Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Noah Leonardo Wach

born in Ludwigsburg (Germany)

2021

Data Re-Uploading on Qudits

This Bachelor Thesis has been carried out by Noah Leonardo Wach at the
Kirchhoff-Institute in Heidelberg
under the supervision of
Prof. Dr. Fred Jendrzejewski

Abstract

The aim of this thesis is to explore the capabilities of qudits in regard of quantum machine learning. After a general description of the dynamics of qudits, a brief explanation of how to realize qudits as a collective bosonic spin follows. Next, an introduction into the used quantum machine learning method follows, data re-uploading. The focus here lies on the realization of data re-uploading using qudits and what benefits and disadvantages this introduces. Additional to the theoretical description, the models will be simulated and trained on various data sets. The final training will be on parts of the famous “MNIST Handwritten Digits Data Set”. In the end, the results will be discussed and compared to a classical machine learning model.

Kurzfassung

Das Ziel dieser Bachelorarbeit ist es, die Möglichkeiten von Qudits im Zusammenhang mit dem Konzept des ”Quantum Machine Learning” zu erforschen. Nach einer kurzen Beschreibung von Qudits folgt eine Erklärung, wie diese mit einem kollektiven Spin von Bosonen realisiert werden können. Daraufhin folgt eine Einführung in das in dieser These verwendete Machine Learning Modell, data re-uploading. Anschließend wird das Modell von Qubits auf Qudits erweitert. Hier wird kurz auf die Vor- und Nachteile der Implementation von Qudits im Gegensatz zu Qubits eingegangen. Im Folgenden wird das theoretisch eingeführte Modell simuliert und auf verschiedene Datensätze trainiert. Für einen finalen Test wird das Modell versuchen Teile des bekannten “MNIST Handwritten Digits Data Set” zu lernen. Zum Schluss werden die Ergebnisse diskutiert und mit einem klassischen Machine Learning Modell verglichen.

Contents

Bachelor Thesis on Data Re-Uploading on Qudits

Noah Wach

1 Introduction

In 1980 Paul Benioff proposed a quantum model of a Turing machine [1] and thus set the starting point for a potentially revolutionizing new branch in physics and computer science. Richard Feynman later suggested that the proposed model of a quantum computer can outperform classical computers [2] in certain tasks. After the theoretical foundation was laid, Peter Shor proposed that an algorithm run on a quantum computer can solve the NP-hard problem of prime-factorization in polynomial time [3]. Not only the prime-factorization problem but also searching an unsorted database can be less complex using quantum computers [4].
It took almost 40 years until this proposition came true; in 2019 scientists from Google claimed to have reached quantum supremacy [5]. Only 200 seconds of computation were needed to solve a problem a classical supercomputer would need 10.000 years to compute[1]. Even after this milestone a powerful and fault tolerant quantum computer which can solve a variety of problems is still in the distant future. The quantum computers available today belong to the so called NISQ (Noisy-Intermediate-Scale-Quantum) devices [7]. The question that arises is what computations can be done using NISQ devices.

To build a quantum computer one needs highly controllable quantum systems, which can interact with each other. The most widely used approach in this regard are superconducting qubits [8], which is also the technology Google used in its quantum supremacy experiment [5]. Other realizations such as trapped ions [9], photons [10] or cold atoms [11] are also being pursued by researchers around the world.

The explosion in popularity of quantum computers has given rise to many fields that try to utilize the computational advantages

1: It was later discovered that a modern supercomputer using tensornetworks can indeed outperform the quantum computer Google used [6].

quantum computation promises. Some of these fields are quantum cryptography, quantum metrology, quantum simulation and lastly quantum machine learning on which this thesis will lay its focus. Quantum machine learning is the bridge between the power of classical machine learning and the usage of quantum systems to speed up certain processes. It refers to three different branches, namely studying a quantum system using classical machine learning algorithms (1), using quantum algorithms (2) and investigating classical data using quantum algorithms (3). In this thesis the latter (3) will be discussed.

Since quantum computers only excel classical computers in certain types of applications, many quantum machine learning models still rely on some sort of classical computation. This ensures that the resulting hybrid model can take advantage of the strengths of both, the classical and quantum methods [12].
As already mentioned, reliable quantum computers are still in the distant future. The issue quantum machine learning is facing is whether models can be trained using today's NISQ devices and how they perform compared to purely classical models.

Outline of the thesis

This thesis investigates the quantum machine learning model data re-uploading and expands the qubit approach onto qudits. This method can be run on today's NISQ device, since it only requires very few qubits.

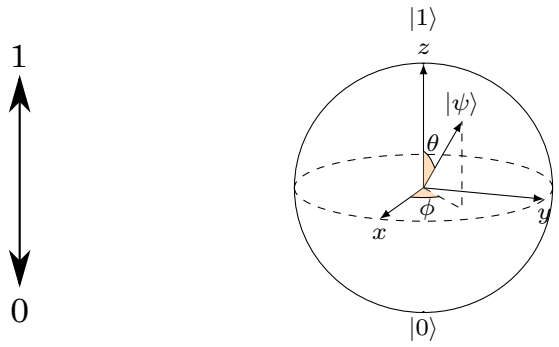
- **Section 2** will lay the theoretical foundation of collective spins treated as qudits. Additionally, a way to visualize the quantum state of a qudit is introduced.
- The investigated method of data re-uploading is explained in **Section 3**. This method is then extended on the earlier presented concept of qudits.
- In **Section 4** the quantum machine learning model is first trained on simple data sets. As a final test the model will learn parts of the MNIST Handwritten digits data set. Lastly the model is tested against a classical machine learning algorithm.
- The results from the simulations are discussed in **Section 5** and an outlook on further improvements and on the experimental implementation is given.

2 Theory

In the following the concept of qudits will be presented and compared to the more standard approach of using qubits. First, the mathematical foundation of qudits is introduced, while there will also be a brief discussion on how to realize a qudit experimentally with cold atoms. The second part will focus on the visualization of quantum states of the qudit and how to squeeze a collective spin.

2.1 Bits vs Qubits vs Qudits

The smallest processing unit of classical computer is a bit, which can either occupy the state 0 or 1 (**Figure 1a**). Contrary to this approach, a conventional quantum computer uses so called "qubits". Qubits can not only be in state 0 or 1, but also in any arbitrary superposition of those two states. This fact makes quantum computers extremely fast in some specific calculations. The state of a qubit can be represented by a vector on a Bloch sphere, which was introduced by physicist Felix Bloch. This representation is quite useful in the context of quantum computing, since it illustrates the similarities between classical bits and qubits and also visualizes the concept of superposition. **Figure 1b** shows a Bloch sphere.



(a) Classical bit

(b) Qubit

Figure 1: Comparison of a classical and a quantum bit.

The state of the qubit can be described as a vector of length 1 and the two independent variables θ and ϕ.

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|1\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|0\rangle \tag{1}$$

Here $|\psi\rangle$ is a vector in the 2-dimensional Hilbert space $\mathbb{C}^2$ with the orthonormal basis $\{|0\rangle, |1\rangle\}$.
A qudit is an expansion of the idea of the qubit. It is not represented by a vector in the 2-dimensional Hilbert space, but a vector in the d-dimensional Hilbert space $\mathbb{C}^d$ with the orthonormal basis $\{|0\rangle, |1\rangle, ... |d-1\rangle\}$.

The state of a qudit can therefore be written similar to **Equation 1**[13]:

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle + ...c_{d-1} |d-1\rangle = \begin{pmatrix} c_0 \\ c_1 \\ ... \\ c_{d-1} \end{pmatrix} \tag{2}$$

with $\sum_{i=0}^{d-1} |c_i|^2 = 1$.

2.2 Qudits as a Collective Spin

There are many approaches to realize a qudit experimentally such as ions [14] or photons [15], however this thesis will focus on the realization using cold atoms and treat the qudit as a collective spin [11].

Lets consider N_A bosonic atoms in an optical tweezer. The many body system can efficiently be described by the creation and annihilation operators in second quantization, $\hat{a}_m$ and $\hat{a}_m^\dagger$. Here m denotes the internal states 0 and 1 of the atom[2]. A collective spin is formed by the atoms via the Schwinger representation:

2: For example the spin or the hyperfine state of the atom.

$$\hat{L}_z = \frac{1}{2}[\hat{a}_1^\dagger \hat{a}_1 - \hat{a}_0^\dagger \hat{a}_0] \tag{3a}$$

$$\hat{L}_+ = \hat{a}_1^\dagger \hat{a}_0 \tag{3b}$$

$$\hat{L}_- = \hat{a}_0^\dagger \hat{a}_1 \tag{3c}$$

$$\hat{L}_x = \frac{1}{2}[\hat{L}_+ + \hat{L}_-] \tag{3d}$$

$$\hat{L}_y = \frac{-i}{2}[\hat{L}_+ - \hat{L}_-] \tag{3e}$$

The eigenstates of $\hat{L}_z$ are indicated by $|m_l\rangle$, where m_l is a (half) integer spin and is ranging from $m_l = -l, ..., l$ with $l = \frac{N_A}{2}$. The computational basis can be expressed with $|j\rangle \equiv |-l + j\rangle$ and $j = 0, 1, ..., N_A$ [3]. The operators $\hat{L}_i$ are a $2l+1$ dimensional representation of the generators of the $SU(2)$ Lie algebra. The commutators of the operators are the following:

3: The collective spin can now be brought back to a qubit structure with $N_A = 1$, which leads to $\{|-1/2\rangle, |1/2\rangle\}$ as the basis for a 2-dimensional Hilbert space.

$$[\hat{L}_i, \hat{L}_j] = i\epsilon_{ijk}\hat{L}_k \tag{4}$$

For the case of $l = 1$ the operators are:

$$\hat{L}_x = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \hat{L}_z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \hat{L}_z^2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

A given state $|\psi_0\rangle$ evolves while the operators act on it for a certain time. The time evolution is described via the Hamiltonian[4], which was experimentally realized in [16], containing $\hat{L}_x$, $\hat{L}_z$ and $\hat{L}_z^2$.

$$H = \chi \hat{L}_z^2 + \Delta \hat{L}_z + \Omega \hat{L}_x \tag{5}$$

The Hamiltonian is able to generate all unitary operations if the variables χ, Δ and Ω can be tuned independently. In the following these parameters are either set to 0 or 1, to obtain a gate based approach[5]. The Hamiltonian acts on the state via the time evolution operator.

$$|\psi(\theta)\rangle = e^{-i\theta H} |\psi\rangle \tag{6}$$

The following notation for the different independently tuned Hamiltonians is introduced at this point:

$$\hat{R}_x(\theta) = e^{-i\theta L_x} \tag{7a}$$
$$\hat{R}_z(\theta) = e^{-i\theta L_z} \tag{7b}$$
$$\hat{R}_z^2(\theta) = e^{-i\theta L_z^2} \tag{7c}$$

To achieve a universal set of quantum gates[6], at least one entangling gate is needed, in this case this is the $\hat{R}_z^2(\theta)$ gate. The set of these three operations[7] represents a universal set of quantum gates [11].

4: It is also called the one-axis twisting Hamiltonian.

5: Instead of thinking of the operations on the qudit as the Hamiltonian acting on the collective spin via microwave pulses one can think of it as gates (**Equation** 7a) acting on the qudit (similar to logic gates in electrical engineering).

6: A set of quantum gates is universal if any unitary operation can be approximated to an arbitrary accuracy using these gates[17].

7: $\hat{R}_x \hat{R}_z \hat{R}_z^2$

2.3 Qudits on a Bloch Sphere

Similar to the states of a qubit the states of a qudit can also be visualized by a vector on the Bloch sphere[8] as it is illustrated in **Figure** 2. Nevertheless, it makes more sense to visualize a quasi probability distribution on the Bloch sphere, namely the Husimi Q distribution[18]. In contrary to the 2-dimensional Hilbert space of the qubit, the qudits Hilbert space dimension is higher and can thus not entirely be represented on the surface of a sphere. For a d-dimensional Hilbert space a representation with $d-1$ independent variables is needed.

8: In this case the vector would show to the point with the highest probability but would not represent the entire probability distribution.

