

# Experimenting with machine learning algorithms on photonic quantum computers

MSc thesis

Nagy Dániel

nagy.dani@wigner.hu

Supervisors:

Zoltán Zimborás (Wigner RCP)

István Csabai (ELTE)



Eötvös Loránd University, Budapest

2021

## **Abstract**

Possibly the most influential achievements of modern computer science are the inventions of different machine learning algorithms, especially deep neural networks, which were able to solve problems that were previously intractable for computers, for example recognizing different animals on photos. On the other hand, in the last few decades we were witnessing an enormous improvement in quantum computing, especially quantum hardware development. In 2019 Google's quantum computer achieved quantum computational supremacy, which means the beginning of a new era in the history of computer science. Combining classical machine learning methods with the power of quantum computing gives rise to a new field called quantum machine learning. In this thesis, we present a few quantum machine learning algorithms, which use the continuous-variable paradigm of quantum computing.

# Acknowledgement

I would like to thank my supervisors, Zoltán Zimborás and István Csabai for their efforts to help me with their guidance and for supporting my work during the writing of this thesis. I also would like to thank my colleagues, Zsófia Kallus and Péter Hága for their helpful comments and ideas which contributed to this thesis. I would also like to thank the Wigner GPU lab team for providing their high performance computing resources which accelerated our simulations.

# Contents

<b>1</b>	<b>Introduction</b>	<b>8</b>
<b>2</b>	<b>Quantum Computing with continuous variables</b>	<b>9</b>
2.1	Phase-space description . . . . .	10
2.2	Gaussian states . . . . .	12
2.2.1	Measurements . . . . .	13
2.3	CV quantum gates . . . . .	14
<b>3</b>	<b>Machine learning</b>	<b>19</b>
3.1	Supervised Classification and Regression . . . . .	19
3.1.1	Classification . . . . .	20
3.1.2	Regression . . . . .	20
3.2	Deep Neural Networks and backpropagation . . . . .	21
3.2.1	Backpropagation . . . . .	25
<b>4</b>	<b>Quantum Machine Learning</b>	<b>27</b>
4.1	Variational Quantum Eigensolvers . . . . .	27
4.1.1	Stochastic and doubly stochastic GD . . . . .	29
4.2	Quantum Neural Networks . . . . .	30
4.3	Calculating quantum gradients . . . . .	31
4.3.1	Heisenberg-picture representation . . . . .	32
4.3.2	Gradients in CV quantum circuits . . . . .	33
<b>5</b>	<b>Results</b>	<b>36</b>
5.1	Classification . . . . .	36
5.2	Regression . . . . .	39
5.3	CV-VQE for the Bose-Hubbard model . . . . .	42

5.3.1	Solving the model by exact diagonalization . . . . .	43
5.3.2	Variational solutions . . . . .	44
<b>6</b>	<b>Conclusion and future work</b>	<b>47</b>
<b>A</b>	<b>Mathematical preliminaries</b>	<b>48</b>
A.1	Hilbert spaces . . . . .	48
A.2	Linear operators on Hilbert spaces . . . . .	49
A.3	Hermitian Operators, Unitary Operators, Spectral theorem, Hadamard-lemma . . . . .	50
A.4	Pure and mixed quantum states . . . . .	51
<b>B</b>	<b>Coherent states</b>	<b>52</b>

# List of Figures

1	The three parts of a quantum circuit: initial state, unitary transformation and measurement.	15
2	Quantum circuit notation of the displacement gate. . . . .	15
3	Quantum circuit notation of the rotation gate. . . . .	16
4	Quantum circuit notation of the squeezing gate. . . . .	16
5	Quantum circuit notation of the beam-splitter gate. . . . .	17
6	Quantum circuit notation of the Kerr-interaction gate. . . . .	17
7	Quantum circuit notation of the CrossKerr-interaction gate. . . . .	17
8	Visualisation of a single neuron cell (image from Wikipedia). . . . .	22
9	The multilayer perceptron architecture. Green circles are the input features which can be scalar or vector values, the yellow circles represent the weights and biases of the perceptrons of the hidden layers, while the red circle is the output. Data flow is noted with black arrows. . . . .	23
10	The three most frequently used activation functions in deep neural networks. Left: the sigmoid function, middle: rectified linear unit (ReLU), right: leaky ReLU. . . . .	23
11	Computational graph of the backpropagation method for a feedforward neural network with a single hidden layer $\mathbf{h}$ , activation function $f$ and loss function $\mathcal{L}$ . . . . .	26
12	The four classes in which all machine learning tasks fit. The yellow box is actually containing all of the classical machine learning tasks, while the other three classes of algorithms somehow exploit quantumness. . . . .	27
13	Optimization loop of a hybrid-quantum classical variational model. . . . .	28
14	A single quantum layer of a general $N$ -mode fully connected QNN. $U_1$ and $U_2$ are general $N$ -port interferometers, while $D, S, K$ are single-mode displacement, squeezing and Kerr operators, respectively. . . . .	31
15	Visualization of the parameter-shift rule for a quantum circuit. . . . .	34
16	Ansatz circuit used in the classification task for the moons and circles datasets. . . . .	36
17	Ansatz circuit used in the classification task for the blobs dataset. . . . .	37

18	Results on the classification tasks on three datasets: blobs (a), circles (b), moons (c). The middle row shows the test set accuracy for each dataset with different number of layers. The bottom row shows comparisons between different SGD results with $k \in \{50, 100, 500, 1000\}$ shots and the exact baseline. . . . .	38
19	Ansatz circuit for the regression task. We use displacement encoding and six consecutive identical layers with a homodyne measurement at the end. . . . .	40
20	Comparison of the regression circuit performance on the test set with different number of layers. . . . .	40
21	The top row shows how the regression circuits perform on the test set. The figures show loss curves with different number of shots for the tanh function (a), the I-V curve (b) and the $x^2$ function (c). The bottom rows show inference tests with 1000 shots both for the tanh function (d), the I-V curve (d) and finally the $x^2$ function (e). All simulations were run with $\alpha = 0.001$ and mini-batch size 4. . . . .	41
22	(a) Visual representation of a sparse Hamiltonian matrix $H_{jj'}$ for a Bose-Hubbard model with $\dim(\mathcal{H}) = 35$ . Darker points show the non-zero elements in the matrix (b) Numerical solutions of the ground-state energy for different $U$ and $t$ values. . . . .	43
23	Illustration of a variational ansatz circuit with its first layer. Initially, each of the four qumodes contain a single photon. The layer is built up from Kerr-gates, CrossKerr gates and Beamsplitters, which are passive optical gates, so the total number of photons is preserved. . . . .	44
24	Comparison of the performance of the Bose-Hubbard VQE with different number of layers in the ansatz circuit. . . . .	45
25	Results of the Bose-Hubbard VQE. We see the simple stochastic results in the top row with shot numbers ranging between 5 and 5000. The bottom shows our results on the same system, when using the doubly stochastic gradient descent. In these figures, the blue continuous lines reoresent the $k \rightarrow \infty$ limit, while the dashed lines are the running medians of the stochastic results with pale regions showing the 5-95 percentile regions of the running window. We used window size of 30 steps in these plots. . . . .	46

# 1 Introduction

Quantum computing is probably the most promising emerging technology with many possible applications across all domains of science and business. The idea of quantum computing was first proposed by Richard P. Feynman as a method to simulate quantum mechanics [1]. Since the size of the Hilbert-space grows exponentially with the complexity of the quantum system, and thus the calculations become intractable on any classical computer, Feynmans idea was to simulate quantum physics on devices that behave themselves according to the rules of quantum physics.

Soon after Feynmans proposal, scientists became to establish the theoretical background of quantum information and quantum computing. Some of the quantum algorithms are proven to have an advantage over any known classical algorithm. One of the first such quantum algorithm is the Deutsch-Jozsa algorithm [2], which decides if a binary function  $f$  is balanced or constant. Probably the most notable quantum algorithm was proposed by Peter W. Shor in 1994, which is a polynomial-time quantum algorithm for factoring large integers [3]. This is the first quantum algorithm with a real-life application, because most of the public-key cryptosystems could be broken with an efficient algorithm for integer factoring. Another important quantum algorithm is the Grover's algorithm proposed by Lov K. Grover in 1996 for searching an unordered database [4]. While the best known classical algorithm for searching unordered databases runs in  $O(N)$  time, Grover's algorithm solves this problem with  $O(\sqrt{N})$  oracle queries. Beyond quantum algorithms, many quantum-inspired algorithms were discovered [5, 6, 7].

The attempts for physical realizations of quantum computers are rather diverse. Some companies like IBM use the superconducting qubit architecture [8] which is the most successful attempt to build large-scale fault tolerant quantum computer. Another promising way towards building quantum computers is by the use of trapped ions that are manipulated with lasers [9]. A bunch of companies placed their bets on photonics. The Canadian startup, Xanadu tries to realize fault-tolerant quantum computation with photonic GKP-states [10] on integrated photonic circuits [11]. Another Silicon-Valley based startup, PsiQuantum wants to build a commercially useful photonic quantum computer based on currently available CMOS technology [12]. Although there are many attempts to build quantum computers that are superior to the best classical computers, so far only the superconducting and the photonic architectures were able to achieve *quantum supremacy* [13, 14]. Achieving quantum computational supremacy does not mean that we have a large-scale universal quantum computer, in fact we are still in the era of noisy intermediate-scale quantum devices, commonly called the NISQ-era [15].

In this thesis we present our numerical results for some basic quantum machine learning tasks, with the use of a photonic quantum simulator. Therefore, we will briefly review the theoretical foundations of the continuous-variable quantum computation in chapter 2, then the basic ideas of machine learning and neural networks in chapter 3. Then in chapter 4 we describe the basics of the combination of quantum computing with machine learning, and finally in cahpter 5 we present our numerical results on three different quantum machine learning problems.



## 2 Quantum Computing with continuous variables

In this chapter, we introduce the mathematical formalism used in the description of continuous-variable (CV) quantum computing. This formalism is quite different from the formalism used in qubit-based quantum computing literature, mostly because CV systems have infinite degrees of freedom. We assume that the reader has basic knowledge of quantum mechanics, Hilbert-spaces, density matrices, bracket notation, etc. Some of this basic mathematical background is covered in appendix A. CV systems have infinite degrees of freedom, and therefore the corresponding Hilbert-space is of infinite dimensions. Such systems can be modeled by  $M$  harmonic oscillators, which are usually called *modes*. With  $H_k$  denoting the Hilbert-space of the  $k$ th mode, the Hilbert-space of the full CV system is their direct product:

$$\mathcal{H} = \bigotimes_{k=1}^M \mathcal{H}_k. \quad (1)$$

The Hamiltonian of mode  $j$  can be described by introducing bosonic creation and annihilation operators  $a_j^\dagger$  and  $a_j$ , obeying the following canonical commutation relations:

$$[\hat{a}_j, \hat{a}_k^\dagger] = \delta_{jk}, \quad [\hat{a}_j^\dagger, \hat{a}_k^\dagger] = [\hat{a}_j, \hat{a}_k] = 0. \quad (2)$$

The effect of the creation and annihilation operators on the Fock-basis states can be described by the following identities:

$$\begin{aligned} \hat{a}_k^\dagger |n_1, \dots, n_k, \dots\rangle &= \sqrt{n_k + 1} |n_1, \dots, n_k + 1, \dots\rangle \\ \hat{a}_k |n_1, \dots, n_k, \dots\rangle &= \sqrt{n_k} |n_1, \dots, n_k - 1, \dots\rangle \end{aligned} \quad (3)$$

We also introduce the number operator  $\hat{n}_j = \hat{a}_j^\dagger \hat{a}_j$ :

$$\hat{a}_k^\dagger \hat{a}_k |n_1, \dots, n_k, \dots\rangle = \hat{n}_k |n_1, \dots, n_k, \dots\rangle = n_k |n_1, \dots, n_k, \dots\rangle \quad (4)$$

Each single-mode Hilbert-space  $\mathcal{H}_k$  is an infinite-dimensional Fock-space spanned by the eigenvectors of the number operator  $\hat{n}_k = \hat{a}_k^\dagger \hat{a}_k$  labeled  $\{|n\rangle\}_k$ . It is often useful to introduce the canonical position operators  $\hat{X}_j$  and momentum operators  $\hat{P}_j$  which are related to the creation and annihilation operators by the following transformations:

$$\hat{X}_k = \frac{\hat{a}_k + \hat{a}_k^\dagger}{\sqrt{2}} \quad \hat{a}_k = \frac{\hat{X}_k + i\hat{P}_k}{\sqrt{2}} \quad (5)$$

$$\hat{P}_k = \frac{\hat{a}_k - \hat{a}_k^\dagger}{i\sqrt{2}} \quad \hat{a}_k^\dagger = \frac{\hat{X}_k - i\hat{P}_k}{\sqrt{2}} \quad (6)$$

Operators  $\hat{X}_k$  and  $\hat{P}_k$  are called quadrature operators. Following the notation often used in literature, we group these operators into a vector

$$\mathbf{R}^\top = (\hat{X}_1, \hat{P}_1, \hat{X}_2, \hat{P}_2, \dots, \hat{X}_M, \hat{P}_M) \quad (7)$$

We can calculate the commutators  $[\mathbf{R}_k, \mathbf{R}_l]$  by introducing a symplectic form  $\mathbf{\Omega}$  as

$$\mathbf{\Omega} = \bigoplus_{j=1}^M \boldsymbol{\omega}, \quad \boldsymbol{\omega} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \quad (8)$$

Using this notation, we can write

$$[\mathbf{R}_k, \mathbf{R}_l] = i\mathbf{\Omega}_{kl} \quad (9)$$

Apart from the Fock-basis, there is another useful alternative basis, the basis of *coherent states*, which are eigenstates of the annihilation operator  $\hat{a}_k$ :

$$\hat{a}_k |\alpha\rangle_k = \alpha |\alpha\rangle_k, \quad (10)$$

with  $\alpha \in \mathbb{C}$ . From definition this, we can derive that the Fock-basis expansion of the coherent state  $|\alpha\rangle_k$  is

$$|\alpha\rangle_k = e^{-\frac{1}{2}|\alpha|^2} \sum_{n=1}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle_k. \quad (11)$$

The derivation of the above formula is explained in detail in Appendix B. By introducing the single-mode Weyl-displacement operator

$$\hat{D}_k(\alpha) = e^{\alpha \hat{a}_k^\dagger - \alpha^* \hat{a}_k}, \quad (12)$$

we can create coherent states from the vacuum state:  $|\alpha\rangle_k = \hat{D}_k(\alpha) |0\rangle_k$ . Of course, we can similarly define multimode coherent states

$$|\alpha_1\rangle_1 \otimes |\alpha_2\rangle_2 \otimes \cdots \otimes |\alpha_N\rangle_N \equiv |\boldsymbol{\xi}\rangle = \hat{D}(\boldsymbol{\xi}) |0\rangle, \quad (13)$$

where  $\hat{D}(\boldsymbol{\xi})$  is the multi-mode Weyl-displacement operator given as

$$\hat{D}(\boldsymbol{\xi}) = e^{i\mathbf{R}^\top \mathbf{\Omega} \boldsymbol{\xi}}, \quad \boldsymbol{\xi} \in \mathbb{R}^{2N}. \quad (14)$$

For a single mode,

$$\hat{D}_k(\boldsymbol{\xi}) = \hat{D}_k \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} = e^{i(\xi_2 \hat{X}_k - \xi_1 \hat{P}_k)}, \text{ if we set } \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \text{Re}(\alpha_k) \\ \text{Im}(\alpha_k) \end{pmatrix}, \quad (15)$$

Eq. (14) reproduces Eq. (12). The introduction of the Weyl-displacement operator is necessary to carry out the phase-space description of CV-systems, which we will do in the next chapter.

## 2.1 Phase-space description

The description of CV quantum systems with the help of canonical creation and annihilation operators in the Fock-basis is equivalent to the description of these systems in the quadrature basis, which is called the phase-space description. The phase-space description is useful, because it makes possible to

calculate quantities easier than it would be in the Fock-basis description.

A CV quantum state  $\rho$  can be described in Fock-space by an infinite-sized matrix. This is problematic if we want to carry out actual calculations or simulate such systems. However, phase-space description can be used to deal with these problems. Instead of giving the elements of the density matrix  $\rho$ , we can use the *s-ordered characteristic functions* (see [16]) to describe the state:

$$\chi_\rho^s(\boldsymbol{\xi}) = \text{Tr} \left[ \rho \hat{D}(\boldsymbol{\xi}) \right] e^{s\|\boldsymbol{\xi}\|^2/2}, \quad \boldsymbol{\xi} \in \mathbb{R}^{2N}. \quad (16)$$

The  $2N$ -dimensional vectors  $\boldsymbol{\xi}$  lie in the *quantum phase-space*  $\Gamma$ . The quantum phase-space  $\Gamma = (\mathbb{R}^{2N}, \boldsymbol{\Omega})$  is a real  $2N$ -dimensional space equipped with the symplectic form  $\boldsymbol{\Omega}$ . While  $\Gamma$  is the quantum phase-space of the  $N$ -mode system, it can be decomposed into a direct sum of single-mode quantum phase spaces:

$$\Gamma = \bigoplus_{k=1}^N \Gamma_k, \quad (17)$$

where  $\Gamma_k = (\mathbb{R}^2, \boldsymbol{\omega})$  are the 2-dimensional phase spaces associated with the  $k$ th mode. Unlike in classical mechanics, where a state of the system is a single point in its phase-space, quantum states are not single points, but a region in the phase space. This is a consequence of the Heisenberg uncertainty principle, and the phase-space region corresponding to a state depends on the product of the uncertainties of the quadrature operators.

By applying a Fourier-transform to the  $s$ -ordered characteristic function, we can define a corresponding quasi-probability function:

$$W_\rho^s(\boldsymbol{\xi}) = \frac{1}{\pi^{2N}} \int_{\mathbb{R}^{2N}} \chi_\rho^s e^{i\boldsymbol{\kappa}^\top \boldsymbol{\Omega} \boldsymbol{\xi}} d^{2N} \boldsymbol{\kappa}. \quad (18)$$

$W_\rho^s(\boldsymbol{\xi})$  are called quasi-probabilities, because they sum up to 1, but they can have negative values or even be singular in some cases. There are a bunch of notable quasi-probability distributions associated to a state  $\rho$ : for  $s = -1$ ,

$$W_\rho^{-1} = \frac{1}{\pi} \langle \boldsymbol{\xi} | \rho | \boldsymbol{\xi} \rangle \quad (19)$$

is called the Husimi Q-function, and for  $s = 1$ , we have the so called P-representation or P-function, which is the representation of  $\rho$  in the coherent basis:

$$\rho = \int_{\mathbb{R}^{2N}} W_\rho^1(\boldsymbol{\xi}) |\boldsymbol{\xi}\rangle \langle \boldsymbol{\xi}| d^{2N} \boldsymbol{\xi}. \quad (20)$$

Perhaps the most important quasiprobability is the  $s = 0$  case, for which  $W_\rho^0 \equiv W_\rho$  is called the *Wigner-function*, and  $\chi_\rho^0 \equiv \chi_\rho$  is simply called the characteristic function. The Wigner-function can be expressed in the quadrature-basis, namely the basis spanned by the eigenvectors of the  $\hat{X}_j$  operators (the vectors satisfying  $\hat{X}_j = X_j | \mathbf{x} \rangle$ ), which we will denote  $| \mathbf{x} \rangle$ :

$$W_\rho(\mathbf{x}, \mathbf{p}) = \frac{1}{\pi^N} \int_{\mathbb{R}^N} \langle \mathbf{x} + \mathbf{y} | \rho | \mathbf{x} - \mathbf{y} \rangle e^{-2i\mathbf{p}\mathbf{y}} d^N \mathbf{y} \quad (21)$$

The Wigner-function is normalized, i.e

$$\int_{\mathbb{R}^{2N}} W_\rho(\mathbf{x}, \mathbf{p}) d^N \mathbf{x} d^N \mathbf{p} = \text{Tr} [\rho] = \chi_\rho(0) = 1, \quad (22)$$

and it can be used to calculate the *purity* of the state  $\mu_\rho$ :

$$\mu_\rho = \text{Tr} [\rho^2] = (2\pi)^N \int_{\mathbb{R}^{2N}} (W_\rho(\mathbf{x}, \mathbf{p}))^2 d^N \mathbf{x} d^N \mathbf{p} = \int_{\mathbb{R}^{2N}} |\chi_\rho(\boldsymbol{\xi})|^2 d^{2N} \boldsymbol{\xi}. \quad (23)$$

The Wigner-function also can be used to calculate expectation values of the quadrature operators, for example  $\langle X_1 \rangle$  can be calculated by marginalizing over all other variables, that is,

$$\langle X_1 \rangle = \int_{\mathbb{R}^{2N-1}} W_\rho(\mathbf{x}, \mathbf{p}) dp_1 \cdots dp_N dx_2 \cdots dx_N. \quad (24)$$

## 2.2 Gaussian states

Gaussian states are ubiquitous in quantum optics: in fact coherent states from lasers or thermal states from black-body radiation are all Gaussian states, as well as the vacuum state itself. In this chapter we briefly review the properties of Gaussian states.

