AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

INSTYTUT *INFORMATYKI*

Praca dyplomowa

Assessment of TensorFlow Quantum

Ocena środowiska wspierającego kwantowe uczenie maszynowe TensorFlow Quantum

Autor:	*Justyna Zawalska*
Kierunek studiów:	Informatyka
Opiekun pracy:	*dr inż. Katarzyna Rycerz*

Kraków, 2021

Streszczenie

Wraz z pojawieniem się dziedziny kwantowego uczenia maszynowego wytworzyła się potrzeba stworzenia specjalistycznego oprogramowania, które umożliwiłoby przełożenie teoretycznych rozważań dotyczących tej dziedziny na praktyczne rozwiązania. Obecne badania nad kwantowym uczeniem maszynowym są zdominowane przez rozwiązania skupiające się na hybrydowych kwantowo-klasycznych modelach wykorzystujących zarówno kwantowe, jak i klasyczne dane i algorytmy. Chociaż istniejące biblioteki służące do uczenia maszynowego, takie jak TensorFlow, doskonale sprawdzają się w zastosowaniach bazujących na komponentach klasycznych, same nie mogą wchodzić w interakcje z komponentami kwantowymi. Aby jednak umożliwić taką integrację powstała biblioteka TensorFlow Quantum łącząca klasyczne i kwantowe ekosystemy.

Technologia hybrydowych obliczeń kwantowo-klasycznych nie jest w tej chwili rozwiązaniem w pełni rozwiniętym. Z tego powodu celem niniejszej pracy jest ocena przydatności wykorzystania TensorFlow Quantum do typowych, reprezentatywnych problemów naukowych oraz ocena możliwości trenowania hybrydowych sieci neuronowych. W niniejszej pracy zostały zaproponowane dwa eksperymenty oparte na zastosowaniu algorytmu przybliżonej optymalizacji kwantowej (ang. Quantum Approximate Optimization Algorithm, QAOA) do rozwiązania problemu komiwojażera przy użyciu sparametryzowanych obwodów kwantowych. Uzyskane wyniki pokazują, że TensorFlow Quantum jest narzędziem, które z powodzeniem można wykorzystać do rozwiązywania problemów optymalizacji kombinatorycznej, takich jak problem komiwojażera z użyciem algorytmu QAOA. Ponadto na podstawie przeglądu istniejących zastosowań tej biblioteki można stwierdzić, że wykazuje ona potencjał, aby dzięki jej użyciu dokonać ekscytujących odkryć w dziedzinie kwantowo-klasycznego uczenia maszynowego.

Słowa kluczowe– TensorFlow Quantum, hybrydowe kwantowo-klasyczne sieci neuronowe, algorytm przybliżonej optymalizacji kwantowej (QAOA), optymalizacja kombinatoryczna, problem komiwojażera

Abstract

With the advent of quantum machine learning, the need for specialized software that can transfer theoretical considerations into practical solutions arose. Current research on quantum machine learning is dominated by solutions focusing on hybrid quantum-classical models utilizing quantum and classical data and algorithms. Although the existing machine learning libraries, such as TensorFlow, excel at applications that require classical components, they alone cannot interact with quantum components. As a result, the TensorFlow Quantum library that integrates the classical and quantum ecosystems was created.

The technology of hybrid quantum-classical computations at the moment is not a fully developed solution. For this reason, this thesis aims to assess the usefulness of using TensorFlow Quantum for typical, representative scientific problems and evaluate the trainability of hybrid neural networks. We propose two experiments based on applying the Quantum Approximate Optimization Algorithm (QAOA) to solve the Traveling Salesman Problem using parameterized quantum circuits. The obtained results show that TensorFlow Quantum can be successfully used for solving combinatorial optimizations problems such as the Traveling Salesman Problem with the QAOA. Additionally, based on an overview of existing applications of this library, it can be concluded that TensorFlow Quantum used as a research tool has the potential to pave the way for exciting discoveries in the field of quantum-classical machine learning.

Keywords— TensorFlow Quantum, hybrid quantum-classical neural networks, Quantum Approximate Optimization Algorithm, combinatorial optimization, Traveling Salesman Problem

Acknowledgements

I would like to express my genuine gratitude to my supervisor, dr inż. Katarzyna Rycerz for helping me advance scientifically and providing me with a 'superposition' of guidance, support, and encouragement while I was preparing this thesis.

This research was supported in part by PLGrid Infrastructure.

Contents

Chapter 1

Introduction

This chapter provides a brief introduction to recent developments in the area of quantum computing and defines the goals for this thesis.

1.1 Motivation

Quantum computing is an up-and-coming field that can significantly impact the way of processing information. Scientists believe that for some problems quantum computer algorithms can achieve a speedup impossible to obtain even on the best classical supercomputers. The power of quantum computing arises from utilizing quantum mechanics principles and phenomena. One of the features that make quantum computing different from classical computing is the use of quantum bits (qubits) instead of classical bits. While a classical bit can be represented as 0 or 1, a qubit is a superposition of both 0 and 1 states. As a result, only n qubits are needed to describe a state of 2^n complex numbers [1].

Prospective applications of quantum computing are very impressive and range from improving cybersecurity to discovering materials or drugs. What researchers both from academia and industry are seeking to prove is "quantum supremacy." Quantum supremacy will be achieved when a problem unsolvable in any reasonable time on a classical computer will be solved on a quantum computer. At the end of 2019, Google researchers claimed to obtain this milestone [2]. They presented a computational task that took around 200 seconds on their 53–qubit quantum processor that would otherwise take about 10 000 years on then the best supercomputer — Summit. However, further research [3] proved that the same task could be performed on Summit within two and a half days. Nevertheless, performing this task on a quantum computer seems to be considerably faster than using high-performance computing.

There are a few quantum algorithms that have the potential to change computer science. One of them, invented more than twenty years ago, is Shor's algorithm [4]. Although this integer factorization algorithm can break the RSA encryption, existing quantum computers are not powerful enough to achieve this breakthrough. Currently developed quantum integrated circuits are considered Noisy Intermediate-Scale Quantum (NISQ) technology [5]. They have only up to a few hundred imperfectly controlled physical qubits with a high error rate. On the one hand, NISQ devices cannot perform very challenging tasks, but they are well suited for performing hybrid quantum-classical algorithms.

Hybrid quantum-classical algorithms do not need to use many qubits but are very potent because they combine the power of quantum and classical computing. Hybrid computation is comprised of tasks that can be split into parts solved on a quantum device and parts solved classically. In the family of hybrid algorithms, researchers distinguish Variational Quantum Algorithms [6] such as Variational Quantum Eigensolver [7] and Quantum Approximate Optimization Algorithm [8]. The former is mainly used for quantum simulation, while the latter is used for solving combinatorial optimization problems. With such a broad range of potential use, the hybrid algorithms are introducing the first valuable applications for current quantum devices.

Furthermore, hybrid quantum-classical models are a foundation of one of the branches of quantum machine learning. Merging ideas from two such important fields as machine learning and quantum computing can speed up the process of inventing approaches to solving significant problems. Although classical machine learning techniques have proven their ability to generalize patterns in classical data, they alone cannot process quantum data. For this task, specialized software libraries have been proposed; one of them is TensorFlow Quantum [9].

1.2 Goals

Although the area of quantum machine learning is gaining more and more attention, its potential remains undiscovered. Experts both from academia and industry wonder if the development of this branch of science can lead to important discoveries. To accelerate the research suitable software is needed. At present, there exist a few new software solutions, however one of the most compelling questions is whether they are eligible to solve a relevant problem with quantum machine learning. Therefore, the goal of this thesis is to evaluate the capabilities of one of the quantum machine learning libraries — TensorFlow Quantum. This task involves in particular:

- providing an outline of TensorFlow Quantum;
- investigating the types of quantum algorithms that can be implemented using TensorFlow Quantum;
- assessing the actual capabilities of the TensorFlow Quantum environment for a representative scientific problem. For this purpose, one of the most popular quantum optimization algorithms was chosen, namely the Quantum Approximate Optimization Algorithm. We have chosen the Traveling Salesman Problem as a

typical use case for that algorithm.

1.3 Thesis structure

Chapter 2 presents the quantum computing basics and introduces two important hybrid quantum-classical algorithms: the Variational Quantum Algorithm and the Quantum Approximate Optimization Algorithm. Chapter 3 outlines quantum machine learning, especially state-of-the-art quantum neural networks. Moreover, an overview of software for quantum machine learning is provided. Chapter 4 overviews the TensorFlow Quantum library by discussing its design and significant features. With the aim of assessing the usability of this framework, a study of using a quantum neural network aimed at solving the Traveling Salesman Problem with the use of the Quantum Approximate Optimization Algorithm is presented in Chapters 5 (problem formulation) and Chapter 6 (experiments). Finally, in Chapter 7, a summary of the examined TensorFlow Quantum properties and achieved results is provided.

Chapter 2

Quantum Algorithms

The first part of this chapter presents basic terms related to quantum computing. Then descriptions of the most promising hybrid quantum-classical algorithms for the near-term quantum devices are provided.

2.1 Preliminaries

The quantum computing model of computation is fundamentally different from classical computing. To introduce the field of quantum computing, selected quantum computing basics and necessary terminology are described below.

