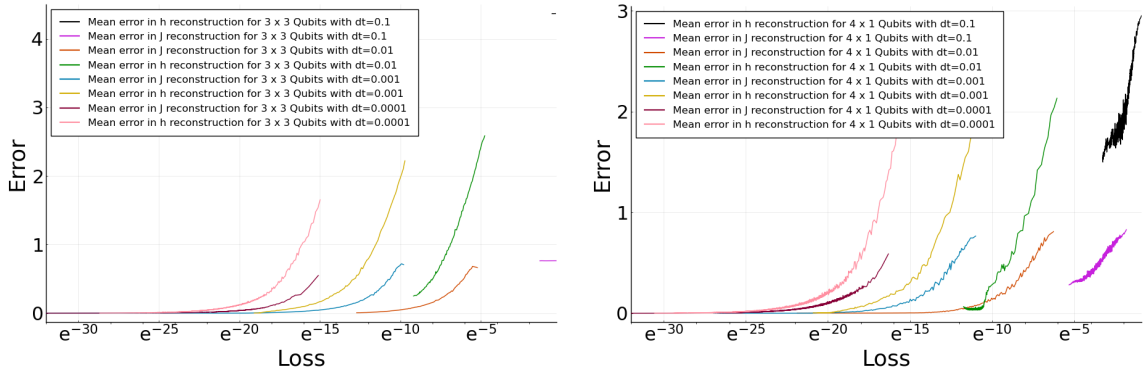
Figure 4.3: Error dependency of loss (KL-divergence) for the same number of Qubits with different $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$ for the $J$ parameters and analogues for $h$ parameters where the absolute values are taken of the entries from the resulting matrix/vector. A sufficient reconstruction (error $\approx 0$) can be found after a decrease by approximately a factor of $e^{-10}$ from the highest loss (loss at which the algorithm started). For $\Delta t = 0.1$, the reconstruction is the worst in all cases. Also, with increasing $\Delta t$, the according curves of each system are shifted to the left. Still, same systems have similar(/identical) error curves. It can be seen that the error curves for $h$ have higher slopes than for $J$, meaning that while reconstructing the error for $h$ is higher than for $J$ at the same loss value.




(a) $3 \times 1$ qubits



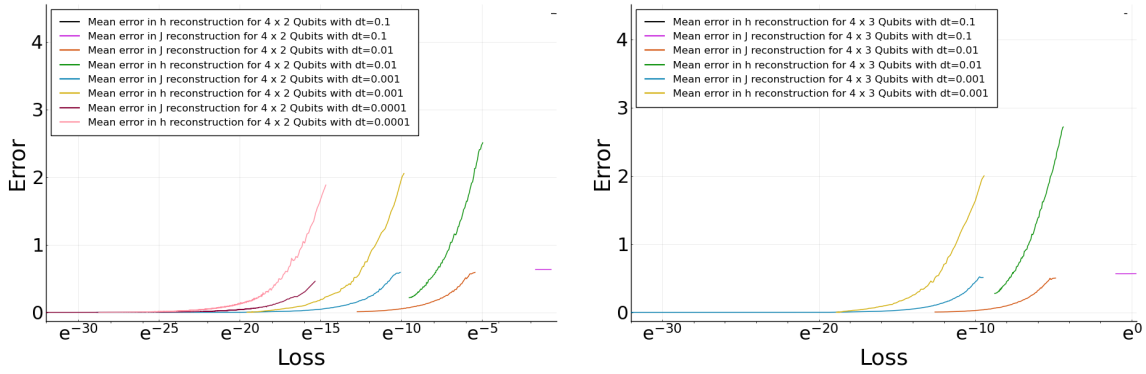

(b) $3 \times 2$ qubits




(c) $3 \times 3$ qubits




(d) $4 \times 1$ qubits




(e) $4 \times 2$ qubits




(f) $4 \times 3$ qubits

Figure 4.4: Error dependency of loss (KL-divergence) for the same number of Qubits with different $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$ for the $J$ parameters and analogues for $h$ parameters where the absolute values are taken of the entries from the resulting matrix/vector. A sufficient reconstruction (error $\approx 0$) can be found after a decrease by approximately a factor of $e^{-10}$ from the highest loss (loss at which the algorithm started). For $\Delta t = 0.1$, the reconstruction is the worst in all cases. Also, with increasing $\Delta t$, the according curves of each system are shifted to the left. Still, same systems have similar(/identical) error curves. It can be seen that the error curves for $h$ have higher slopes than for $J$, meaning that while reconstructing the error for $h$ is higher than for $J$ at the same loss value.

```
using Flux

mse(phi1,ref) = Flux.Losses.mse(abs2.(phi1), abs2.(ref))
mseHHH(phi1,ref) = Flux.Losses.mse(abs2.(hadamardsFront*phi1), abs2.(
    hadamardsFront*ref))
mae(phi1,ref) = Flux.Losses.mae(abs2.(phi1), abs2.(ref))
maeHHH(phi1,ref) = Flux.Losses.mae(abs2.(hadamardsFront*phi1), abs2.(
    hadamardsFront*ref))
huber_loss(phi1,ref) = Flux.Losses.huber_loss(abs2.(phi1), abs2.(ref))
huber_lossHHH(phi1,ref) = Flux.Losses.huber_loss(abs2.(hadamardsFront*
    phi1), abs2.(hadamardsFront*ref))
```

Figure 4.5: Different loss functions which were used. The loss functions with the HHH in the end were used in the first phase, the according loss function without the HHH in the second phase.