**Definition 1.** *A Gaussian state is a quantum state  $\rho$  for which the characteristic function  $\chi_\rho$  and the Wigner-function  $W_\rho$  are Gaussian-distributions in the quantum phase-space  $\Gamma$ .*

A Gaussian state  $\rho$  can be defined by giving the first and second moments. For a state  $\rho$ , the first moment  $\mathbf{d}$  is given by

$$d_j = \langle \hat{R}_j \rangle_\rho, \quad (25)$$

while the second moment  $\boldsymbol{\sigma}$  (which is also called covariance matrix) is defined as

$$\sigma_{ij} = \langle \hat{R}_i \hat{R}_j + \hat{R}_j \hat{R}_i \rangle_\rho - 2\langle \hat{R}_i \rangle_\rho \langle \hat{R}_j \rangle_\rho. \quad (26)$$

The first moment  $\mathbf{d}$  is a real vector of length  $2N$ , while  $\boldsymbol{\sigma}$  is a real  $2N \times 2N$  symmetric matrix. With these notations, we can express the characteristic function and Wigner-function of a Gaussian state, as

$$\chi_\rho(\boldsymbol{\xi}) = e^{-\frac{1}{4}\boldsymbol{\xi}^\top \boldsymbol{\Omega} \boldsymbol{\sigma} \boldsymbol{\Omega}^\top \boldsymbol{\xi} - i(\boldsymbol{\Omega} \mathbf{d})^\top \boldsymbol{\xi}}, \quad (27)$$

$$W_\rho(\boldsymbol{\xi}) = \frac{1}{\pi^N \sqrt{\det \boldsymbol{\sigma}}} e^{-(\boldsymbol{\xi} - \mathbf{d})^\top \boldsymbol{\sigma}^{-1} (\boldsymbol{\xi} - \mathbf{d})}. \quad (28)$$

As an example the first and second moments of a single-mode coherent state  $|\alpha\rangle$  are

$$\mathbf{d} = \begin{pmatrix} \text{Re}(\alpha) \\ \text{Im}(\alpha) \end{pmatrix}, \quad \boldsymbol{\sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \quad (29)$$

We mentioned, that the vacuum state  $|0\rangle$  is a Gaussian state. This is true, since the vacuum state is also a coherent state, since it is the eigenstate of the annihilation operator  $\hat{a}_k$  with eigenvalue 0:

$|0\rangle_k = 0|0\rangle_k$ . We can calculate the Wigner-function for the single-mode vacuum state:

$$\hat{a}|0\rangle = \frac{1}{\sqrt{2}}(\hat{X} + i\hat{P}) \int dX |X\rangle \langle X|0\rangle = \int dX |X\rangle \left(X + \frac{\partial}{\partial X}\right) \underbrace{\langle X|0\rangle}_{=\psi_0(X)}, \quad (30)$$

and therefore,

$$\left(X + \frac{\partial}{\partial X}\right) \psi_0(X) = 0 \implies \psi_0(X) = \frac{1}{\pi^{1/4}} e^{-\frac{X^2}{2}}, \quad (31)$$

which is obviously a Gaussian function. From  $\psi_0(X)$ , the Wigner-function reads

$$W_{|0\rangle}(X, P) = \frac{1}{\pi} e^{-X^2 - P^2}. \quad (32)$$

### 2.2.1 Measurements

We work with two types of measurements in quantum mechanics, the projective measurements and POVMs. The projective measurements (a.k.a. von Neumann measurements) are defined by a set of positive Hermitian operators  $\{\hat{\Pi}_i\}$ , which sum up to the identity and are mutually orthogonal:

$$\sum_i \hat{\Pi}_i = \mathbb{1}, \quad \hat{\Pi}_i \hat{\Pi}_j = \delta_{ij} \hat{\Pi}_j. \quad (33)$$

Projective measurement are mapping the quantum state  $\rho$  into state

$$\rho_i = \frac{\hat{\Pi}_i \rho \hat{\Pi}_i}{\text{Tr} [\hat{\Pi}_i \rho \hat{\Pi}_i]}, \quad (34)$$

with probability  $p_i = \text{Tr} [\hat{\Pi}_i \rho \hat{\Pi}_i]$  [16]. We can measure the subsystem of a composite system as well. Let us consider a quantum state  $\rho_{AB}$  of a joint quantum system. In this case if we measure subsystem  $A$  with projective operator  $\hat{\Pi}_i = \hat{\Pi}_{i,A} \otimes \mathbb{1}_B$ , the subsystem  $B$  will be projected into

$$\rho_{B, \hat{\Pi}_i} = \text{Tr}_B \left[ \frac{\hat{\Pi}_i \rho_{AB} \hat{\Pi}_i}{\text{Tr} [\hat{\Pi}_i \rho_{AB} \hat{\Pi}_i]} \right]. \quad (35)$$

POVMs consist of a set of positive operators  $\{\hat{\Pi}_i\}$ , which sum up to identity, just as projective measurements, but the orthogonality condition is not required. While projective measurements are ideal measurements, POVMs provide better mathematical description of the measurements actually happening in an experimental setup, hence, POVMs are widely used in quantum information theory. We mention a special class of measurements:

**Definition 2.** A measurement on a CV quantum system, which projects Gaussian states into Gaussian states is called a **Gaussian measurement**.

In CV quantum computing, we usually encounter the following three types of measurements:

- Photon-number resolving measurements. These measurements are mapping the quantum state to Fock-space eigenstates  $|n_1, \dots, n_N\rangle$ . Photon number-resolving measurements are ideal projective measurements summing up to the identity:

$$\sum_j \hat{\Pi}_j = \sum_{n_1, \dots, n_N} |n_1, \dots, n_N\rangle \langle n_1, \dots, n_N| = \mathbb{1}. \quad (36)$$

In theory, by using photon counting measurements we could be able to differentiate between any two excitation levels, but these measurements are quite challenging to realize experimentally.

- Homodyne detections. Homodyne measurements are realized by mixing the initial state  $\rho$  with a coherent state  $\alpha$  with the help of a beam-splitter and then applying photodetectors (see [17]). These are Gaussian measurements, which measure the quadrature operator  $\hat{X}_\phi = \cos \phi \hat{X} + \sin \phi \hat{P}$ , with outcome probabilities

$$p(\phi) = \langle x_\phi | \rho | x_\phi \rangle, \quad (37)$$

where  $|x_\phi\rangle$  are eigenstates of  $\hat{X}_\phi$ .

- Heterodyne detection. A heterodyne detection (on a single mode) consists in a measurement by POVMs defined as

$$\mathbb{1} = \int_{\mathbb{C}} \frac{|\alpha\rangle \langle \alpha|}{\pi} d^2\alpha, \quad (38)$$

which map quantum states into coherent outcome state  $|\alpha\rangle$  with probabilities  $p(\alpha) = \frac{1}{\pi} \langle \alpha | \rho | \alpha \rangle$ .

In quantum computing literature, we often use the term **shot** for a single measurement outcome. In practice, quantum measurements are always composed from a finite number of measurement outcomes, and therefore the probabilities of  $p_k$  associated to measurement operators  $\hat{\Pi}_k$  are always approximated by

$$\tilde{p}_k = \frac{N_k}{N_{\text{shots}}}, \quad (39)$$

where  $N_k$  is the number of individual measurements in which the  $k$ -th outcome happened.

## 2.3 CV quantum gates

There are several different models of computation in the theory of quantum computing, but the most used among these is the *gate-based* model of quantum computation. This is very similar to the description of the digital logic circuits, where the wires represent actual electrical wires and the gates represent simple digital logic units built of transistors. In the circuit model of quantum computation, wires represent the free time-evolution of a qubit, or qumode, and the gates represent some operator acting on the system. In quantum photonics, the free time-evolution is usually the propagation of photons in a waveguide, hence the wires in this case are usually representing these waveguides. In the circuit model of quantum computation the system always starts in some initial state  $|\psi_0\rangle$ , then a unitary transformation  $\hat{U}$  is applied to the initial state, and finally some measurement is done on the system (see Fig. 1).

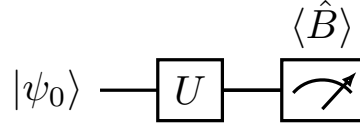


Figure 1. The three parts of a quantum circuit: initial state, unitary transformation and measurement.

In this chapter we present the most important quantum gates used in continuous variable quantum computing. We follow the notation preferred in the work of Adesso et al. [16], which also used in the documentation of Strawberry Fields [18, 19]. In the previous chapter we saw that a special class of quantum states is the set of Gaussian states, and this is true for quantum operations too: we differentiate between Gaussian and non-Gaussian quantum operations. A general Gaussian operation  $\mathcal{G}$  can be described as

$$\mathcal{G} : \begin{bmatrix} \mathbf{x} \\ \mathbf{p} \end{bmatrix} \mapsto \mathbf{M} \begin{bmatrix} \mathbf{x} \\ \mathbf{p} \end{bmatrix} + \begin{bmatrix} \text{Re}(\boldsymbol{\alpha}) \\ \text{Im}(\boldsymbol{\alpha}) \end{bmatrix}, \quad (40)$$

where  $\boldsymbol{\alpha} \in \mathbb{C}^N$  and  $\mathbf{M}$  is a symplectic matrix. A matrix  $\mathbf{M}$  is called symplectic if  $\mathbf{M}^\top \boldsymbol{\Omega} \mathbf{M} = \boldsymbol{\Omega}$ , is the  $2N \times 2N$  symplectic form similar to Eq. (8):

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & \mathbb{1} \\ -\mathbb{1} & 0 \end{bmatrix}. \quad (41)$$

Gaussian operations include displacement, squeezing, rotation and beam-splitters, while the Kerr and the cross-Kerr gates are non-Gaussian. Another important feature of CV operations worth mentioning is whether they are active or passive. Active gates can change the total energy of the system, while passive gates leave its energy unchanged. For example the squeezing and displacement gates are active transformations, while the rotation, beam-splitter, Kerr- and cross-Kerr gates are passive.

**Displacement.** The displacement gate was basically already introduced in eq. (12), and is described as a translation of the state in phase-space, without modifying the shape of the distribution. The gate notation of the displacement operator is shown on fig. 2.

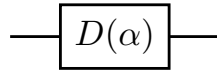


Figure 2. Quantum circuit notation of the displacement gate.

The effect of the displacement operator on a single mode can be expressed with the canonical ladder operators as

$$\hat{D}(\alpha) = e^{\alpha \hat{a}^\dagger - \alpha^* \hat{a}} = \exp \left[ r \left( e^{i\phi} \hat{a}^\dagger - e^{-i\phi} \hat{a} \right) \right], \quad (42)$$

with  $\alpha = r e^{i\phi} \in \mathbb{C}$ ,  $r \geq 0$ ,  $\phi \in [0, 2\pi)$ . It can also be expressed in the quadrature basis as an affine transformation:

$$\hat{D}(\alpha) : \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} x + \text{Re}(\alpha) \\ p + \text{Im}(\alpha) \end{bmatrix}. \quad (43)$$

**Rotation.** Rotation gates, as the name suggests, are rotating the quantum state in phase space with angle  $\phi$ . See fig. (3) for the circuit notation of rotation gates.

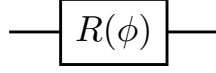


Figure 3. Quantum circuit notation of the rotation gate.

The Fock-basis and quadrature basis representations of a single-mode rotation gate are

$$\hat{R}(\phi) = e^{i\phi\hat{a}^\dagger\hat{a}} = \exp\left[\frac{\phi}{2}\left(\frac{\hat{X}^2 + \hat{P}^2}{\hbar} - \hat{I}\right)\right], \quad (44)$$

while the effect of  $\hat{R}(\phi)$  on a state is

$$\hat{R}(\phi) : \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix}. \quad (45)$$

**Squeezing.** The squeezing operator reduces the variance of the state to some degree in some phase  $\phi$ , while it increases the variance in the direction perpendicular to the former, to not violate the Heisenberg uncertainty principle. The circuit notation of the squeezing gate is shown on fig. 4.

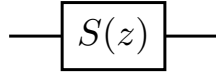


Figure 4. Quantum circuit notation of the squeezing gate.

The Fock-basis representation of the single-mode squeezing operator  $\hat{S}(z)$  is

$$\hat{S}(z) = \exp\left[\frac{1}{2}\left(z^*\hat{a}^2 - z\hat{a}^{\dagger 2}\right)\right] = \exp\left[\frac{r}{2}\left(e^{-i\phi}\hat{a}^2 - e^{i\phi}\hat{a}^{\dagger 2}\right)\right], \quad (46)$$

with  $z = re^{i\phi} \in \mathbb{C}$ ,  $r \geq 0$ ,  $\phi \in [0, 2\pi)$ , while its effect on a quantum state in the phase-space formalism is

$$\hat{S}(z) = \hat{S}(r, \phi) : \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} \cosh(r) + \cos(\phi) \sinh(r) & \sin(\phi) \sinh(r) \\ \sin(\phi) \sinh(r) & \cosh(r) - \cos(\phi) \sinh(r) \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix}, \quad (47)$$

or in the phase-free case

$$\hat{S}(r) : \begin{bmatrix} x \\ p \end{bmatrix} \mapsto \begin{bmatrix} e^{-r} & 0 \\ 0 & e^r \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix} \quad (48)$$

**Beam-splitter.** The beam-splitter operator is a general two-mode Gaussian operation, which has two real parameters and its circuit notation is shown on fig. 5.



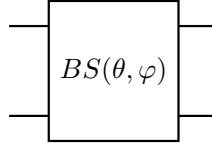


Figure 5. Quantum circuit notation of the beam-splitter gate.

The Fock-basis representation of the beam-splitter acting between modes  $j$  and  $k$  is

$$\hat{BS}(\theta, \varphi) = \exp \left[ \theta \left( e^{i\varphi \hat{a}_j \hat{a}_k^\dagger} - e^{-i\varphi \hat{a}_j^\dagger \hat{a}_k} \right) \right], \quad (49)$$

while the effect of a phase-free ( $\varphi = 0$ ) beam-splitter in the phase-space is

$$\hat{BS}(\theta) : \begin{bmatrix} x_j \\ x_k \\ p_j \\ p_k \end{bmatrix} \mapsto \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & \cos(\theta) & -\sin(\theta) \\ 0 & 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_j \\ x_k \\ p_j \\ p_k \end{bmatrix}. \quad (50)$$

**Kerr-interaction.** The Kerr-interaction described by the Hamiltonian  $\hat{H} = \hat{n}^2$  is a non-linear effect [20], and therefore the Kerr-gate is a non-Gaussian operator (see fig. 6 for the circuit notation).

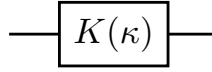


Figure 6. Quantum circuit notation of the Kerr-interaction gate.

The Kerr-gate is a one-parameter single-mode gate, which can be described in the Fock-basis with the formula

$$\hat{K}(\kappa) = e^{i\kappa \hat{n}^2} = e^{i\kappa (\hat{a}^\dagger \hat{a})^2}, \quad (51)$$

and since it is a non-Gaussian gate, it is a non-linear transformation of the phase-space variables  $(x, p)$ .

**Cross-Kerr interaction.** The cross-Kerr interaction (CK) describes a two-mode non-linear interaction between modes  $j$  and  $k$  with the Hamiltonian  $\hat{H} = \hat{n}_j \hat{n}_k$ . The circuit notation of the CK-gate is shown on fig. 7.

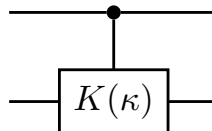


Figure 7. Quantum circuit notation of the CrossKerr-interaction gate.

The cross-Kerr interaction can be expressed with number operators  $\hat{n}_j$  and  $\hat{n}_k$  in the Fock-basis as

$$\hat{CK}(\kappa) = e^{i\kappa \hat{n}_j \hat{n}_k} = e^{i\kappa \hat{a}_j^\dagger \hat{a}_j \hat{a}_k^\dagger \hat{a}_k}, \quad (52)$$

and describes a nonlinear transformation of phase-space variables  $(x_j, x_k, p_j, p_k)$ .

## 3 Machine learning

Machine learning is simply said, a new paradigm in software development. Traditional software have a predefined behavior, which does not change over time. In contrast, machine learning models are iteratively "trained", i.e., updated to achieve better performance by the use of data. These algorithms are designed to learn patterns from the data available and update themselves with new experience. Although there are a lot of machine learning algorithms, they usually fit into one of the following three main classes:

- **Supervised Learning (SL)** algorithms need labeled data, and based on the input-output pairs previously seen by the algorithm, it learns to produce correct outputs for previously unseen inputs. Some very basic examples of supervised learning are the regression models or support vector machines for classification, but there are a lot of complex models like CNNs for image classification, which are trained in a supervised manner.
- **Unsupervised Learning (UL)** does not require labeled data, instead of that, it is trying to learn from a lot of unlabeled data points. These algorithms include a set of clustering methods, like k-means or mixture models. Another example of unsupervised learning is the usage of latent variables or latent distributions for feature extraction or data denoising. A great example of latent distribution learning are Variational Autencoders [21] which have been adopted for a variety of tasks.
- **Reinforcement Learning (RL)** is very different from the previously mentioned two classes of machine learning method. In RL the goal is not to learn from a given set of data, but rather to solve a specific task in an environment. In RL, an *agent* learns to perform a task by observing the state (or some subspace of the state) of the surrounding environment, and executing actions in order to maximize an objective. RL is inspired by the process of learning from mistakes or rewards. The environment in RL can be any type of physical simulation, or even a computer game. Examples for RL algorithms include Deep Q-networks [22] or AlphaGo [23].

### 3.1 Supervised Classification and Regression

The two most common SL tasks are **classification** and **regression**. Both classification and regression can be performed with traditional algorithms or deep neural networks. In this chapter we present the basic ideas and formalism of classification and regression which will be useful in the following chapters.

In supervised learning the data points consist of pairs  $(\mathbf{x}_i, \mathbf{y}_i)$ , where  $\mathbf{x}_i$  are called feature vectors and  $\mathbf{y}_i$  are called labels. Both the feature vectors and the labels are real-valued vectors of any dimension. Feature vectors can be pre-processed to achieve better performance. A dataset is usually split into two or three parts. The first part is usually the largest, which is called the training set, and this is used to train the model. The second dataset is called testing or evaluation set and is used to evaluate how the model performs when it is fed with previously unseen data. There might be an additional set called

the validation set, which is used to verify the model's performance after training is finished. While the testing set is used to monitor the learning process during training, the validation set is used only after the full training process is finished.