2.1.1 Quantum states

A state in quantum mechanics is represented as a vector from a Hilbert space taken over the complex numbers. The standard notation for quantum states is the *Dirac notation* (also known as the *bra-ket notation*) that distinguishes by using '$|\cdot\rangle$' and '$\langle\cdot|$' symbols. Table 2.1 presents a summary of this notation.

Table 2.1: The Dirac notation for concepts from linear algebra [10].

Notation	**Description**
$z^* = a - bi$	Complex conjugate of the number $z = a + bi, z \in \mathbb{C}$.
$\|\psi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{bmatrix}$, where $\alpha_i \in \mathbb{C}$ and $\sum_i \|\alpha_i\|^2 = 1$	$\|\psi\rangle$ — a vector, also known as a *ket*. In the complex vector space it is identified with a column vector.
$\langle\psi\| = \begin{bmatrix} \alpha_0^* & \alpha_1^* & \dots & \alpha_{n-1}^* \end{bmatrix}$	$\langle\psi\|$ — Hermitian conjugate of $\|\psi\rangle$, also known as a *bra*. In the complex vector space identified with a row vector.
$\langle\psi\| \phi\rangle = \begin{bmatrix} \alpha_0^* & \alpha_1^* & \dots & \alpha_{n-1}^* \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{bmatrix}$	Inner product between the vectors $\|\psi\rangle$ and $\|\phi\rangle$.
$\|\psi\rangle \langle\phi\| = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} \begin{bmatrix} \beta_0^* & \beta_1^* & \dots & \beta_{n-1}^* \end{bmatrix}$	Outer product between the vectors $\|\psi\rangle$ and $\|\phi\rangle$.
$\|\psi\rangle \otimes \|\phi\rangle = \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} \otimes \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_{n-1} \end{bmatrix} = \begin{bmatrix} \alpha_0\beta_0 \\ \vdots \\ \alpha_0\beta_{n-1} \\ \vdots \\ \alpha_{n-1}\beta_0 \\ \vdots \\ \alpha_{n-1}\beta_{n-1} \end{bmatrix}$	Tensor product of $\|\psi\rangle$ and $\|\phi\rangle$ (abbreviated forms: $\|\psi\rangle \|\phi\rangle$, $\|\psi\phi\rangle$).

In quantum computing, the simplest unit of information is a two-state system, called a quantum bit — *qubit*. Usually, the qubit's state $|\psi\rangle$ is mathematically described as

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{bmatrix}\alpha_0 \\ \alpha_1\end{bmatrix}, \quad \text{where } \alpha_0, \alpha_1 \in \mathbb{C} \text{ and } |\alpha_0|^2 + |\alpha_1|^2 = 1 \tag{2.1.1}$$

which is a *superposition* (linear combination) of the computational basis states

$$|0\rangle = \begin{bmatrix}1 \\ 0\end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix}0 \\ 1\end{bmatrix}. \tag{2.1.2}$$

The coefficients α_0 and α_1 are called *amplitudes.* Based on the fact that the amplitudes are complex numbers and the sum of their squares equals 1, there exist angles $\gamma, \theta, \phi \in \mathbb{R}$ that satisfy

$$|\psi\rangle = e^{i\gamma}(cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle). \tag{2.1.3}$$

The global phase $e^{i\gamma}$ is irrelevant, so Equation 2.1.3 can be rewritten as

$$|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle. \tag{2.1.4}$$

As a result, the state of a single qubit can be represented graphically as a point on the *Bloch sphere* (cf. Figure 2.1), using the spherical coordinates

$$(sin\theta cos\phi, sin\theta sin\phi, cos\theta) \in \mathbb{R}^3. \tag{2.1.5}$$

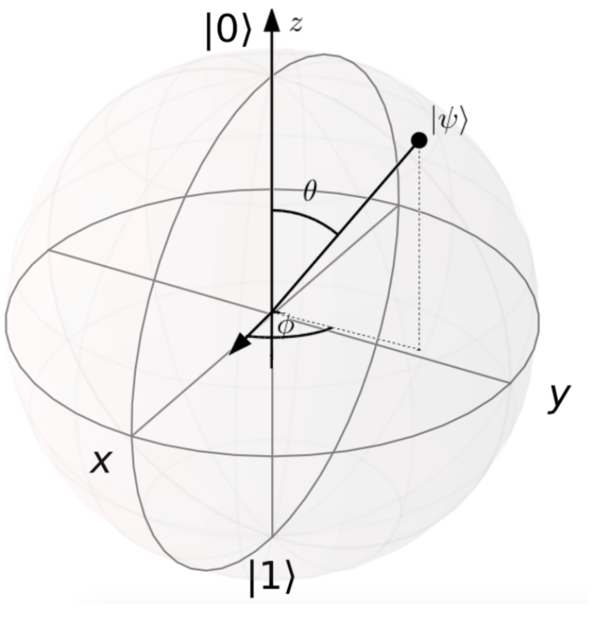


Figure 2.1: The Bloch sphere.

When we have n classical bits, there are 2^n possible states

$$00..0, 00..1, \ldots, 1..11 \text{ (equiv. } 0, 1, \ldots, 2^n - 1). \tag{2.1.6}$$

Correspondingly, a n–qubit quantum system consists of 2^n computational basis states

$$|00..0\rangle, |00..1\rangle, \ldots, |1..11\rangle \text{ (equiv. } |0\rangle, |1\rangle, \ldots, |2^n - 1\rangle). \tag{2.1.7}$$

A multi-qubit state is created as a tensor product of qubits. E.g., for qubits $|\psi\rangle$ and $|\phi\rangle$

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle, |\phi\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle \tag{2.1.8}$$

,

the 2–qubit state is

$$\begin{aligned} |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{bmatrix} &= \alpha_0\beta_0 |00\rangle + \alpha_0\beta_1 |01\rangle + \alpha_1\beta_0 |10\rangle + \alpha_1\beta_1 |11\rangle \\ &= \alpha_0\beta_0 |0\rangle + \alpha_0\beta_1 |1\rangle + \alpha_1\beta_0 |2\rangle + \alpha_1\beta_1 |3\rangle \end{aligned} \tag{2.1.9}$$

where $\sum_{i,j=0}^{1} |\alpha_i\beta_j|^2 = 1$.

An important term in the quantum mechanics terminology is an *operator*. The operator A is a function that transforms one state vector into another. Suppose

$$A|\psi\rangle = a|\psi\rangle, \tag{2.1.10}$$

where $a \in \mathbb{R}$, then A has the eigenstate $|\psi\rangle$ and the eigenvalue a.

Given k independent eigenvectors $v_0, v_1, \ldots, v_{k-1}$ and corresponding to them eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_{k-1}$, the $k \times k$ operator A can be found with the following formula

$$A = V\Lambda V^{-1}, \tag{2.1.11}$$

where V is a diagonal matrix containing eigenvalues, and Λ is a matrix of concatenated eigenvectors

$$\Lambda = \begin{bmatrix} \lambda_0 & 0 & \ldots & 0 \\ 0 & \lambda_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_{k-1} \end{bmatrix}, V = \begin{bmatrix} v_0 v_1 \ldots v_{k-1} \end{bmatrix}.$$

Quantum states are not directly observable. A qubit or multi-qubit system has to be measured to provide information. Usually, this process refers to the *measurement* *in the computational basis*. For example, suppose a qubit is in the state described by Equation 2.1.1. After the measurement in the computational basis, the result is labeled as the classical bit 0 with probability $|\alpha_0|^2$ and 1 with probability $|\alpha_1|^2$, and due to wave function collapse, the new quantum state is $|0\rangle$ or $|1\rangle$ depending on the measurement outcome. Additionally, a quantum state cannot be copied, so it is impossible to execute multiple independent measurements. Figure 2.2 illustrates the process of measuring a qubit.

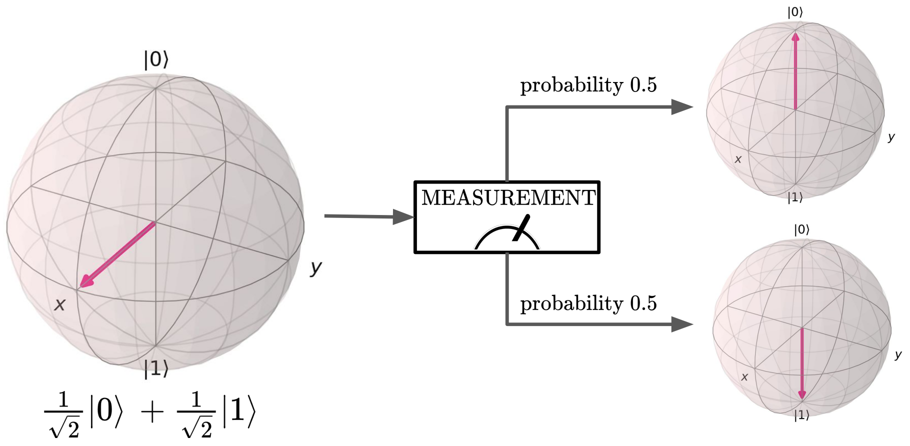


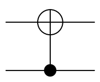
Figure 2.2: Measuring a qubit. The probability of obtaining the output state $|0\rangle$ is 0.5, and the output state $|1\rangle$ is 0.5.