## 4.3 Different loss functions

In addition, different loss functions were used to test whether better results can be achieved. Therefore, the mean squared error (MSE), mean absolute error (MAE) and the huber loss [Hub64] were used and tested as possible loss functions.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.1}$$

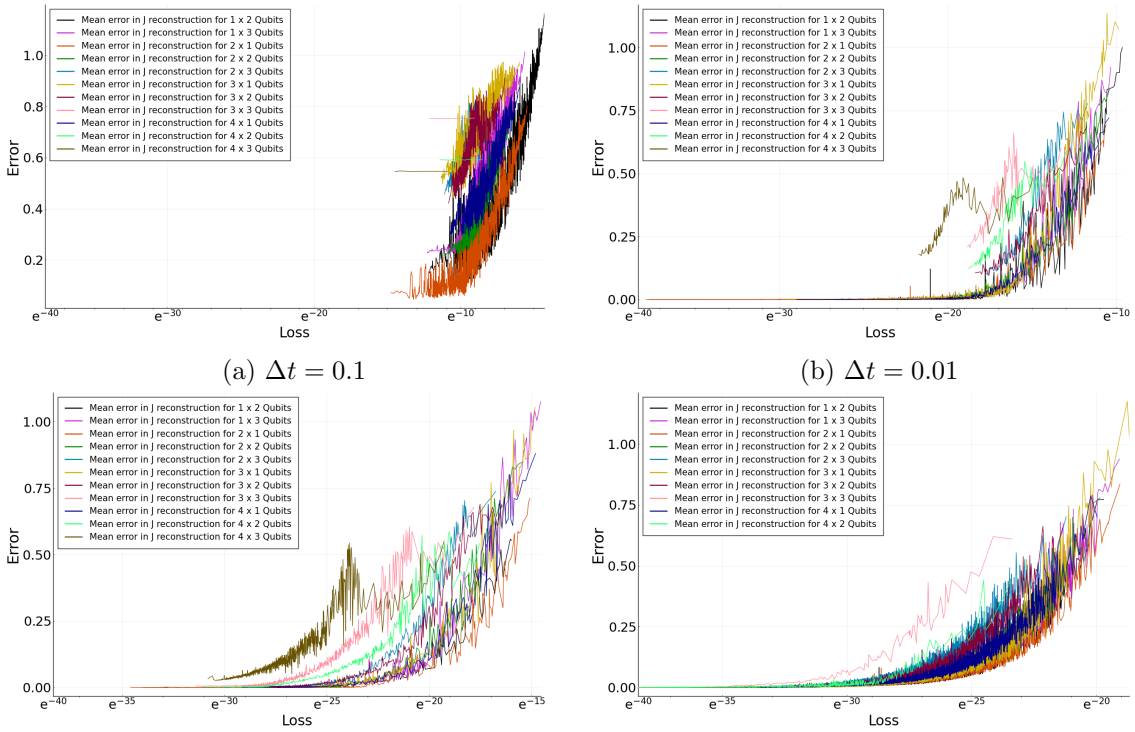
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{4.2}$$

$$\text{huber loss}_\delta(Y,\hat{Y}) = \begin{cases} \frac{1}{2}(Y-\hat{Y})^2 & \text{for } |Y-\hat{Y}| \leq \delta \\ \delta(|Y-\hat{Y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}. \tag{4.3}$$

In fig. 4.5 can be seen how they were selected and prepared in Julia.

The algorithm was then run for the same settings and configurations as in 4.2 with the only difference that this time, only 5 samples for each run were taken and analyzed. The results for these runs can be seen in fig. 4.6, 4.7, 4.8, 4.9, 4.10 and 4.11.

It can clearly be seen in all figures that the reconstructions for $\Delta t = 0.1$ are the worst. This could already be seen from the reconstructions with the KL-divergence. This underlines that the bad reconstruction for runs with $\Delta t = 0.1$ isn't due to the loss function, but as already written above (sec. 4.2) due to the bad approximation through the trotter formula (eq. 2.20). Also, the largest share of the curves is afflicted with moderate fluctuations, which can be led back with high probability, to the fact that only 5 samples for each test of a loss were taken. Also like the KL-divergence, the loss values for the different loss functions decreases with decreasing $\Delta t$. Coupled curves can also be seen in the figures 4.8, 4.7, 4.9 and 4.11 of the different loss functions, but not as clear as in fig. 4.1 and 4.2. In fig. 4.6 and 4.10 they are not seen that clearly as the fluctuations here are high and due this, the single curves cannot be distinguished that good. This is most likely due to the small sample sizes, which give rise to higher fluctuations. The interesting thing is that, on the contrary