2.3.1 Husimi Q Distribution

The Husimi-Q distribution is the trace of the density matrix $\hat{\rho} = |\psi\rangle \langle\psi|$ over the basis of the coherent states $|q\rangle$.

$$|q(\theta, \phi)\rangle = \sum_n c(\theta, \phi) |n\rangle \tag{8}$$

The states $|q\rangle$ depend on the two independent variables ϕ and θ, which make up the Bloch sphere.

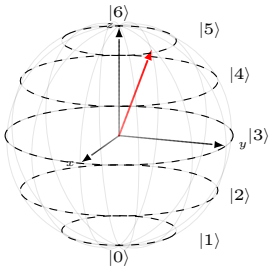


Figure 2: Qudit with $d = 7$

$|n\rangle$ are the basis states of the qudit. The formula for the Husimi Q distribution is:

$$Q(q) = \frac{1}{\pi} \langle q|\hat{\rho}|q\rangle \tag{9}$$

When evaluating the Husimi Q distribution over $\phi \in [0, 2\pi)$ and $\theta \in [0, \pi]$ it can be mapped on a sphere. This is comparable to the Bloch sphere representation for qubits.

2.3.2 Qudits and Squeezing

The entanglement for qudits is achieved via squeezing, which is done by applying $\hat{L}_z^2$. If a state is squeezed in z-direction, the variance of $\hat{L}_z$ gets smaller while the variance in $\hat{L}_x$ or $\hat{L}_y$ gets larger. This is due to the Heisenberg uncertainty principle.

$$\Delta\hat{L}_i \Delta\hat{L}_j = \frac{1}{2} \tag{10}$$

In [16] it is demonstrated that squeezed qudit states can achieve a variance below the shot noise. **Figure 3a** shows the experimental data from [16] and the simulated data[9] using the spin matrices from **Section 2.2**. Squeezing proves to be very important later on, since it can be used to improve the classifier[10].

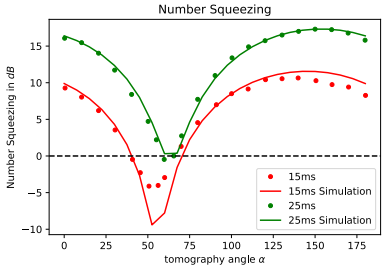
9: The black line indicates the shot noise.

10: This is shown in **Section 5.1.4**.

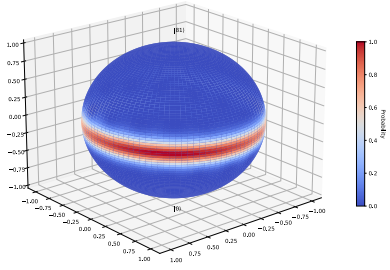
2.3.3 Qudits and Spin Matrices

Figure 3b shows a squeezed state. Here, red indicates a high probability of finding the qudit in this state, whereas blue indicates a low probability. This state is the result of applying a rotation around the y-axis to reach an unstable fixed point at the equator[11]. Then the one-axis twisting Hamiltonian (**Equation 5**) acts on the state with $\Omega = 0$. As a final step the state is rotated around the x-axis to obtain a non-Gaussian state.

11: Similar to the state seen in **Figure 4b**.



(a) Simulation of the Data in [16]



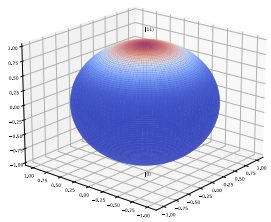
(b) Probability distribution of a squeezed non-Gaussian qudit state.

Figure 3: Squeezed state for a collective spin with variance below the shot noise.

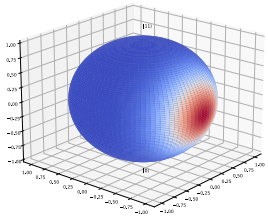
To get a better understanding of what effects the spin matrices have on the qudit, **Figure 4** shows the evolution of a qudit under the following operations.

$$|\psi\rangle = \hat{R}_z^2(\pi/15)\hat{R}_z(\pi/4)\hat{R}_x(\pi/2)\,|d-1\rangle \tag{11}$$

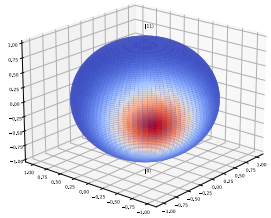
(a) The qudit starts in the position of the collective spin up $|d-1\rangle$ (**Figure 4a**).
(b) This state evolves under $\hat{R}_x$ to a coherent superposition of the basis states. The probability distribution is located on the equator (**Figure 4b**).
(c) Then $\hat{R}_z$ acts on it, which rotates the distribution around the z-axis (**Figure 4c**).
(d) Finally $\hat{R}_z^2$ squeezes the state (**Figure 4d**).



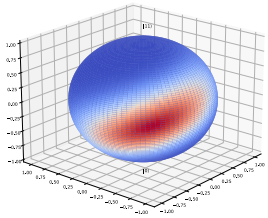
(a) Basis state ($|d-1\rangle$)



(b) State after applying $\hat{R}_x(\pi/2)$



(c) State after applying $\hat{R}_z(\pi/4)$



(d) State after applying $\hat{R}_z^2(\pi/15)$

Figure 4: Illustration of an evolving qudit state

To get a variance below the shot noise, the state has to be rotated again around the x-axis. This results in a state similar to the one shown in (**Figure 3b**). Here the variance in $\hat{L}_z$ is below the shot noise, while the variance in $\hat{L}_x$ and $\hat{L}_y$ are much greater than the shot noise.

In conclusion, qudits are represented by a vector in the d-dimensional Hilbert space. A qudit can be realized experimentally using cold atoms where the individual spins of the atoms are treated as a collective spin. Entangling the individual spins allows to reach a variance below the shot noise in one measurement basis. This is called squeezing which is beautifully visualized by the Husimi Q distribution on a Bloch sphere.

3 Data Re-Uploading

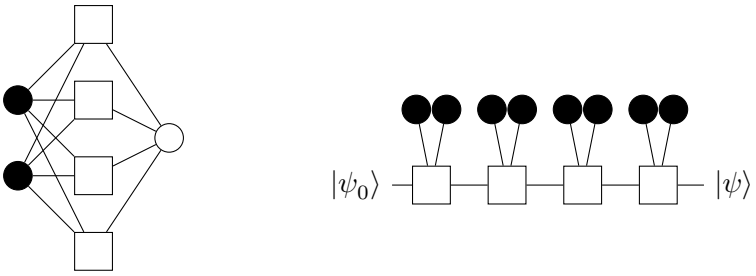
In theory, most quantum algorithms need a number qubits or qudits well beyond the capabilities of current experimental platforms to run on instances of practical interest. Since a quantum computer with a large number of qubits/qudits, a high fidelity and a low error rate has yet to be built, the question arises what algorithms can be run on the quantum systems that are present at the moment, notably NISQ (Noisy-Intermediate-Scale-Quantum) devices [7]. Data re-uploading charges this problem at the extreme, A. Pérez-Salinas introduces the idea of only using **one** qubit and states that this, in combination with a classical subroutine, is sufficient to built a universal quantum classifier [19]. Data re-uploading maps the input vector $\vec{x}$ via rotations onto the qubit. A second set of operators rotates the state onto target states. This process is repeated multiple times, hence the name data re-uploading. Data re-uploading belongs to the class of Variational Quantum Algorithms [20].
This section of the thesis will introduce the fundamentals of data re-uploading and will later expand this approach onto qudits.

3.1 Data Re-Uploading on Qubits

The first problem in only using one qubit to construct a universal classifier is the processing and encoding of the classical input data onto the qubit. For once, a qubit has only two degrees of freedom[12] so no quantum classifier in higher dimensions can be created. Secondly once the input is loaded onto the qubit the only possible operations are Bloch sphere rotations. Due to these two limitations, it is not possible to classify any non-trivial patterns, with a single data input.

12: This is illustrated nicely in **Equation 1**, here the degrees of freedom are θ and ϕ.

The solution to both of these problems is inspired by neural networks. Here the input is processed multiple times before it reaches the output layer. This motivated the idea to introduce the input data several times during the classification process.



(a) Simple Neural Network **(b)** Quantum Classifier

Figure 5: Comparison of a classical neural net and a quantum classifier using data re-uploading. The black circles indicate the input data.

In [19] it is proven that with data re-uploading a single qubit classifier can classify any classification function to an arbitrary accuracy.

3.1.1 Circuit Structure

The input vector $\vec{x}$ is introduced as a rotation on the Bloch sphere. Assuming an input vector of dimension d, the data is mapped onto the qubit as d independent rotations. For a three dimensional input vector the unitary operation applied onto the qubit would be the following:

$$\mathcal{U}(\vec{x}) = \hat{R}_i(x_1)\hat{R}_j(x_2)\hat{R}_i(x_3) \tag{12}$$

Due to the Euler angles, only two different rotational operations are needed to make an arbitrary rotation. These are denoted as i and j. The unitary $\mathcal{U}(\vec{x})$ is referred to as the input rotation.

Additional to the input rotations, an operation with learnable parameters is needed, to create a classifier. These parameters are comparable to the weights in a neural network and are denoted as $\vec{\theta}_n$. Rotations stemming from $\vec{\theta}_n$ have a similar structure as the input rotation in **Equation 12**. The following equation shows a unitary operation composed of three independent variables.

$$\mathcal{U}(\vec{\theta}_n) = \hat{R}_i(\theta_{n_3})\hat{R}_j(\theta_{n_2})\hat{R}_i(\theta_{n_1}) \tag{13}$$

In the following the unitary operator $\mathcal{U}(\vec{\theta}_n)$ is referred to as the parameter rotation. $\mathcal{U}(\vec{\theta}_n)$ and $\mathcal{U}(\vec{x})$, the two unitary operations, make up a processing layer.

$$\mathcal{L}(\vec{x}, \vec{\theta}_n) = \mathcal{U}(\vec{\theta}_n)\mathcal{U}(\vec{x}) \tag{14}$$

$\mathcal{L}(\vec{x}, \vec{\theta}_n)$ can be applied multiple times one after another with different $\vec{\theta}_n$.

$$\mathscr{U}(\vec{x}, \vec{\theta}) = \mathcal{L}(\vec{x}, \vec{\theta}_n)...\mathcal{L}(\vec{x}, \vec{\theta}_1) = \mathcal{U}(\vec{\theta}_n)\mathcal{U}(\vec{x})...\mathcal{U}(\vec{\theta}_1)\mathcal{U}(\vec{x}) \tag{15}$$

As the expressivity of the circuit grows the more accurate the classifier will be. If the input vector $\vec{x}$ has a higher dimension, additional rotation gates are added to the input rotation, such that rotation i and j alternate.

Every time the circuit is executed the qubit starts in state $|0\rangle$. From this starting point each unitary operation is applied to obtain the final state $|\psi(\vec{\theta}, \vec{x})\rangle$.

$$|\psi(\vec{\theta}, \vec{x})\rangle = \mathscr{U}(\vec{x}, \vec{\theta})\,|0\rangle = \mathcal{L}(\vec{x}, \vec{\theta}_n)...\mathcal{L}(\vec{x}, \vec{\theta}_1)\,|0\rangle \tag{16}$$

3.1.2 Loss Function and Gradient Descent

Having a similar approach to classifying as a classical neural network, there is also the need for a loss function and a classical minimization method. First a loss function has to be defined. To increase a models performance the loss function needs to be minimized, which is done via the gradient of the loss function. The gradient tells the optimizer how to adjust the parameters such that

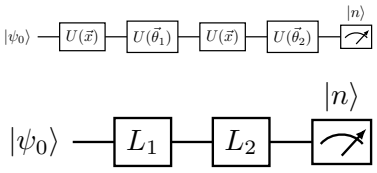


Figure 6: Circuit Diagrams for a 2 layer classifier.

the loss function is minimized. The parameters to optimize are in this case the learnable rotations $\mathcal{U}(\vec{\theta}_i)$.