2.1.2 Quantum circuit model

There are several equivalent models of quantum computation. One of them is the quantum circuit model based on using quantum logic gates. The quantum gates enable manipulating the state of a quantum system.

Quantum circuits consist of quantum wires represented by horizontal lines and quantum gates depicted by boxes with symbols. In Table 2.2, selected quantum gates with their symbolical and matrix representations are introduced. Among the listed quantum operators, only the Controlled-NOT gate is a 2-qubit gate, and the rest are single-qubit gates. Also, what is worth noting is that the rotation gates are parameterized.

Table 2.2: Quantum gates.

Name	**Symbol**	**Unitary Matrix**
Identity	$-\boxed{I}-$	$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Hadamard	$-\boxed{H}-$	$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Pauli-X	$-\boxed{X}-$	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y	$-\boxed{Y}-$	$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z	$-\boxed{Z}-$	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Rotation-X	$-\boxed{Rx(\theta)}-$	$R_x\theta = \begin{bmatrix} cos\frac{\theta}{2} & -isin\frac{\theta}{2} \\ -isin\frac{\theta}{2} & cos\frac{\theta}{2} \end{bmatrix}$
Rotation-Y	$-\boxed{Ry(\theta)}-$	$R_y(\theta) = \begin{bmatrix} cos\frac{\theta}{2} & -sin\frac{\theta}{2} \\ sin\frac{\theta}{2} & cos\frac{\theta}{2} \end{bmatrix}$
Rotation-Z	$-\boxed{Rz(\theta)}-$	$R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$
Controlled-NOT		$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

To describe the behavior of the quantum circuit computations, we will examine the examples of using the Pauli-X gate and creating combinations of quantum gates. The classical logic NOT gate implements logical negation, the quantum operator of NOT – the Pauli-X gate mimics this behavior, which can be written as

$$X|0\rangle = |1\rangle\,, X|1\rangle = |0\rangle\,. \tag{2.1.12}$$

Apart from the symbolical representation, the Pauli-X gate can also be represented as a unitary matrix

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.1.13}$$

or as a quantum circuit

$$-\boxed{X}-\,.$$

The mathematical representation below

$$\begin{aligned} X|\psi\rangle &= X(\alpha|0\rangle + \beta|1\rangle) = X(\alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix}) \\ &= \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\\beta\end{bmatrix} = \begin{bmatrix}\beta\\\alpha\end{bmatrix} = \beta\begin{bmatrix}1\\0\end{bmatrix} + \alpha\begin{bmatrix}0\\1\end{bmatrix} \\ &= \beta|0\rangle + \alpha|1\rangle \end{aligned} \tag{2.1.14}$$

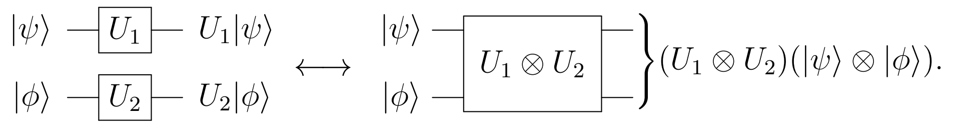
is equivalent to the following quantum circuit

$$\alpha|0\rangle + \beta|1\rangle \; -\boxed{X}- \; \beta|0\rangle + \alpha|1\rangle\,.$$

Combinations of quantum gates — e.g., U_1, U_2 – some single qubit quantum gates — create serially wired circuits

$$|\psi\rangle \; -\boxed{U_1}-\boxed{U_2}- \; U_2U_1|\psi\rangle \quad \longleftrightarrow \quad |\psi\rangle \; -\boxed{U_2 \cdot U_1}- \; U_2U_1|\psi\rangle\,,$$

and parallel circuits e.g.,



Quantum circuits that perform operations on input qubits and end with measurement are called quantum algorithms.

2.2 Variational Quantum Algorithms

Near-term quantum computing devices offer only a limited number of qubits and are exposed to noise (thus not fault-tolerant). Considering these drawbacks, current quantum algorithms cannot be efficient enough to outperform classical algorithms. However, a paradigm of hybrid quantum-classical algorithms that utilize both quantum and classical computational resources has emerged as a method to provide the quantum advantage. Usually, in the hybrid approach, a quantum computer aims to prepare and evaluate the quantum state, while a classical computer offers pre-processing and post-processing and an optimization algorithm, cf. Figure 2.3. An example of a hybrid quantum-classical algorithm is the class of Variational Quantum Algorithms (VQAs).

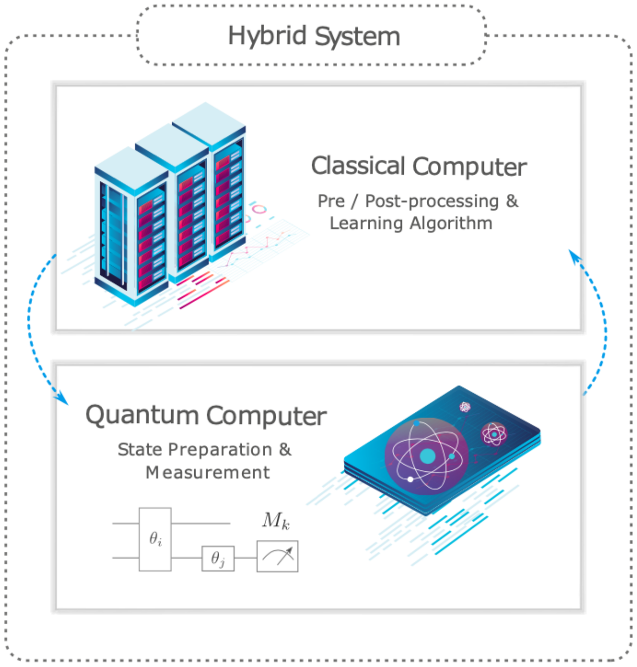


Figure 2.3: Hybrid system consisting of classical and quantum computers. Reprinted from Ref. [11].

The VQA aims at approximating the minimum of a cost function that encodes a selected problem [6]. For this task, it employs a hybrid loop that consists of

- a *parameterized quantum circuit* (PQC), also known as a *variational circuit*, containing both parameterized and fixed quantum gates that enable the manipulation of quantum information;
- a classical optimizer that adjusts the parameters of the PQC to minimize the value of the cost function.

A simplified diagram of the VQA is presented in Figure 2.4 and described below.

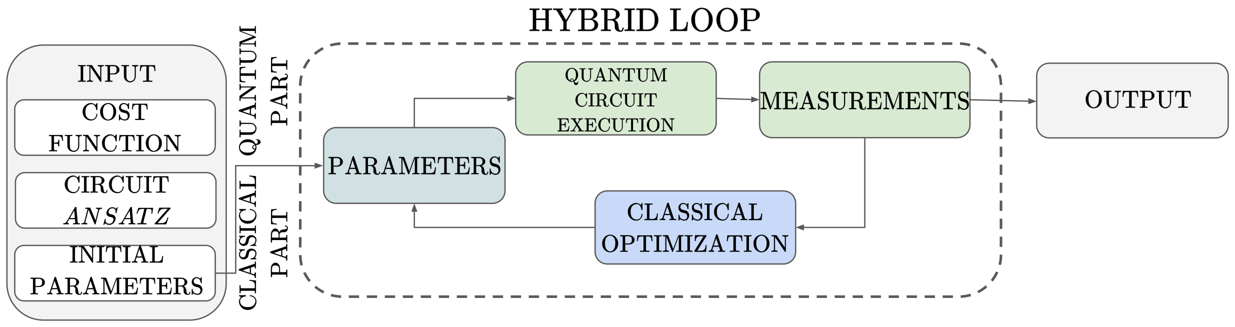


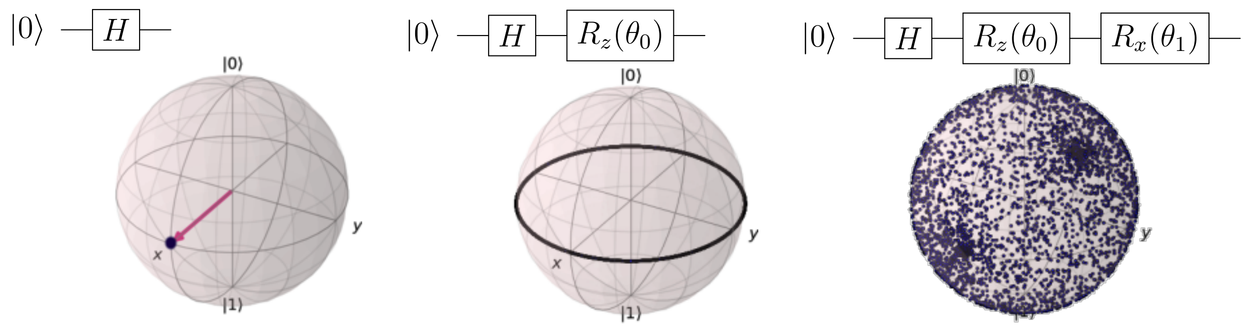
Figure 2.4: Variational Quantum Algorithm scheme.