### 3.1.1 Classification

In classification, each point  $\mathbf{x}$  is in one of  $K$  predefined categories. That is, the labels are integers less or equal to  $K$ :  $y_i \in \{1, \dots, K\}$ . The goal of a classifier is to learn the connection between  $\mathbf{x}_i$  and  $y_i$ , in other words, to learn, how to put each vector  $\mathbf{x}_i$  into the right category. The performance of a classifier is usually measured with categorical accuracy i.e. what percentage of the predicted labels are in the right category. Apart from accuracy, it is often useful to introduce the categorical cross-entropy. In general, the crossentropy of two probability distributions  $p$  and  $q$  is defined as

$$H(p, q) = - \int p(x) \log q(x) dx, \quad (53)$$

and roughly saying, it measures the difference between the distribution  $q$  and the target distribution  $p$ . To transform this general formula of cross-entropy to fit the case of classification, we need to introduce one-hot encoding. One-hot encoding maps integer labels to binary vectors in a way that vector corresponding to the label  $j$  will contain 1 at the  $j$ th position and 0 everywhere else. The length of the resulting vectors is  $K$ . For example the one-hot encoding of labels 1 and 3 in case of three categories would look like

$$\text{OneHot}(1, 3) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{OneHot}(3, 3) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (54)$$

In machine learning, especially in deep learning one-hot encoding of categorical features or labels is proven to be very useful. Now, with one-hot encoded labels, we can easily define the categorical crossentropy:

$$\text{CrossEntropy} : \mathbb{R}^K \times \mathbb{R}^K \rightarrow \mathbb{R}, \text{CrossEntropy}(\mathbf{y}^{\text{pred}}, \mathbf{y}^{\text{true}}) = - \sum_{i=1}^K y_i^{\text{true}} \log y_i^{\text{pred}} \quad (55)$$

The crossentropy essentially measures how the predicted distribution of labels differs from the true distribution, hence it is widely used in machine learning. Applications of classification include image recognition, image segmentation, speech and voice recognition, fraud detection and a lot more.

### 3.1.2 Regression

Regression models are usually used for forecasting and prediction, or simply to uncover the relationships between two or more features. Opposed to classification, in regression, the output of the model is not restricted to a discrete set of predefined classes, instead the set of possible outputs is a bounded or

unbounded subset of  $\mathbb{R}^d$ . Therefore, classification is a special case of regression where the set of possible values of the outcomes is a finite set of integers. Apart from this difference, the purpose of regression is the same: to find a connection between the features  $\mathbf{x}$  and labels  $\mathbf{y}$ . The elements of feature vectors  $x_i$  are usually called independent variables or predictors, while the elements of the label vectors  $y_i$  are called dependent variables. Formally the regression model is a function approximator, which is itself a function that can be parametrized by a set of real parameters  $\boldsymbol{\theta} \in \mathbb{R}^m$ . Given a set of observations  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$  a function approximator  $f(\cdot; \boldsymbol{\theta})$ , we can write

$$y_i = f(\mathbf{x}_i; \boldsymbol{\theta}) + \epsilon_i, \quad (56)$$

where  $\epsilon_i$  denotes a random statistical noise, usually assumed to follow Gaussian distribution. The goal is to find an optimal set of parameters  $\boldsymbol{\theta}^*$  which minimizes the mean squared error:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{k=1}^N (y_k - f(\mathbf{x}_k; \boldsymbol{\theta}))^2. \quad (57)$$

The mean squared error is itself a good measure of the performance of the regression model and is often used in machine learning and deep learning. Another scalar value for getting some information about the goodness of fit is the coefficient of determination or often called  $R^2$ -score. The  $R^2$ -score is a real value usually falling between zero and one (the latter meaning perfect fit), although sometimes it can be negative. The  $R^2$ -score measures the proportion of variance in the predictions that can be explained by the data. To calculate the  $R^2$ -score, we first introduce the total sum of squares

$$SS_{\text{tot}} = \sum_i (y_i^{\text{true}} - \langle y^{\text{true}} \rangle)^2, \quad (58)$$

where  $\langle y^{\text{true}} \rangle = \frac{1}{N} \sum_k y_k^{\text{true}}$  is the mean of the dependent variable in the dataset. Then we calculate the sum of squared residuals, or sum of squared errors:

$$SS_{\text{res}} = \sum_i (y_i^{\text{true}} - y_i^{\text{pred}})^2. \quad (59)$$

With these two quantities, we can now define the  $R^2$ -score as follows:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}. \quad (60)$$

In this thesis, we will use the above definition of  $R^2$ -score to measure the performance of our models.

## 3.2 Deep Neural Networks and backpropagation

Probably the most successful machine learning models are deep neural networks, which enjoy a great popularity in recent years because of their flexibility and expressive power [24]. In this chapter we give a short introduction to deep neural networks by describing the basic ideas behind training neural networks

for various tasks.

The human brain is very good at some tasks in which computers are quite bad. While a computer can calculate that  $567,213 \times 789,133 = 447,606,496,329$  in a few milliseconds, it can not decide whether a picture shows a dog or a cat. In contrary humans are very good at recognizing pictures and deciding whether a picture shows a dog or a cat takes less than a second for a brain. Therefore the idea to mimic the brain with computers and solve tasks with brain-like algorithms seems plausible. Although the basic idea is very simple, it took scientists decades to make practically useful neural networks and training neural networks still requires huge amounts of computational power. To mimic the neural network of the brain, we need to come up with a mathematical model that is simple enough to be simulated on computers, but still complex enough to be able to learn how to do difficult tasks. Not surprisingly each neuron will be modeled by a parametric function  $f(\mathbf{x}; \mathbf{W}, \mathbf{b})$ . Before defining the function  $f$  explicitly we should review, how biological neurons work.

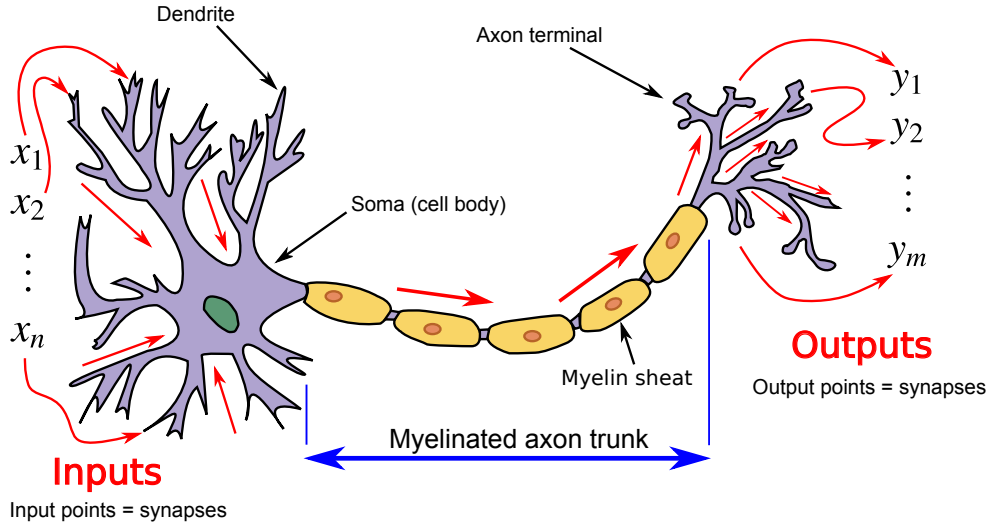


Figure 8. Visualisation of a single neuron cell (image from [Wikipedia](#)).

Figure 8 shows a biological neuron with input and output synapses. Each of the input and output synapses is connected to other cells forming the neural network with billions of individual cells. Without diving into the biochemical details, we can say that information is carried down the axon in the form of spike-like electric pulses, and transmitted through the synapse with the help of ions neurotransmitters and positive ions. The electrical potential in the cells is typically  $-70\text{mV}$ . When a synapse receives a signal, this potential slightly increases, and when it reaches a threshold of  $\approx -55\text{mV}$ , the cell fires, and a signal will be sent to the output synapses. Different input synapses have different importance, and thus they contribute to the output signal with different coefficients. Therefore, we can say that the output frequency of the  $i$ -th neuron is modeled by

$$y_i = \theta \left( \sum_j J_{ij} x_j - U_0 \right), \quad (61)$$

where  $\theta(x)$  is the Heaviside-step function,  $x_j$  are input frequencies and  $U_0 = -55\text{mV}$  is the threshold potential. From this model, we can derive a more general neuron model, which will be used to train

deep neural networks:

$$\mathbf{y} = g(\mathbf{W}^\top \mathbf{x} + \mathbf{b}), \quad (62)$$

where  $g$  is called *activation function*,  $\mathbf{W}$  and  $\mathbf{b}$  are the *weights* and *biases*, respectively. Some commonly used activation functions are shown on Fig. 10. This model of a single neuron is called a *perceptron*.

The simplest form of deep neural network uses many perceptrons and is called a *multilayer-perceptron* (MLP, see Fig. 9) or *feedforward neural network*. In this case the input vectors are fed into many perceptrons, and the outputs of these is fed into the perceptrons of the next layer. This architecture is often called fully connected or dense neural network, and the layers of perceptrons are called hidden layers. The outputs of an MLP can be scalars vector, or in some cases tensors of higher rank. The fully connected neural network structure is a very powerful function approximator, in fact a neural network composed of affine transformations and continuous non-polynomial activation functions can approximate any “well-behaved” function [25].

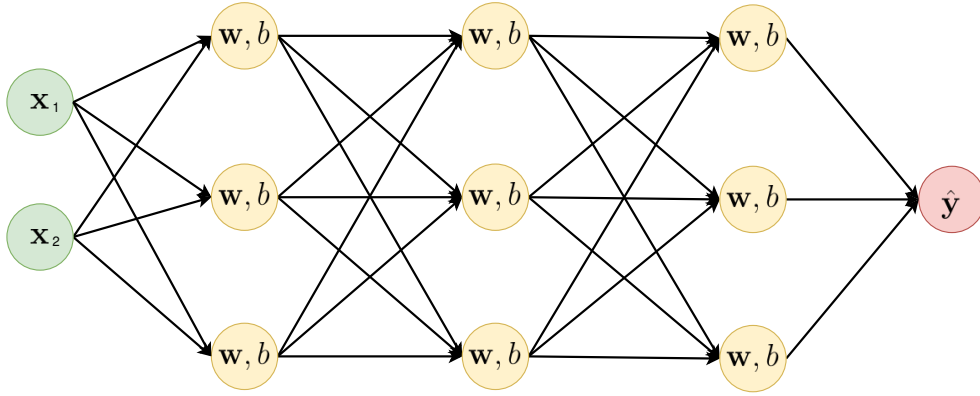


Figure 9. The multilayer perceptron architecture. Green circles are the input features which can be scalar or vector values, the yellow circles represent the weights and biases of the perceptrons of the hidden layers, while the red circle is the output. Data flow is noted with black arrows.

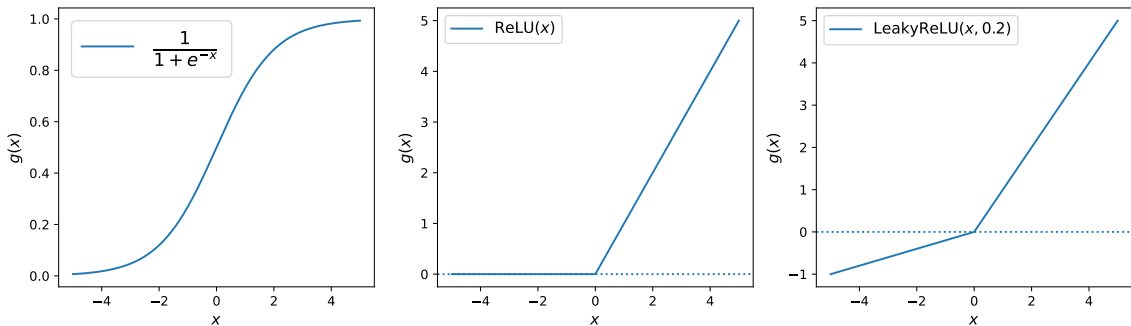


Figure 10. The three most frequently used activation functions in deep neural networks. Left: the sigmoid function, middle: rectified linear unit (ReLU), right: leaky ReLU.

To use deep neural networks in a classification task, it is often useful to introduce the **Softmax**

activation function

$$\text{Softmax} : \mathbb{R}^m \rightarrow \mathbb{R}^m, [\text{Softmax}(\mathbf{y})]_j = \frac{e^{y_j}}{\sum_k e^{y_k}}. \quad (63)$$

This activation function comes handy, because the outputs of a neural network  $y_i^{\text{pred}}$  usually are not forming a probability distribution, however, categorical loss functions like **CrossEntropy** (see Eq. (55)) require  $\mathbf{y}^{\text{pred}}$  to be a valid probability distribution. The **Softmax** function turns a sequence of arbitrary real values into a probability distribution, and therefore it is often used in classification tasks.

There are many more sophisticated neural network architectures like convolutional neural networks (CNNs) [26], which are useful for image processing tasks, recurrent neural networks with internal long term memory [27] for speech recognition or time-series forecasting, or transformers [28] for machine translation. To dive into the details of these complex deep neural network architectures is out of the scope of this thesis. We will, however explain the gradient descent (GD) algorithm, which makes it possible to train complex neural networks with the help of backpropagation, which we will explain in chapter 3.2.1. Essentially, training neural networks means searching for the best weights  $(\mathbf{W}^*, \mathbf{b}^*) \equiv \boldsymbol{\theta}^*$ , for which a defined loss function  $\mathcal{L}$  has minimum:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta})), \quad (64)$$

where  $f(\cdot; \boldsymbol{\theta})$  represents the neural network. The idea of gradient descent is to find  $\boldsymbol{\theta}^*$  by starting at a random point  $\boldsymbol{\theta}^{(0)}$  and at each timestep update theta in the direction of gradient. In the standard version of gradient-descent, we calculate the average loss  $L = \frac{1}{N} \sum_{k=1}^N \mathcal{L}(\mathbf{y}_k, f(\mathbf{x}_k; \boldsymbol{\theta}^{(t)}))$  for the entire dataset to do a single step of gradient descent (see algorithm 1).

---

**Algorithm 1** Gradient Descent

---

```

1: procedure GRADIENTDESCENT( $\{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^N, f, \boldsymbol{\theta}^{(0)}, \mathcal{L}, \alpha^{(0)}, T$ )
2:   for all  $t \in \{0, \dots, T\}$  do
3:     Calculate  $L = \frac{1}{N} \sum_{k=1}^N \mathcal{L}(\mathbf{y}_k, f(\mathbf{x}_k; \boldsymbol{\theta}^{(t)}))$ 
4:     for all  $\theta_j \in \boldsymbol{\theta}$  do
5:       Calculate the gradients  $\left. \frac{\partial L}{\partial \theta_j} \right|_{\theta_j = \theta_j^{(t)}}$ 
6:       Update the parameter  $\theta_j^{(t+1)} \leftarrow \theta_j^{(t)} - \alpha_j^{(t)} \left. \frac{\partial L}{\partial \theta_j} \right|_{\theta_j = \theta_j^{(t)}}$ 
7:     end for
8:   end for
9: end procedure

```

---

To apply gradient-descent methods, we need to require  $f(\cdot; \boldsymbol{\theta})$  and  $\mathcal{L}$  to be differentiable in each of the parameters  $\theta_j$ . In many cases, the dataset is very large, and it turned out to be useful to split the dataset into mini-batches of a small size and calculate the losses and gradients for each individual min-batch. This version of gradient descent is called mini-batch stochastic gradient descent or mini-batch SGD, sometimes simply SGD. Using SGD instead of GD not only improves the overall perform



ance of the models, but usually reduces time needed for convergence. In algorithm 1,  $\alpha_j^{(t)}$  are called learning rates, which determine the magnitude of the gradient step. We used this notation because most modern GD algorithms like Adam [29] use different learning rates for each parameter, and the value of learning rates is updated after each iteration. Initially, the learning rates are typically in the range  $[10^{-4}, 10^{-1}]$ . Since the loss function  $\mathcal{L}$  is in most cases not a globally convex hyperplane, it is very hard to find its global minimum by GD. In fact, it is not even possible to tell if the optimum found by a deep neural network is a global or local optimum. Therefore it is possible for a network to get stuck in a local minimum, which is suboptimal. There are two solutions to this problem: one can run the same model parallelly with different initial parameters  $\theta^{(0)}$ , or we can use a cyclic learning rate. Using a cyclic learning rate means, that when the variance of  $\mathcal{L}$  reaches close to zero, we increase the learning rate, so we try to push the model out of the minimum, in the hope that it will find a better solution.

### 3.2.1 Backpropagation

In the previous chapter we introduced the gradient-descent algorithm, which is an effective method to optimize neural networks. This method requires one to calculate the gradients  $\frac{\partial L}{\partial \theta_j}$  for each parameter  $\theta_j$  of the network. Modern deep neural networks have millions of parameters, so an efficient method for gradient calculation is crucial in the development of deep learning models. The training of large neural networks with gradient-descent became possible after the proposal of backpropagation algorithm in 1986 by Geoffrey Hinton et al. [30]. Modern machine learning frameworks like TensorFlow [31] and Pytorch [32] implement the back-propagation algorithm, a method to calculate the gradients automatically from the loss. In this chapter we shortly present the idea behind backpropagation.

Since we required the neural network  $f(\cdot; \theta)$  to be differentiable in each of the parameters, gradients always exist, theoretically at least, although they can 'explode' in some cases due to numerical instability. Backpropagation is an algorithm, which is essentially using the chain-rule of differentiation, since the neural network is composed of a few basic functions which are themselves differentiable.

Instead of diving into the mathematical details of the backpropagation algorithm, we show, how it works in the simplest case. Let us define a model with a single scalar input vector  $\mathbf{x}$ , a single weight matrix  $\mathbf{W}$  and an activation function  $f(x) = \text{sigmoid}(x)$  which is applied to the hidden units element-wise. Furthermore, let us use **CrossEntropy** loss function on the output of this simple neural network. We will have to calculate a single derivative,  $\frac{\partial \mathcal{L}}{\partial W_{kl}}$ , in order to train this model with GD. To calculate  $\frac{\partial \mathcal{L}}{\partial W_{kl}}$ , we only need to store the results obtained in the forward step and use the chain rule. Figure 11 shows the *computational graph* depicting backpropagation for this simple case and table 1 contains the key equations for the forward and backward step as well.

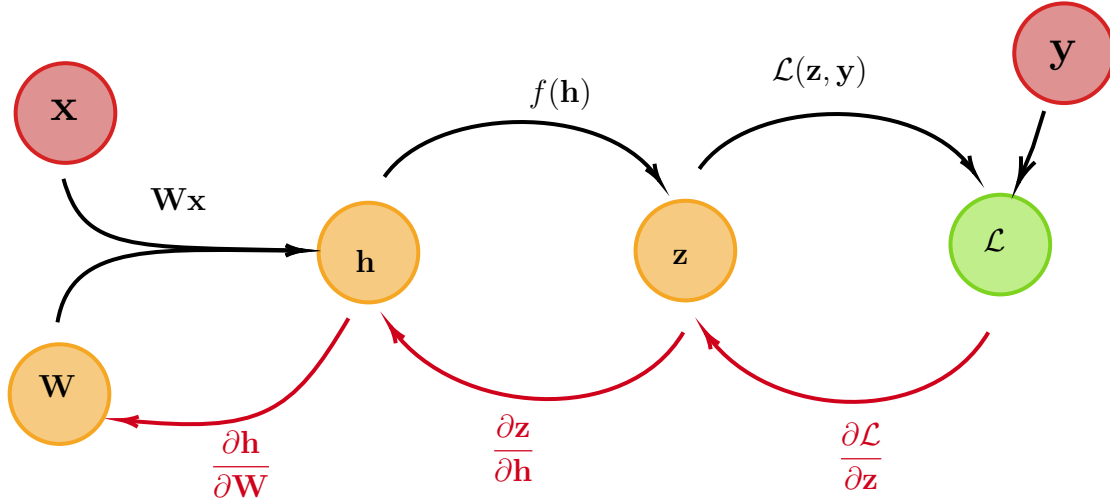


Figure 11. Computational graph of the backpropagation method for a feedforward neural network with a single hidden layer  $\mathbf{h}$ , activation function  $f$  and loss function  $\mathcal{L}$ .

Forward step	Backward step
$h_i = \sum_j W_{ij} x_j$	$\frac{\partial h_i}{\partial W_{kl}} = \sum_j x_j \delta_{ik} \delta_{jl} = x_l \delta_{ik}$
$z_i = f(h_i) = \frac{1}{1 + e^{-h_i}}$	$\frac{\partial z_i}{\partial h_i} = \frac{e^{-h_i}}{(1 + e^{-h_i})^2} = f(h_i)^2 e^{-h_i}$
$\mathcal{L}(\mathbf{z}, \mathbf{y}) = - \sum_i y_i \log z_i$	$\frac{\partial \mathcal{L}}{\partial z_i} = - \sum_i y_i \frac{1}{z_i}$

Table 1. Key equations to calculate the forward and backward step for a single layer feedforward NN.