The goal of the rotations introduced in **Section 3.1.1** is to rotate the qubit state on a specific spot on the Bloch sphere. In other words: Maximize the fidelity of the resulting state after the rotations and a specific point on the Bloch sphere. For each input vector $\vec{x}_n$ the following equation needs to be maximized:

$$\mathcal{F}_n(\vec{\theta}) = |\langle\tilde{\psi}_s|\psi(\vec{\theta},\vec{x}_n)\rangle|^2 \tag{17}$$

Here $|\tilde{\psi}_s\rangle$ corresponds to the label state of the class of data point $\vec{x}_n$. Since the gradient descent optimizer "Adam"[21] is used, the equation needs to be put in such a form, that the desired outcome corresponds to minimizing the equation. This leads to:

$$\chi^2_f(\vec{\theta}) = \sum_{n=1}^{M}(1 - |\langle\tilde{\psi}_s|\psi(\vec{\theta},\vec{x}_n)\rangle|^2) \tag{18}$$

M denotes the size of the data set with input vectors $\vec{x}_n$, so the sum over each input vector needs to be minimized. In the most simple example the classifier is binary, such that the label classes are $|0\rangle$ and $|1\rangle$. The circuit maximizes the fidelity such that an input $\vec{x}_j$ with the class 0 is rotated onto $|0\rangle$ after running the circuit with its input and the learned parameters $\vec{\theta}$. If the state is rotated exactly on $|0\rangle$ the loss is:

$$\chi^2_f(\vec{\theta}) = (1 - |\langle 0|\psi(\vec{\theta},\vec{x}_j)\rangle|^2) = 1 - 1 = 0$$

3.1.3 Learning Process

The parameters $\vec{\theta}$ are initialized with random values between 0 and $\pi/2$. After that, the data set is split into a test and a training set. In each epoch the model splits a randomized training set into batches from which it will use the Adam [21] optimizer to optimize for these data points. A batch is a subset of the training data. Batch learning is used in Machine Learning, because when computing the gradient over the entire data set a lot of information is lost due to averaging. Once the optimizer has received and used every batch of the data set (the entire training data set) an epoch is over. Finally, the model is tested using the test data set.

3.1.4 Measurement

Once the model has processed the input it outputs a state vector for a given data point, from which the model has to make a prediction. For this the probability $P_0(|\psi\rangle)$ of the state being in $|0\rangle$[13] and $P_1(|\psi\rangle)$ of the state being in $|1\rangle$ is computed. This is just:

$$P_n(|\psi\rangle) = |\langle n|\psi\rangle|^2 \tag{19}$$

13: All measurements are the expectation value of $\hat{L}_z$.

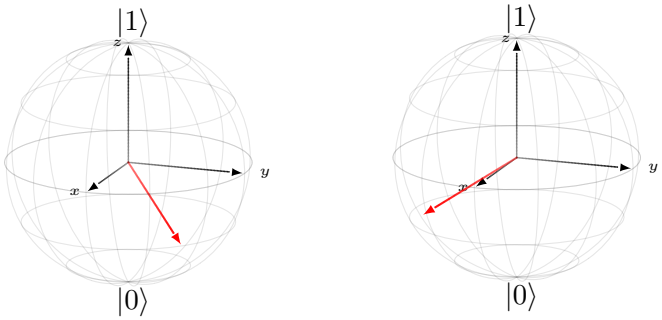
An input is classified as 0 if $P_0(|\psi\rangle) > P_1(|\psi\rangle)$ and 1 otherwise. On a quantum computer this is done by a majority vote of the measurement results.

3.1.5 Classification Illustration

This section will briefly illustrate the classifying process of a two layer binary classifier. The data point $\vec{x}$ is labeled as 1 and needs to be mapped on $|1\rangle$. The following operations with the already learned parameters $\vec{\theta}_n$ rotate the input to the desired state. The qubit is initialized into state $|0\rangle$.

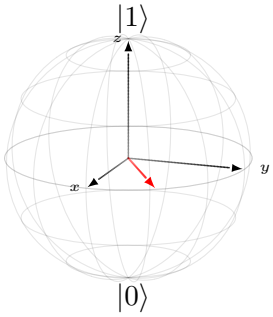
$$\begin{aligned}|\psi(\vec{x},\vec{\theta})\rangle &= \mathscr{U}(\vec{x},\vec{\theta})\,|0\rangle \\ &= \mathcal{L}(\vec{x},\vec{\theta}_2)\mathcal{L}(\vec{x},\vec{\theta}_1)\,|0\rangle \\ &= \mathcal{U}(\vec{\theta}_2)\mathcal{U}(\vec{x})\mathcal{U}(\vec{\theta}_1)\mathcal{U}(\vec{x})\,|0\rangle\end{aligned}$$

The resulting rotations can be seen in **Figure 7**. The input rotations in both layers are the same, hence the name data re-uploading. The parameter rotations are different in each layer. **Figure 7d** shows the final state of the qubit. This state is used to compute the probabilities which are later used to classify the input data.

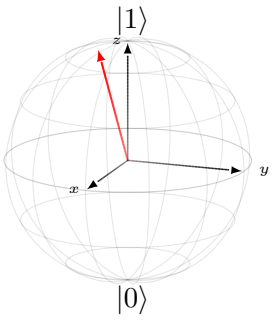


(a) 1st input rotation $\mathcal{U}(\vec{x})$

(b) 1st parameter rotation $\mathcal{U}(\vec{\theta}_1)$



(c) 2nd input rotation $\mathcal{U}(\vec{x})$



(d) 2nd parameter rotation $\mathcal{U}(\vec{\theta}_2)$

Figure 7: Illustration of an input being classified.
Probabilities of the final state:
$P_0(\vec{x}) = 0.05$
$P_1(\vec{x}) = 0.95$

3.2 Data Re-Uploading on Qudits

The concepts described in **Section 3.1** can now be extended onto a qudit. This introduces some advantages and disadvantages. For once the qudit is not limited to a Hilbert space dimension of only 2 and one can assume that it is easier to build a multi class classifier with a qudit compared to a single qubit[14]. However if the state is just rotated on the qudit Bloch sphere, the fidelity of the states not lying at the poles can never reach 1. This is due to the fact that only the eigenstates $|0\rangle$ and $|d-1\rangle$ of $\hat{L}_z$ can be reached by simply rotating the qudit around the x-axis. The other states the qudit reaches are not eigenstates of $\hat{L}_z$. Additionally the variance of $\hat{L}_z$ has its maximum at the equator, which results in the state $|(d-1)/2\rangle$ having the smallest maximal fidelity. In **Figure 8** the fidelities for each basis state[15] are shown when the state is rotated from the north pole to the south pole.

14: In [19] it is described to use maximal orthogonal states on the Bloch sphere to label more than 2 classes. This leads to the well known problem of quantum state discrimination[22]. Even though this problem is solved in some cases it is hard to realize experimentally. It is far easier to measure the collective spin length of a qudit **Section 2.2.**

15: $|0\rangle, |1\rangle, |2\rangle, |3\rangle, |4\rangle$

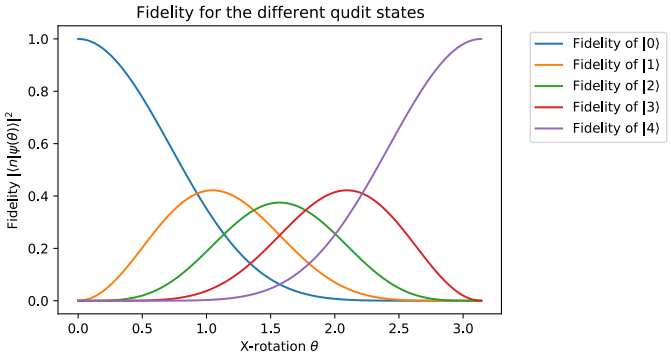


Figure 8: Fidelities of each basis state of a qudit with dimension 5.

This is where the squeezing, which was introduced in **Section 2.3.2** comes into play. Squeezing a state decreases the variance of $\hat{L}_z$ and thus increases the fidelity of a given label state. When only labeling the poles of the qudit this will worsen the performance, since squeezing in this case only decreases the fidelity[16]. Nevertheless, when labeling more states of the qudit, squeezing should improve the speed of learning and the accuracy, because the fidelity of the states that do not lie on a pole will reach higher values **(Equation 18)**.

16: The basis states $|0\rangle$ and $|d-1\rangle$ can reach a fidelity of 1 without squeezing since these are already eigenstates of $\hat{L}_z$.

3.2.1 Circuit Structure

In the example from [19] three instances of two different rotations are sufficient to approximate any unitary operation on a single qubit. For qudits the $\hat{L}_z^2$ operation is needed to approximate an arbitrary unitary operation[17] on a qudit with $d > 2$. Later, whenever a circuit with squeezing is mentioned, it will have the following structure[18].

$$\mathcal{U}(\vec{\theta}_n) = \hat{R}_z^2(\theta_{n_4})\hat{R}_i(\theta_{n_3})\hat{R}_j(\theta_{n_2})\hat{R}_i(\theta_{n_1}) \tag{20}$$

17: As seen in [11] appendix B.

18: This structure proved to be best when testing different circuits structures.

Here i and j are replaced by x and z, the rotations, that are mentioned in [11]. The input rotation is the same as in (**Equation 12**). A circuit without squeezing has the same structure as the one mentioned in **Equation 13**.

3.2.2 Label Classes

As previously stated, the approach of classifying multiple classes using maximally orthogonal states (as described in [19]) is hard to realize experimentally. By using qudits, one already has a higher dimensional Hilbert space and thus each, or every other, computational basis state can be used to label classes. In principle one is only limited by the dimensionality of the qudit, when building a multi class classifier.

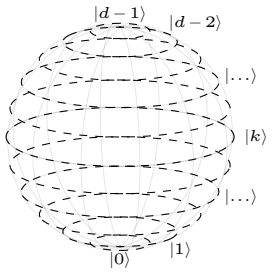


Figure 9: In principle each computational basis state $|0\rangle, \dots, |d-1\rangle$ can be labeled as a class to build a d class classifier.

3.2.3 Learning Process

The learning process is almost exactly the same as described in **Section 3.1.3**. The only difference is the fact that the parameter θ_{n_4} for $\hat{R}_z^2$ is not set randomly but is set to 0 at the beginning of the training. This is due to the fact that models that have randomly set variables for θ_{n_4} converged much slower to a minimum, because it is very hard for the model to "unsqueeze" a state. In addition the squeezing parameter is orders of magnitude smaller than the other parameters.

3.2.4 Measurement

The measurement is also quite similar as described in **Section 3.1.4**, but instead of only computing the probability of the two basis states $|0\rangle, |1\rangle$, now the probability of each labeled state[19] is computed. This leads to n probabilities for a n class classifier, from which the maximum is chosen as the prediction.

To summarize, the quantum machine learning model of data reuploading can be realized on a single qubit or qudit. The classifier consists of multiple layers, in each layer are two rotational operations. The first one decodes the input data onto the qudit. The second rotational operation has learnable parameters which map the state of the qudit onto a label state. The state resulting from these rotations is later used to classify the data point by computing the overlap with the different label states. The prediction is based on the label state with the highest fidelity. The more layers the classifier consists of the more powerful the classifier becomes.

19: Of course for a binary classifier the measurement falls back to having only 2 probabilities and thus being the same as in **Section 3.1.4**, with the slight difference that the measured states are $|0\rangle$ and $|d-1\rangle$

4 Simulations

Before benchmarking the classifier on the famous MNIST Handwritten Digits Data Set [23], it is beneficial to replicate some of the results from [19], such as the classification of a circle and of multiple simple 2D plane problems. This leads to a better understanding of the classifier and will provide insight into the effect of the different parameters for the training on the MNIST data set. When constructing the quantum classifier the following questions need to be answered, such that an optimal performance on the MNIST data set can be obtained:

- Which qudit dimensions are best for a binary/multi class classifier?
- How many layers are necessary to learn simple/more complex patterns?
- Which circuit structure[20] poses the best results?
- Does squeezing improve the performance?

20: For example $\hat{R}_z\hat{R}_x\hat{R}_z$ or $\hat{R}_x\hat{R}_z\hat{R}_x$.

4.1 Used Libraries

This section will shortly introduce the python packages that were used for the simulations. The code can be found in the this Git Repository. A helpful blog post and introduction into the topic of data re-uploading can be found on the pennylane website [24].