- Input
  - Cost function $C(\boldsymbol{\theta})$ — a function of continuous or discrete parameters $\boldsymbol{\theta}$ that encodes the solution to the problem.
  - Circuit ansatz — a circuit template that is expressible enough to cover the solution space adequately. For this reason, the circuit ansatz consists of gates with parameters (e.g., rotation gates).

    Figure 2.5 presents the expressibility of simple 1-qubit ansatze. The circuit in Figure 2.5a is not parameterized and has very low expressibility. The circuit in Figure 2.5b has only one parameterized gate (R_z), so the cost function will be parameterized with one parameter θ_0. Figure 2.5c depicts a circuit with two parameterized gates (R_z and R_x) so the cost function will be parameterized with two parameters θ_0, and θ_1. The ansatz determines the number and meaning of the cost function parameters $\boldsymbol{\theta}$.

    Usually, the ansatz design is dictated either by the problem whose solution it should encode or the hardware on which the circuit will be executed. In the ideal case, the ansatz circuit should cover the whole solution space, be shallow (the more gates, the more noise), and not have too many parameters to optimize [12]. However, finding the appropriate ansatz for specific use-cases might be challenging.
  - Initial parameters $\boldsymbol{\theta}$ — the variational parameters that are applied to parameterized gates. Since there are not many rules on choosing them, they can be set to zeros [13] or random numbers [14]. Another practice is utilizing the *layerwise learning* [15] technique which is based on optimizing only the subset of the variational parameters at first and then gradually increasing the number of the trained parameters.
- Hybrid loop — aims at solving the optimization task [6]

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}). \tag{2.2.1}$$



(a) The input state $|0\rangle$ after the Hadamard gate always becomes the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ (represented as a single dot).

(b) The outputs of the circuit consisting of the Hadamard gate and the parameterized Rotation-Z gate are located on the equator of the Bloch sphere.

(c) The outputs of this circuit are distributed all over the Bloch sphere but the distribution is not uniform.

Figure 2.5: Expressibility of 1-qubit circuit ansatze after sampling them with 1000 different pairs of parameters. The results are superimposed on the Bloch sphere. Based on Ref. [12].

- Quantum part
  - Quantum circuit execution — the parameter values are applied to the parameterized circuit, then the circuit is executed.
  - Measurements — the circuit execution enables to evaluate the cost function $C(\boldsymbol{\theta})$ or its gradient.
- Classical part — optimization
  The classical device is responsible for optimizing the parameters $\boldsymbol{\theta}$. Usually, it is a gradient–based optimization. The choice of the optimizer type has a significant impact on the obtained results.

- Output — when the termination condition is met (e.g., the cost function value is small enough or the number of loop repetitions is reached), an approximate solution to the problem is returned. The solution can have the form of, among other things, a string of bits or probability distribution.

All things considered, the hybrid quantum-classical structure of the Variational Quantum Algorithms can be seen as a groundwork for developing other algorithms. An example algorithm based on the VQA is the Quantum Approximate Optimization Algorithm (QAOA).

2.3 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) [8] was proposed for approximating the solutions of the combinatorial optimization problems. These problems can be expressed in the form of the cost function

$$C(z) = \sum_{a=1}^{m} C_a(z), \tag{2.3.1}$$

where z is an n-bit binary string $z = \{0,1\}^n$, m is the number of clauses/constraints that should be satisfied, and $C_a(z)$ – a Boolean function ($f : \{0,1\}^n \to \{0,1\}$) such that

$$C_a(z) = \begin{cases} 1, & \text{if } z \text{ satisfies the constraint } a, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3.2}$$

The goal is to find a bit string z that maximizes the number of satisfied constraints

$$C_{\max} = \max_{z \in \{0,1\}^n} C(z). \tag{2.3.3}$$

However, the Quantum Approximate Optimization Algorithm is designed for minimization tasks, hence

$$C_{\max} = \min_{z \in \{0,1\}^n} -C(z). \tag{2.3.4}$$

To solve a combinatorial optimization problem with the QAOA, it is necessary to translate the problem's cost function acting on bit strings into a Hamiltonian acting on qubits.

There are 2^n possible different inputs z. In the quantum setting they can be expressed as an n-qubit quantum system consisting of 2^n computational basis states (cf. eq. 2.1.6 – 2.1.7).

The Hamiltonian is a quantum operator that describes the energy of a physical system. As a result, knowing the Hamiltonian enables exploring, i.a. the behavior and states of physical systems. It is sometimes beneficial to think of the Hamiltonian as a matrix because Hamiltonian operators have associated with them Hamiltonian matrices (energy matrices).

The Hamiltonian H_C represents the function C acting on an n-bit string z if for each input $z \in \{0,1\}^n$ with the corresponding computational basis state $|z\rangle$ it satisfies

$$H_C |z\rangle = C(z) |z\rangle . \tag{2.3.5}$$

The operator H_C has eigenvectors $|z\rangle$ and eigenvalues $C(z)$. Suppose we find all the eigenvalues and order them from the lowest to the highest. In that case, the lowest eigenvalue is the optimal value of the cost function C_{max}, and the eigenvector associated with this eigenvalue corresponds to the solution.

To provide an example of creating a Hamiltonian for a function, we will examine the Boolean identity function

$$f(x) = x, \, x \in \{0,1\}. \tag{2.3.6}$$

Based on Eq. 2.3.5 we obtain

$$\begin{aligned} H_f \left|0\right\rangle &= f(0)\left|0\right\rangle = 0\left|0\right\rangle \\ H_f \left|1\right\rangle &= f(1)\left|1\right\rangle = 1\left|1\right\rangle . \end{aligned} \tag{2.3.7}$$

Operator H_f has the following eigenvalues and eigenvectors

$$\lambda_0 = 0, v_0 = \left|0\right\rangle = \begin{bmatrix}1\\0\end{bmatrix} \text{ and } \lambda_1 = 1, v_1 = \left|1\right\rangle = \begin{bmatrix}0\\1\end{bmatrix}. \tag{2.3.8}$$

Using Eq. 2.1.11, we obtain the matrix representation of the Hamiltonian H_f

$$H_f = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}. \tag{2.3.9}$$

In general, every Hamiltonian operator for a Boolean or pseudo-Boolean function ($f : \{0,1\}^n \to \mathbb{R}$) can be expressed as a linear combination of the gates I, Z, and $Z \otimes Z$ acting on proper qubits. As a result, the matrix representation of H_f is equivalent to the Hamiltonian operator $\frac{1}{2}(I - Z)$ since

$$\frac{1}{2}(I - Z) = \frac{1}{2}\Big(\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} - \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\Big) = \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}. \tag{2.3.10}$$

Based on the above observations, the Hamiltonian H_f represents the Boolean clause $f(x) = x$ and

$$x \longleftrightarrow \frac{1}{2}(I - Z). \tag{2.3.11}$$

Nevertheless, the Boolean functions themselves are not expressible enough for defining complex combinatorial optimization problems. For this purpose, pseudo-Boolean functions are used. The Quadratic Unconstrained Binary Optimization (QUBO) [16] problems represent a significant class of the pseudo-Boolean functions. In the QUBO, the aim is to find a minimum or maximum of the function [17]

$$f(x) = a + \sum_{j=1}^{n} c_j x_j + \sum_{j \leq k} d_{jk} x_j x_k, \tag{2.3.12}$$

where $a, c_j, d_{jk} \in \mathbb{R}, x_j \in \{0,1\}$. If we substitute all the variables x from Equation 2.3.12 using Property 2.3.11, we obtain the Hamiltonian [17]

$$H_f = (a + c + d)I - \frac{1}{2}\sum_{j=1}^{n}(c_j + d_j)Z_j + \frac{1}{4}\sum_{j<k} d_{jk} Z_j Z_k, \tag{2.3.13}$$

where $c = \frac{1}{2}\sum_{j=1}^{n} c_j, d = \frac{1}{4}\sum_{j \leq k} d_{jk},$ and $d_j = \sum_{k:k \neq j} d_{jk}$ with $d_{jk} = d_{kj}$. Based on the Variational Quantum Algorithm scheme defined in Section 2.2, below are described the steps of the QAOA.

- Input
  - Cost function

    The first step is express the cost function as the cost Hamiltonian H_C (sometimes denoted as the phase Hamiltonian). Usually, it is helpful to start with defining the QUBO formulation of the problem and then map it to a Hamiltonian form (cf. Eq. 2.3.12 and 2.3.5).