Combining the results calculated in the forward step (see table 1) with the chain rule, we can calculate the gradient of the weights  $W_{kl}$ :

$$\frac{\partial \mathcal{L}}{\partial W_{kl}} = \frac{\partial \mathcal{L}}{\partial z_i} \frac{\partial z_i}{\partial h_i} \frac{\partial h_i}{\partial W_{kl}} = - \sum_i y_i \frac{1}{z_i} f(h_i)^2 e^{-h_i} x_l \delta_{ik} = -e^{-h_k} y_k z_k x_l \quad (65)$$

Note that the values  $z_k$  and  $h_k$  do not need to be calculated in the backward step since they are already stored in the memory. What we presented above is the simplest case of backpropagation, but the same technique works for neural networks with many consecutive layers and much more parameters as well. Frameworks such as TensorFlow or Pytorch usually do a lot of optimization steps, to make the calculation of the gradients faster and more stable numerically. Although the backpropagation algorithm was invented in the late 80s, the training of neural networks with backpropagation can be very computationally demanding even with today's powerful computers, because modern neural networks contain a huge amount of trainable parameters. To tackle this issue, researchers use special hardware designed to do these calculations more efficiently. GPUs were initially designed to accelerate graphical computations and image rendering, but are also useful to train neural networks more efficiently. Some companies manufacture tensor processing units (TPUs) which are specifically designed to accelerate matrix calculations for deep learning.

## 4 Quantum Machine Learning

One of the most promising branch of quantum computing is quantum machine learning, which means the application of quantum computing for traditional machine learning tasks i.e. to process classical data, or the application of classical machine learning methods to solve quantum-related problems like circuit optimization, error correction or quantum control. Beyond these, quantum machine learning includes scenarios where both the data and the algorithm are quantum for example using a quantum computer to validate quantum states [33].

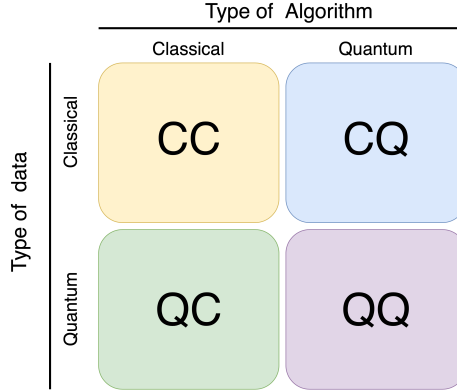


Figure 12. The four classes in which all machine learning tasks fit. The yellow box is actually containing all of the classical machine learning tasks, while the other three classes of algorithms somehow exploit quantumness.

After the discovery of the first quantum algorithms which have proven speedups compared to their classical counterparts, the search for possible realizations of quantum machine learning started. Many classical machine learning algorithms require matrix inversion, or the solution of large systems of linear equations. In 2009, Harrow, Hassidim and Lloyd found a quantum algorithm for solving systems of linear equations which has exponential speedup over classical algorithms [34]. A few years later Lloyd, Rebentrost and Mohseni designed the quantum versions of the principal component analysis [35] and the support vector machine [36], which also offer exponential speedups in some cases. Schuld and Killoran described the theoretical foundations of using the classically intractably large Hilbert space of quantum systems as a feature space by encoding classical data into quantum states, then using known kernel methods [37]. The list of quantized classical machine learning methods continues with Quantum Hopfield Networks [38], Quantum Boltzmann Machines [39], and continuous-variable quantum neural networks [40], which will be presented in more detail in chapter 4.2.

### 4.1 Variational Quantum Eigensolvers

The basic idea behind variational quantum circuits is as follows. First we fix the structure of a parametric quantum circuit called the ansatz, defined by a unitary  $\hat{U}(\theta)$ , where  $\theta \in \mathbb{R}^m$  are real parameters. Then we start with an initial state  $|\psi_0\rangle$ , which is usually  $|0\rangle^{\otimes M}$  in the qubit setting. For the continuous variable

setting, the initial state can be the vacuum state, a displaced state, a squeezed state, or a photon number eigenstate. After preparing the circuit, we evaluate the expectation value of some operator  $\hat{B}$  and then construct the loss as a function of  $\langle \hat{B} \rangle_\rho$ :

$$\mathcal{L}(\boldsymbol{\theta}) = f\left(\langle \hat{B} \rangle_\rho\right), \quad (66)$$

or if the state is pure, we can replace  $\langle \hat{B} \rangle_\rho$  with  $\langle \psi_0 | \hat{U}^\dagger(\boldsymbol{\theta}) \hat{B} \hat{U}(\boldsymbol{\theta}) | \psi_0 \rangle$ . The goal is to minimize the loss  $\mathcal{L}$  by iteratively updating the parameters  $\boldsymbol{\theta}$  with some update rule. This update rule can be any gradient-free method [41] or one can use gradient decent to update the weights. Variational quantum circuits are called hybrid quantum-classical algorithms, because the values of the parameters  $\boldsymbol{\theta}$  are stored in the memory of a classical computer and the classical CPU calculates the updated parameters. In this setting, the quantum device is an auxiliary device or accelerator, like a GPU in PCs. Figure 13 shows the workflow of a variational quantum circuit.

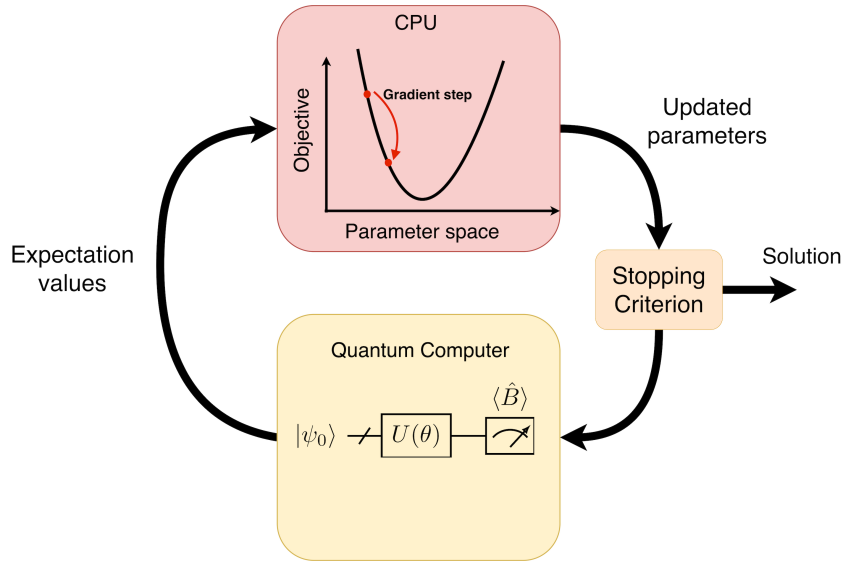


Figure 13. Optimization loop of a hybrid-quantum classical variational model.

Variational Quantum Eigensolvers [42] are a special class of variational quantum circuits, where the loss function is the Hamiltonian of some system:

$$\mathcal{L}(\boldsymbol{\theta}) = \langle \hat{H} \rangle_\rho. \quad (67)$$

VQEs are designed to find the lowest energy eigenvalue of some Hamiltonian  $\hat{H}$ , and are one of the most successful and most studied quantum algorithms because of their expected use cases in quantum chemistry and materials science [43, 44, 45, 46]. To use gradient-descent optimization in the training of variational quantum circuits, we need to evaluate the gradient of an expectation value,  $\nabla_{\boldsymbol{\theta}} \langle \hat{B} \rangle$ , which can be challenging. A method for obtaining such gradients called the *parameter-shift rule* is described in details in chapter 4.3. Algorithm 2 describes a finite-shot variational quantum eigensolver with a general gradient-based learning method. Notice that we wrote  $\alpha_j^{(t)}$  for the learning rate, since in most modern optimizers like Adam [29], each parameter has its own learning rate and it may depend on the number of iterations already done.

### 4.1.1 Stochastic and doubly stochastic GD

Variational quantum eigensolvers can be trained with stochastic gradient descent i.e. by using a finite number of circuit evaluations to approximate the expectation value  $\langle \hat{H} \rangle_\rho$ . As before, we denote a  $k$ -shot estimator of the expectation value of  $\hat{H}$  in quantum state  $\rho$  as  $\langle \hat{H} \rangle_{\rho,k}$ . By stochastic gradient descent, we mean replacing  $\partial_\mu \langle \hat{H} \rangle_\rho$  with  $\partial_\mu \langle \hat{H} \rangle_{\rho,k}$ . The  $k$ -shot VQE algorithm is presented below.

---

**Algorithm 2**  $k$ -shot VQE

---

```

1: procedure K-VQE( $H, k, T, \alpha$ )
2:   for all  $t \in \{1, \dots, T\}$  do
3:     Prepare the ansatz  $\hat{U}(\boldsymbol{\theta})$ 
4:     Estimate the loss  $\mathcal{L} = \langle \hat{H} \rangle_{\rho,k}$  by averaging  $k$  shots
5:     for all  $\theta_j \in \boldsymbol{\theta}$  do
6:       Calculate the quantum gradient  $\partial_{\theta_j} \mathcal{L}$ 
7:       Update the parameter  $\theta_j \leftarrow \theta_j - \alpha_j^{(t)} \partial_{\theta_j} \mathcal{L}$ 
8:     end for
9:   end for
10: end procedure

```

---

We can however define doubly stochastic gradient descent as in [47], which further reduces the total number of shots required to do a gradient step in some cases. The idea of doubly stochastic gradient descent for VQE is as follows. We consider a system with Hamiltonian

$$\hat{H} = \sum_{j=1}^M \hat{H}_j, \quad (68)$$

that is a system for which the Hamiltonian can be written as a sum of Hamiltonian terms. Then, we can decompose the expectation value of  $\hat{H}$  as

$$\langle \hat{H} \rangle_\rho = \sum_{j=1}^M \langle \hat{H}_j \rangle_\rho. \quad (69)$$

Applying the rules of differentiation to the above formula yields

$$\frac{\partial \langle \hat{H} \rangle_\rho}{\partial \theta_i} = \sum_{j=1}^M \frac{\partial \langle \hat{H}_j \rangle_\rho}{\partial \theta_i}, \quad (70)$$

which means that an estimator for the derivative of  $\langle \hat{H} \rangle_\rho$  with respect to the circuit parameters can be constructed as a linear combination of estimators for the derivatives of  $\langle \hat{H}_j \rangle_\rho$  [47]. The doubly stochastic algorithm is essentially the method of sampling mini-batches from the set of individual parts of the Hamiltonian  $\{\hat{H}_j\}_{j=1}^M$  and updating the circuit parameters based on the average of the derivatives calculated from these mini-batches (see algorithm 3).

**Algorithm 3** (k,n) doubly stochastic VQE

---

```
1: procedure KN-VQE( $H, k, n, T, \alpha$ )
2:   for all  $t \in \{1, \dots, T\}$  do
3:     Prepare the ansatz  $\hat{U}(\boldsymbol{\theta})$ 
4:     From  $\{\hat{H}_j\}_{j=1}^M$  sample a mini-batch  $\mathcal{B}$  of size  $n$ 
5:     for all  $\hat{H}_j$  in  $\mathcal{B}$  do
6:       Estimate the loss  $\langle \hat{H}_j \rangle_{\rho, k}$  by averaging  $k$  shots
7:     end for
8:     for all  $\theta_i \in \boldsymbol{\theta}$  do
9:       Calculate the quantum gradient  $\sum_{j=1}^{|\mathcal{B}|} \frac{\partial \langle \hat{H}_j \rangle_{\rho}}{\partial \theta_i}$ 
10:      Update the parameter  $\theta_i \leftarrow \theta_i - \alpha_i^{(t)} \sum_{j=1}^{|\mathcal{B}|} \frac{\partial \langle \hat{H}_j \rangle_{\rho}}{\partial \theta_i}$ 
11:    end for
12:  end for
13: end procedure
```

---

## 4.2 Quantum Neural Networks

In this chapter we present the basic ideas behind CV quantum neural networks, which we used in our numerical experiments. CV quantum neural networks are introduced by Killoran et al. [40], and we rely on their work in this chapter. We saw in chapter 3.2 that neural networks require linear operations and non-linear activation functions, and the composition of these transformations lead to an universal function approximator. In chapter 2.3 we saw that CV operations fall into two main categories: Gaussian and non-Gaussian operations, furthermore we saw that Gaussian operations correspond to affine transformations in the quantum phase-space, while non-Gaussian operators are non-linear transformations of the phase-space variables  $(\mathbf{x}, \mathbf{p})$ . Combining these two ideas, it is plausible to define the quantum analogue of the fully connected neural networks with CV quantum gates and quadrature basis measurements.

Just like in the classical case, where we are able to build a universal function approximator with linear transformations and nonlinear activations, we can build a universal unitary approximator with a finite set of CV quantum gates. A universal unitary approximator is a CV model built from CV gates which is able to approximate any unitary of the form

$$\hat{U} = e^{-it\hat{H}(\hat{\mathbf{X}}, \hat{\mathbf{P}})}, \quad (71)$$

where  $\hat{H}(\hat{\mathbf{X}}, \hat{\mathbf{P}})$  is a polynomial of the quadrature operators with arbitrary, but fixed degree [40]. It turns out that the smallest set of gates necessary to build an universal CV unitary approximator consists of all the Gaussian gates defined by equations (42), (44), (46) and (49) plus one single non-Gaussian gate like the Kerr-gate (51) [48, 40]. Therefore we will build quantum neural networks from this set of gates and we will show how to train such QNNs with gradient-descent method. We define a single  $N$ -mode

quantum layer similar to [40] as

$$L = K^{\otimes N}(\boldsymbol{\kappa})U_2(\boldsymbol{\theta}_2, \boldsymbol{\varphi}_2)S^{\otimes N}(\mathbf{z})U_1(\boldsymbol{\theta}_1, \boldsymbol{\varphi}_1)D^{\otimes N}(\boldsymbol{\alpha}), \quad (72)$$

where  $(\boldsymbol{\kappa}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\theta}_1, \boldsymbol{\varphi}_1, \boldsymbol{\theta}_2, \boldsymbol{\varphi}_2)$  are the trainable parameters of the layer (see Fig. 14). We use this architecture (with a few consecutive identical layers) both for the classification and the regression task in chapter 5. After the sequence of layers applied to the initial state, we can apply different homodyne measurements, depending on the task.

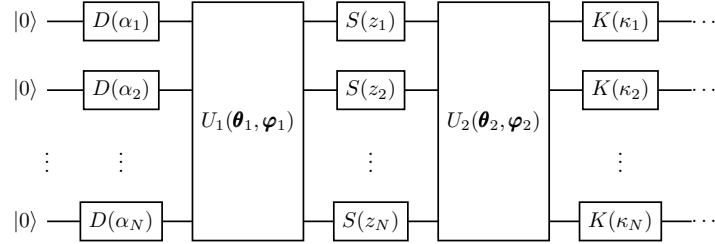


Figure 14. A single quantum layer of a general  $N$ -mode fully connected QNN.  $U_1$  and  $U_2$  are general  $N$ -port interferometers, while  $D, S, K$  are single-mode displacement, squeezing and Kerr operators, respectively.

In chapter 5, where we implement different quantum circuits for different tasks, we will specify the measurements more precisely, but all measurements are performed in the quadrature basis. The structure of the quantum layer defined with equation (72) is very similar to the classical perceptron model. If we denote by  $\boldsymbol{\xi} = (\mathbf{x}, \mathbf{p})$ , the phase-space variables, and recall that a Gaussian gate corresponds to a symplectic transformation plus a phase-space translation (see Eq. (40)), then we can write

$$L(\boldsymbol{\xi}) = \Phi(\boldsymbol{\xi} + \boldsymbol{\alpha}), \quad \Phi = K_{\boldsymbol{\kappa}}^{\otimes N}, \quad (73)$$

which very much resembles Eq. (62). This means that if we encode classical data into the quantum phase space variables, then we can apply non-linear operations on them, without breaking the unitary evolution of the quantum state. With this in mind, we could build more complex quantum neural networks, like quantum residual networks, quantum convolutional networks and so on [40]. Finally we need to mention data encoding, which is a key component of quantum machine learning. Throughout this thesis, we use displacement encoding, which means that a classical feature vector  $\mathbf{x}$  is mapped into a quantum state by the mapping  $\mathbf{x} \mapsto D(\mathbf{x})|0\rangle$ .

### 4.3 Calculating quantum gradients

To use the gradient descent method for training quantum neural network, we obviously need to calculate the gradients  $\partial_{\mu}\mathcal{L}$ ,  $\mu \in \boldsymbol{\theta}$ , however, using automatic differentiation presented in chapter 3.2.1 on quantum circuits is not possible, because we do not have direct access to the intermediate states, only the measurement statistics. Of course, one could argue that the gradients  $\partial_{\mu}\mathcal{L}$  can be approximated

numerically by black-box evaluations of the quantum circuit using the finite difference method,

$$\partial_\mu \mathcal{L} \approx \frac{\mathcal{L}(\mu + \Delta\mu/2) - \mathcal{L}(\mu - \Delta\mu/2)}{\Delta\mu}, \quad (74)$$

where  $\Delta\mu$  is sufficiently small. However, the devices are noisy and therefore, measuring small differences would require intractably many measurement shots under some circumstances [49]. In this chapter, we introduce the theory of calculating quantum gradients of the circuit parameters with repeated circuit evaluations using a method called the *parameter-shift rule*. The parameter-shift rule allows us to calculate analytic gradients of the quantum gates by shifting their parameters by a finite, non-infinitesimal value  $s$ . An open-source software package called PennyLane [50] implements the parameter-shift rule for both the continuous-variable as well as the qubit setting. Here we restrict ourselves to the discussion of the continuous-variable case. In this chapter, we follow the notation of Schuld et al. [49], and we refer to this paper for a more rigorous description of the ideas presented in our work.

We consider a variational quantum circuit defined by a unitary  $\hat{U}(\boldsymbol{\theta})$ ,  $\boldsymbol{\theta} \in \mathbb{R}^m$ , with an initial state  $|\psi_0\rangle$  and an observable  $\hat{B}$ . We can view this variational circuit as a mapping  $f : \mathbb{R}^m \rightarrow \mathbb{R}$ , which maps the circuit parameters to the expectation value of operator  $\hat{B}$ :

$$f(\boldsymbol{\theta}) = \langle \psi_0 | \hat{U}^\dagger(\boldsymbol{\theta}) \hat{B} \hat{U}(\boldsymbol{\theta}) | \psi_0 \rangle \quad (75)$$

In gradient-based optimizations, we need the gradient  $\nabla_{\boldsymbol{\theta}} f$ , and this eventually means calculating the partial derivatives  $\partial_\mu f$ ,  $\mu \in \boldsymbol{\theta}$ .

### 4.3.1 Heisenberg-picture representation

In the continuous variable setting, observables can be expressed as polynomials of the canonical quadrature operators  $\hat{X}_j$  and  $\hat{P}_j$ . We introduce the *Heisenberg-picture* of Gaussian gates as in [49]. The Heisenberg-picture relies on defining an infinite-length vector  $\hat{\mathbf{C}}$  that contains all quadrature monomials:

$$\hat{\mathbf{C}} = (\mathbb{1}, \hat{X}_1, \hat{P}_1, \hat{X}_2, \hat{P}_2, \dots, \hat{X}_n, \hat{P}_n, \hat{X}_1^2, \hat{X}_1 \hat{P}_1, \dots). \quad (76)$$

If we consider the conjugation of operator  $\hat{C}_j$  by gate  $\hat{\mathcal{G}}$ , we get a linear transformation which we denote with  $\Omega^{\mathcal{G}}$ :

$$\Omega^{\mathcal{G}}[\hat{C}_j] \stackrel{\text{def}}{=} \hat{\mathcal{G}}^\dagger \hat{C}_j \hat{\mathcal{G}} = \sum_i M_{ij}^{\mathcal{G}} \hat{C}_i, \quad (77)$$

where  $M_{ij}^{\mathcal{G}}$  are real matrices depending on the gate parameters. Subsequent conjugations are also linear transformations, therefore they correspond to multiplying matrices  $M^{\mathcal{G}}$ . For example if we conjugate  $\hat{C}_j$  with gate  $\hat{V}$  and then with gate  $\hat{U}$ , the result is

$$\Omega^U[\Omega^V[\hat{C}_j]] = \Omega^U[\hat{V}^\dagger \hat{C}_j \hat{V}] = \sum_{i,k} M_{ik}^U M_{kj}^V \hat{C}_i. \quad (78)$$



Since conjugation by a Gaussian gate, does not increase the degree of quadrature polynomials, we can define a vector of length  $2n + 1$ ,

$$\hat{\mathbf{D}} = (\mathbb{1}, \hat{X}_1, \hat{P}_1, \dots, \hat{X}_n, \hat{P}_n), \quad (79)$$

for an  $n$ -mode system, since the Gaussian operators will map the subspace spanned by  $\hat{\mathbf{D}}$  into itself:

$$\Omega^{\mathcal{G}}[\hat{D}_j] = \sum_{i=0}^{2n} M_{ij}^{\mathcal{G}} \hat{D}_i. \quad (80)$$

The Heisenberg-picture is useful, since it allows us to represent any  $n$ -mode Gaussian operator  $\mathcal{G}$  with a  $(2n + 1) \times (2n + 1)$  matrix  $M^{\mathcal{G}}$ , furthermore we can calculate the gradients  $\partial_{\mu} \hat{\mathcal{G}}$  by calculating the gradients of the matrices,  $\partial_{\mu} M^{\mathcal{G}}$ .