4.1.1 Scikit-learn

`scikit-learn` [25] was used to split the data sets into respective training and test sets and shuffle them. It was also used to run the classical machine learning algorithm in **Section 4.5.3** to compare the quantum classifier to a classical one. But most importantly the Principle Component [26] analysis of `scikit-learn` was used to reduce the dimensionality of the MNIST data set (**Section 4.5.1**).

4.1.2 Jax

This package is probably the most important one in terms of code speedup. `jax` [27] is a Python library for automatic differentiation (autograd) and accelerated linear algebra (XLA) for native Python and NumPy code. It computes the gradient of the used circuits. `jax` was also used to vectorize the code and compile it "just in time". This sped up the code by a huge margin.

4.2 Simulation Method

To be able to analyze a models performance accurately, each model was trained over 40 epochs at least 20 times. The number 20 was

chosen because it produced statistically meaningful results while still being computationally feasible. Then the mean and the variance of those 20 simulations was computed. The linear binary classifier was only trained for 20 epochs since after 20 epochs, the model had already converged.

4.3 Binary Classification

As an easy classification problem some simple data sets were chosen. The premise of each is the same, N points with $x \in \mathbb{R}^2$ are randomly sampled in a square with $\vec{x}_i \in [-1, 1]^2$. Each point is labeled with a class, depending on the position in the 2D plane. In this way, linear and non-linear data sets can be generated.

For the binary classifier the points are either labeled as 0 (blue) or 1 (red). These two classes are mapped to the label classes $|0\rangle$ and $|d-1\rangle$ of a qudit with dimensionality d. This is illustrated in **Figure 10**.

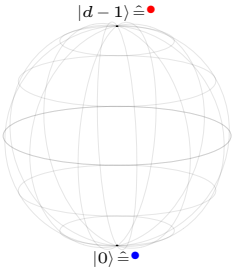


Figure 10: Labels of the qudit/qubit for the binary models.

The following plots and tables compare the quantum classifier for different circuit depths and learning rates[21].

21: The learning rate refers to the step size the optimizer takes towards the point of steepest descent, given by the gradient. A large learning rate can lead to the optimizer "stepping" over the global minimum, whereas a small learning rate can lead to the optimizer getting caught in a local minimum.

4.3.1 Linear Model

The following example leads to a linearly divided data set with the labels given by

$$y_i = \begin{cases} 0 : & x_1^i < 0 \\ 1 : & x_1^i \geq 0 \end{cases} \tag{21}$$

First of all it needs to be checked if the classifier can replicate the performance of [19] using the same structure, i.e. with a qudit dimensionality of 2.

The following table shows the accuracies after 20 epochs of training and a learning rate of 0.05. The accuracy is defined as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{22}$$

Layers	Accuracy
1	100.0 %
2	99.6 %
3	98.0 %
4	99.6 %
5	98.4 %
6	98.8 %
7	99.2 %

Table 1: Performance of model on linear data set

The reason for the missing variance in accuracy in **Table 1** is, because the model always converged to the same final parameters,

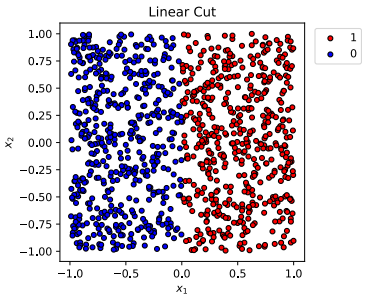
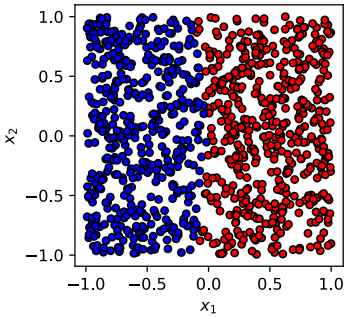


Figure 11: Resulting truth data from **Equation 21**.

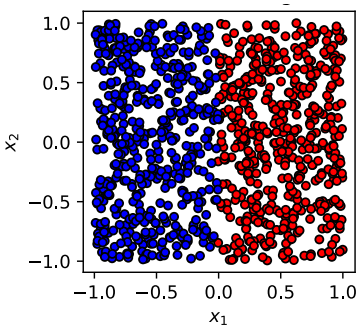
independent of the chosen starting parameters for $\vec{\theta}_n$. The reason for this being the simple structure of the data set.

From **Table 1**, one can deduce that one layer is sufficient to classify the linear data set. More layers minimally decrease the accuracy. This is due to the fact that with one layer the only possible separation is a linear one whereas multiple layers are able to create a non-linear decision boundary. This leads to overfitting the training data set and in turn to a reduction of the test performance. This simple problem is easily classified with a high accuracy by the model.

Below are some examples from the linear binary classifier.



(a) Predictions of a model with 2 layers.
Accuracy = 99.6%



(b) Predictions of a model with 5 layers.
Accuracy = 98.4%

Figure 12: Examples from the models trained on a binary linear data set after 20 epochs and a learning rate of 0.05.

4.3.2 Non-Linear Model

A more complex problem is the following

$$y_i = \begin{cases} 0 & : \quad \|\vec{x}_i\| < r \\ 1 & : \quad \|\vec{x}_i\| \geq r \end{cases} \tag{23}$$

with r being the radius.
First the effect of the qudit dimensionality and the consequences of different number of layers is investigated. In each case the label states are $|0\rangle$ and $|d-1\rangle$.

Figure 14 shows the accuracy and the loss depending on different qudit dimensions and number of layers. It can be seen, that lower qudit dimensions perform significantly better and reach a lower loss.

This result is not very surprising since by increasing the dimensionality of the qudit the optimization problem becomes more complex which leads to a worse performance of the classifier. The increased complexity leads to a loss function, that has a higher minimal loss and thus the optimizer is not able to reach lower losses than at lower dimensions. Additionally, with an increasing Hilbert space dimension more commutators are needed to approximate an arbitrary unitary operation.

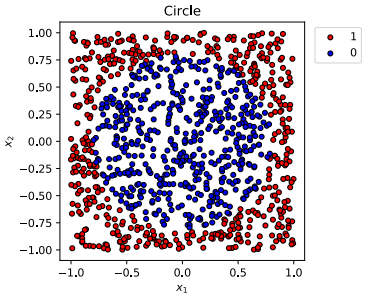
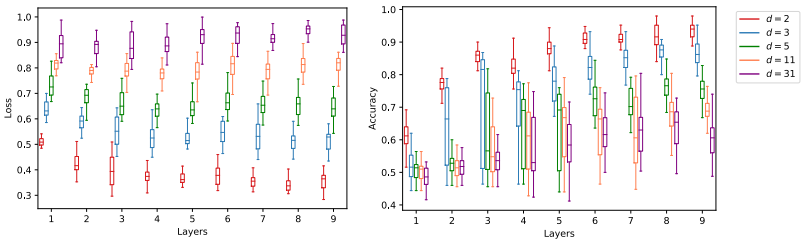


Figure 13: Resulting truth data from **Equation 23**.



(a) Loss dimension comparison of non-linear model.

(b) Accuracy dimension comparison of non-linear model.

Figure 14: Performance comparison for different qudit dimension on the binary non-linear data set.

The following figure shows the results after 40 epochs of training, a learning rate of 0.05 and a qudit dimension of 2. The dimension is set to 2 since, as stated above, the model performs significantly better with a lower qudit dimension.

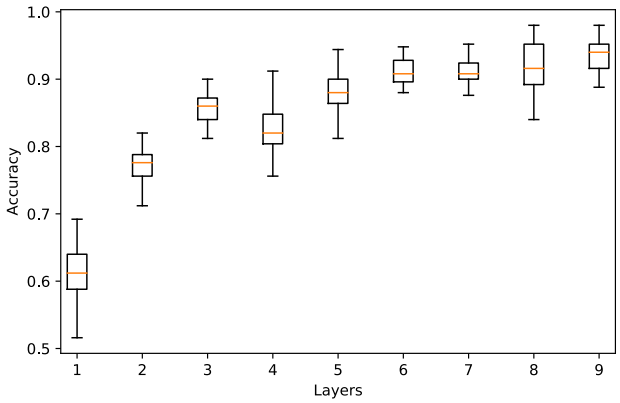
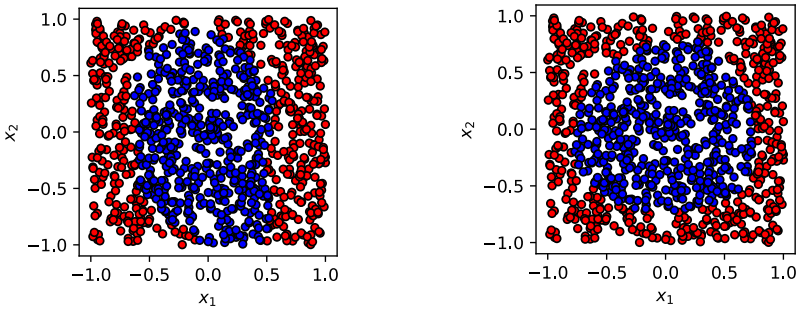


Table 2: Performance of the model on a non-linear data set with $d = 2$.

Layers	Accuracy
1	$60.7 \pm 4.9\%$
2	$77.2 \pm 2.8\%$
3	$85.5 \pm 2.9\%$
4	$82.8 \pm 4.4\%$
5	$87.9 \pm 3.1\%$
6	$90.9 \pm 2.1\%$
7	$91.2 \pm 2.3\%$
8	$91.9 \pm 3.9\%$
9	$93.5 \pm 2.7\%$

Figure 15: Boxplot accuracy for $d = 2$.

Additionally, the lower dimension of the qudit leads to a faster computation, since computing many high dimensional matrix exponentials is resource expensive. As described in [19], one needs at least two layers to approximate a non-linear data set with an accuracy above 50%. This is due to the fact, that one layer can only linearly cut a data set. With six or more layers, an accuracy above 90% is possible. In **Figure 16** the prediction results of two different models can be seen. While the three layer model struggles to classify a round circle, eight layers achieve a quite high accuracy.



(a) Predictions of a model with 3 layers.
Accuracy = 85.6%

(b) Predictions of a model with 8 layers.
Accuracy = 96.2%

Figure 16: Examples from the binary non-linear data set trained models over 40 epochs and a learning rate of 0.05.

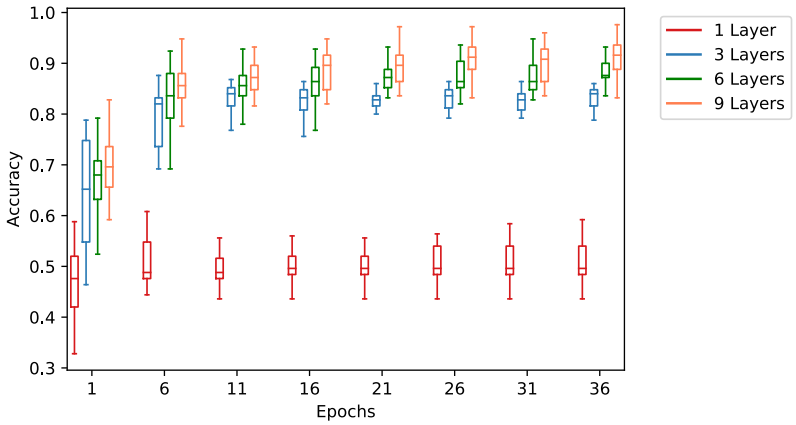


Figure 17: Accuracy during the learning process for different circuit depths of the 2-dimensional model.

Figure 17 shows the accuracy over the epochs for different circuit depths. One can see that the models achieve a high accuracy after 10 epochs and then reaches what looks like a plateau. The improvements in accuracy after 30 epochs are only marginally.

4.4 Multi Class Classification

Since the "MNIST Handwritten Digits data set" is a multi class data set, the model has to be tested on a data set containing multiple classes. For this, the approach from **Section 4.3** was expanded. Rather than only having two different label classes, the model now needs to learn three or more.