    Apart from the cost Hamiltonian, a mixer Hamiltonian H_M is needed. The simplest mixer Hamiltonian has the following form

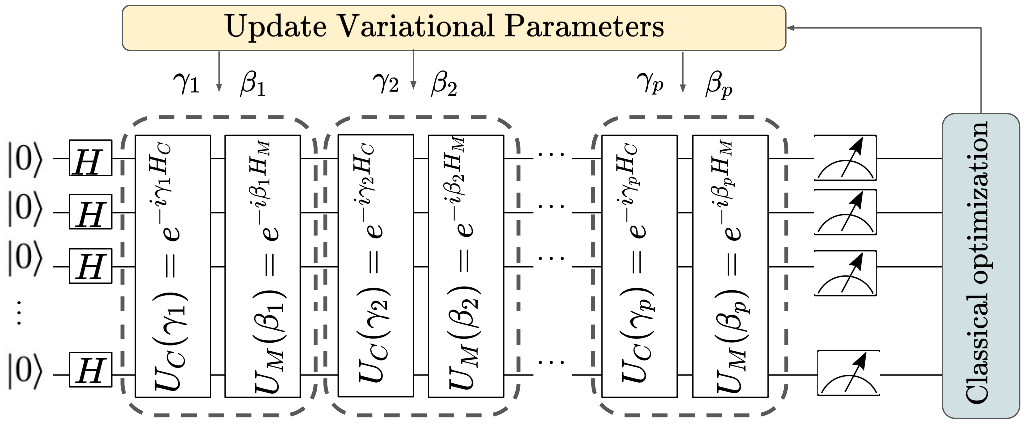
    $$H_M = \sum_{j=1}^{n} X_i, \tag{2.3.14}$$

    where X_i is the Pauli-X gate acting on the i-th qubit. H_M improves the number of states that can be potentially reached.
  - Circuit ansatz

    The QAOA uses the *Alternating Operator Ansatz* which contains p alternating layers that consist of U_C and U_M

    $$U_C(\gamma) = e^{-i\gamma H_C}, \tag{2.3.15}$$
    $$U_M(\beta) = e^{-i\beta H_M}. \tag{2.3.16}$$

    U_C and U_M can be translated into quantum gates. The hyperparameter p defines the depth of the circuit. The layered architecture of the QAOA is presented in Figure 2.6.



Update Variational Parameters
γ_1 β_1
γ_2 β_2
γ_p β_p
$|0\rangle$
H
$U_C(\gamma_1) = e^{-i\gamma_1 H_C}$
$U_M(\beta_1) = e^{-i\beta_1 H_M}$
$U_C(\gamma_2) = e^{-i\gamma_2 H_C}$
$U_M(\beta_2) = e^{-i\beta_2 H_M}$
$U_C(\gamma_p) = e^{-i\gamma_p H_C}$
$U_M(\beta_p) = e^{-i\beta_p H_M}$
Classical optimization

Figure 2.6: Hybrid optimization loop containing the p-layer quantum ansatz.

  - Initial state and parameters

    The initial quantum state is the uniform superposition on all qubits, defined as

    $$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle, \tag{2.3.17}$$

where $|s\rangle$ can be obtained through applying the Hadamard gate in parallel on each qubit.

Based on the QAOA's hyperparameter p there are $2p$ initial parameters (angles) that need to be optimized $\gamma_1, \ldots, \gamma_p, \equiv \gamma$ and $\beta_1, \ldots, \beta_p \equiv \beta$. Generally, there are not any predefined initialization values, so this could be zeros [13] or random numbers [14] if there are no better propositions.

- Hybrid loop
  The goal is to find $2p$ angles γ and β that minimize the expectation value

$$E(\gamma, \beta) = \langle \gamma, \beta| H_C |\gamma, \beta\rangle \tag{2.3.18}$$

  - (Quantum computer) Construct the quantum state with the p-level QAOA

$$|\gamma, \beta\rangle = U_M(\beta_p)U_C(\gamma_p) \cdots U_M(\beta_1)U_C(\gamma_1) |s\rangle . \tag{2.3.19}$$

  - (Quantum computer) Estimate the expectation value $E(\gamma, \beta)$.
  - (Classical computer) Use an optimizer to vary the angles γ and β in order to obtain angles γ^* and β^* that

$$(\gamma^*, \beta^*) = \arg\min_{\gamma,\beta} E(\gamma, \beta). \tag{2.3.20}$$

- Output — $|\gamma, \beta\rangle$ measured in the computational basis to obtain the bit string that is the solution to the problem.

To sum up, the QAOA is an algorithm that searches for an optimal bit string that satisfies a specified combinatorial optimization problem. What is characteristic is that the ansatz is based on two Hamiltonians applied in alteration. The QAOA is a very suitable algorithm for near-term quantum devices because it enables easy control of the depth of the quantum circuit.

2.4 Summary

This chapter introduced the building blocks of quantum algorithms for the quantum circuit model. One of the most interesting aspects of the Variational Quantum Algorithms is their hybrid quantum-classical design. The next chapter will explore the connection between the Variational Quantum Algorithms and quantum machine learning.

Chapter 3

Quantum Machine Learning

This chapter introduces recent developments in the area of quantum machine learning. The first part presents selected state-of-the-art quantum neural network algorithms, and the second focuses on available software solutions.

3.1 Quantum neural networks

A quantum neural network (QNN) can be perceived as a type of an artificial neural network that incorporates quantum computations. Some of the classical neural networks have inspired their quantum analogs. Below is an overview of the most popular artificial neural networks and inspired by them quantum neural networks.

3.1.1 Quantum feedforward neural networks

An artificial neuron (Fig. 3.1) converts several input values into a single output value. To compute the output, first, the neuron's inputs x are multiplied by real numbers called weights w that represent the importance of each input value, and the weighted sum $\sum_i w_i x_i$ is calculated. Then, to the weighted sum is added a bias term b. Next, an activation function g that transforms the data before passing it to the following layer is applied $g(b + \sum_i w_i x_i)$. The activation function (usually nonlinear) defines the node's output, and the bias term allows for shifting this function's curve.

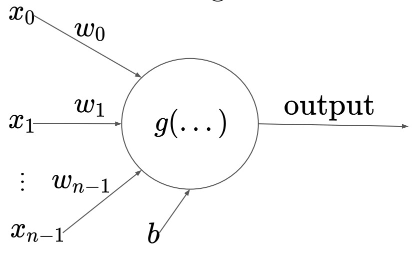


Figure 3.1: Artificial neuron.

Neural networks training consists of constant adjustments of network's parameters (weights and biases) to fit the data. It is usually achieved through the backpropagation algorithm [18]. Backpropagation uses the gradient methods to optimize the loss function (measure of the network's performance) by changing the weights and biases. Feedforward neural networks are types of neural networks in which the neurons of one layer are only connected with the neurons from the next layer; consequently, there are no cycles. The information is 'fed forward' through the input layer, then the hidden layers (if present), and lastly through the output layer, cf. Figure 3.2.

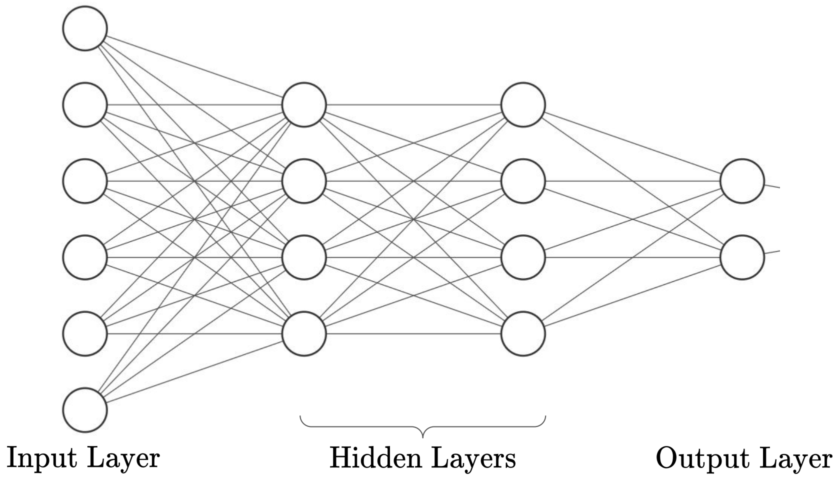


Figure 3.2: Feedforward neural network design. Based on Ref. [19].

Quantum feedforward neural networks [20] have a very similar design to the classical feedforward neural networks, cf. Figure 3.3. They consist of consecutive layers containing quantum nodes that are implemented using quantum hardware.

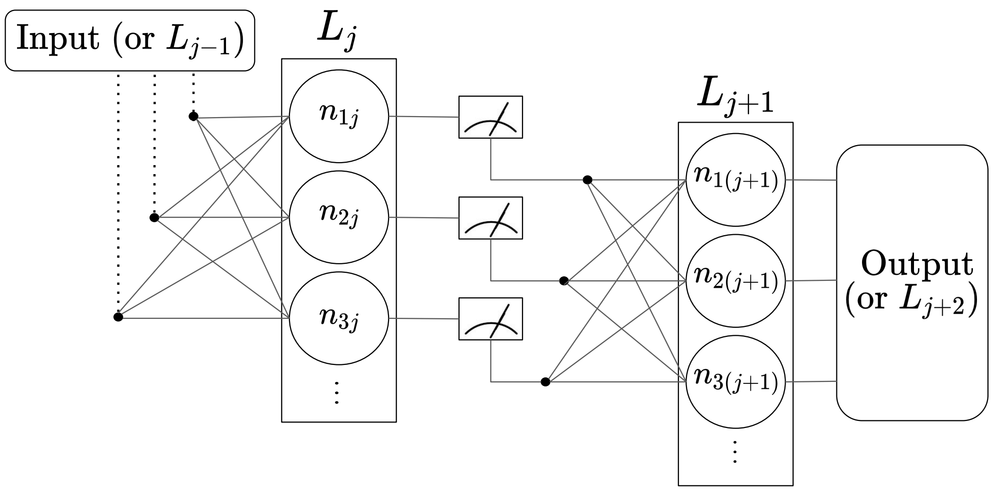


Figure 3.3: The results of measurements performed on quantum neurons are passed to the successive layer. Based on Ref. [20].