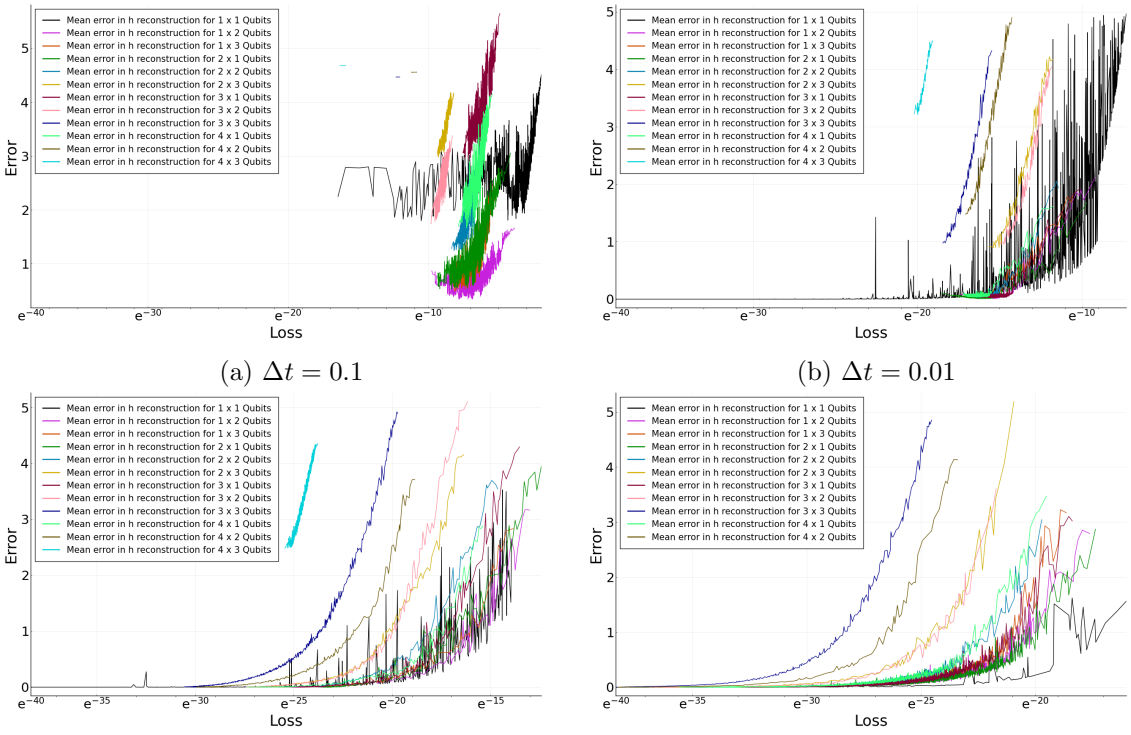

(a) $\Delta t = 0.1$

(b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$

(d) $\Delta t = 0.0001$

Figure 4.6: J error dependency of loss (mean squared error) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$ where the absolute values are taken of the entries from the resulting matrix. High fluctuations can be seen in the figures. This is most likely due to the small sample size of 5 or the high resolution of the y-axis. The reconstruction for $\Delta t = 0.1$ and $\Delta t = 0.01$ are the worst. With increasing system size, the curves are shifted to the left.




(a) $\Delta t = 0.1$

(b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$

(d) $\Delta t = 0.0001$

Figure 4.7: h error dependency of loss (mean squared error) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|h_{\text{true}} - h_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting vector. The reconstruction for $\Delta t = 0.1$ and $\Delta t = 0.01$ are the worst. In general, the reconstruction for the system $(4 \times 3)$ seems to be the worst here. With increasing system size, the curves are shifted to the left.

to fig. 4.1 and 4.2 in these figures, the curves of systems of similar sizes aren't coupled (i.e. $(3 \times 3)$ and $(2 \times 4)$).

In contrast to the KL-Divergence loss (fig. 4.1 and 4.2), for the figures 4.6, 4.7, 4.8, 4.10 and 4.11, it can be seen that with increasing system sizes the curves are shifted to the left instead to the right. Meaning that with increasing system size, the loss values are getting lower. This is probably due to that that basically the MSE, MAE and the huber loss compare the resulting probability distribution with the true probability just by taking the difference of both and square or take the absolute value of the result. By making the system larger, the probabilities to encounter a possible state gets averagely smaller. Therefore, the single summands of the losses get smaller and, when squaring, even smaller, as the probabilities and the corresponding differences are all below or equal 1.

For the tests of the MSE and the huber loss, it can be observed that the loss is generally lower than for the KL-divergence and the MAE. Also, both of these losses have a worse reconstruction for $\Delta t = 0.01$ compared to the KL-divergence and MAE. The bad reconstruction cannot be afflicted to a too small data batch size, as all the data batches had the same sizes for each according test run. Also, for the MSE and the huber loss, there are outliers in fig. 4.7b, 4.7c, 4.11b and 4.11c as the reconstruction for the $h$ parameter of $(4 \times 3)$ is inaccurate. The symmetry can be explained due to that the MSE and the huber loss have strong similarities per definition.

Also, the reconstructions of the $J$ parameters are averagely occupied with higher fluctuations, which can be seen by comparing the fig. 4.6, 4.8 and 4.10 with 4.7, 4.9 and 4.11. This can be due to statistical variance, as only 5 samples were taken into account to construct the mean.

The MAE loss seems to be the most accurate of the here tested losses, as it reaches the highest accuracies on average. Also, it produces the largest losses on average, which can be advantageous as this is more vivid.

All in all, it can be seen that these losses can be used for training, but in the tests, which were done here, they perform on average the same or worse than the KL-divergence loss. As it is worked with probability distributions due to the nature of quantum measurements, the KL-divergence might be the best loss function for this use case, as this was designed for comparing probability distributions.