As described in **Section 3.2**, in theory, squeezing should improve the performance of the model. Since squeezing does not introduce any beneficial aspects regarding a binary classifier[22], the circuit only from now on has the additional $\hat{R}_z^2(\theta)$ gate in each but the last layer. In preliminary tests, this structure proved to be best.

22: This is due to the fact that if the poles are used for the labels, the variance can only increase at these points leading to a worse fidelity.

$$\mathcal{U}(\vec{\theta}_n) = \hat{R}_z^2(\theta_{n_4})\hat{R}_i(\theta_{n_3})\hat{R}_j(\theta_{n_2})\hat{R}_i(\theta_{n_1}) \tag{24}$$

The last layer has the following structure.

$$\mathcal{U}'(\vec{\theta}_n) = \hat{R}_i(\theta_{n_3})\hat{R}_j(\theta_{n_2})\hat{R}_i(\theta_{n_1}) \tag{25}$$

For comparison purposes models with and without the additional squeezing gate in the parameter rotation were trained. Additionally, the effect of the circuit structure ($\hat{R}_z\hat{R}_x\hat{R}_z$ and $\hat{R}_x\hat{R}_z\hat{R}_x$) is investigated. The results for a linear and a non-linear data set are presented below.

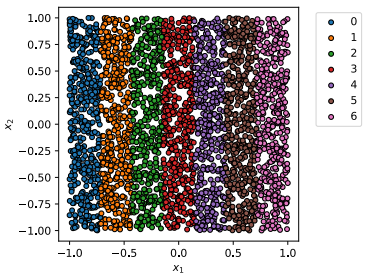


Figure 18: Resulting truth data from **Equation 26**.

4.4.1 Linear Multi Class Classification

The following equation cuts the data set into m equally sized regions:

$$y_i = n \ : \frac{n}{m} \leq x_1^i + 1 < \frac{n+1}{m} \tag{26}$$

Figure 18 shows the resulting data set for $m = 7$. This data set was also used to train the classifier and the results are shown below.

First of all the effect of the qudit dimensionality is investigated. From **Figure 14b** one suspects that higher dimensions decrease the performance. On the other hand one could argue that labeling every other basis state could be beneficial, since then the classifier has a bigger margin of error to the next class.
Both models are trained with the same 7 classes. The 7-dimensional classifier uses every possible computational basis state to label the 7 classes (**Figure 19**), whereas the 13-dimensional classifier uses the following states as label states: $|0\rangle, |2\rangle, |4\rangle, |6\rangle, |8\rangle, |10\rangle, |12\rangle$ (**Figure 20**).
Table 3 shows the results for a 7-dimensional and a 13-dimensional classifier.

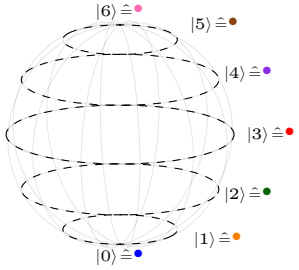


Figure 19: Labels of the qudit for 7 classes and a qudit dimension of 7.

Layers	Accuracy $d = 7$	$d = 13$
2	72.0 ±4.3 %	57.0 ±9.3 %
3	88.2 ±9.0 %	76.3 ±13.6 %
4	86.3 ±9.0 %	76.4 ±10.6 %
5	91.2 ±4.5 %	80.3 ±12.7 %
6	88.1 ±9.5 %	83.9 ±9.5 %
7	88.6 ±10.0 %	83.9 ±11.8 %
8	92.2 ±4.6 %	75.3 ±14.7 %
9	91.6 ±6.3 %	79.2 ±10.7 %

Table 3: Comparison of two classifiers using different qudit dimensions.
100 epochs, learning rate: 0.001

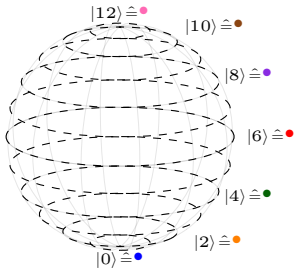


Figure 20: Labels of the qudit for 7 classes and a qudit dimension of 13.

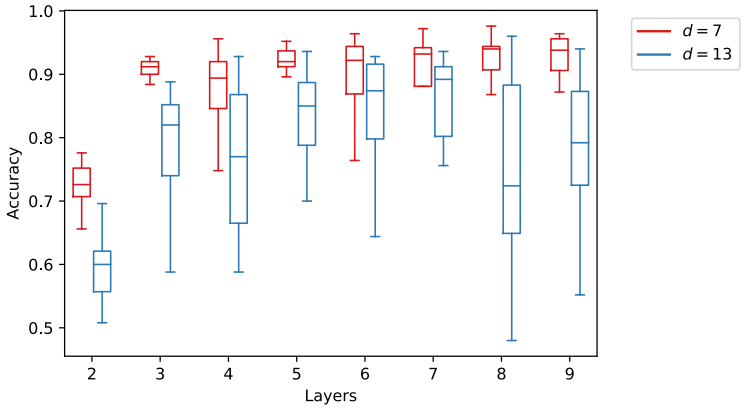


Figure 21: Boxplot showing the accuracy for $d = 7$ and $d = 13$.
100 epochs, learning rate: 0.001

Both classifiers use squeezing and the circuit structure is $\hat{R}_z \hat{R}_x \hat{R}_z$. The two models were trained over 100 epochs, to ensure the optimizer had converged. **Table 3** clearly shows that the ansatz with lower dimensions greatly outperforms the one using twice as many basis states as label classes. The reason for this is the same as mentioned in **Section 4.4.1**.

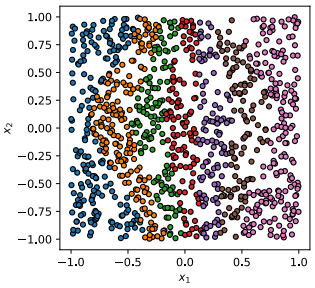
Next, the effect of squeezing on the model is examined. As a result of the simulations from **Table 3** and because the data set

has seven classes, a qudit dimension of $d = 7$ is chosen. Since the computation of all four models over several epochs and model instances is computationally expensive, the models were only trained for 40 epochs, instead over 100 as above.

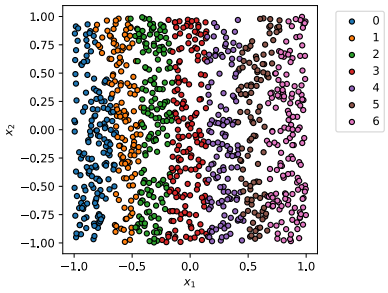
Table 4: Performance of the model on a linear data set with 7 classes and a qudit dimension of 7.
$*$ since the last layer of each circuit has no squeezing gate a single layer classifier with squeezing is not possible.

Layers	No Squeezing		Squeezing	
	$\hat{R}_z\hat{R}_x\hat{R}_z$	$\hat{R}_x\hat{R}_z\hat{R}_x$	$\hat{R}_z\hat{R}_x\hat{R}_z$	$\hat{R}_x\hat{R}_z\hat{R}_x$
1	65.0 ±2.0 %	64.8 ±2.3%	-*	–*
2	52.9 ±3.5 %	52.0 ±3.1%	70.1 ±5.9 %	67.1 ±5.3%
3	66.4 ±7.4 %	60.6 ±6.6%	87.6 ±9.3 %	89.5 ±8.2%
4	61.4 ±4.2 %	61.2 ±6.4%	85.8 ±6.0 %	87.9 ±3.1%
5	62.2 ±5.2 %	67.0 ±7.0%	88.1 ±3.7 %	89.1 ±5.0%
6	65.4 ±6.1 %	70.1 ±7.5%	87.4 ±10.7 %	86.2 ±10.0%
7	65.2 ±6.8 %	71.1 ±5.9%	88.8 ±8.0 %	90.3 ±5.5%
8	66.9 ±4.9 %	72.4 ±5.9%	87.8 ±7.4 %	86.3 ±9.8%
9	70.8 ±5.9 %	72.3 ±7.6%	87.8 ±8.7 %	82.7 ±12.0%

Overall the model containing squeezing performs significantly better than the one without. The variance for the models with fewer layers is smaller than the ones with more or less layers. A reason for the higher variance of the models with more layers could be the loss function (**Equation 18**). Due to the structure of the loss function a lower loss does not necessarily lead to a more accurate classifier.

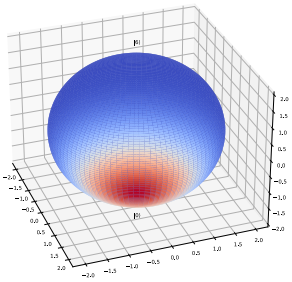


(a) Predictions of a model with 3 layers without squeezing.
Accuracy = 71.6%

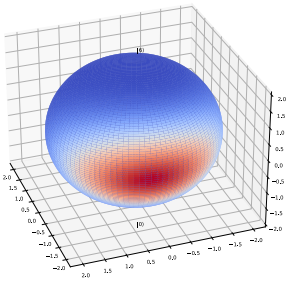


(b) Predictions of a model with 3 layers with squeezing.
Accuracy = 90.0%

Figure 22: Examples from the multi class non-linear data set trained models after 40 epochs and a learning rate of 0.05



(a) Model without squeezing, True label (3), classified as (2)



(b) Model with squeezing, True label (3), classified as (3)

Figure 23: Example states from the 3 layer multi class linear data set trained models.

It is very interesting to see that the classifier without squeezing prefers the classes which are labeled with the poles. This can be seen when comparing **Figure 22a** to **Figure 22b** and is not very surprising, since without squeezing the label states at the poles can achieve the highest fidelity[23]. In **Figure 23** example states from the classifiers with and without squeezing are shown. The correlation between the maximal fidelity and the predicted classes can be seen in **Figure 24**.

23: as seen in **Figure 8**.

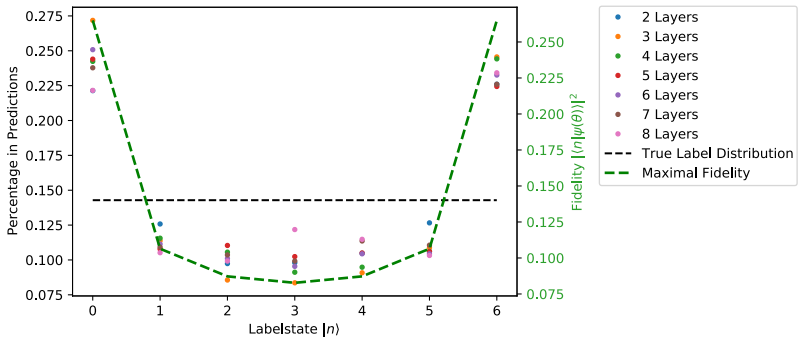


Figure 24: Comparing the maximal fidelity of different qudit states with the occurrence of the state in the predictions of the linear multi class classifier.

Here the y-axis on the left represents the portion of predicted states of the classifier without squeezing, whereas the right y-axis shows the maximal fidelities of the basis states. The simulated predictions without squeezing seem to correlate strongly with the maximal possible fidelity without squeezing.

Since the data set is uniformly distributed, a classifier which is not limited by the maximal fidelity of its qudit states would show uniformly distributed predictions, just like the data set. However, the classifier without squeezing shows this correlation (**Figure 24**) between the maximal fidelity and the share of the predictions for each label.

Another approach to remove the bias resulting from the maximal possible fidelities would be to rescale the loss function such that every label state has the same maximal possible fidelity.

4.4.2 Non-Linear Multi Class Classification

The last preparation before learning the handwritten digits will be a non-linear multi class classifying problem. For this a similar data set as in **Section 4.3.2** is generated with the difference being that there are two circles, one inside the other

$$y_i = \begin{cases} 0 & : \quad \|\vec{x}_i\| < r_1 \\ 1 & : \quad r_1 \leq \|\vec{x}_i\| \leq r_2 \\ 2 & : \quad \|\vec{x}_i\| > r_2 \end{cases} \tag{27}$$

with $r_1 < r_2$. The way these classes are mapped on the qudit is visualized in **Figure 26**.