The creation of fully quantum neurons is very challenging since the power of neural networks lies in the nonlinear activation functions while quantum computers perform only linear operations. However, it is possible to implement quantum circuits that approximate nonlinear functions using quantum oracles and measurements [21].

3.1.2 Quantum convolutional neural networks

Convolutional neural networks (CNNs) were proposed as object recognition systems [22]. One of their first applications was handwritten digits recognition. Additionally, deep CNNs also proved to be tremendously successful in the image classification task [23], winning the ImageNet contest in 2012, and as a result bringing much attention to the deep learning field. In general, CNNs are highly recommended when working with grid-structured data topology, especially two-dimensional images [24]. Usually, a CNN consists of alternating *convolution layers* and *pooling layers* followed by a fully connected layer at the end of the model.

A convolution layer uses filters (kernels) to detect features in pictures. Essentially, the image is divided into regular-shaped (possibly overlapping) regions covering the entire image. Then, these regions are convolved (multiplied) by the elements of the kernel. This process detects objects but associates them with their location, i. e. the object will not be recognized when we add a small translation to the input. The purpose of the pooling layer is to provide invariance to input translations and reduce data size for easier processing in the next layers.

CNNs directly inspired two quantum neural network models: quantum convolutional neural networks [25] and quanvolutional neural networks [26].

Quantum convolutional neural networks (QCNNs) translate the CNNs structure to the quantum architecture.
The components of a QCNN [25, 27]:

1. Input: a quantum state $|\psi\rangle_{in}$.
2. Convolution circuit: applying unitary operators (U_i).
3. Pooling circuit: applying unitary operators (V_i). This step reduces the size of the quantum system.
4. Repeating steps 2.–3. until the system size is reduced enough.
5. Fully connected layer: applying a multi-qubit unitary operator (F) to the remaining qubits.
6. Measuring output qubits.

This process is illustrated in Figure 3.4.

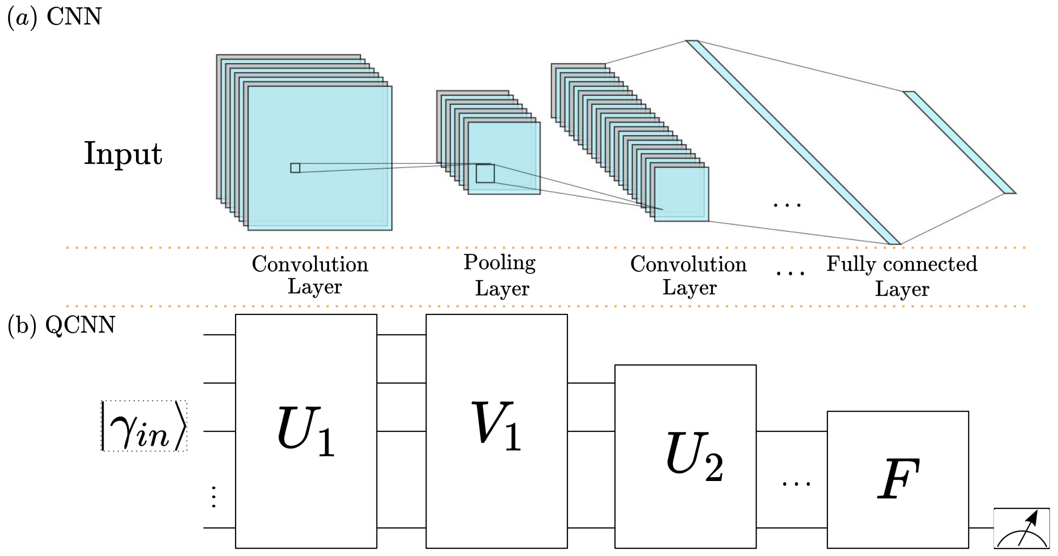


Figure 3.4: Simplified architecture comparison of CNNs and QCNNs. Quantum error correction (QEC) might be applied in pooling layers. Based on Ref. [27] and [20].

During the training phase the parameters of the unitary gates are learned, the number of convolutions and pooling layers are treated as predetermined model hyperparameters.

Quanvolutional neural networks (QNNs). While QCNNs use only the quantum environment, the quanvolutional neural networks are a hybrid quantum-classical algorithm. The classical CNN structure remains almost the same except that instead of a classical convolution, a quantum convolution (quanvolution) is used.

The building blocks of a quanvolution layer [26, 27]:

1. Circuit input: encoding a small section of input image data into a quantum circuit.
2. Performing quantum computations on a learnable quantum circuit to find the hidden state.
3. Measuring the quantum system to obtain classical (decoded) data.
4. Repeating steps 1.–3. on remaining image sections to obtain a new feature map.

This procedure is illustrated in Figure 3.5.

The main advantage is that the quantum convolution potentially can produce very complex, classically unachievable kernels. What is more, the quanvolutions are error-resilient, and their architecture is suitable for quantum computers that have only a handful of qubits. Current research results [26, 27] proven that QCNNs can have almost the same learning performance as CNNs on the MNIST [28] handwritten digits classification dataset. However, the potential of obtaining a significant quantum advantage is still being investigated.

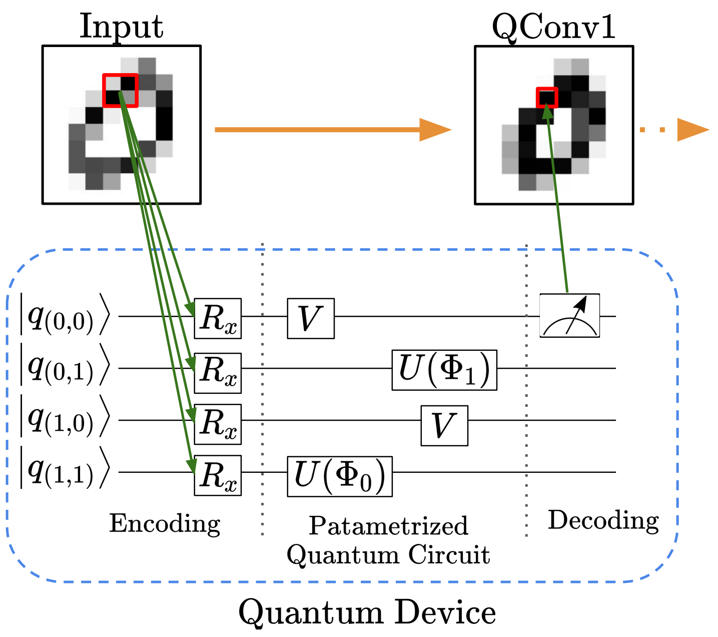


Figure 3.5: Quanvolutions encoding and decoding. Based on Ref. [27].

3.1.3 Quantum generative adversarial networks

Generative adversarial networks [29] (GANs) refer to models that can generate new data instances that look like original training data. Some of its most successful applications include creating visually-realistic images [30] or videos [31], and increasing image resolution [32]. GANs are composed of a generator and a discriminator. The generator's role is to generate synthetic data resembling the actual one, while the purpose of the discriminator is to distinguish between the fake and real data.

At first, the generator model — provided with a usually random, drawn from data distribution input — generates an artificial sample. Then, the discriminator model is fed with an example taken either from the genuine input or from the generator. After that, the discriminator classifies its input as either real or fake. Based on the discriminator's feedback, the generator gradually improves and produces more realistic samples. The training process usually consists of alternating phases of updating the discriminator and updating the generator. This procedure is repeated until the discriminator cannot differentiate between true and false inputs. As a result, the generator is successfully trained to imitate the original data [24].

Quantum Generative Adversarial Networks [33, 34] are a counterpart of their classical predecessor. The authors of [33] reviewed the main approaches to designing a Quantum GAN (QGAN) :

- Data: quantum, generator: quantum, discriminator: quantum.
  A fully quantum setting is as effective at learning to generate quantum data as a

classical GAN is at learning to generate classical data.

- Data: quantum, generator: classical, discriminator: quantum/classical. The classical generator might not be able to reproduce quantum data when a quantum supremacy system produces the data.
- Data: classical, generator: quantum, discriminator: quantum. In this setting, there is a possibility of observing a quantum advantage. A quantum processor requires only $log(N)$ qubits to represent a N-dimensional input vector, and $O(poly(log(N))$ time to perform operations. Consequently, QuGANs might be an alternative to GANs when the data is high-dimensional.

A closer inspection to implementing a fully quantum QGAN where both the generator and discriminator are trainable quantum circuits, and the data is quantum was presented in [34]. The authors introduced a method to obtain a quantum version of conditional GANs. As opposed to plain GANs, conditional GANs produce samples conditional on a class label λ (conditional distribution). It allows controlling the types of generated data. In the conditional QGAN, the generator is a parameterized quantum circuit that takes as input two quantum states: a label and noise. The purpose of noise is to provide entropy and control how the fluctuation of this parameter will affect the output. The generator is also a parameterized quantum circuit and it aims is to output a quantum state distinguishing whether the data is real or fake. Figure 3.6 depicts that a conditional QGAN is like a GAN except for the fact that it operates on quantum data (represented by Dirac notation $|\cdot\rangle$).