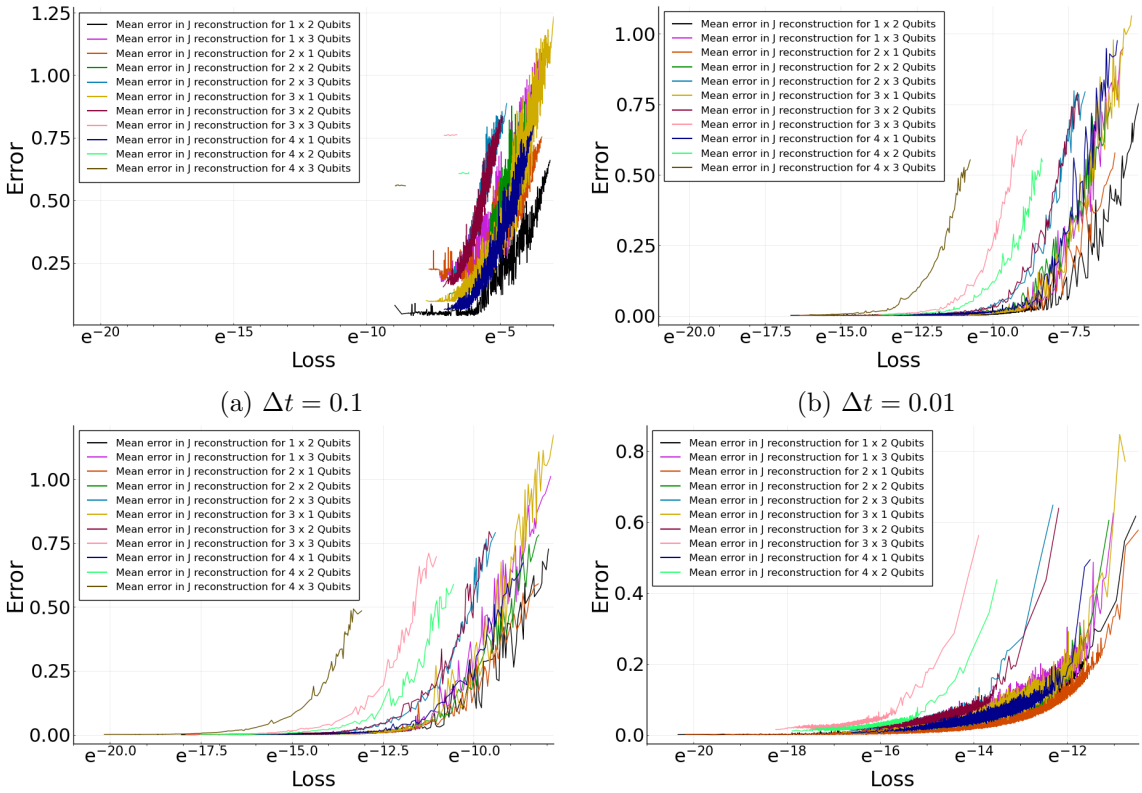
## 4.4 Different circuits for training

Furthermore, it was tested if it isn't possible to reconstruct the Hamiltonian with a more general parametrized quantum circuit and than later, maybe to use these for more general Hamiltonians. Therefore, variations of variational quantum circuits were build and tested. An example can be found at fig. 4.12, which can be applied several times to increase the expressiveness.

The results for this weren't sufficient enough as the reconstruction weren't accurate and 'wrong' Hamiltonians were reconstructed. Theoretically, this circuit should be able to represent with the correct parameters, any possible circuit $U$ [NC10]:

$$U = e^{i\alpha} R_{\vec{v}}(\Theta) \tag{4.4}$$

with $R_{\vec{v}}(\Theta)$ the general rotation gate. In fig. 4.12 the overall phase factors of eq. 4.4 aren't getting reconstructed, which wouldn't even be possible as the phase factors don't have