Just as deducted in **Section 4.4.1** it is now assumed that the

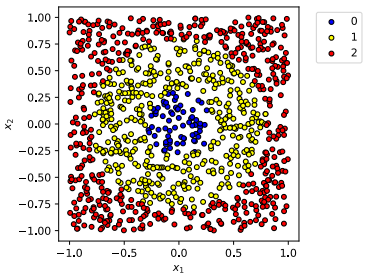


Figure 25: Resulting truth data from **Equation 27**.

model with squeezing will perform significantly better than the one without squeezing.

Table 5: Performance of the model on a non-linear data set with 3 classes. $*$ since the last layer of each circuit has no squeezing gate a single layer classifier with squeezing is not possible.

Layers	No Squeezing		Squeezing	
	$\hat{R}_z\hat{R}_x\hat{R}_z$	$\hat{R}_x\hat{R}_z\hat{R}_x$	$\hat{R}_z\hat{R}_x\hat{R}_z$	$\hat{R}_x\hat{R}_z\hat{R}_x$
1	53.1 ±1.6%	53.0 ±2.1%	-*	–*
2	55.6 ±0.5%	55.5 ±0.5%	55.2 ±2.3%	59.6±1.2%
3	76.3 ±2.3%	78.6 ±2.5%	73.0 ±3.0%	77.1±3.3%
4	78.8 ±1.7%	79.2 ±2.0%	75.1 ±3.5%	75.6±2.3%
5	81.2 ±3.0%	80.6 ±3.6%	78.2 ±3.2%	80.9±3.2%
6	80.6 ±4.5%	79.8 ±3.0%	85.1 ±4.2%	87.1±4.7%
7	82.5 ±3.4%	79.6 ±3.2%	85.7 ±3.0%	85.4±4.3%
8	81.2 ±2.3%	80.2 ±2.4%	88.6 ±3.8%	87.9±3.4%
9	79.2 ±1.9%	78.7 ±3.0%	88.8 ±4.5%	89.4±5.1%

In this case, squeezing does increase the accuracy but not as much as in **Section 4.4.1**. This is because for the ternary classifier there are only three classes and the improvements squeezing introduces are only beneficial for the classes not labeled at the poles, which is in this case only one. Nevertheless, squeezing improves the classifiers accuracy.

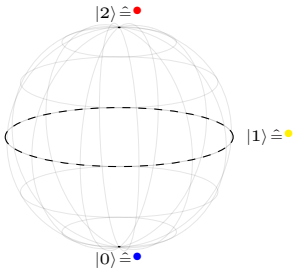
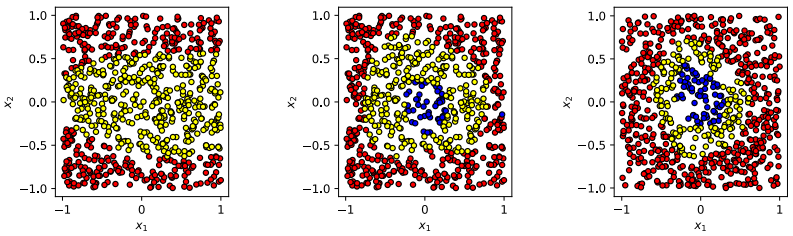


Figure 26: Labels of the qudit for 3 classes and a qudit dimension of 3.



(a) Predictions of a model with 3 layers with squeezing. Accuracy = 72.8%

(b) Predictions of a model with 8 layers with squeezing. Accuracy = 91.1%

(c) Predictions of a model with 8 layers without squeezing. Accuracy = 76.5%

Figure 27: Examples from the model trained on the multi class non-linear data set after 40 epochs and a learning rate of 0.05.

Table 5 also shows a similar trend as **Table 4** concerning the circuit structure. The arrangement of the gates $\hat{R}_x$ and $\hat{R}_z$ does not seem to have a significant effect on the performance of the classifier. This is expected since in both cases there are sufficiently many commutators to approximate any arbitrary unitary operation U. When comparing **Figure 27b** to **Figure 27c** one can again see that without squeezing class 2 (yellow) is underrepresented. The reason for this can be seen in **Figure 24**. The model with squeezing however does not have this bias.

4.5 MNIST Classification

The "MNIST Handwritten Digits data set" is **the** data set to test a new algorithm in machine learning.

4.5.1 Preparation

Since the digits in the MNIST data set are of the shape 8×8, which would lead to 64 input parameters, the dimensionality of the input vectors needs to be reduced. For this a Principle Component Analysis (PCA) [26] was used. PCA reduces the dimensionality of a given data set to a set number of dimensions. In this process, some of the information gets lost, but since it is not numerically feasible to introduce 64 input dimensions (d_i) to the model this trade-off had to be made.

Using the whole MNIST data set with all ten numbers would require bigger computational resources, so only five of the 10 numbers were selected to train the classifier.

The numbers were labeled on the qudit as follows (visualized in **Figure 28**):

- $|0\rangle \mathrel{\hat{=}} 1$
- $|1\rangle \mathrel{\hat{=}} 7$
- $|2\rangle \mathrel{\hat{=}} 4$
- $|3\rangle \mathrel{\hat{=}} 2$
- $|4\rangle \mathrel{\hat{=}} 0$

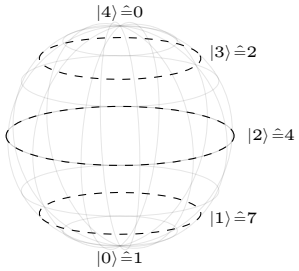


Figure 28: Label states of the qudit for five digits.

This specific labeling was chosen because it produced the best results when running preliminary tests.

For these simulations the lessons learned form above were applied:

- Low qudit dimensions

  $\Rightarrow$ A qudit dimension of $d = 5$ was used

- Sufficiently many layers

  $\Rightarrow$ Simulations with at least 3 layers were run

- Squeezing

  $\Rightarrow$ The models used squeezing

4.5.2 Results

Each model was trained 10 times over 500 epochs with a learning rate of 0.005. Additionally to varying the layer size, models with different input dimensions were trained. Each of these models received the data with input dimensions d_i of 3 to 7. In **Section 4.3** and **Section 4.4** the input data was mapped via $\hat{R}_z(x_1)\hat{R}_x(x_2)$. Since the input dimension has now changed, for each additional input dimension an additional rotation gate $\hat{R}_i(x_n)$ is added to the

input rotation unitary operation. The type of rotation (either $\hat{R}_x$ or $\hat{R}_z$) depends on the previous rotation operations such that an alternating rotation pattern emerges:

$$\hat{R}_z(x_1)\hat{R}_x(x_2)\hat{R}_z(x_3)\hat{R}_x(x_4)\ldots \tag{28}$$

Table 6: Performance of the model on parts of the MNIST data set.

Layers	Accuracy of Input Dimension d_i				
	3	4	5	6	7
3	69.0±7.3%	70.6 ±6.5%	71.0±5.8%	65.5±8.6%	63.5±7.7%
4	76.1±3.3%	77.1±3.1%	74.1±4.2%	70.1±5.8%	72.0±3.9%
5	76.9±6.9%	79.0±4.1%	75.6±6.4%	73.0±6.4%	70.8±3.8%
6	83.4±1.9%	81.6±4.4%	79.6±3.8%	75.3±4.7%	72.4±4.8%
7	83.8±3.1%	82.7±2.8%	80.3±3.6%	80.0±2.1%	73.7±3.9%
8	82.1±4.6%	82.9±2.6%	80.5±3.2%	79.7±4.2%	72.8±6.6%
9	85.3±2.7%	86.0±3.1%	80.0±2.5%	79.9±2.9%	73.7±5.0%

Table 6 shows the results from the MNIST classifier. One can see that more input dimensions do not necessarily lead to a better performance. There is a clear downwards trend in the accuracy with an increasing number of input dimensions.

This result is rather surprising since an increasing number of informative input dimensions allows the classifier to separate the data more efficiently and thus achieve higher accuracies. The results from **Table 6** and **Figure 29** show the exact opposite. It can be suggested that the method of data re-uploading breaks down with an increasing number input dimensions using this circuit structure and loss function. To answer the question why this happens and how to avoid this, further research is needed.

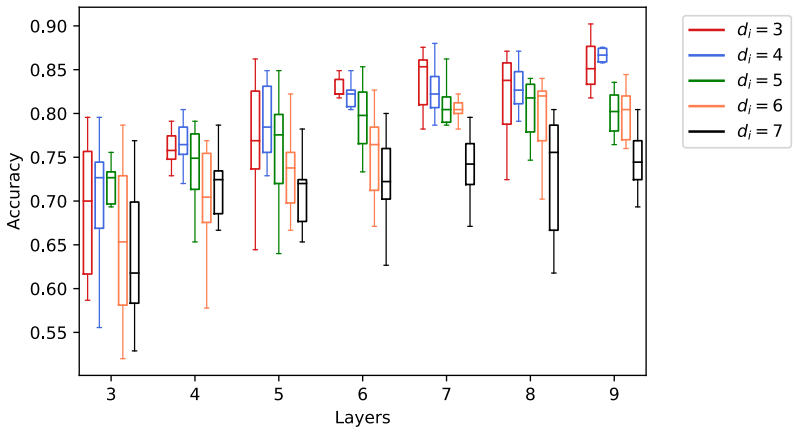


Figure 29: Accuracy of the MNIST classifier with different layers and input dimensions d_i.

Figure 29 visualizes the data from **Table 6** in a boxplot. From this plot one can deduct that the optimal number input dimensions for this data set and classifier structure is 3 or 4. More layers proved to increase the accuracy for these input dimensions. Even more layers would probably produce even more accurate results, but training

the model with $d_i = 4$ and 9 layers 10 times took over 4 hours. For further investigation in this regard, more powerful computers are necessary.

In **Figure 30**, the accuracy and the loss of three selected models is plotted. The red boxplot shows the in the mean best performing model[24]. The higher the input dimension d_i the lower is the accuracy. Another surprising observation is the widely scattered loss of the model with 7 input dimensions. The whiskers of the boxplot are fairly large when comparing them to models with less dimensions. This fact can be used to further investigate the effect of an increasing number of input dimension on the data encoding and the loss function.

24: $d_i = 4$, 9 layers

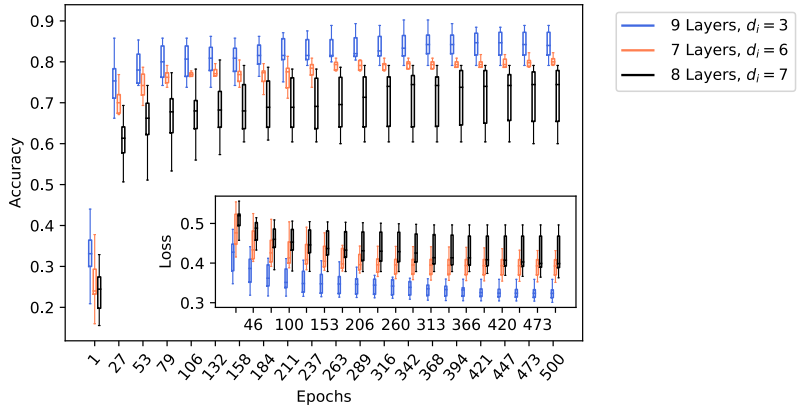


Figure 30: Performance comparison for different input dimension of the MNIST classifier.

4.5.3 Comparison to a Classical Machine Learning Algorithm

This section will provide a comparison between the results obtained from the quantum classifier and a classical machine learning approach. There are many machine learning algorithms that could be used to compare the results. This thesis will use random forests to classify the same part of the MNIST data set and then compare the results. Random forests are very versatile and "meet-[ing] the requirements for serving as an off-the-shelf procedure for data mining" [28].

4.5.4 Random Forests

The concept of random forests was first introduced by Tin Kam Ho in 1995 [29] and later improved by Leo Breiman [30]. Random forests consist of many decision trees[25]. While training a decision tree, the data is split at features and feature values, which are determined by a metric, such as the Gini coefficient or information gain. The split data is then assigned to one of the children of the node. The child turns into a leaf as soon as the data assigned to it consists of only one class. After training, a decision tree represents the data

25: In the context of computer science trees are acyclic graphs. Each node which is not a leaf has at least one child.

where the labels are derived by simple rules as specified through the feature splits. Each leaf node represents a certain subset of the data.