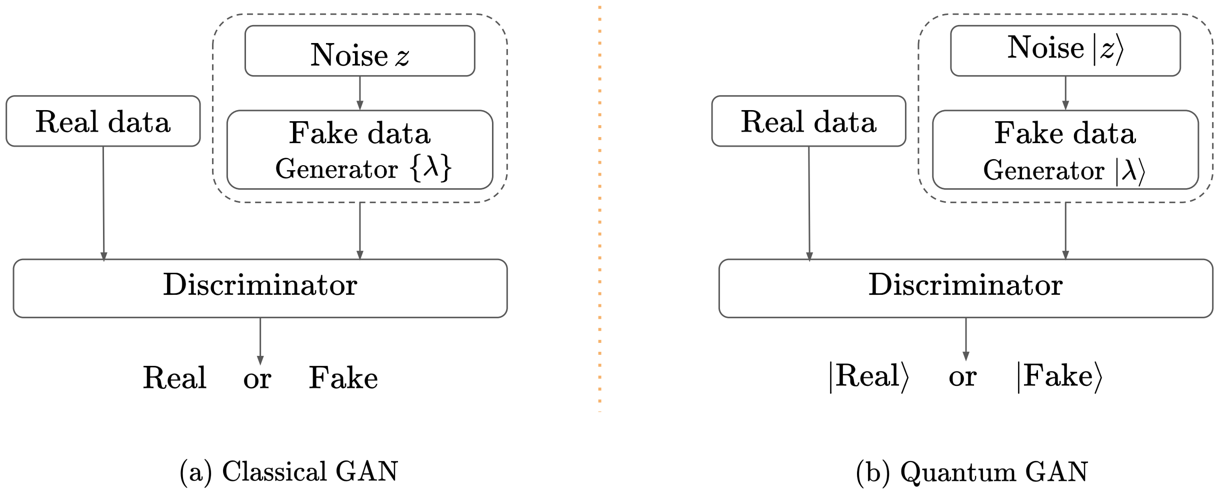


Figure 3.6: Design similarities between GANs and QGANs. Based on Ref. [34].

3.2 Software

Although the software enabling to develop quantum algorithms was introduced a few years ago, until recently, implementing quantum machine learning models was not an easy task. The challenge was to find a missing link between the design of quantum circuits and neural networks. Classical neural networks are usually trained using the backpropagation algorithm that is a particular case of automatic differentiation. Automatic differentiation refers to a set of techniques related to calculating derivatives of numeric expressions in programs [35]. The most popular classical machine learning libraries such as TensorFlow [36], PyTorch [37], and Theano [38] also use automatic differentiation as their backbone. Consequently, the desired internal design of a quantum machine learning library should also facilitate automatic differentiation that will enable the training of hybrid quantum-classical neural networks. Below are described the most current software tools for quantum machine learning except TensorFlow Quantum that is described in Chapter 4.

PennyLane [39] is a Python quantum machine learning library developed by Xanadu. It can perform optimization and machine learning tasks on quantum and hybrid quantum-classical hardware. This library aims at being a quantum counterpart of classical machine learning libraries such as TensorFlow or PyTorch.

A PennyLane program is composed of classical and quantum submodules called *nodes*. Each node can have tunable parameters — *variables* that are trained during learning or optimization. Intending to solve deep learning problems, PennyLane utilizes a differentiable programming paradigm to calculate gradients of quantum circuits through automatic differentiation.

What distinguishes this library is that apart from its own environment and simulator devices, it offers seamless integration with external quantum software and hardware via plugins. There are twelve officially supported plugins to software development kits/libraries incl. Google Cirq[1], IBM Q[2], Rigetti Forest[3], Amazon Braket[4], and IonQ[5]. By design, it should be compatible with any gate-based quantum computing backend. Additionally, PennyLane facilitates the co-design of quantum and classical nodes with the top machine learning libraries — TensorFlow, PyTorch, and JAX [40].

Qiskit [41] is a quantum software development kit written in Python. It consists of several components incl. Terra — the core of quantum programs, Ignis — tools for experiments, Aer — quantum computing simulators, and Aqua (depreciated) — application modules. Recently, Qiskit Aqua (Algorithms for QUantum computing Applications) has morphed into the following elements: chemistry, finance, optimization, and machine learning. Although Qiskit's machine learning module is in its early development, it provides a basic QML circuits library and some sample datasets. It also contains a predefined, universal interface that simplifies the creation of neural networks.

[1]`https://quantumai.google/cirq/`
[2]`https://quantum-computing.ibm.com/`
[3]`https://www.rigetti.com/`
[4]`https://aws.amazon.com/braket/`
[5]`https://ionq.com/`

For the purpose of performing QML tasks, Qiskit Gradients Framework is used to calculate derivatives with automatic differentiation. Additionally, Qiskit integrates with PyTorch to facilitate creating hybrid quantum-classical models.

QuantumFlow [42] is a Python framework that offers a co-design of neural networks and quantum circuits. Its power comes from providing a lot of out of the box components. First of all, this framework has a few predefined layers that simplify composing the neural computational model. Additionally, it supports the automated creation of quantum circuits for defined quantum neural networks models. What is more, it provides a training environment that enables forward and backward propagation. It is possible to integrate PyTorch for the network's training.

Although this framework is not widespread, the proof of concept demonstrated a network designed to classify the MNIST dataset that has very high accuracy (94.09%) and reduces the cost 10.85 times compared to classical computer performance.

Yao [43] is a framework for developing quantum algorithms written in Julia. Its functionality is based on a quantum block intermediate representation (QBIR). The QBIR defines a quantum program using a domain-specific abstract-syntax tree, and as a result, provides a hardware-agnostic generalization of quantum circuits. By design, it supports differentiable programming through a built-in engine. This framework efficiently implements reverse and forward modes of automatic differentiation. It is also possible to integrate it with an external automatic differentiation engine (e.g., Zygote [44]), but its internal mechanisms are proven to be more efficient for very deep circuits. Additionally, Yao uses batched quantum registers to store quantum states processed by quantum circuits (possible parallelization on CUDA devices). This feature combined with the QBIR enables efficient training of quantum machine learning models.

What differentiates this framework is the programming language it uses. The authors chose Julia for many reasons:

- it is easy to write generic programs due to the dynamic type system and multiple dispatch mechanism;
- it supports metaprogramming;
- it offers good integration with other programming languages;
- it enables high performance programming.

To summarize, a comparison of all the above-mentioned software solutions for QML is presented in Table 3.1.

Table 3.1: Quantum Machine Learning libraries.

SDK/Library	PennyLane	Qiskit	QuantumFlow	Yao
Institution	Xanadu	IBM	University of Notre Dame	QuantumBFS
Language	Python	Python	Python	Julia
Integrated external ML tools	TensorFlow	TensorFlow, PyTorch	PyTorch	Zygote
Possible quantum hardvare integration?	Yes	Yes	Yes	Yes
Open Source?	Yes	Yes	Yes	Yes
Implements Automatic Differentiation?	Yes	Yes	Yes	Yes

3.3 Summary

Quantum machine learning has the potential of becoming one of the first near–term applications that can provide a quantum advantage. For NISQ era devices, the most promising approach is building hybrid quantum-classical machine learning models since they allow combining the power of both quantum and classical approaches and benefiting from the extended hardware resources.

Although this chapter has focused on a few QML libraries, an overview of the framework being the main subject of this thesis is presented in the next chapter.

Chapter 4

Overview and Assessment of TensorFlow Quantum

This chapter introduces TensorFlow Quantum — an open-source quantum machine learning framework developed by Google. The first part of this chapter focuses on the design of this framework, while the second contains its assessment.

4.1 Design

To enable hybrid quantum-classical machine learning, TensorFlow Quantum [9] (TFQ) bridges the capabilities of two other Google products: Tensorflow [36] (TF) — one of the top machine learning libraries for classical data and Cirq [45] — a quantum programming framework. As a result, TFQ enables the rapid creation of machine learning models for both classical and quantum data.

Architecture. The TFQ architecture by design enables interleaving interactions between components of TensorFlow Quantum with components of TensorFlow and Cirq, cf. Figure 4.1. At the top of the software stack is data. TensorFlow is responsible for handling classical data, and TensorFlow Quantum adds the functionality of processing quantum data. The following two layers represent the ability to compose neural network models (Keras models) consisting of classical and quantum layers. TF Keras can manage automatic differentiation, but it also needs the TFQ module to perform automatic differentiation on quantum data. The next layer incorporates TF and TFQ Operations that enable conducting computations on Tensors (multidimensional arrays). Tensors are the core computational elements of TF, so it was crucial for TFQ to provide the functionality of expressing quantum circuits and Hamiltonian operators as tensors. The components of the next layer denote that the execution of quantum parts of the model can be simulated either using the Cirq's simulator or simulator provided by TFQ – qsim. Additionally, the bottom layer representing hardware indicates the possibility of circuit execution on an actual quantum processing unit (QPU).