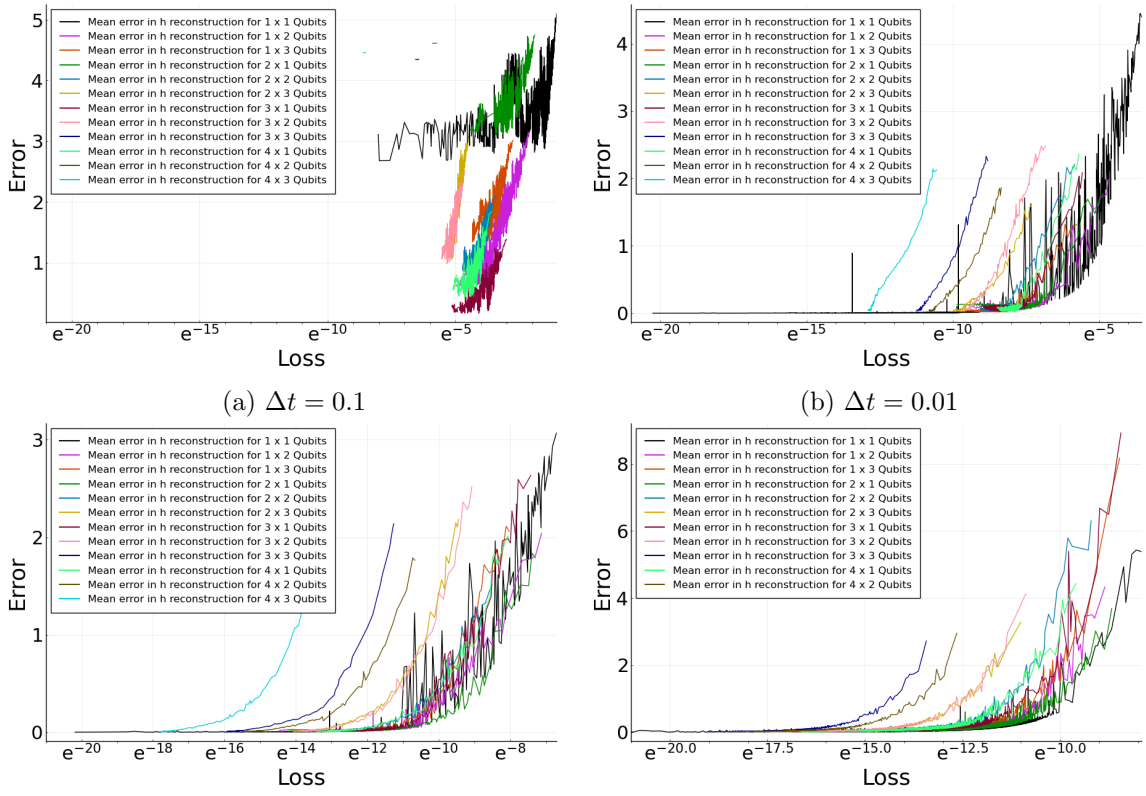

(a) $\Delta t = 0.1$

(b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$

(d) $\Delta t = 0.0001$

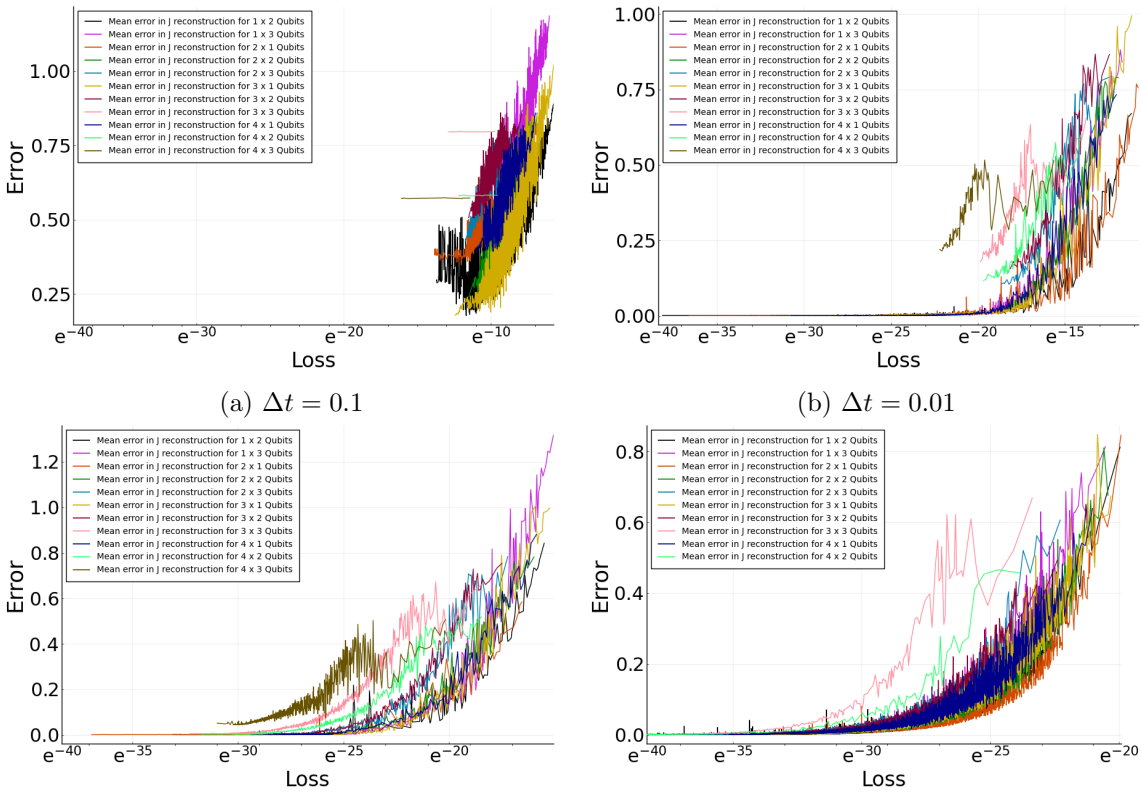
Figure 4.8: J error dependency of loss (mean absolute error) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting matrix. High fluctuations can be seen in the figures. This is most likely due to the small sample size of 5 or the high resolution of the y-axis. The reconstruction for $\Delta t = 0.1$ is the worst. With increasing system size, the curves are shifted to the left.




(a) $\Delta t = 0.1$ (b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$ (d) $\Delta t = 0.0001$

Figure 4.9: h error dependency of loss (mean absolute error) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|h_{\text{true}} - h_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting vector. The reconstruction for $\Delta t = 0.1$ is the worst. With increasing system size, the curves are shifted to the left.