For example, the single-mode phaseless squeezing gate  $\hat{S}(r, \phi = 0)$  can be represented in Heisenberg-picture, as

$$M^S(r) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & e^{-r} & 0 \\ 0 & 0 & e^r \end{pmatrix}, \quad (81)$$

and therefore its derivative  $\partial_r M^S(r)$  is

$$\partial_r M^S(r) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -e^{-r} & 0 \\ 0 & 0 & e^r \end{pmatrix}. \quad (82)$$

The matrix  $\partial_r M^S(r)$  does not correspond to the Heisenberg-representation to any single Squeezing gate, however it can be decomposed as a linear combination of two Squeezing gates with different parameters [49]:

$$\partial_r M^S(r) = \frac{1}{2 \sinh(s)} [M^S(r+s) - M^S(r-s)], \quad (83)$$

where  $s$  is a fixed non-zero real number. Therefore,

$$\partial_r [\hat{S}^\dagger(r) \hat{D}_j \hat{S}(r)] = \frac{1}{2 \sinh(s)} [\hat{S}^\dagger(r+s) \hat{D}_j \hat{S}(r+s) - \hat{S}^\dagger(r-s) \hat{D}_j \hat{S}(r-s)], \quad j \in \{0, 1, 2\}. \quad (84)$$

### 4.3.2 Gradients in CV quantum circuits

If we want to calculate the partial derivative  $\partial_{\mu} \hat{U}(\boldsymbol{\theta})$ , where  $\hat{U}(\boldsymbol{\theta})$  is a parametric gate sequence containing Gaussian and non-Gaussian quantum gates. We split this gate sequence into three parts as  $\hat{U}(\boldsymbol{\theta}) = \hat{V} \hat{\mathcal{G}}(\mu) \hat{W}$  (see Fig. 15). Furthermore let us assume that the observable  $\hat{B}$  can be expressed in terms of first-degree quadrature monomials as  $\hat{B} = \sum_i b_i \hat{D}_i$ . Then,

$$f(\boldsymbol{\theta}) = \langle 0 | \hat{U}^\dagger(\boldsymbol{\theta}) \hat{B} \hat{U}(\boldsymbol{\theta}) | 0 \rangle = \langle 0 | \hat{W}^\dagger \hat{\mathcal{G}}^\dagger(\mu) \hat{V}^\dagger \hat{B} \hat{V} \hat{\mathcal{G}}(\mu) \hat{W} | 0 \rangle \quad (85)$$

$$= \sum_i b_i \langle 0 | \hat{W}^\dagger \hat{\mathcal{G}}^\dagger(\mu) \hat{V}^\dagger \hat{D}_i \hat{V} \hat{\mathcal{G}}(\mu) \hat{W} | 0 \rangle. \quad (86)$$

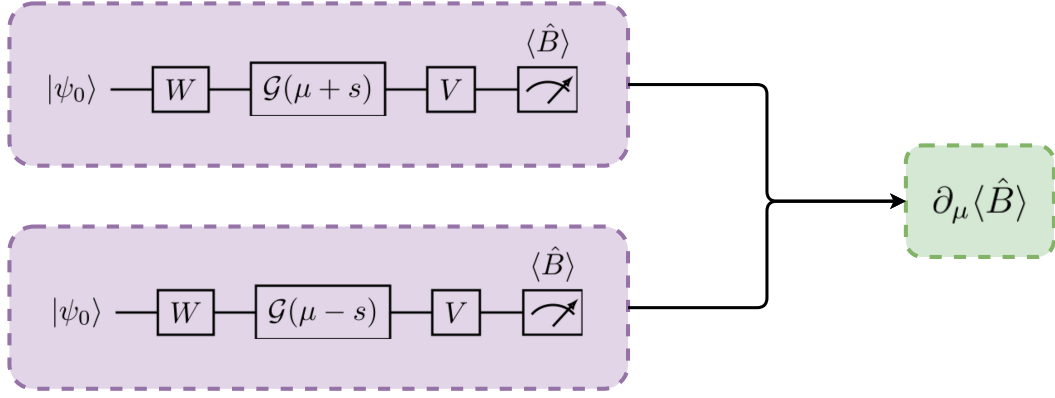


Figure 15. Visualization of the parameter-shift rule for a quantum circuit.

Assuming that  $\hat{V}$  and  $\mathcal{G}$  is Gaussian and using equations (77), (78) and (80), we can write

$$f(\boldsymbol{\theta}) = \sum_{i,j,k} \langle 0 | \hat{W}_k \hat{D}_k \hat{W} | 0 \rangle M_{kj}^{\mathcal{G}}(\mu) M_{ji}^V b_i. \quad (87)$$

In possession of this formula, we can calculate the derivative  $\partial_\mu f$  with the help of the derivative of  $M_{kj}^{\mathcal{G}}(\mu)$ :

$$\partial_\mu f(\boldsymbol{\theta}) = \sum_{i,j,k} \langle 0 | \hat{W}_k \hat{D}_k \hat{W} | 0 \rangle (\partial_\mu M^{\mathcal{G}}(\mu))_{kj} M_{ji}^V b_i. \quad (88)$$

If the derivative of the matrix  $\partial_\mu M^{\mathcal{G}}(\mu)$  can be expressed as a linear combination of shifted matrices, that is,

$$\partial_\mu M^{\mathcal{G}}(\mu) = \sum_{\ell} \gamma_{\ell} M^{\mathcal{G}}(\mu + s_{\ell}), \quad (89)$$

we can simply write the derivative as

$$\partial_\mu f(\boldsymbol{\theta}) = \sum_{i,j,k,\ell} \langle 0 | \hat{W}_k \hat{D}_k \hat{W} | 0 \rangle \gamma_{\ell} M_{kj}^{\mathcal{G}}(\mu + s_{\ell}) M_{ji}^V b_i. \quad (90)$$

The above formula is the parameter-shift rule for CV quantum circuits. Note that we assumed that the subcircuit  $\hat{V}$  is Gaussian, i.e., there are only Gaussian gates between  $\mathcal{G}$  and measurement, but the subcircuit  $\hat{W}$  does not need to be Gaussian in this case. In table 2 we summarize the Heisenberg-representations of the most common Gaussian gates, and in table 3 we present the parameter-shift rules for these gates.

The parameter shift rule can be applied in cases where the observable  $\hat{B}$  is a higher order polynomial of quadrature operators, since these can be reduced to products of first-order operators:

$$\Omega^{\mathcal{G}}[\hat{D}_i \hat{D}_j] = \hat{\mathcal{G}}^{\dagger} \hat{D}_i \hat{D}_j \hat{\mathcal{G}} = \hat{\mathcal{G}}^{\dagger} \hat{D}_i \hat{\mathcal{G}} \hat{\mathcal{G}}^{\dagger} \hat{D}_j \hat{\mathcal{G}} = \Omega^{\mathcal{G}}[\hat{D}_i] \Omega^{\mathcal{G}}[\hat{D}_j]. \quad (91)$$

However, when the operator  $\hat{\mathcal{G}}$  is non-Gaussian, or the subcircuit  $\hat{V}$  contains non-Gaussian gates, we can no longer use the first-degree subspace spanned by operators  $\hat{\mathbf{D}}$ , instead of that, we need a larger subspace of  $\hat{\mathbf{C}}$ . For example we may need to include the second-order monomials as well, which will make the matrices  $M^{\mathcal{G}}$  and  $M^V$  larger, introducing extra computational overhead.

Gate $\mathcal{G}$	Heisenber-representation $M^{\mathcal{G}}$
$R(\phi)$	$M^R(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{pmatrix}$
$D(r, \phi)$	$M^D(r, \phi) = \begin{pmatrix} 1 & 0 & 0 \\ 2r \cos \phi & 1 & 0 \\ 2r \sin \phi & 0 & 1 \end{pmatrix}$
$S(r)$	$M^S(r) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & e^{-r} & 0 \\ 0 & 0 & e^r \end{pmatrix}$
$BS(\theta, \varphi)$	$M^{BS}(\theta, \varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \cos \theta & 0 & -\alpha & -\beta & \\ 0 & 0 & \cos \theta & \beta & -\alpha \\ 0 & \alpha & -\beta & \cos \theta & 0 \\ 0 & \beta & \alpha & 0 & \cos \theta \end{pmatrix}$

Table 2. Heisenberg representation and partial derivatives of the most common Gaussian gates. In the case of beam-splitter, we introduced  $\alpha = \cos \varphi \sin \theta, \beta = \sin \varphi \sin \theta$ . We included the phase-free squeezing,  $S(r)$  only, because the phase-dependent squeezing can be decomposed as  $S(r, \phi) = R(\frac{\phi}{2})S(r)R(-\frac{\phi}{2})$ .

Heisenberg-representation	Parameter-shift rule
$M^R(\phi)$	$\partial_\phi M^R(\phi) = \frac{1}{2} [M^R(\phi + \frac{\pi}{2}) - M^R(\phi - \frac{\pi}{2})]$
$M^D(r, \phi)$	$\partial_r M^D(r, \phi) = \frac{1}{2s} [M^D(r + s, \phi) - M^D(r - s, \phi)]$ $\partial_\phi M^D(r, \phi) = \frac{1}{2} [M^D(r, \phi + \frac{\pi}{2}) - M^D(r, \phi - \frac{\pi}{2})]$
$M^S(r)$	$\partial_r M^S(r) = \frac{1}{2 \sinh(s)} [M^S(r + s) - M^S(r - s)]$
$M^{BS}(\theta, \varphi)$	$\partial_\theta M^{BS}(\theta, \varphi) = \frac{1}{2} [M^{BS}(\theta + \frac{\pi}{2}, \varphi) - M^{BS}(\theta - \frac{\pi}{2}, \varphi)]$ $\partial_\varphi M^{BS}(\theta, \varphi) = \frac{1}{2} [M^{BS}(\theta, \varphi + \frac{\pi}{2}) - M^{BS}(\theta, \varphi - \frac{\pi}{2})]$

Table 3. Parameter-shift rules for the most common Gaussian gates.

## 5 Results

In this chapter, we present the results of our numerical experiments. In chapter 5.2 we present the results obtained by applying a CV QNN to some regression tasks, in chapter 5.1 we discuss the performance of the QNN-s on various classification examples. Finally, in chapter 5.3 we present the performance of a continuous-variable VQE on finding the ground-state energy of a Bose-Hubbard model. These numerical experiments were run using Strawberry Fields [18, 51], an open-source software for simulating photonic quantum circuits and TensorFlow [31], Google’s open-source framework for calculating gradients via automatic differentiation. There exists a framework called PennyLane [50], which calculates quantum gradients with the parameter-shift rule, however this introduces a  $2n$  computational overhead in case of  $n$  quantum gates. This  $2n$  overhead would cost a lot of time, therefore we used TensorFlow to calculate gradients of the parameters with the help of automatic differentiation, although this would not be possible when using a physical hardware. Although these software packages are very powerful and efficient for computing matrix algebra and simulating quantum circuits, due to the exponential scaling of the Hilbert-space of CV quantum systems, we can not efficiently simulate quantum circuits with more than a few qumodes. As we saw in chapter 2, the dimension of the Hilbert-space of a CV system is infinite, and because of this, quantum simulation software like Strawberry Fields [18] use a truncated Hilbert space which has  $d^N$  dimensions, where  $d$  is a *cutoff dimension* and  $N$  is the number of modes. We use cutoff dimension  $d = 10$  for the regression and classification tasks where  $N = 2$  and  $N = 3$  respectively, and  $d = 4, N = 4$  for the Bose-Hubbard model.

### 5.1 Classification

In this chapter we present the architecture we used to solve different classification tasks using quantum neural networks and of course we present the results as well. We tested the same ansatz circuit with the same loss function on three different synthetic datasets: blobs, moons and circles (see Fig. 18 a-c). The datasets were generated with scikit-learn [52], a python package which contains a lot of implementations of data preprocessing and machine learning algorithms. We used an ansatz circuit architecture inspired by [40], composed of beamsplitters, displacement and squeezing gates and Kerr nonlinearities. Figures 16 and 17 shows the ansatz circuits we used with their first layers.

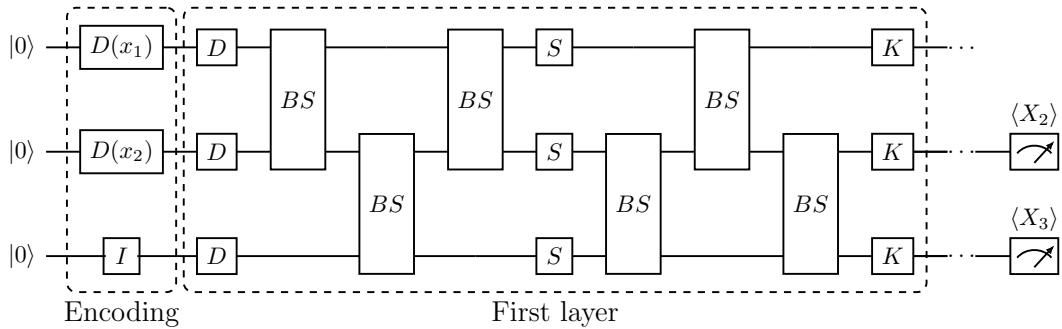


Figure 16. Ansatz circuit used in the classification task for the moons and circles datasets.

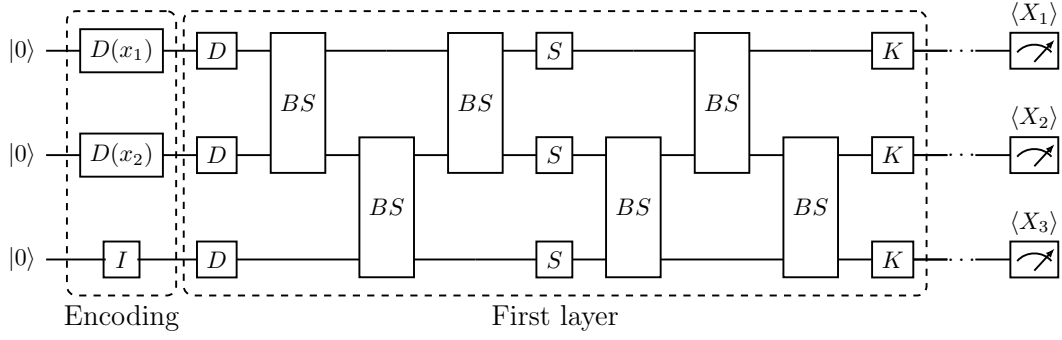


Figure 17. Ansatz circuit used in the classification task for the blobs dataset.

In all three datasets we used, the feature vectors are two-dimensional spacial coordinates  $\mathbf{x} = (x_1, x_2)$ , and we applied displacement encoding to map these vectors into quantum states. We used a three-mode quantum neural network, and therefore the encoding function is

$$(x_1, x_2) \mapsto \hat{D}_1(x_1) \otimes \hat{D}_2(x_2) \otimes I_3 |0, 0, 0\rangle, \quad (92)$$

and we can write  $\hat{U}(\mathbf{x}; \boldsymbol{\theta})$  as a product:

$$\hat{U}(\mathbf{x}; \boldsymbol{\theta}) = \hat{U}'(\boldsymbol{\theta}) \left( \hat{D}_1(x_1) \otimes \hat{D}_2(x_2) \otimes I_3 \right). \quad (93)$$

Assuming that the output states of the QNN defined above are pure, we can write

$$\rho = |\psi\rangle\langle\psi| \text{ with } |\psi\rangle = \hat{U}(\mathbf{x}; \boldsymbol{\theta}) |\psi_0\rangle. \quad (94)$$

In most cases this assumption is true in our simulations, because we use  $L^2$  normalization on the active weights of the network, and thus  $\text{Tr}[\rho^2] \approx 1$ . To construct the loss function, we used homodyne measurements with **Softmax** and **CrossEntropy**. Depending on the number of classes in the dataset we measured modes  $\{1, 2, 3\}$  in the case of blobs or  $\{2, 3\}$  in the case of moons and circles. The loss is simply

$$\mathcal{L} = \text{CrossEntropy}(\mathbf{y}^{\text{pred}}, \mathbf{y}^{\text{true}}) + \frac{\lambda}{|\mathcal{B}|} \sum \|W^{\text{active}}\|^2, \quad (95)$$

where

$$\mathbf{y}^{\text{pred}} = \text{Softmax} \begin{bmatrix} \text{abs}(\langle \hat{X}_1 \rangle_{\rho, k}) \\ \text{abs}(\langle \hat{X}_2 \rangle_{\rho, k}) \\ \text{abs}(\langle \hat{X}_3 \rangle_{\rho, k}) \end{bmatrix} \text{ or } \mathbf{y}^{\text{pred}} = \text{Softmax} \begin{bmatrix} \text{abs}(\langle \hat{X}_2 \rangle_{\rho, k}) \\ \text{abs}(\langle \hat{X}_3 \rangle_{\rho, k}) \end{bmatrix}, \quad (96)$$

furthermore  $W^{\text{active}}$  are the weights of the active gates in the QNN and  $\mathcal{B}$  is a mini-batch. In the above equation  $\langle \hat{B} \rangle_{\rho, k}$  are the  $k$ -shot estimates of the expectation value  $\langle \hat{B} \rangle$  for state  $\rho$ .

We used stochastic and doubly stochastic gradient descent methods (see chapter 4.2) to train these QNN classifiers. We also run the simulations with exact expectation values, this case corresponds to the  $k \rightarrow \infty$  limit, and these simulations were as a baseline for comparison against the stochastic results.