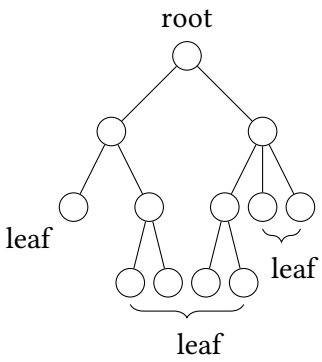


Figure 31: Single decision tree

In a random forest, many decision trees are trained, where each decision tree is trained with a randomly selected subset of the data and operates only on a randomly selected subset of the features. The final classification result can be derived from the individual trees by various aggregation methods. The `scikit-learn` implementation averages the probabilistic predictions of each tree in the ensemble. When the forest receives an input to classify, it forwards the input to each decision tree. Each tree then compares a characteristic value at each node and makes the decision based on the seen data to classify it to a certain child. Once the input reaches a leaf the decision tree can make a prediction[26].

26: Remember that during the training a leaf was created only when data points from a single class were present, thus the tree can now make prediction based on which class reached this specific leaf.

4.5.5 Results

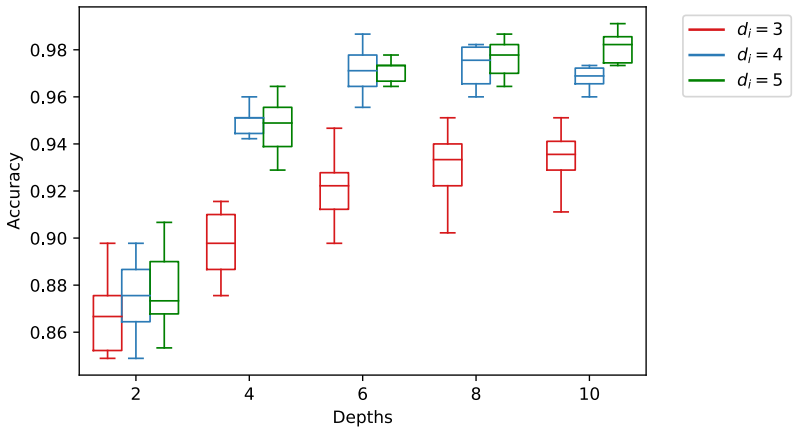


Figure 32: Accuracy of the random forest with different depths and input dimensions d_i.

The boxplot below shows the results for different forest depths[27] and input dimensions d_i. Each depth was trained 10 times with a randomly selected training and testing batch and a different random state for each forest.

27: The maximum depth of a tree.

The random forests perform very well when reducing the data set to at least four dimensions. Once a depth of 6 is reached, the classifier reaches a median accuracy of over 95%.

4.5.6 Comparison

Figure 29 and **Figure 32** show that the classical algorithm random forest is still superior to data re-uploading. Not only when looking at the accuracy but also regarding the computation time. While the random forests only took seconds to train the quantum model trained for about 10 to 40 minutes depending on the input dimensions and layer depths. Nevertheless, an accuracy of around 85.0% with a 9 layer model is not as bad when comparing it to a classical algorithm, a random forest with depth 9. The difference is only about 8 percentage points. The most surprising discovery concerning a real world data classifier is the fact that an increasing number of input dimension decreases the accuracy of the quantum classifier.

This section first tested the qudit classifier on simple classification problems that became increasingly more difficult. The results for the qubit classifier from [19] were recreated. The preliminary simulations showed four things: squeezing improves the classifier, the circuit structure is not as important for the accuracy, lower qudit dimensions achieve a higher accuracy and at least three layers are necessary to classify a non-linear data set. Finally the model was trained on parts of the MNIST Handwritten Digits data set and compared to the classical machine learning algorithm of random forests. The qudit model reached a maximal accuracy of 86%, whereas the classical algorithm achieved accuracies above 95%.

5 Conclusion and Perspectives

5.1 Optimal Parameters

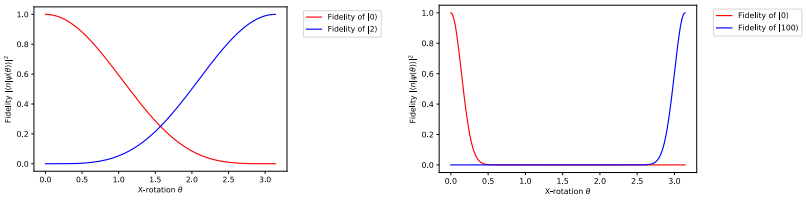
5.1.1 Which qudit dimensions are best for a binary/multi class classifier?

Binary Data Set The simulations in Section 4.3.2 are quite clear. The qudit dimension does not only impact the computation time, but also the accuracy of the model. It proved to be better to choose a lower dimension rather than a higher one.

Multi Class Data Set When classifying a multi class data set the results were similar to the one observed in the binary case. The performance of the model was significantly better when the qudit dimension was equal to the number of classes in a data set.

For the optimal performance the relation $m = d$ needs to be chosen, where d is the qudit dimension and m the number of classes in a data set.

Qudit Dimensions and the Loss Function The reason for the performance issues of the higher dimensional qudit is the loss function. The rate at which the fidelity of a given qudit state approaches 0 is dependent on the qudit dimension.



(a) Fidelity of $|0\rangle$ and $|2\rangle$ for $d = 3$. **(b)** Fidelity of $|0\rangle$ and $|100\rangle$ for $d = 101$.

Figure 33: Comparison of the pole fidelities for different qudit dimensions.

Since the fidelity vanishes much quicker for higher dimensions the loss function does not reach values as small as for lower qudit dimensions. Additionally, the optimizer only sees a plateau (Barren Plateau) at the region where both fidelities are 0. Thus the gradient is zero over a vast region of the Hilbert space. A similar problem with high dimensional Hilbert spaces and learning is discussed in [31]. In [19] a weighted loss function is presented, where not only the fidelity of the correct label state is maximized, but also the fidelity of the wrong states is minimized.

Another approach to overcome the plateaus of the loss function for high dimensional qudits could be, to label more than one qudit state with a single class. This would lead to a coarse grained loss

function that could have Gaussian shapes over the desired qudit states.

5.1.2 How many layers are necessary to learn simple/more complex patterns?

Binary Linear Data Set The preliminary simulations in **Section 4.3.1** showed that one layer is already sufficient to classify the binary linear data set. Even though the model was trained 20 times with each layer, no variance in the accuracies was recorded. This is due to the fact that the data set is so simple that the optimizer converges to the global minimum each time.

Binary Non-Linear Data Set Since the non-linear data set is more challenging for the classifier three layers are needed to classify at least 85% correctly. To correctly classify more than 90% more than five layers were needed. When comparing these results to the data in [19] (Table 1) one can see that this classifier performs quite similar to the qubit classifier[28]. To further improve the classifier a weighted loss function is described in [19]. The limits of the chosen loss function is discussed below (**Section 5.2**).

28: Which is not very surprising since a qudit with dimension 2 is in principle just a qubit.

Multi Class Linear Data Set The classification for the multi class linear data shows that at least three layers are needed to achieve an accuracy above 80%. Adding more layers did not record any better results regarding the raw performance. There is however a sweet spot of 4-5 layers concerning the variance.

Multi Class Non-Linear Data Set The results from this data set show a linear trend when looking at the accuracy over the layer depth. The more layers the circuit consists of, the more accurately it can classify the data set. To achieve an accuracy above 85% at least six layers are needed.

5.1.3 Which circuit structure poses the best results?

Section 4.4.1 and **Section 4.4.2** paint a clear picture in this regard. The differences in performance using the two circuit structure are not significant.

5.1.4 Does Squeezing improve the performance?

The results from **Table 4** and **Table 5** show a clear result, squeezing does significantly improve the classifier. In the case of the linear data set, squeezing improved the classifier by about 20 percentage points, whereas in the case of the non-linear multi class data set of about 10 percentage points. Squeezing removes the bias of the

classifier to prefer the classes which are labeled with the poles of the qudit.

5.2 The Loss Function and its Limits

Ideally, the loss function behaves in such a way, that when the loss after an epoch was minimized the accuracy improves. This however is not always the case considering the loss function introduced in **Section 3.1.2**. It has the undesired feature of not always producing better predictions when minimizing the loss. This is illustrated by the following individual data from the simulation with squeezing, three layers and the structure of $\hat{R}_z\hat{R}_x\hat{R}_z$ from **Table 5**.

- Run 3
  - **Loss**: 0.374
  - **Accuracy**: 76.8 %
- Run 7
  - **Loss**: 0.307
  - **Accuracy**: 70.0 %

Even though **Run 3** has a higher loss the accuracy is larger compared to **Run 7**.

Lets consider a model which has already optimized its parameters such that half of the data set is barely classified correctly, more precisely

$$P_c^1(|\psi\rangle) - P_w^1(|\psi\rangle) > 0 \tag{29}$$

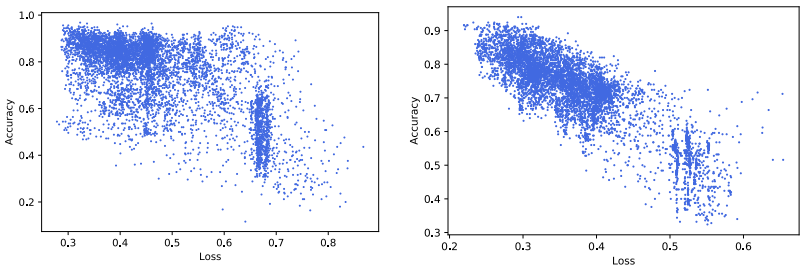
with P_w^1 as the probability of a wrong state and P_c^1 of a correct one out of the first half of the data set. The other half is classified completely wrong:

$$P_c^2(|\psi\rangle) - P_w^2(|\psi\rangle) \ll 0 \tag{30}$$

In some cases the optimizer only "sees" a plateau concerning the second half of the data set and is not able to optimize the model for these data points. However, the loss of the first half of the data set can be minimized even further, such that $P_w^1(|\psi\rangle) \ll P_c^1(|\psi\rangle)$. But once $P_w(|\psi\rangle) < P_c(|\psi\rangle)$ the further minimization is not as beneficial as for $P_w(|\psi\rangle) \geq P_c(|\psi\rangle)$ for the prediction. This is visualized in **Figure 34a**, where the accuracy is plotted over the loss of the linear multi class classifier with squeezing. Each dot represents an epoch.

With a better loss function, the scatter plot would show a clear linear trend. The correlation of loss and accuracy is more linear (**Figure 34b**) when looking at the binary or ternary classifier, since the loss function is not a complex in these cases.

In [19] a weighted loss function that also minimizes the fidelity of the wrong label states is introduced. This approach would probably also improve the MNIST classifier a lot. Further research is



(a) Accuracy over Loss for the linear multi class classifier.

(b) Accuracy over Loss for the non-linear multi class classifier.

Figure 34: Visualization of the accuracy and loss correlation.

needed to implement the weighted sum for data re-uploading on qudits.

5.3 Qubits vs. Qudits

The simulations from **Section 4** have shown that lower dimension preform significantly better. When comparing the performance of a binary qubit and a binary qudit classifier the results are almost identical. This is not very surprising since the 2-dimensional qudit is basically a qubit. The advantages of the qudit become present when constructing a multi class classifier. It is far easier to experimentally measure a qudit state, compared to maximally orthogonal states on a qubit. In addition, it is conceptually easier to visualize and understand the processes taking place when classifying multiple classes.

5.4 Labeling the Data correctly

A question that always arises when dealing with machine learning algorithms is how to properly format the data. In the case of the quantum classifier this boils down to the question how to label the qudit states. The straightforward approach is to label the qudit in the order the data is structured[29]. This becomes increasingly more difficult when dealing with a higher dimensional data set, such as the MNIST data set. The correct labeling of the classes in this case needs further investigation since this could improve the classifier. This could be done using methods such as t-Distributed Stochastic Neighbor Embedding [32] (t-SNE) or Uniform Manifold Approximation and Projection for Dimension Reduction [33] (UMAP).

29: As it is done in **Figure 19**.