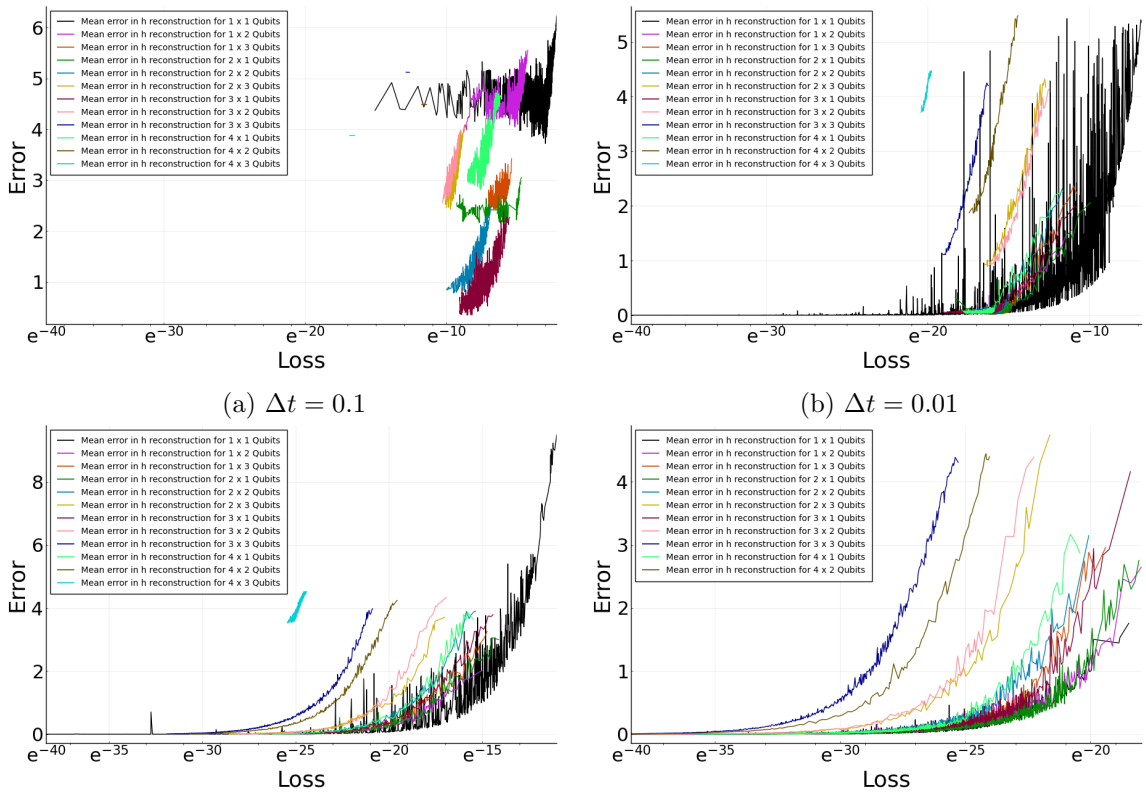

(a) $\Delta t = 0.1$

(b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$

(d) $\Delta t = 0.0001$

Figure 4.10: J error dependency of loss (huber loss) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|J_{\text{true}} - J_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting matrix. High fluctuations can be seen in the figures. This is most likely due to the small sample size of 5 or the high resolution of the y-axis. The reconstruction for $\Delta t = 0.1$ and $\Delta t = 0.01$ are the worst. With increasing system size, the curves are shifted to the left.




(a) $\Delta t = 0.1$ (b) $\Delta t = 0.01$

(c) $\Delta t = 0.001$ (d) $\Delta t = 0.0001$

Figure 4.11: h error dependency of loss (huber loss) for different number of Qubits with same $\Delta t$. The x-axis is scaled logarithmically. The error was computed as $mean(|h_{\text{true}} - h_{\text{reconstructed}}|)$, where the absolute values are taken of the entries from the resulting vector. The reconstruction for $\Delta t = 0.1$ and $\Delta t = 0.01$ are the worst. In general, the reconstruction for the system $(4 \times 3)$ seems to be the worst here. With increasing system size, the curves are shifted to the left.

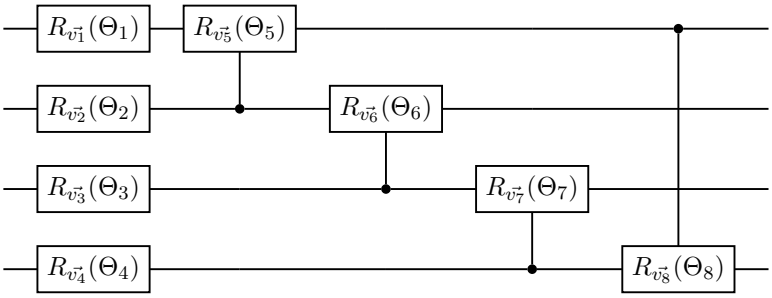



Figure 4.12: Variational quantum circuit for 4 qubits with $R_{\vec{v_n}}(\Theta_n)$ representing the general rotation gate. For more or less qubits, the circuit needs only to be scaled with the given structure.

an impact on the probability distribution. This is another reason why this approach isn't sufficient, as it could reconstruct the evolution operator up to a phase factor, which then has an impact on the reconstruction of the underlying Hamiltonian, as the reconstructed one will probably differ to the true one. Due to this, the question rises if the different Hamiltonians represent the same system sufficiently. Because as when the same probabilities arise, it doesn't mean that the underlying states are the same or that the systems are underlying the same evolution. But as this approach wasn't pursued, the question wasn't pursued either.

Also, due to the many degrees of freedom, the loss function has too many local optima for this kind of circuit in which the algorithm can get stuck. Due to this, it is very hard to use general circuits for general physical problems without having some knowledge about the system to which some restrictions can be applied.

Even for small system sizes i.e. $(2 \times 1)$, where the loss function should have less local optima (as there are less parameters), the algorithm reconstructed the evolution operator inaccurate and therefore the Hamiltonian wrong.

That's why the trotterized circuit was used as this pushes the algorithm already in a direction and imposes a structure which restricts the degrees of freedom as well as the occurrence of a phase factor. This results in less local optima and eases an accurate reconstruction.

# 5 Conclusion

In this work, an approach was proposed and investigated on how to do QPT on spin glasses (Ising model included) via time delayed measurements in combination with quantum computing. Therefore, the foundations which are needed for this are explained as well as the problem with the traditional QPT, as this scales exponentially for increasing system sizes. A representation was derived how an Ising model and then a spin glass evolution of a time step, can be described via a quantum circuit. Due to this it was possible to build parametrized quantum circuits where less parameters need to be reconstructed and which were optimized to resemble the evolution of the spin glass (Ising model) system, leading to a speed up as instead of $2^{4n}$ parameters of the $\chi$-matrix only $3n$ parameters at most need to be reconstructed. From the parameters of the quantum circuits, the according Hamiltonians then could be extracted in the end. This optimization was done on data, which consists of initial states and their evolutions. The machine learning algorithm is fed with the initial states and then the outcome probabilities are compared with the true probabilities of the evolutions. For the reconstruction, the time steps of the initial states need to be measured in the $\sigma^Z$ as well as in the $\sigma^X$-basis, due to unwanted symmetries. The proposed algorithm and its implementation was presented and it could be shown that it is feasible in a relatively low amount of time on common hardware for system sizes up to 12 qubits. Unfortunately, it wasn't possible to test it for more qubits, as the hardware wasn't capable to do this large amount of computations. For well chosen configurations, it could also be shown that the accuracy of the reconstruction is high enough. Also, different loss functions were compared and it could be found that the KL-divergence loss performed averagely as the best with the highest accuracy.