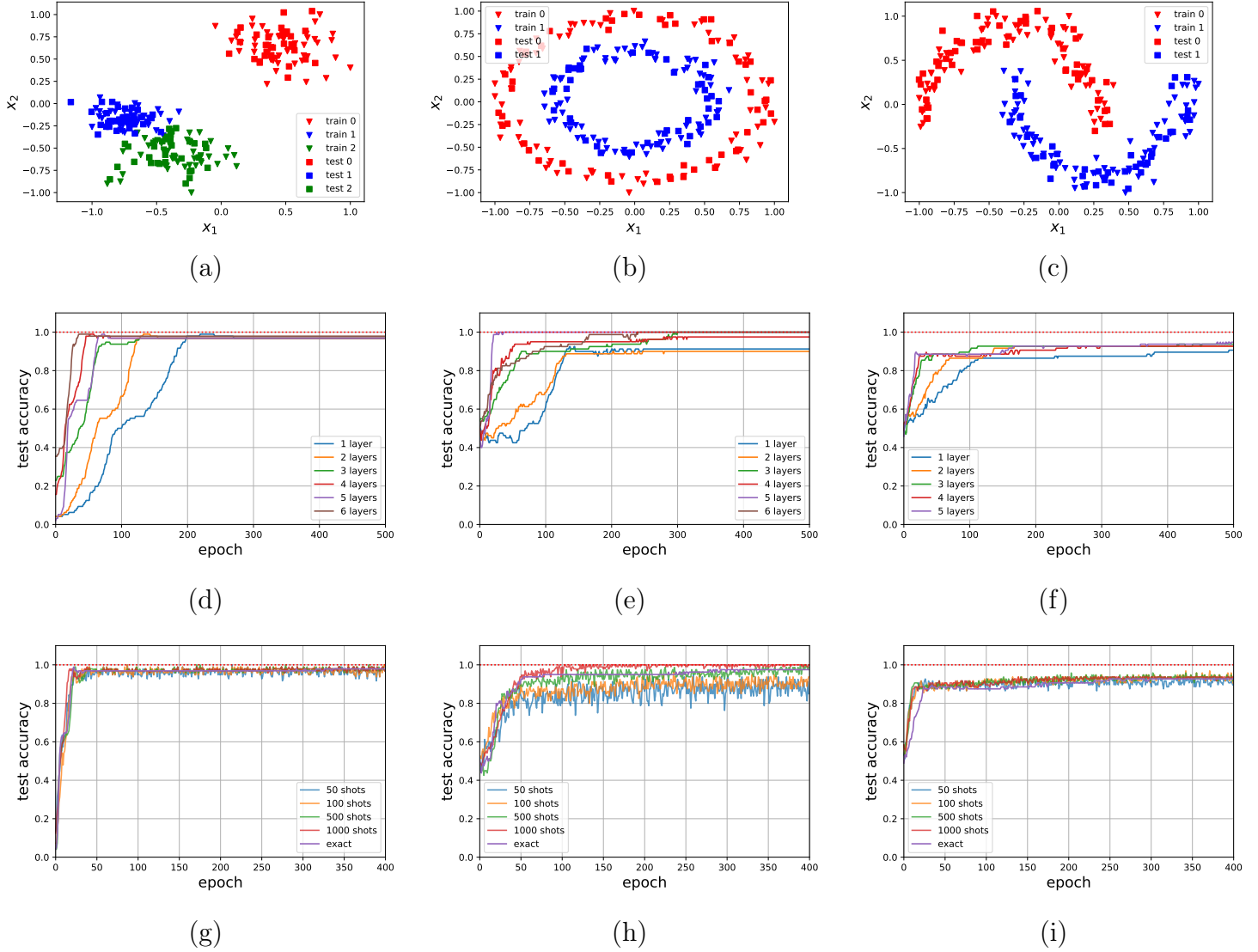


Figure 18. Results on the classification tasks on three datasets: blobs (a), circles (b), moons (c). The middle row shows the test set accuracy for each dataset with different number of layers. The bottom row shows comparisons between different SGD results with  $k \in \{50, 100, 500, 1000\}$  shots and the exact baseline.

The results of these simulations are shown on figures 18 (d) to (i), and table 4 contains the hyper-parameters of the experiments. First, we tested the performance of the QNN architecture with different number of layers and different learning rates. As seen from the results, the easiest of the three datasets is the blobs dataset, hence a QNN with only three layers is perfectly able to fit the data even with only  $k = 50$  shots. The circles dataset is a bit more difficult requiring at least 4 layers for a good convergence, and also requires more shots. However, a QNN with 4 layers trained with at least 500 shots was able to achieve accuracy  $\approx 98\%$  within 400 training epochs. The hardest of the three is the moons dataset, for which even a 5-layer QNN was not able to perform better than 90% on the test set. We found that QNNs with 4 and 5 layers perform almost identically on this dataset and therefore we chose the 4-layer architecture to run SGD test. Increasing the width of the QNN might give better results, but the size of the Hilbert-space with  $N = 4$  qumodes and cutoff dimension  $d = 10$  is  $10^4$ , which would lead to

intractably slow calculations on our computers.

	blobs	circles	moons
#training data	144	144	144
#test data	96	96	96
#layers	3	4	4
#qumodes	3	3	3
#shots	50 to 1000	50 to 1000	50 to 1000
$\lambda$	0.1	0.1	0.1
lr	0.005	0.01	0.01

Table 4. Hyperparameters of the classification experiments. batch=összes

## 5.2 Regression

As specified earlier, the aim of regression is to predict or forecast a value  $y$  for previously unseen feature vectors  $\mathbf{x}$  by inferring a function  $f$  from given example pairs  $(\mathbf{x}_k, y_k)$ . In this chapter, we present our results on testing regression with parametric quantum circuits on a photonic CV architecture. We test our methods on synthetic datasets, as well as on a real-world dataset. In the simulations presented hereby, both the feature and the dependent value are scalars. Data normalization is usually a very important step in order to achieve good performance in machine learning. Therefore, we normalized our datasets so that most of the  $x$  and  $y$  values fall into the  $[-1, 1]$  interval. In chapter 4.2 we saw that encoding classical data into a quantum state is a crucial step in quantum machine learning. Therefore we tested different encodings and found that displacement encoding works the best in each of the simulations. Hence, in all of the simulations presented in this thesis we use displacement encoding, which means that the feature  $x$  is encoded into the quantum system by applying a single-mode Weyl-displacement operator  $\hat{D}(x)$  on one or more modes. In the regression task, we apply the same displacement on all of the modes:

$$x \mapsto \hat{D}_1(x) \otimes \hat{D}_2(x) |0, 0\rangle. \quad (97)$$

This way the unitary matrix in equation (99) can be written as

$$\hat{U}(x; \boldsymbol{\theta}) = \hat{U}'(\boldsymbol{\theta}) \left( \hat{D}_1(x) \otimes \hat{D}_2(x) \right), \quad (98)$$

where  $\hat{U}'(\boldsymbol{\theta})$  does not depend on the value of  $x$ . After data encoding, we apply a number of identical quantum layers composed of beamsplitters, rotations, displacements, squeezing gates and Kerr nonlinearities. These gates are arranged in a structure similar to that proposed in [40]. Another important step to consider is weight normalization. Since passive photonic gates do not change the overall energy of the system, we do not apply any kind of normalization to the parameters of the passive gates. Active gates however like displacement and squeezing change the energy of the system (or photon number), and thus they can push the state into a mixed state [40], which we want to avoid as much as possible. Therefore we apply an  $L^2$ -regularization to the active weights  $W^{\text{active}}$  of the network. Assuming that

the output state is pure, we can write

$$\rho = |\psi\rangle\langle\psi|, \text{ with } |\psi\rangle = \hat{U}(x; \boldsymbol{\theta}) |\psi_0\rangle, \quad (99)$$

and we can define the loss function as

$$\mathcal{L} = \frac{1}{|\mathcal{B}|} \sum_{(x,y) \in \mathcal{B}} \left( y - \langle \hat{X}_2 \rangle_{\rho,k} \right)^2 + \frac{\lambda}{|\mathcal{B}|} \sum \|W^{\text{active}}\|^2, \quad (100)$$

where  $\mathcal{B}$  is a mini-batch of size  $|\mathcal{B}|$  and  $\rho$  is the output of the quantum neural network. Some regression tasks work better if we use  $L^1$  regularization, or even both  $L^1$  and  $L^2$ , but we found that  $L^2$  regularization works perfectly fine in our cases.

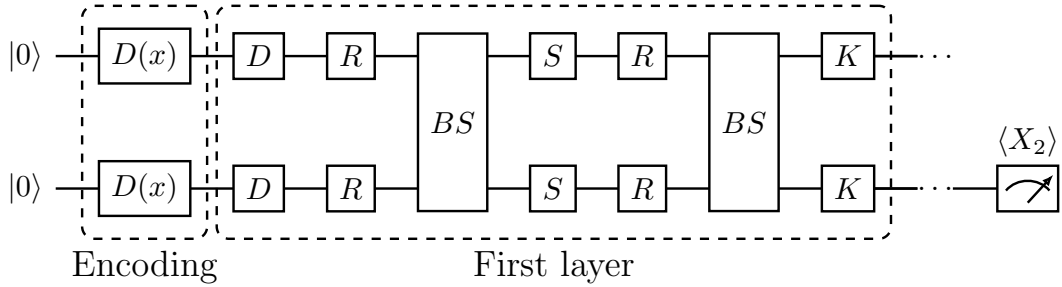


Figure 19. Ansatz circuit for the regression task. We use displacement encoding and six consecutive identical layers with a homodyne measurement at the end.

We tested the quantum neural network described above on two three datasets: two of them were synthetic datasets and one of them was a real-world dataset. The synthetic datasets were a sample from the set  $\{(x_i, y_i) \mid y_i = \tanh(\pi x_i), x_i \in [-1, 1]\}$ , and another sample from  $\{(x_i, y_i) \mid y_i = x_i^2, x_i \in [0, 1]\}$ , while the real-world dataset was created from an I-V curve measurement of a photodiode. We split these datasets into a larger training set and a smaller testing set, so that the training sets contained approximately twice as much data points as the testing sets (see table 5).

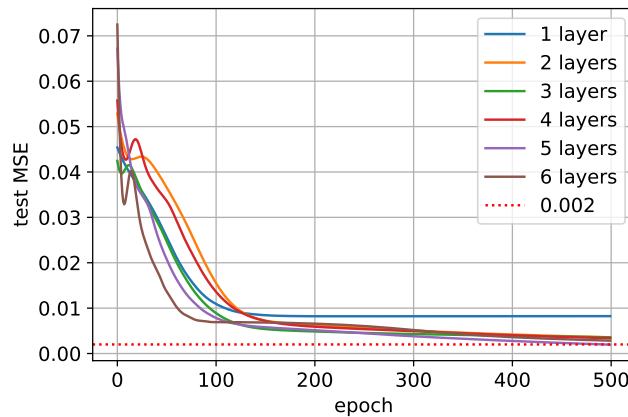


Figure 20. Comparison of the regression circuit performance on the test set with different number of layers.



We tested the performance of the QNN with different number of layers (see Fig. 20), and concluded that we need at least six layers to achieve the target MSE of 0.002 on the test set. The quantum neural network we use in the regression task is shown on Fig. 19.

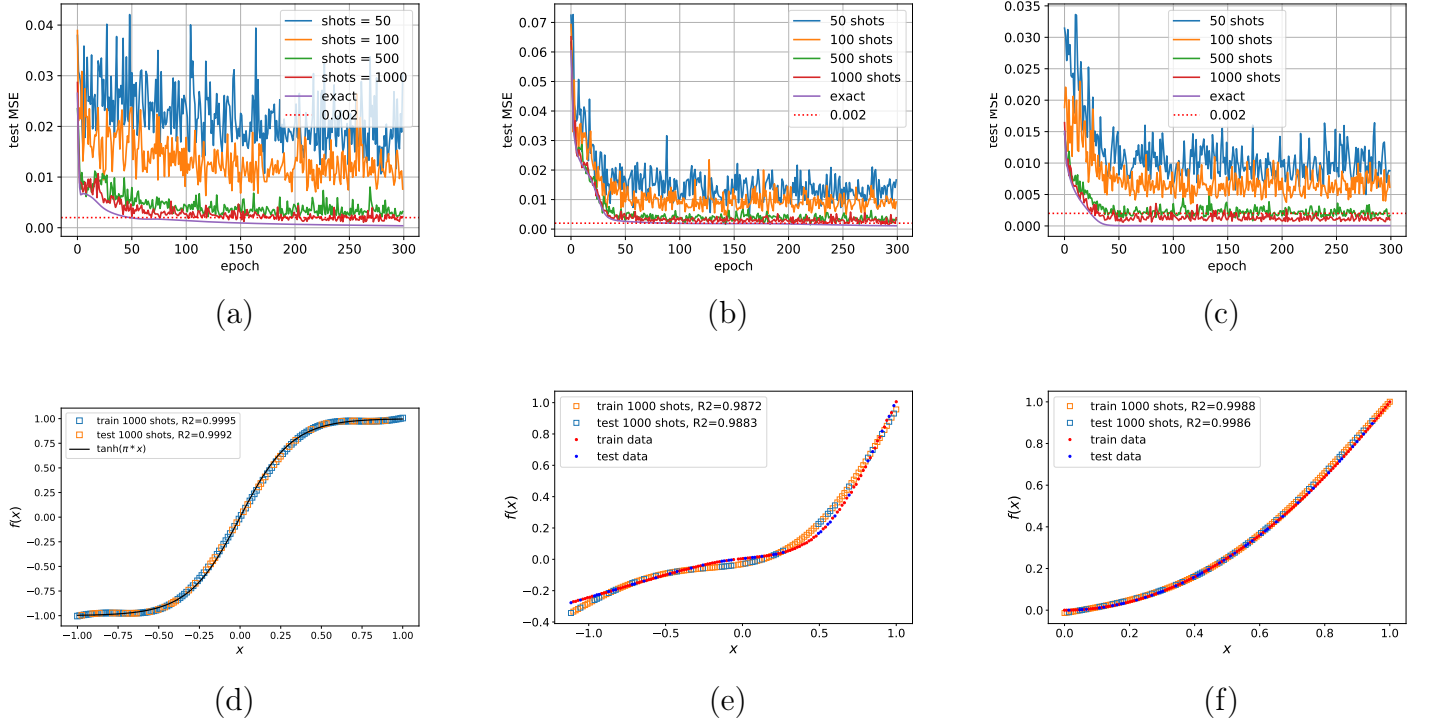


Figure 21. The top row shows how the regression circuits perform on the test set. The figures show loss curves with different number of shots for the tanh function (a), the I-V curve (b) and the  $x^2$  function (c). The bottom rows show inference tests with 1000 shots both for the tanh function (d), the I-V curve (d) and finally the  $x^2$  function (e). All simulations were run with  $\alpha = 0.001$  and mini-batch size 4.

We tested the performance of the proposed QNN on the three datasets by running mini-batch SGD with exact expectation value as a baseline, similarly to what we did in the case of classification presented in the previous chapter. Besides that we also tested doubly stochastic gradient descent with  $k$ -shot estimates of the expectation value  $\langle X_2 \rangle$ , however we found that for the regression task, we need more shots to achieve good performance. In fact, we needed at least 500 shots to achieve the target MSE of 0.002. Figures 21 (a) to (c) show the MSE part of the loss curves during the training with different number of shots. The final results inferred with the best set of parameters. In all experiments the QNNs were able to achieve an  $R^2$ -score larger than 0.98 in the worst case, which means that these models have a good expressive power even with very few trainable parameters. The details of the hyperparameters used in our experiments are summarized in table 5.

	$\tanh(\pi x)$	I-V	$x^2$
#training data	105	100	105
#test data	45	38	45
#layers	6	6	6
#qumodes	2	2	2
#shots	50 to 1000	50 to 1000	50 to 1000
mini-batch size	4	4	4
$\lambda$	0.1	0.1	0.1
lr	0.001	0.001	0.001

Table 5. Hyperparameters of the regression experiments

### 5.3 CV-VQE for the Bose-Hubbard model

The Bose-Hubbard model introduced first by Gersch and Knollman in 1963 for the description of interacting spinless bosons on a lattice, such as ultracold bosonic atoms on an optical lattice [53, 54]. This model has also gained popularity because it is found to be an adequate model for the description of the Superfluidity-Mott insulator (SF-MI) transition, which has been experimentally demonstrated [55, 56]. We chose this model, because an exact diagonalization of the Bose-Hubbard Hamiltonian with respect to a finite dimensional Fock-basis is possible and thus the model is numerically solvable. This enables the verification of the correctness of the results given by our variational quantum solver for small problem instances.

The Bose Hubbard model which we used in our simulations is defined by the following Hamiltonian expressed with the bosonic creation and annihilation operators:

$$\hat{H} = -t \sum_{i=1}^{M-1} (\hat{a}_i^\dagger \hat{a}_{i+1} + \hat{a}_{i+1}^\dagger \hat{a}_i) + \frac{U}{2} \sum_{i=1}^M \hat{n}_i^2, \quad (101)$$

where  $M$  is the number of bosonic modes in the system. If we restrict the total number of bosons in the system to be  $N$ , the dimension of the restricted Fock-space is

$$D = \frac{(N + M - 1)!}{N!(M - 1)!}, \quad (102)$$

which explosively grows with the system size. Therefore finding the groundstate of the Bose-Hubbard Hamiltonian for larger systems can be intractable even on classical supercomputers, making quantum computers, especially continuous-variable quantum computers a potential candidate for solving these problems in the future. Compared to the original definition of the Bose-Hubbard Hamiltonian, we omitted the term  $-\mu \sum_i \hat{n}_i$ , because it is not relevant in our particular experiments.

### 5.3.1 Solving the model by exact diagonalization

If we want to calculate the groundstate of a Bose-Hubbard model, then we need to solve the eigenvalue-problem of Hamiltonian (101). Of course a general Fock-space corresponding to a bosonic system is infinite-dimensional, therefore we have to restrict the system so that a maximum of  $N$  particles are allowed in each mode:

$$|0, 0, \dots, 0\rangle, |0, 0, \dots, N\rangle, |0, 0, \dots, 1, N\rangle, \dots |N, N, \dots, N\rangle.$$

Then, we can relabel the Fock-states so that each Fock basis has its own unique label  $j$ :

$$|n_{M-1}, n_{M-2}, \dots, n_1, n_0\rangle = \left| \sum_{k=0}^{M-1} n_k (N+1)^k \right\rangle. \quad (103)$$

This is useful, because with these relabeled states we can calculate the matrix elements of the new Hamiltonian:

$$H_{jj'} = \langle j | \hat{H} | j' \rangle = \langle n_{M-1}, n_{M-2}, \dots, n_1, n_0 | \hat{H} | n'_{M-1}, n'_{M-2}, \dots, n'_1, n'_0 \rangle \quad (104)$$

We can further reduce the dimension of the Hilbert space if we prescribe that the total number of photons must be  $N$ :

$$\sum_k n_k = N. \quad (105)$$

In this case for example if the number of modes is  $M = 4$  and the total number of particles is  $N = 4$ , the dimension of the restricted Hilbert-space is only 35. The resulting matrix  $H_{jj'}$  is usually a sparse matrix (see fig. 22 (a)), and can be diagonalized using existing software. Figure 22 (b). shows the solutions of a Bose-Hubbard model by exact diagonalization for different  $U$  and  $t$  values, with  $M = 4$  and  $N = 4$ .

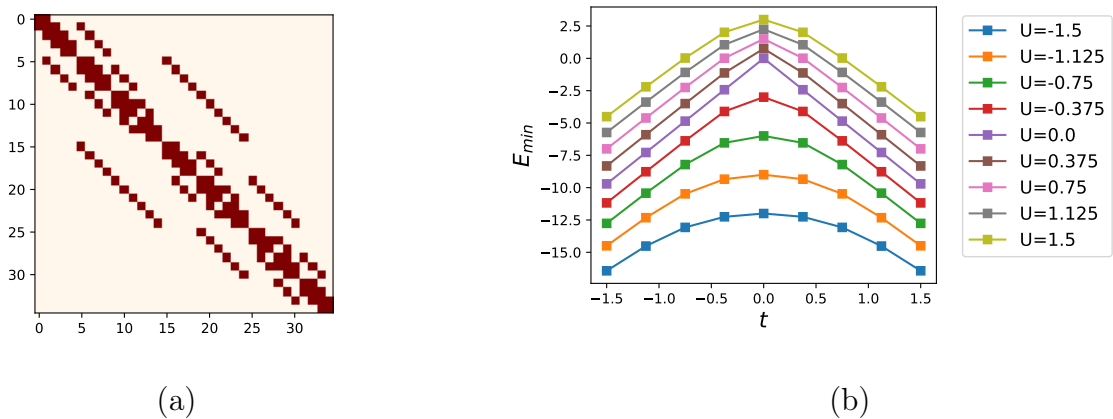


Figure 22. (a) Visual representation of a sparse Hamiltonian matrix  $H_{jj'}$  for a Bose-Hubbard model with  $\dim(\mathcal{H}) = 35$ . Darker points show the non-zero elements in the matrix (b) Numerical solutions of the ground-state energy for different  $U$  and  $t$  values.

### 5.3.2 Variational solutions

VQEs introduced in chapter 4.1 are shown to be effective for simulating different fermionic quantum systems, like molecules or metals [43, 44, 45, 46]. In this chapter we demonstrate a photonic VQE for finding the ground-state energy of a Bose-Hubbard model. Furthermore, we show that stochastic gradient descent methods can be applied to the VQE to reduce the number of shots required, and thus reduce the overall cost of finding the ground state energy on a quantum hardware.