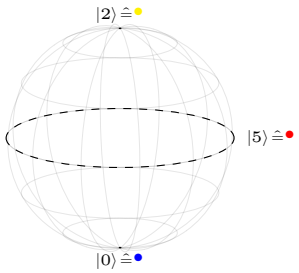


Figure 35: Does this labeling compared to **Figure 26** improve the classifier?

5.5 MNIST Classification

When testing a new (quantum) machine learning model, the first benchmark test is usually done on the MNIST Handwritten Digits data set. The training in **Section 4.5** has shown that the qudit model can indeed reach a reasonable accuracy in the mid 80% regime. This accuracy can probably reach even higher values when

implementing improvements mentioned in this section, such as labeling the data according to the distance to one another and improving the loss function. The most surprising finding of the training is the fact that the way data re-uploading was implemented in this thesis, it struggles with higher dimensional input vectors. The accuracy decreased significantly when introducing more informative input dimensions. This is contrary to what one would expect and observes when training classical models. To investigate the reason of the worse performance caused by higher input dimensions further research is needed. The classical approach of using random forests outperformed the ansatz of using a qudit classifier. Not only did the classical model outperform the quantum model regarding accuracy but also in computation time (when simulated on a classical computer). The next steps in research would be to run the qudit classifier on a near term NISQ device such as the "NaLi Machine" from [34]. The advantages of this approach would be that the Hilbert space of the system can reach $1.000.000$ and a single circuit takes only up to one minute to execute. To make use of the larger Hilbert space and not decrease the performance, as simulations in **Section 4.4.1** have shown, a better loss function needs to be implemented.

5.6 Outlook

This thesis has presented the implementation of the quantum machine learning model data re-uploading on qudits. First the theoretical foundation was laid to show that data re-uploading is possible on qudits and might pose some advantages over the more conventional approach of using qubits. This hypothesis was tested by training the qudit classifier on a variety of different data sets, finishing with a classification of the "MNIST Handwritten Digits Data Set".
Using qudits for data re-uploading showed to have an advantage over qubits in the regime of multi class classifying since the higher dimensional Hilbert space is advantageous when classifying multi class data sets, contrary to the approach of using maximal orthogonal states.

Finally the classifier was compared to a classical algorithm, random forests. The classical approached proved to be superior not only in the accuracy but also in the training time.

The next steps concerning a realization of the theoretical work done in this thesis would be to run the classifier on an experiment. This would be possible on the "NaLi Machine" from [34]. A challenge when running the model on an experimental setup is to determine the gradient of a given circuit. On classical computers the gradient can be computed numerically. When the model is run on an experimental setup only the expectation value of $\hat{L}_z$ can be mea-

sured. To be able to find the a minimum from these measurements one can use a stochastic gradient descent optimizer [35] or a cross-entropy optimizer [36]. Both optimizers try to find the direction of steepest descent from a random sample of data points. Additionally it is interesting to explore and quantify the data encoding mechanism of the qudit classifier using an approach described in [37].

6 Bibliography

[1] Paul Benioff. 'The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines'. In: *Journal of Statistical Physics* 22.5 (May 1980), pp. 563–591. DOI: 10.1007/BF01011339. (Visited on 08/11/2021) (cited on page 1).

[2] Richard P. Feynman. 'Simulating physics with computers'. In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. DOI: 10.1007/BF02650179. (Visited on 08/11/2021) (cited on page 1).

[3] Peter W. Shor. 'Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer'. In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. (Visited on 08/11/2021) (cited on page 1).

[4] Lov K. Grover. 'A fast quantum mechanical algorithm for database search'. In: *arXiv:quant-ph/9605043* (Nov. 19, 1996). (Visited on 08/21/2021) (cited on page 1).

[5] Frank Arute et al. 'Quantum supremacy using a programmable superconducting processor'. In: *Nature* 574.7779 (Oct. 24, 2019), pp. 505–510. DOI: 10.1038/s41586-019-1666-5. (Visited on 08/11/2021) (cited on page 1).

[6] Feng Pan and Pan Zhang. 'Simulating the Sycamore quantum supremacy circuits'. In: *arXiv:2103.03074 [physics, physics:quant-ph]* (Mar. 4, 2021). (Visited on 08/11/2021) (cited on page 1).

[7] John Preskill. 'Quantum Computing in the NISQ era and beyond'. In: *Quantum* 2 (Aug. 6, 2018), p. 79. DOI: 10.22331/q-2018-08-06-79. (Visited on 07/29/2021) (cited on pages 1, 8).

[8] John Clarke and Frank K. Wilhelm. 'Superconducting quantum bits'. In: *Nature* 453.7198 (June 2008), pp. 1031–1042. DOI: 10.1038/nature07128. (Visited on 08/12/2021) (cited on page 1).

[9] Rainer Blatt and David Wineland. 'Entangled states of trapped atomic ions'. In: *Nature* 453.7198 (June 2008), pp. 1008–1015. DOI: 10.1038/nature07125. (Visited on 08/12/2021) (cited on page 1).

[10] E. Knill, R. Laflamme, and G. J. Milburn. 'A scheme for efficient quantum computation with linear optics'. In: *Nature* 409.6816 (Jan. 2001), pp. 46–52. DOI: 10.1038/35051009. (Visited on 08/12/2021) (cited on page 1).

[11] Valentin Kasper et al. 'Universal quantum computation and quantum error correction with ultracold atomic mixtures'. In: *arXiv:2010.15923 [cond-mat, physics:quant-ph]* (Oct. 29, 2020). (Visited on 07/28/2021) (cited on pages 1, 4, 5, 12, 13).

[12] Tariq M. Khan and Antonio Robles-Kelly. 'Machine Learning: Quantum vs Classical'. In: *IEEE Access* 8 (2020), pp. 219275–219294. DOI: 10.1109/ACCESS.2020.3041719. (Visited on 07/08/2021) (cited on page 2).

[13] Yuchen Wang et al. 'Qudits and High-Dimensional Quantum Computing'. In: *Frontiers in Physics* 8 (Nov. 10, 2020), p. 589504. DOI: 10.3389/fphy.2020.589504. (Visited on 07/28/2021) (cited on page 4).

[14] Hsuan-Hao Lu et al. 'Quantum Phase Estimation with Time-Frequency Qudits in a Single Photon'. In: *Advanced Quantum Technologies* 3.2 (Feb. 2020), p. 1900074. DOI: 10.1002/qute.201900074. (Visited on 08/21/2021) (cited on page 4).

[15] A. B. Klimov et al. 'Qutrit quantum computer with trapped ions'. In: *Physical Review A* 67.6 (June 26, 2003), p. 062313. DOI: 10.1103/PhysRevA.67.062313. (Visited on 08/21/2021) (cited on page 4).

[16] H. Strobel et al. 'Fisher information and entanglement of non-Gaussian spin states'. In: *Science* 345.6195 (July 25, 2014), pp. 424–427. DOI: 10.1126/science.1250147. (Visited on 07/29/2021) (cited on pages 5, 6).

[17] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge ; New York: Cambridge University Press, 2000. 676 pp. (cited on page 5).

[18] Kôdi Husimi. *Some Formal Properties of the Density Matrix*. 1940. URL: https://doi.org/10.11429/ppmsj1919.22.4_264 (visited on 07/28/2021) (cited on page 5).

[19] Adrián Pérez-Salinas et al. 'Data re-uploading for a universal quantum classifier'. In: *Quantum* 4 (Feb. 6, 2020), p. 226. DOI: 10.22331/q-2020-02-06-226. (Visited on 06/06/2021) (cited on pages 8, 12–15, 17, 27–30).

[20] M. Cerezo et al. 'Variational Quantum Algorithms'. In: *arXiv:2012.09265 [quant-ph, stat]* (Dec. 16, 2020). (Visited on 08/25/2021) (cited on page 8).

[21] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *arXiv:1412.6980 [cs]* (Jan. 29, 2017). (Visited on 07/08/2021) (cited on page 10).

[22] Stephen M. Barnett and Sarah Croke. 'Quantum state discrimination'. In: *arXiv:0810.1970 [quant-ph]* (Oct. 10, 2008). (Visited on 08/13/2021) (cited on page 12).

[23] Corinna Cortes Yann LeCun. *The MNIST Database of Handwritten Digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (cited on page 14).

[24] Ahmed Shahnawaz. *Data-reuploading classifier*. Quantum Machine Learning. Jan. 19, 2021. URL: https://pennylane.ai/qml/demos/tutorial_data_reuploading_classifier.html (cited on page 14).

[25] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cited on page 14).

[26] Ian T. Jolliffe and Jorge Cadima. 'Principal component analysis: a review and recent developments'. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (Apr. 13, 2016), p. 20150202. DOI: 10.1098/rsta.2015.0202. (Visited on 08/02/2021) (cited on pages 14, 23).

[27] James Bradbury et al. *JAX: composable transformations of Python+NumPy*. Version 0.2.5. 2018 (cited on page 14).

[28] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction: with 200 full-color illustrations*. Springer series in statistics. New York: Springer, 2001. 533 pp. (cited on page 25).

[29] Tin Kam Ho. 'Random decision forests'. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. 3rd International Conference on Document Analysis and Recognition. Vol. 1. Montreal, Que., Canada: IEEE Comput. Soc. Press, 1995, pp. 278–282. DOI: 10.1109/ICDAR.1995.598994. (Visited on 08/12/2021) (cited on page 25).

[30] Leo Breiman. 'Random Forests'. In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. (Visited on 08/22/2021) (cited on page 25).

[31] Jarrod R. McClean et al. 'Barren plateaus in quantum neural network training landscapes'. In: *Nature Communications* 9.1 (Dec. 2018), p. 4812. DOI: 10.1038/s41467-018-07090-4. (Visited on 08/20/2021) (cited on page 28).

[32] Tejas Khot. 'Visualizing high-dimensional data'. In: *XRDS: Crossroads, The ACM Magazine for Students* 23.2 (Dec. 15, 2016), pp. 66–67. DOI: 10.1145/3021604. (Visited on 08/24/2021) (cited on page 31).

[33] Leland McInnes, John Healy, and James Melville. 'UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction'. In: *arXiv:1802.03426 [cs, stat]* (Sept. 17, 2020). (Visited on 08/24/2021) (cited on page 31).

[34] Alexander Mil et al. 'A scalable realization of local U(1) gauge invariance in cold atomic mixtures'. In: *Science* 367.6482 (Mar. 6, 2020), pp. 1128–1130. DOI: 10.1126/science.aaz5312. (Visited on 07/08/2021) (cited on page 32).

[35] David Saad, ed. *Online learning in neural networks*. Publications of the Newton Institute. Cambridge, [Eng.] ; New York: Cambridge University Press, 1998. 398 pp. (cited on page 33).

[36] Pieter-Tjerk de Boer et al. 'A Tutorial on the Cross-Entropy Method'. In: *Annals of Operations Research* 134.1 (Feb. 2005), pp. 19–67. DOI: 10.1007/s10479-005-5724-z. (Visited on 08/25/2021) (cited on page 33).

[37] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. 'The effect of data encoding on the expressive power of variational quantum machine learning models'. In: *Physical Review A* 103.3 (Mar. 24, 2021), p. 032430. DOI: 10.1103/PhysRevA.103.032430. (Visited on 08/25/2021) (cited on page 33).

7 Acknowledgments

Many people supported me during my work on this bachelor thesis. To all the people who contributed directly or indirectly, thank you! Especially I would like to mention

- Fred Jendrzejewski for giving me the opportunity to work on this thesis and for supporting me during the entire time I spent in the group.
- Martin Gärttner for taking the time and being the 2nd corrector of my thesis.
- Yannick Deller for always answering the many questions I had (and still have), for giving me many new ideas every day (which I unfortunately was not able to test all) and for proofreading my thesis.
- The NaLi's SoPa's and NaKa's for the good food, the enjoyable lunches and dinners and for the fun working atmosphere.
- Sebastian Schmitt for the valuable discussions about machine learning, optimizers and for proofreading my thesis.
- My family. Mama, Papa und Marvin vielen Dank für Eure Unterstützung über die letzten drei Jahre. Ohne Eure Unterstützung hätte ich es nicht dahin geschafft wo ich jetzt bin.

Erklärung

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

Heidelberg, den 30. August 2021