## 5.1 Future work

The algorithm was run on simulated data and therefore only little noise was present in the data. It would be interesting to see how the algorithm performs on real world data, as this will be more noisy. Still, it could be shown that, in theory, this algorithm is capable of performing QPT on spin glasses and therefore should, in theory, also work on real data. The next step would be to set up an experiment where data according to the evolution of some spin glass is gathered. Then the algorithm would be run on these data to be analyzed, and could be improved according to the results.

It would also be interesting to execute this algorithm in a hybrid form, where the quantum circuit is run on a real quantum computer and the optimization part is executed on a classical computer. This could lead to a significant increase in efficiency and therefore a decrease in run time, as the classical computer wouldn't be affected anymore by the large matrix/vector computations as this will be executed on the quantum computer. As a consequence, larger system sizes could be feasible as well. Unfortunately, quantum computers aren't yet developed enough so that sheer runs will already be afflicted by much noise, making the algorithm unfeasible. Still, this could well be a goal for the future.

Nevertheless, with regard to the future of quantum computing, the area of QPT will become more and more important as quantum hardware is improving rapidly and QPT is used in benchmarking of quantum hardware [TWA+20, BAH+10]. Maybe even a modified version of the proposed algorithm could be used to benchmark and characterize spin qubits [VE19], which are used as a foundation for quantum computers.

# List of Figures

# List of Tables

# Bibliography


[ABJ+03] Altepeter, J. B. ; Branning, D. ; Jeffrey, E. ; Wei, T. C. ; Kwiat, P. G. ; Thew, R. T. ; O'Brien, J. L. ; Nielsen, M. A. ; White, A. G.: Ancilla-Assisted Quantum Process Tomography. In: *Physical Review Letters* 90 (2003), May, Nr. 19. http://dx.doi.org/10.1103/physrevlett.90.193601. – DOI 10.1103/physrevlett.90.193601. – ISSN 1079–7114

[BAH+10] Bialczak, R. C. ; Ansmann, M. ; Hofheinz, M. ; Lucero, E. ; Neeley, M. ; O'Connell, A. D. ; Sank, D. ; Wang, H. ; Wenner, J. ; Steffen, M. ; al. et: Quantum process tomography of a universal entangling gate implemented with Josephson phase qubits. In: *Nature Physics* 6 (2010), Apr, Nr. 6, 409–413. http://dx.doi.org/10.1038/nphys1639. – DOI 10.1038/nphys1639. – ISSN 1745–2481

[BEKS17] Bezanson, Jeff ; Edelman, Alan ; Karpinski, Stefan ; Shah, Viral B.: Julia: A Fresh Approach to Numerical Computing. In: *SIAM Review* 59 (2017), Nr. 1, 65-98. http://dx.doi.org/10.1137/141000671. – DOI 10.1137/141000671

[BLSF19] Benedetti, Marcello ; Lloyd, Erika ; Sack, Stefan ; Fiorentini, Mattia: Parameterized quantum circuits as machine learning models. In: *Quantum Science and Technology* 4 (2019), Nov, Nr. 4, 043001. http://dx.doi.org/10.1088/2058-9565/ab4eb5. – DOI 10.1088/2058–9565/ab4eb5. – ISSN 2058–9565

[BMR+19] Bisognin, R. ; Marguerite, A. ; Roussel, B. ; Kumar, M. ; Cabart, C. ; Chapdelaine, C. ; Mohammad-Djafari, A. ; Berroir, J.-M. ; Bocquillon, E. ; Plaçais, B. ; al. et: Quantum tomography of electrical currents. In: *Nature Communications* 10 (2019), Jul, Nr. 1. http://dx.doi.org/10.1038/s41467-019-11369-5. – DOI 10.1038/s41467–019–11369–5. – ISSN 2041–1723

[CCG21] Cuvelier, Thibaut ; Combes, Richard ; Gourdin, Eric: *Statistically Efficient, Polynomial Time Algorithms for Combinatorial Semi Bandits.* 2021

[Dra20] Dral, Pavlo O.: Quantum Chemistry in the Age of Machine Learning. In: *The Journal of Physical Chemistry Letters* 11 (2020), Nr. 6, 2336-2347. http://dx.doi.org/10.1021/acs.jpclett.9b03664. – DOI 10.1021/acs.jpclett.9b03664. – PMID: 32125858

[EA75] Edwards, S F. ; Anderson, P W.: Theory of spin glasses. In: *Journal of Physics F: Metal Physics* 5 (1975), may, Nr. 5, 965–974. http://dx.doi.org/10.1088/0305-4608/5/5/017. – DOI 10.1088/0305–4608/5/5/017

[FIM+21] Ferrolho, Henrique ; Ivan, Vladimir ; Merkt, Wolfgang ; Havoutis, Ioannis ; Vijayakumar, Sethu: *Inverse Dynamics vs. Forward Dynamics in Direct Transcription Formulations for Trajectory Optimization.* 2021

[Gal99] In: Gallavotti, Giovanni: *Coexistence of Phases/Exactly Soluble Models*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999. – ISBN 978–3–662–03952–6, 175–231

[GM21] Gutiérrez, Irene L. ; Mendl, Christian B.: *Real time evolution with neural-network quantum states*. 2021