The variational circuit  $\hat{U}(\theta)$  can be realized with a sequence of CV quantum gates. Since we want the total number of particles in the system to be preserved during the simulations, we must use photon-number preserving gates only. We composed each quantum layer from Beamsplitters, Kerr-gates and Cross-Kerr-gates (see fig. 23). Although the set of CV quantum gates we use in this case do not form a universal gate set as we saw in chapter 4.2, we will see that the ansatz architecture shown on Fig. 23 is capable of finding the ground-state of a Bose-Hubbard model. Contrary to the classification and regression tasks, where we set the initial state of the system to  $|\psi_0\rangle = |0\rangle^{\otimes M}$ , here we put a single photon into each mode i.e  $|\psi_0\rangle = |1\rangle^{\otimes M}$ , because we want to find the lowest energy  $M$ -photon state.

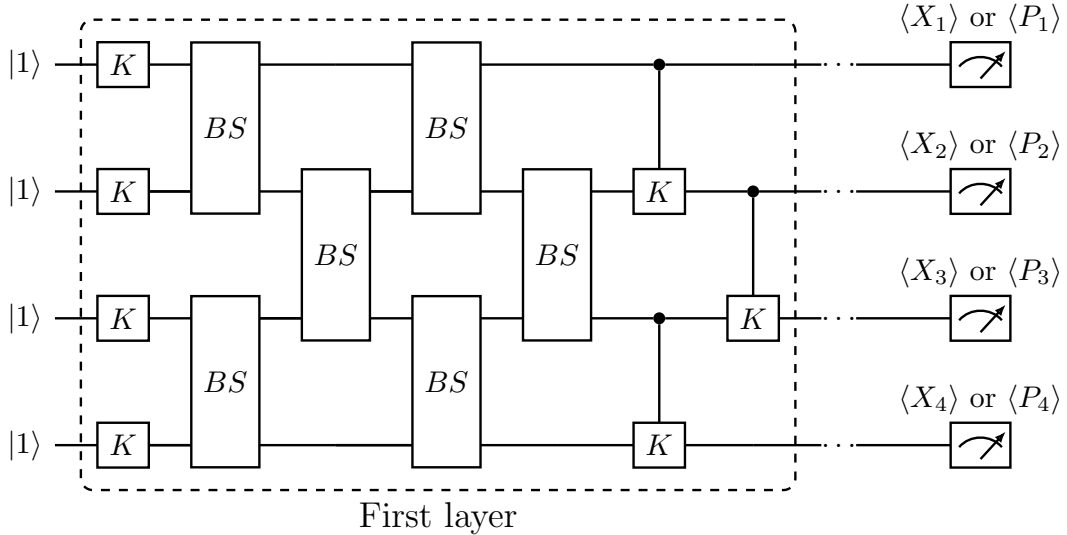


Figure 23. Illustration of a variational ansatz circuit with its first layer. Initially, each of the four qumodes contain a single photon. The layer is built up from Kerr-gates, CrossKerr gates and Beamsplitters, which are passive optical gates, so the total number of photons is preserved.

Similarly to what we did in the case of classification and regression, we were searching for the lowest number of layers required to find the ground state (see Fig. 24), and found that we need at least five layers with exact expectation values. However, when using stochastic or doubly stochastic gradient descent, the five layers proved too few, and therefore we run these experiments using six-layer ansatz circuits.

As we see on Fig. 23, we apply quadrature basis measurements at the end of the ansatz circuit, and this is because quadrature measurements are less challenging experimentally compared to the photon-number resolving measurements. However, the Bose-Hubbard Hamiltonian defined with Eq. (101) is

in the Fock-basis, and we need to transform this Hamiltonian into the quadrature-basis. Using the Jordan-Wigner transformation introduced in Eq. (6), it is straightforward to rewrite the Bose-Hubbard Hamiltonian in the quadrature basis as:

$$\begin{aligned} \hat{H} = & -\frac{t}{2\hbar} \sum_{j=1}^{M-1} \left[ \hat{X}_j \hat{X}_{j+1} + \hat{X}_{j+1} \hat{X}_j + \hat{P}_j \hat{P}_{j+1} + \hat{P}_{j+1} \hat{P}_j \right] \\ & + \frac{U}{8\hbar^2} \sum_{j=1}^M \left[ (\hat{X}_j^2 + \hat{P}_j^2)^2 - 2\hbar(\hat{X}_j^2 + \hat{P}_j^2) \right] + \frac{UM}{8}, \end{aligned} \quad (106)$$

which can be measured with a number of homodyne measurements. Despite the measurements being homodyne, we could not use the Gaussian backend of Strawberry Fields, because the ansatz circuit contains non-Gaussian gates, and therefore, we needed to run all the simulations on the Fock-state simulator.

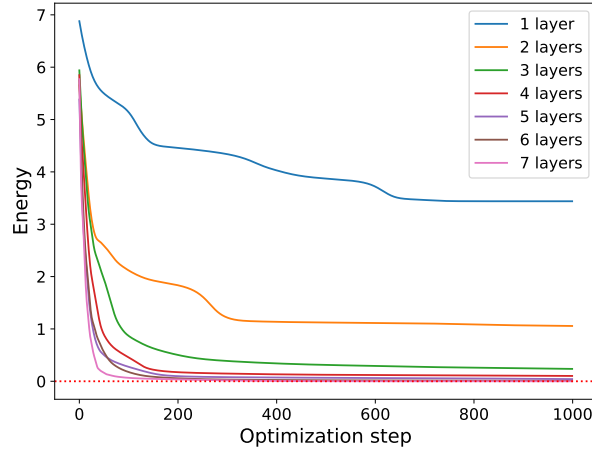
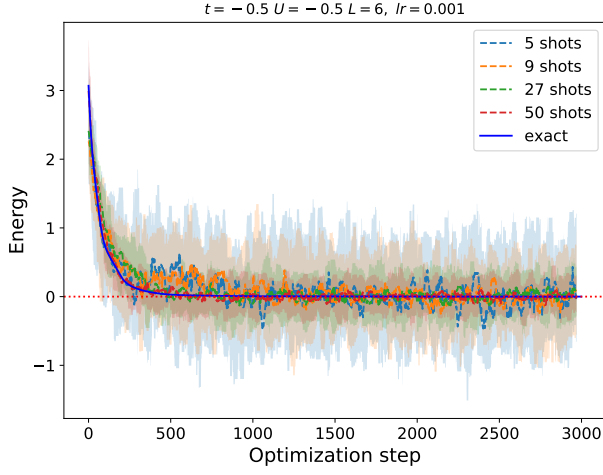
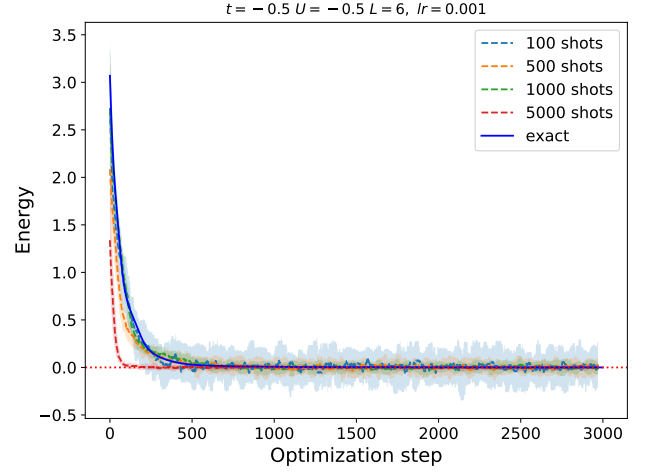


Figure 24. Comparison of the performance of the Bose-Hubbard VQE with different number of layers in the ansatz circuit.

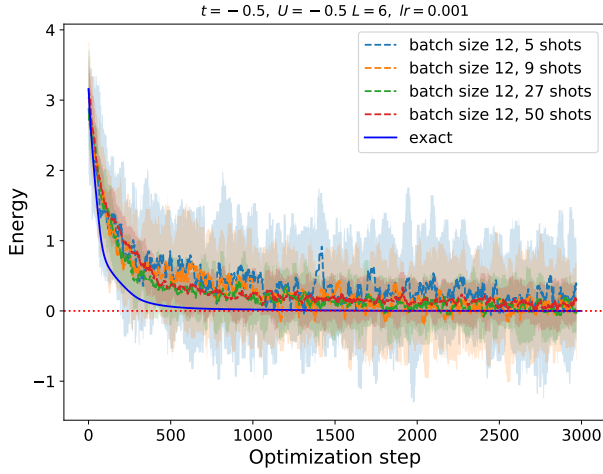
We tested the VQE proposed above with a wide variety of shots ranging between 5 and 5000, and we also tested the doubly stochastic gradient descent method with batch size 12. Figure 25 (a) and (b) show the VQE solutions of the Bose Hubbard model with parameters  $t = -0.5$ ,  $U = -0.5$ , using few-shot measurements (a) and measurements with higher number of shots (b). As we expected, the SGD with very few shots is noisy, but the fluctuations decrease quickly as we increase the number of shots. We can also see on figures (c) and (d) that measuring only a half of the Hamiltonian in each gradient step still gives acceptable convergence, however in this case we need to wait more epochs compared to the simple SGD. We found that by using the simple stochastic GD, we need at least 750 optimization steps in order to find the ground-state of the system, while more than 2500 steps are required to get close to the ground-state when measuring only the half of the Hamiltonian. Fig. 25 (c) and (d) show that when using doubly stochastic GD, the optimization did not reach the exact ground-state, but converges to a value close to it. This issue might be solved by using more layers, or even higher number of shots which might be investigated in our future work.



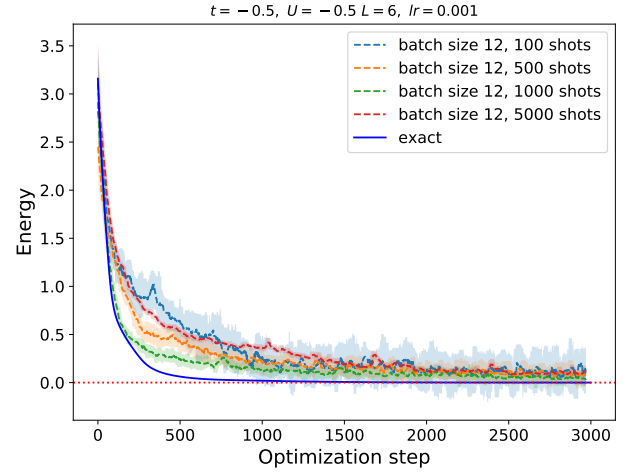
(a)



(b)



(c)



(d)

Figure 25. Results of the Bose-Hubbard VQE. We see the simple stochastic results in the top row with shot numbers ranging between 5 and 5000. The bottom shows our results on the same system, when using the doubly stochastic gradient descent. In these figures, the blue continuous lines represent the  $k \rightarrow \infty$  limit, while the dashed lines are the running medians of the stochastic results with pale regions showing the 5-95 percentile regions of the running window. We used window size of 30 steps in these plots.

## 6 Conclusion and future work

In this thesis, we presented the basic mathematical background of continuous- variable quantum computation. Furthermore, we described the ideas behind deep neural network and the gradient-descent method of machine learning. We saw that deep neural networks with great learning capabilities can be constructed from basic building blocks, and can be trained in a supervised manner, when sufficient amount of data is available. We also presented classification and regression as the two main problem classes targeted by neural networks, and defined common loss functions and performance metrics.

After reviewing how classical neural networks work, we proceeded to defining the quantum analogue of classical neural networks in the CV quantum computing paradigm. We saw that a small set of Gaussian quantum gates, together with a non-Gaussian gate can be used to construct a universal unitary approximator, which is the quantum analogue for the universal function approximator in classical machine learning. We also presented methods for calculating gradients of parametric quantum circuits, which is essential to training quantum neural networks.

Then we saw three different examples of applications of quantum neural networks: a QNN trained to solve three regression tasks, another QNN trained to classify spacial data points, and finally a variational quantum eigensolver to find the ground state of a Bose-Hubbard model. We showed that we can use a finite number of shots to approximate expectation values, without losing accuracy, however the minimum number of shots required to achieve a target performance varies from task to task, and a general answer to this question is yet to be investigated. We also saw that we can implement doubly stochastic gradient descent, i.e., using a finite number of shots together with mini-batches, to further reduce the overall cost of training quantum circuits. However, we saw that the doubly stochastic method is suboptimal for the VQE, since if we measure only the half of the Hamiltonian, we need considerably more training epochs to achieve the same performance.

In the future, we plan to apply quantum methods to reinforcement learning, which is a very interesting topic in classical AI research. Out of the three classes of classical machine learning (SL, UL and RL), reinforcement learning is the most difficult, and we expect that there will be plenty of room for quantum speedups in this field.

# A Mathematical preliminaries

## A.1 Hilbert spaces

### Definition 3. *Hilbert-space*

Given a field  $T$  (real or complex), a vector space  $\mathcal{H}$  endowed with an inner product, is called a Hilbert-space, if it is a complete metric space with respect to the distance function induced by the inner product. The inner product is a map  $\langle \cdot | \cdot \rangle : \mathcal{H} \times \mathcal{H} \rightarrow T$ , for which  $\forall x, y, z \in \mathcal{H}$ :

- $\langle x | x \rangle \geq 0$
- $\langle x | x \rangle = 0 \iff x = \mathbf{0} \in \mathcal{H}$
- $\langle x | y \rangle = \langle y | x \rangle^*$ , where  $*$  denotes complex conjugation.
- $\langle x | \alpha y + \beta z \rangle = \alpha \langle x | y \rangle + \beta \langle x | z \rangle$ , where  $\alpha, \beta \in T$

The norm induced by this inner product is a map  $\| \cdot \| : \mathcal{H} \rightarrow T$  defined as

$$\|x\| = \sqrt{\langle x | x \rangle},$$

And the metric induced by this norm is defined as

$$d(x, y) = \|x - y\| = \sqrt{\langle x - y | x - y \rangle}.$$

The space  $\mathcal{H}$  is said to be complete if every Cauchy-sequence is convergent with respect to the norm, and the limit is in  $\mathcal{H}$ . That is, each sequence  $x_1, x_2, \dots$ , for which

$$\forall \varepsilon > 0 \exists N(\varepsilon) \text{ so, that } n > m > N(\varepsilon) \implies \|x_n - x_m\| < \varepsilon.$$

### Definition 4. *Linear functional*

Let  $\mathcal{H}$  be a Hilbert-space over the field  $T$ . Then, the map  $\varphi : \mathcal{H} \rightarrow T$  is called a linear functional, if

$$\varphi(\alpha x + \beta y) = \alpha \varphi(x) + \beta \varphi(y), \quad \forall \alpha, \beta \in T, x, y \in \mathcal{H}.$$

### Definition 5. *Dual space*

Given a Hilbert-space  $\mathcal{H}$ , its dual space,  $\mathcal{H}^*$  is the space of all continuous linear functionals from the space  $\mathcal{H}$  into the base field. The norm of an element in  $\mathcal{H}^*$  is

$$\|\varphi\|_{\mathcal{H}^*} \stackrel{\text{def}}{=} \sup_{\|x\|=1, x \in \mathcal{H}} |\varphi(x)|.$$

### Theorem 1. *Riesz representation theorem*



For every element  $y \in \mathcal{H}$ , there exists a unique element  $\varphi_y \in \mathcal{H}^*$ , defined by

$$\varphi_y(x) = \langle y|x \rangle, \quad \forall x \in \mathcal{H}.$$

The mapping  $y \mapsto \varphi_y$  is an antilinear mapping, i.e.,  $\alpha y_1 + \beta y_2 \mapsto \alpha^* \varphi_{y_1} + \beta^* \varphi_{y_2}$ , and the Riesz-representation theorem states that this mapping is an antilinear isomorphism. The inner product in  $\mathcal{H}^*$  satisfies

$$\langle \varphi_x | \varphi_y \rangle = \langle x | y \rangle^* = \langle y | x \rangle.$$

Moreover,  $\|y\|_{\mathcal{H}} = \|\varphi_y\|_{\mathcal{H}^*}$ .

**Definition 6. Dirac-notation**

From now on, the elements in  $\mathcal{H}$  will be denoted by  $|x\rangle$  and their corresponding element in  $\mathcal{H}^*$  as  $\langle x|$ .

## A.2 Linear operators on Hilbert spaces

**Definition 7. Linear operators**

A map  $\hat{A} : \mathcal{H}_1 \rightarrow \mathcal{H}_2$  is a linear operator, if

$$\hat{A}(\alpha |x\rangle + \beta |y\rangle) = \alpha(\hat{A}|x\rangle) + \beta(\hat{A}|y\rangle).$$

**Remark 1.** If not stated otherwise, we will assume that  $\mathcal{H}_1 = \mathcal{H}_2 = \mathcal{H}$ .

**Remark 2.** Operators will be denoted with a hat ( $\hat{\cdot}$ ).

**Definition 8. Bounded linear operators**

A linear operator  $\hat{A} : \mathcal{H} \rightarrow \mathcal{H}$  is bounded, if

$$\exists m \in \mathbb{R} : |\langle v | \hat{A} | v \rangle| \leq m \langle v | v \rangle, \quad \forall |v\rangle \in \mathcal{H}$$

**Remark 3.** The set of all bounded operators on  $\mathcal{H}$  is denoted  $\mathcal{B}(\mathcal{H})$ .

**Definition 9. Commutators and anticommutators**

Since operators usually do not commute, its useful to define their commutator and anticommutator:

$$\begin{aligned} [\hat{A}, \hat{B}] &= \hat{A}\hat{B} - \hat{B}\hat{A} \\ \{\hat{A}, \hat{B}\} &= \hat{A}\hat{B} + \hat{B}\hat{A} \end{aligned}$$

**Definition 10. Operator norm**

The operator norm of an operator  $\hat{A}$  is defined as

$$\|\hat{A}\| \stackrel{\text{def}}{=} \inf\{c \geq 0 : \|\hat{A}|v\rangle\| \leq c\| |v\rangle \|, \quad \forall |v\rangle \in \mathcal{H}\}$$

**Definition 11. Trace-class operators**

An operator  $\hat{A}$  is called trace-class if it admits a well defined and finite trace  $\text{Tr} [\hat{A}] = \sum_j \langle j | \hat{A} | j \rangle$

**Definition 12. Positive operators**

An operator  $\hat{A}$  is called positive if  $\langle v | \hat{A} | v \rangle \geq 0, \forall |v\rangle \in \mathcal{H}$ . If  $\hat{A} = \sum_j \lambda_j |j\rangle \langle j|$  then  $\hat{A}$  is positive if  $\lambda_j \geq 0$ .

**Definition 13. Projections** An operator  $\Pi : \mathcal{H} \rightarrow \mathcal{H}$  is a projection if  $\Pi^2 = \Pi$ .

### A.3 Hermitian Operators, Unitary Operators, Spectral theorem, Hadamard-lemma

**Definition 14. Hermitian adjoint**

Consider a **bounded** linear operator  $\hat{A} : \mathcal{H} \rightarrow \mathcal{H}$ . The hermitian adjoint of  $\hat{A}$  is a bounded linear operator  $\hat{A}^\dagger : \mathcal{H} \rightarrow \mathcal{H}$  which satisfies

$$\langle y | \hat{A} | x \rangle = \left( \langle x | \hat{A}^\dagger | y \rangle \right)^*, \quad \forall |x\rangle, |y\rangle \in \mathcal{H}. \quad (107)$$

**Definition 15. Hermitian operators**

A bounded linear operator  $\hat{H} : \mathcal{H} \rightarrow \mathcal{H}$  is Hermitian if

$$\hat{H} = \hat{H}^\dagger, \text{ i.e., } \hat{H} |x\rangle = \hat{H}^\dagger |x\rangle, \quad \forall |x\rangle \in \mathcal{H}. \quad (108)$$

**Definition 16. Unitary operator**

A bounded linear operator  $\hat{U} : \mathcal{H} \rightarrow \mathcal{H}$  is unitary if

$$\hat{U}\hat{U}^\dagger = \hat{U}^\dagger\hat{U} = 1, \text{ in other words, } \hat{U}^\dagger = \hat{U}^{-1}. \quad (109)$$

**Definition 17. Eigenvalues and eigenvectors**

Consider bounded linear operator  $\hat{A}$ . If exist a vectors  $|k\rangle \in \mathcal{H}$  such that

$$\hat{A} |k\rangle = \lambda_k |k\rangle, \quad (110)$$

then  $|k\rangle$  is called an eigenvector of  $\hat{A}$  and  $\lambda_k$  is the corresponding eigenvalue.