[GSW20] Greener, Joe G. ; Selvaraj, Joel ; Ward, Ben J.: BioStructures.jl: read, write and manipulate macromolecular structures in Julia. In: *Bioinformatics* 36 (2020), 05, Nr. 14, 4206-4207. http://dx.doi.org/10.1093/bioinformatics/btaa502. – DOI 10.1093/bioinformatics/btaa502. – ISSN 1367–4803

[Hub64] Huber, Peter J.: Robust Estimation of a Location Parameter. In: *The Annals of Mathematical Statistics* 35 (1964), Nr. 1, 73 – 101. http://dx.doi.org/10.1214/aoms/1177703732. – DOI 10.1214/aoms/1177703732

[Hus17] Hush, Michael R.: Machine learning for quantum physics. In: *Science* 355 (2017), Nr. 6325, 580–580. http://dx.doi.org/10.1126/science.aam6564. – DOI 10.1126/science.aam6564. – ISSN 0036–8075

[ISF+18] Innes, Michael ; Saba, Elliot ; Fischer, Keno ; Gandhi, Dhairya ; Rudilosso, Marco C. ; Joy, Neethu M. ; Karmali, Tejan ; Pal, Avik ; Shah, Viral: Fashionable Modelling with Flux. In: *CoRR* abs/1811.01457 (2018). https://arxiv.org/abs/1811.01457

[Isi25] Ising, Ernst: Beitrag zur Theorie des Ferromagnetismus. In: *Zeitschrift für Physik* 31 (1925), 253–258. http://dx.doi.org/10.1007/BF02980577. – DOI 10.1007/BF02980577

[KB17] Kingma, Diederik P. ; Ba, Jimmy: *Adam: A Method for Stochastic Optimization*. 2017

[KC01] Kraus, B. ; Cirac, J. I.: Optimal creation of entanglement using a two-qubit gate. In: *Phys. Rev. A* 63 (2001), May, 062309. http://dx.doi.org/10.1103/PhysRevA.63.062309. – DOI 10.1103/PhysRevA.63.062309

[KL51] Kullback, S. ; Leibler, R. A.: On Information and Sufficiency. In: *The Annals of Mathematical Statistics* 22 (1951), Nr. 1, 79 – 86. http://dx.doi.org/10.1214/aoms/1177729694. – DOI 10.1214/aoms/1177729694

[NC10] Nielsen, Michael A. ; Chuang, Isaac L.: *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. http://dx.doi.org/10.1017/CBO9780511976667. http://dx.doi.org/10.1017/CBO9780511976667

[OPG+04] O'Brien, J. L. ; Pryde, G. J. ; Gilchrist, A. ; James, D. F. V. ; Langford, N. K. ; Ralph, T. C. ; White, A. G.: Quantum Process Tomography of a Controlled-NOT Gate. In: *Phys. Rev. Lett.* 93 (2004), Aug, 080502. http://dx.doi.org/10.1103/PhysRevLett.93.080502. – DOI 10.1103/PhysRevLett.93.080502

[PG20] Physik Gruppe, Tohoku U. s.: *Two dimensional ising model.* http://www.cmpt.phys.tohoku.ac.jp/open-campus/2020/ising/Ising2D.png. Version: 2020

[Pre18] Preskill, John: Quantum Computing in the NISQ era and beyond. In: *Quantum* 2 (2018), Aug, 79. http://dx.doi.org/10.22331/q-2018-08-06-79. – DOI 10.22331/q–2018–08–06–79. – ISSN 2521–327X

[RKS+06] Riebe, M. ; Kim, K. ; Schindler, P. ; Monz, T. ; Schmidt, P. O. ; Körber, T. K. ; Hänsel, W. ; Häffner, H. ; Roos, C. F. ; Blatt, R.: Process Tomography of Ion Trap Quantum Gates. In: *Phys. Rev. Lett.* 97 (2006), Dec, 220407. http://dx.doi.org/10.1103/PhysRevLett.97.220407. – DOI 10.1103/PhysRevLett.97.220407

[Sch07] In: *Die Prinzipien der kanonischen Mechanik.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2007. – ISBN 978–3–540–71379–1, 79–169

[SIC13] In: Suzuki, Sei ; Inoue, Jun-ichi ; Chakrabarti, Bikas K.: *Introduction.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–33039–1, 1–11

[Sim15] Simoncini, Valeria: Krylov Subspaces. In: Higham, Nicholas J. (Hrsg.) ; Dennis, Mark R. (Hrsg.) ; Glendinning, Paul (Hrsg.) ; Martin, Paul A. (Hrsg.) ; Santosa, Fadil (Hrsg.) ; Tanner, Jared (Hrsg.): *The Princeton Companion to Applied Mathematics.* Princeton, NJ, USA : Princeton University Press, 2015, S. 113–114

[SLHS21] Shi, Naichen ; Li, Dawei ; Hong, Mingyi ; Sun, Ruoyu: RMSprop converges with proper hyper-parameter. In: *International Conference on Learning Representations*, 2021

[TWA+20] Torlai, Giacomo ; Wood, Christopher J. ; Acharya, Atithi ; Carleo, Giuseppe ; Carrasquilla, Juan ; Aolita, Leandro: *Quantum process tomography with unsupervised learning and tensor networks.* 2020

[VE19] Vandersypen, Lieven M. K. ; Eriksson, Mark A.: Quantum computing with semiconductor spins. In: *Physics Today* 72 (2019), Nr. 8, 38-45. http://dx.doi.org/10.1063/PT.3.4270. – DOI 10.1063/PT.3.4270

[zur10] zureks: *Spin glass model.* https://commons.wikimedia.org/wiki/File:Spin_glass_by_Zureks.svg#/media/File:Spin_glass_by_Zureks.svg. Version: 2010