An important property of Hermitian operators is that they can be diagonalized with real eigenvalues. This is formally stated by the spectral theorem:

**Theorem 2. The Spectral theorem**

Let  $\hat{A}$  be a bounded Hermitian operator on some Hilbert-space  $\mathcal{H}$ . Then there exists an orthonormal basis in  $\mathcal{H}$  which consists of the eigenvectors of  $\hat{A}$  and each eigenvalue of  $\hat{A}$  is real.

This means that any bounded Hermitian operator  $\hat{H}$  can be decomposed as

$$\hat{H} = \sum_k \lambda_k \hat{P}_k = \sum_k \lambda_k |k\rangle \langle k| \quad (111)$$

where  $\lambda_k$  and  $|k\rangle$  are the eigenvalues and eigenvectors of  $\hat{H}$ .

**Definition 18. Exponential of operators** If  $X$  is a linear operator, we can define the exponential of  $X$ :

$$e^X = \sum_{n=0}^{\infty} \frac{X^n}{n!}$$

**Important:** The product of exponentials of operators generally isn't equal to the exponential of their sum:

$$e^X e^Y = e^{Z(X,Y)} \neq e^{X+Y},$$

where  $Z(X, Y)$  is given by the Baker-Campbell-Hausdorff formula:

$$\begin{aligned} Z(X, Y) = & X + Y + \frac{1}{2}[X, Y] + \frac{1}{12}[X, [X, Y]] - \frac{1}{12}[Y, [X, Y]] - \frac{1}{24}[Y, [X, [X, Y]]] \\ & - \frac{1}{720}([[[[X, Y], Y], Y], Y] + [[[Y, X], X], X], X) + \dots \end{aligned}$$

It is however equal if  $[X, Y] = 0$ :

$$\text{if } [X, Y] = 0 \implies e^X e^Y = e^{X+Y}$$

There are 2 important special cases:

**Theorem 3. The Hadamard-lemma**

$$e^X Y e^{-X} = Y + [X, Y] + \frac{1}{2!}[X, [X, Y]] + \frac{1}{3!}[X, [X, [X, Y]]] + \dots$$

**Theorem 4.** If  $X$  and  $Y$  commute with their commutator, i.e.,  $[X, [X, Y]] = [Y, [X, Y]] = 0$ , then:

$$e^X e^Y = e^{X+Y+\frac{1}{2}[X,Y]}$$

**Theorem 5.** If  $[X, Y] = sY$  with  $s \in \mathbb{C}, s \neq 2i\pi n, n \in \mathbb{Z}$  then:

$$e^X e^Y = \exp\left(X + \frac{s}{1 - e^{-s}} Y\right)$$

## A.4 Pure and mixed quantum states

**Definition 19. Quantum states**

A quantum state of a quantum system is a mathematical entity that provides a probability distribution for the outcomes of each possible measurement on the system.

**Definition 20. Pure quantum states**

Pure quantum states are quantum states that can be described by a vector  $|\psi\rangle$  of norm 1.

If one multiplies a pure quantum state by a complex scalar  $e^{i\alpha}$ , then the new state is physically equivalent to the former, thus  $|\psi\rangle$  and  $e^{i\alpha} |\psi\rangle$  are the same pure state. The transformation  $|\psi\rangle \rightarrow e^{i\alpha} |\psi\rangle$

does not change the outcomes of measurements on the state, however the phase  $\alpha$  is important in quantum algorithms.

**Example.** For example, the states  $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi} |1\rangle)$  and  $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\frac{\pi}{2}} |1\rangle)$  are not the same quantum state, but in both states there is 50-50 percent probability of measuring  $|0\rangle$  and  $|1\rangle$ .

**Definition 21. Density Matrix**

A quantum state  $\hat{\rho}$  is a trace-1, self-adjoint, positive semidefinite operator. The set of quantum states is

$$\mathcal{S}(\mathcal{H}) = \{\hat{\rho} : \hat{\rho} \geq 0, \hat{\rho} = \hat{\rho}^\dagger, \text{Tr} [\hat{\rho}] = 1\}$$

A quantum state is pure if and only if  $\hat{\rho}^2 = \hat{\rho}$ . Also, if  $\rho$  is a pure state, then it can be written as  $\hat{\rho} = |\psi\rangle\langle\psi|$ . The operator  $\rho$  is called the density operator or density matrix. It is often useful to measure the purity of a state as  $\mu_\rho = \text{Tr} [\rho^2]$ .

## B Coherent states

A coherent state is the eigenstate of  $\hat{a}$ :

$$\hat{a} |\alpha\rangle = \alpha |\alpha\rangle$$

$$|\alpha\rangle = \sum_{n=0}^{\infty} |n\rangle \langle n|\alpha\rangle$$

we have

$$\hat{a} |\alpha\rangle = \alpha |\alpha\rangle \implies \langle n|\hat{a}|\alpha\rangle = \alpha \langle n|\alpha\rangle$$

and

$$\begin{aligned} \hat{a}^\dagger |n\rangle &= \sqrt{n+1} |n+1\rangle \implies \langle n|\hat{a} = \sqrt{n+1} \langle n+1| \\ \implies \langle n|\hat{a}|\alpha\rangle &= \sqrt{n+1} \langle n+1|\alpha\rangle = \alpha \langle n|\alpha\rangle \\ \implies \langle n|\alpha\rangle &= \frac{\alpha}{\sqrt{n}} \langle n-1|\alpha\rangle = \dots = \frac{\alpha^n}{\sqrt{n!}} \langle 0|\alpha\rangle \end{aligned}$$

therefore,

$$|\alpha\rangle = \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} \langle 0|\alpha\rangle |n\rangle.$$

The value of  $\langle 0|\alpha\rangle$  can be calculated from the normalization condition  $\langle\alpha|\alpha\rangle \stackrel{!}{=} 1$ . Since

$$|\alpha\rangle = \langle 0|\alpha\rangle \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle$$

$$\langle\alpha| = \langle\alpha|0\rangle \sum_{k=0}^{\infty} \frac{(\alpha^*)^k}{\sqrt{k!}} \langle k|,$$

we have

$$\begin{aligned} 1 \stackrel{!}{=} \langle\alpha|\alpha\rangle &= \left( \langle\alpha|0\rangle \sum_{k=0}^{\infty} \frac{(\alpha^*)^k}{\sqrt{k!}} \langle k| \right) \left( \langle 0|\alpha\rangle \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle \right) \\ &= \langle\alpha|0\rangle \langle 0|\alpha\rangle \sum_{k=0}^{\infty} \sum_{n=0}^{\infty} \frac{(\alpha^*)^k \alpha^n}{\sqrt{k!n!}} \langle k|n\rangle \\ &= \langle\alpha|0\rangle \langle 0|\alpha\rangle \sum_{k=0}^{\infty} \sum_{n=0}^{\infty} \frac{(\alpha^*)^k \alpha^n}{\sqrt{k!n!}} \delta_{kn} \\ &= |\langle 0|\alpha\rangle|^2 \sum_{n=0}^{\infty} \frac{(|\alpha|^2)^n}{n!} = |\langle 0|\alpha\rangle|^2 e^{|\alpha|^2}. \\ \implies \langle 0|\alpha\rangle &= e^{i\varphi} e^{-\frac{|\alpha|^2}{2}} \end{aligned}$$

And so, a coherent state can be written as

$$|\alpha\rangle = e^{i\varphi} e^{-\frac{|\alpha|^2}{2}} \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle.$$

## References

- [1] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, June 1982. doi: 10.1007/bf02650179. URL <https://doi.org/10.1007/bf02650179>.
- [2] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907): 553–558, December 1992. doi: 10.1098/rspa.1992.0167. URL <https://doi.org/10.1098/rspa.1992.0167>.
- [3] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press. doi: 10.1109/sfcs.1994.365700. URL <https://doi.org/10.1109/sfcs.1994.365700>.
- [4] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC 1996*. ACM Press, 1996. doi: 10.1145/237814.237866. URL <https://doi.org/10.1145/237814.237866>.

- [5] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. ACM, June 2019. doi: 10.1145/3313276.3316310. URL <https://doi.org/10.1145/3313276.3316310>.
- [6] Chen Ding, Tian-Yi Bao, and He-Liang Huang. Quantum-Inspired Support Vector Machine. *arXiv e-prints*, art. arXiv:1906.08902, June 2019.
- [7] Juan Miguel Arrazola, Alain Delgado, Bhaskar Roy Bardhan, and Seth Lloyd. Quantum-inspired algorithms in practice. *arXiv e-prints*, art. arXiv:1905.10415, May 2019.
- [8] He-Liang Huang, Dachao Wu, Daojin Fan, and Xiaobo Zhu. Superconducting quantum computing: a review. *Science China Information Sciences*, 63(8), July 2020. doi: 10.1007/s11432-020-2881-9. URL <https://doi.org/10.1007/s11432-020-2881-9>.
- [9] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Phys. Rev. Lett.*, 74: 4091–4094, May 1995. doi: 10.1103/PhysRevLett.74.4091. URL <https://link.aps.org/doi/10.1103/PhysRevLett.74.4091>.
- [10] Daniel Gottesman, Alexei Kitaev, and John Preskill. Encoding a qubit in an oscillator. *Phys. Rev. A*, 64:012310, Jun 2001. doi: 10.1103/PhysRevA.64.012310. URL <https://link.aps.org/doi/10.1103/PhysRevA.64.012310>.
- [11] J. Eli Bourassa, Rafael N. Alexander, Michael Vasmer, Ashlesha Patil, Ilan Tzitrin, Takaya Matsuura, Daiqin Su, Ben Q. Baragiola, Saikat Guha, Guillaume Dauphinais, Krishna K. Sabapathy, Nicolas C. Menicucci, and Ish Dhand. Blueprint for a scalable photonic fault-tolerant quantum computer. *Quantum*, 5:arXiv:2010.02905v2, February 2021. doi: 10.22331/q-2021-02-04-392. URL <https://doi.org/10.22331/q-2021-02-04-392>.
- [12] Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Terry Rudolph, and Chris Sparrow. Fusion-based quantum computation. *arXiv e-prints*, art. arXiv:2101.09310, January 2021.
- [13] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit

- Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019. doi: 10.1038/s41586-019-1666-5. URL <https://doi.org/10.1038/s41586-019-1666-5>.
- [14] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020. ISSN 0036-8075. doi: 10.1126/science.abe8770. URL <https://science.sciencemag.org/content/370/6523/1460>.
- [15] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79. URL <https://doi.org/10.22331/q-2018-08-06-79>.
- [16] Gerardo Adesso, Sammy Ragy, and Antony R. Lee. Continuous variable quantum information: Gaussian states and beyond. *Open Systems & Information Dynamics*, 21(01n02):1440001, March 2014. doi: 10.1142/s1230161214400010. URL <https://doi.org/10.1142/s1230161214400010>.
- [17] Alessio Serafini. *Quantum Continuous Variables*. CRC Press, July 2017. doi: 10.1201/9781315118727. URL <https://doi.org/10.1201/9781315118727>.
- [18] Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. Strawberry Fields: A Software Platform for Photonic Quantum Computing. *Quantum*, 3:129, March 2019. ISSN 2521-327X. doi: 10.22331/q-2019-03-11-129. URL <https://doi.org/10.22331/q-2019-03-11-129>.
- [19] URL <https://strawberryfields.ai/photonics/conventions/gates.html>.
- [20] John Kerr LL.D. XI. a new relation between electricity and light: Dielectrified media birefringent. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(332):337–348, 1875. doi: 10.1080/14786447508641302. URL <https://doi.org/10.1080/14786447508641302>.
- [21] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv e-prints*, art. arXiv:1312.6114, December 2013.
- [22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1312.5602, December 2013.
- [23] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go

- with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. doi: 10.1038/nature16961. URL <https://doi.org/10.1038/nature16961>.
- [24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. doi: 10.1038/nature14539. URL <https://doi.org/10.1038/nature14539>.
- [25] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989. doi: 10.1007/bf02551274. URL <https://doi.org/10.1007/bf02551274>.
- [26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL <https://doi.org/10.1162/neco.1989.1.4.541>.
- [27] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. URL <https://arxiv.org/pdf/1706.03762.pdf>.
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.
- [30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. doi: 10.1038/323533a0. URL <https://doi.org/10.1038/323533a0>.
- [31] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
- [32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [33] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and D.G. Cory. Hamiltonian learning and certification using quantum resources. *Physical Review Letters*, 112(19), May 2014. doi: 10.1103/physrevlett.112.190501. URL <https://doi.org/10.1103/physrevlett.112.190501>.



- [34] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009. doi: 10.1103/PhysRevLett.103.150502. URL <https://link.aps.org/doi/10.1103/PhysRevLett.103.150502>.
- [35] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, July 2014. doi: 10.1038/nphys3029. URL <https://doi.org/10.1038/nphys3029>.
- [36] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Phys. Rev. Lett.*, 113:130503, Sep 2014. doi: 10.1103/PhysRevLett.113.130503. URL <https://link.aps.org/doi/10.1103/PhysRevLett.113.130503>.
- [37] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Phys. Rev. Lett.*, 122:040504, Feb 2019. doi: 10.1103/PhysRevLett.122.040504. URL <https://link.aps.org/doi/10.1103/PhysRevLett.122.040504>.
- [38] Patrick Rebentrost, Thomas R. Bromley, Christian Weedbrook, and Seth Lloyd. Quantum hopfield neural network. *Phys. Rev. A*, 98:042308, Oct 2018. doi: 10.1103/PhysRevA.98.042308. URL <https://link.aps.org/doi/10.1103/PhysRevA.98.042308>.
- [39] Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Phys. Rev. X*, 8:021050, May 2018. doi: 10.1103/PhysRevX.8.021050. URL <https://link.aps.org/doi/10.1103/PhysRevX.8.021050>.
- [40] Nathan Killoran, Thomas R. Bromley, Juan Miguel Arrazola, Maria Schuld, Nicolás Quesada, and Seth Lloyd. Continuous-variable quantum neural networks. *Phys. Rev. Research*, 1:033063, Oct 2019. doi: 10.1103/PhysRevResearch.1.033063. URL <https://link.aps.org/doi/10.1103/PhysRevResearch.1.033063>.
- [41] D. Zhu, N. M. Linke, M. Benedetti, K. A. Landsman, N. H. Nguyen, C. H. Alderete, A. Perdomo-Ortiz, N. Korda, A. Garfoot, C. Brecque, L. Egan, O. Perdomo, and C. Monroe. Training of quantum circuits on a hybrid quantum computer. *Science Advances*, 5(10):eaaw9918, October 2019. doi: 10.1126/sciadv.aaw9918. URL <https://doi.org/10.1126/sciadv.aaw9918>.
- [42] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014. doi: 10.1038/ncomms5213. URL <https://doi.org/10.1038/ncomms5213>.
- [43] Shijie Wei, Hang Li, and GuiLu Long. A full quantum eigensolver for quantum chemistry simulations. *Research*, 2020:1–11, March 2020. doi: 10.34133/2020/1486935. URL <https://doi.org/10.34133/2020/1486935>.
- [44] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B. Buckley, David A. Buell, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Benjamin Chiaro, Roberto Collins, William Courtney, Sean Demura, Andrew

- Dunsworth, Edward Farhi, Austin Fowler, Brooks Foxen, Craig Gidney, Marissa Giustina, Rob Graff, Steve Habegger, Matthew P. Harrigan, Alan Ho, Sabrina Hong, Trent Huang, William J. Huggins, Lev Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Cody Jones, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Seon Kim, Paul V. Klimov, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Pavel Laptev, Mike Lindmark, Erik Lucero, Orion Martin, John M. Martinis, Jarrod R. McClean, Matt McEwen, Anthony Megrant, Xiao Mi, Masoud Mohseni, Wojciech Mroczkiewicz, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Hartmut Neven, Murphy Yuezhen Niu, Thomas E. O’Brien, Eric Ostby, Andre Petukhov, Harald Putterman, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Doug Strain, Kevin J. Sung, Marco Szalay, Tyler Y. Takeshita, Amit Vainsencher, Theodore White, Nathan Wiebe, Z. Jamie Yao, Ping Yeh, and Adam Zalcman. Hartree-fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, 2020. ISSN 0036-8075. doi: 10.1126/science.abb9811. URL <https://science.sciencemag.org/content/369/6507/1084>.
- [45] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, September 2017. doi: 10.1038/nature23879. URL <https://doi.org/10.1038/nature23879>.
- [46] Cornelius Hempel, Christine Maier, Jonathan Romero, Jarrod McClean, Thomas Monz, Heng Shen, Petar Jurcevic, Ben P. Lanyon, Peter Love, Ryan Babbush, Alán Aspuru-Guzik, Rainer Blatt, and Christian F. Roos. Quantum chemistry calculations on a trapped-ion quantum simulator. *Phys. Rev. X*, 8:031022, Jul 2018. doi: 10.1103/PhysRevX.8.031022. URL <https://link.aps.org/doi/10.1103/PhysRevX.8.031022>.
- [47] Ryan Sweke, Frederik Wilde, Johannes Meyer, Maria Schuld, Paul K. Faehrmann, Barthélémy Meynard-Piganeau, and Jens Eisert. Stochastic gradient descent for hybrid quantum-classical optimization. *Quantum*, 4:314, August 2020. ISSN 2521-327X. doi: 10.22331/q-2020-08-31-314. URL <https://doi.org/10.22331/q-2020-08-31-314>.
- [48] Seth Lloyd and Samuel L. Braunstein. Quantum computation over continuous variables. *Phys. Rev. Lett.*, 82:1784–1787, Feb 1999. doi: 10.1103/PhysRevLett.82.1784. URL <https://link.aps.org/doi/10.1103/PhysRevLett.82.1784>.
- [49] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Phys. Rev. A*, 99:032331, Mar 2019. doi: 10.1103/PhysRevA.99.032331. URL <https://link.aps.org/doi/10.1103/PhysRevA.99.032331>.
- [50] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, Keri McKiernan, Johannes Jakob Meyer, Zeyue Niu, Antal Száva, and Nathan Killoran. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv e-prints*, art. arXiv:1811.04968, November 2018.

- [51] Thomas R. Bromley, Juan Miguel Arrazola, Soran Jahangiri, Josh Izaac, Nicolás Quesada, Alain Delgado Gran, Maria Schuld, Jeremy Swinarton, Zeid Zabaneh, and Nathan Killoran. Applications of near-term photonic quantum computers: software and algorithms. *Quantum Science and Technology*, 5(3):034010, July 2020. doi: 10.1088/2058-9565/ab8504.
- [52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [53] H. A. Gersch and G. C. Knollman. Quantum cell model for bosons. *Phys. Rev.*, 129:959–967, Jan 1963. doi: 10.1103/PhysRev.129.959. URL <https://link.aps.org/doi/10.1103/PhysRev.129.959>.
- [54] D. Jaksch and P. Zoller. The cold atom Hubbard toolbox. *Annals of Physics*, 315(1):52–79, January 2005. doi: 10.1016/j.aop.2004.09.010.
- [55] D. Jaksch, C. Bruder, J. I. Cirac, C. W. Gardiner, and P. Zoller. Cold bosonic atoms in optical lattices. *Phys. Rev. Lett.*, 81:3108–3111, Oct 1998. doi: 10.1103/PhysRevLett.81.3108. URL <https://link.aps.org/doi/10.1103/PhysRevLett.81.3108>.
- [56] Markus Greiner, Olaf Mandel, Tilman Esslinger, Theodor W. Hänsch, and Immanuel Bloch. Quantum phase transition from a superfluid to a mott insulator in a gas of ultracold atoms. *Nature*, 415(6867):39–44, January 2002. doi: 10.1038/415039a. URL <https://doi.org/10.1038/415039a>.

# NYILATKOZAT

**Név:** Nagy Dániel

**ELTE Természettudományi Kar, szak:** Fizikus Msc

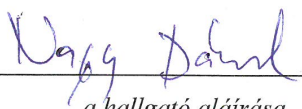
**NEPTUN azonosító:** CXCW8A

**Diplomamunka címe:**

Experimenting with machine learning algorithms on photonic quantum computers

A **diplomamunka** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2021.05.28.

  
a hallgató aláírása