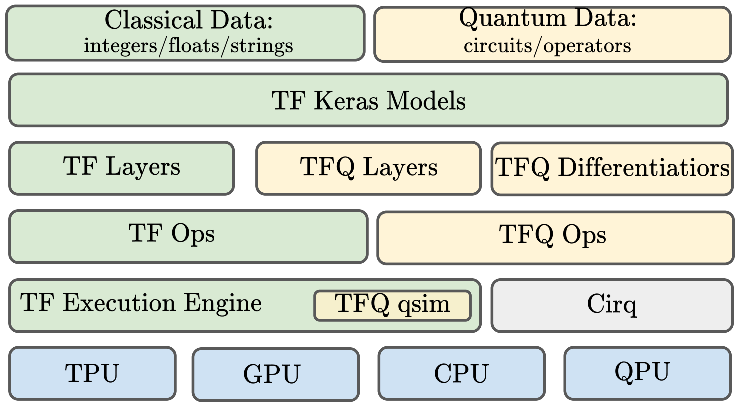


Figure 4.1: TensorFlow Quantum's software and hardware stack. The green boxes depict elements from TensorFlow, the yellow boxes represent TFQ elements, and the blue boxes symbolize hardware. Framework Cirq (the gray box) is mainly responsible for preparing quantum circuits. Based on Ref. [9].

Hybrid quantum-classical neural network model. Quantum neural networks can take as input classical or quantum data. When it comes to the classical data, it can be, e.g., imported from some external source. However, with the current state of quantum technology, quantum data should be directly generated with the use of quantum circuits.

Figure 4.2 presents an example connection between two layers of a hybrid neural network. The values of neurons in the *Layer i* are obtained after the measurement of the quantum circuit parameterized with some values Φ. In classical neural networks, the trainable parameters are weights and biases (θ). However, in quantum-classical neural networks, trainable elements also include parameter values of the parameterized quantum circuit.

One of the crucial steps in training the hybrid quantum-classical neural network is evaluating the cost function ($L_{\theta,\Phi}$). Having calculated the cost function its gradients for both classical and quantum parameters can be determined (assuming that a gradient optimization method is used to update the parameters). Then, the parameters can be adequately updated.

In classical neural networks, the loss function measures the model performance; the lower the cost function, the more accurate the model's outcomes. When using a gradient-based optimization method, the cost function gradients enable determining the direction of tuning the network's parameters.

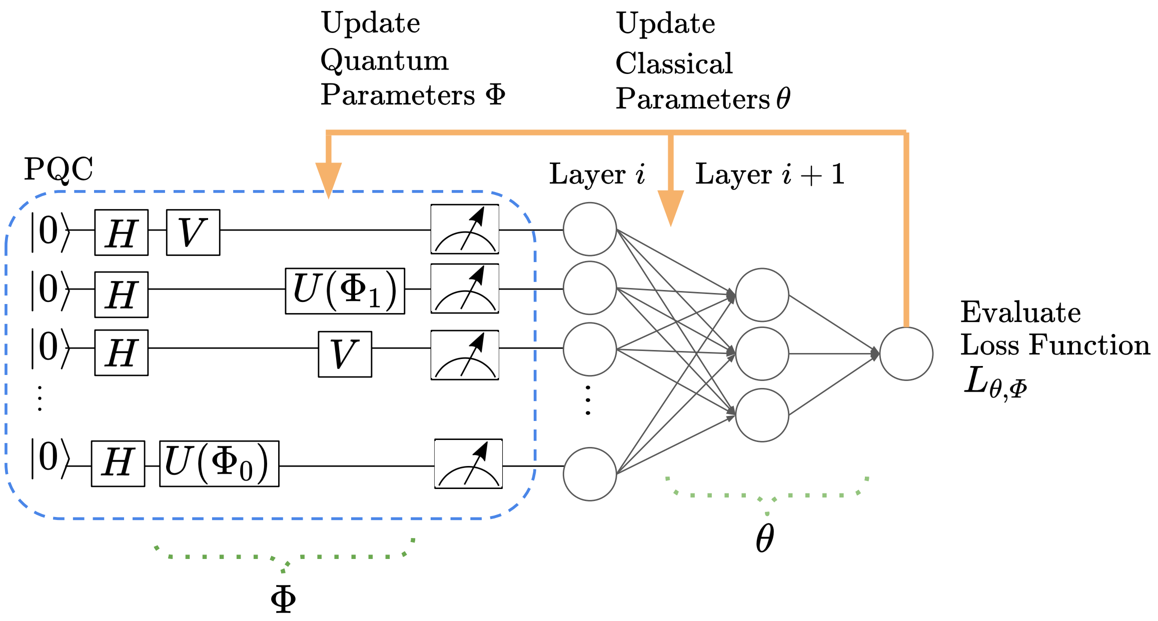

Update Quantum Parameters Φ
Update Classical Parameters θ
PQC
Layer i
Layer $i+1$
$|0\rangle$
H
V
$U(\Phi_1)$
$U(\Phi_0)$
Evaluate Loss Function $L_{\theta,\Phi}$
θ
Φ

Figure 4.2: Quantum data generated by the PQC is extracted through measurement and then post-processed by the neural network layers. Based on Ref. [9].

In the quantum case, an expectation value of the parameterized quantum circuit can be interpreted as the loss function since it directly depends on the circuit parameters values (cf. Eq. 2.3.18). The expectation value of an observable is an average value of all the results obtained from many repetitions of measuring this observable (on duplicates of the initially prepared quantum system). TFQ defines a few differentiation methods, one of the most frequently used is the parameter shift rule. Under certain assumptions, the parameter shift rule states that the derivative of the expectation value $E(\Phi)$ with regard to the gate parameter Φ can be expressed as [46]

$$\frac{d}{d\Phi}E(\Phi) = r[E(\Phi + \frac{\pi}{4r}) - E(\Phi - \frac{\pi}{4r})], \tag{4.1.1}$$

where r is the shift coefficient. One of the most significant advantages of this method is that the same form of the quantum circuit can be used to calculate the expectation function value and its gradient.

4.2 Evaluation

TensorFlow Quantum is a multi-purpose quantum machine learning library. There are many potential areas of applications, including, but not limited to, scientific research, benchmarking, exploration of new concepts, and development of personal projects. Below are discussed the most important aspects and features of this library.

Integration with TensorFlow. TFQ flawlessly integrates with TF — both high-level Keras Application Programming Interface (API) and TFQ API (cf. Table 4.1) can be utilized. As a result, creating complex hybrid neural network models with alternating quantum and classical layers is very fast.

One of the most important features is that TFQ has methods that convert quantum data into TF tensors representation. Another remarkable aspect is that the outputs of quantum measurements can be directly supplied to a classical network. Moreover, the automatic differentiation of hybrid quantum-classical computations is supported and the same classical optimizers can be used to optimize quantum and classical parameters.

Table 4.1: TensorFlow Quantum modules.

Module name	**Description**
Tfq	General-purpose methods.
Datasets	Location for quantum circuits that can be used as datasets.
Differentiators	Implementations of hybrid quantum-classical automatic differentiation schemes.
Layers	Definitions of quantum layers.
Math	Supplementary mathematical operations.
Noise	Methods for the purpose of noisy simulation.
Optimizers	Structure optimizations of quantum circuits.
Util	Module for helper functions.

Access to quantum devices and simulators. TFQ offers a very efficient quantum simulator – qsim. Qsim is much faster than Cirq's quantum simulator, but either of them can be used as a backend to simulate quantum circuits on classical computers.

The TFQ's design (cf. Fig. 4.1) enables the possibility of using quantum hardware as the backend. Fantastic though the option of using QPU might seem, currently Google restricts access to its quantum hardware and hardware from two other companies (AQT and Pasqal) only to selected partners. However, since June 2021, it is possible to access an 11-qubit quantum computer created by the company IonQ via a paid subscription on Google Cloud.

Nevertheless, taking into account that quantum simulations on classical computers have exponential overhead, only algorithms with a limited number of qubits can be tested. From this perspective, using IBM's Qiskit or PennyLane with Qiskit plugin might be a better idea since IBM Quantum Services offer a few free, publicly available quantum systems.

Ease of use is a significant factor that has a notable influence on the library's success. The features that affect the programming experience in TFQ are presented in Table 4.2.

Table 4.2: Aspects of TFQ that affect the software development experience.

Aspect	**Overview**
Installation	TFQ can be installed using Python's package manager pip or built from the source. Additionally, this library can be used interactively in a browser via Google Colab or Jupyter Notebooks.
Language	TFQ is a Python library, so it eliminates the necessity of learning a domain-specific programming language.
Documentation	TFQ has comprehensive documentation full of helpful code snippets. This library is open-source, so it is easy to explore the codebase.
Community and support	There is no community dedicated strictly to TFQ. There exist a very active TF forum, and the questions related to TFQ can be asked there. Also, issues such as bugs or new ideas can be reported on GitHub, where the code is stored.

Applications. TFQ provides minimal working implementations for many cutting-edge hybrid quantum-classical algorithms. Selected examples of algorithms that were implemented with TFQ are overviewed in Table 4.3.

Table 4.3: Algorithms implemented in TFQ.

Algorithm	**Description**	**Inspired by**
MaxCut QAOA	Example of optimizing variational parameters of the **QAOA** for the maximum cut problem.	[8]
Quantum Convolutional Neural Network	Implementation of a quantum version of the **convolutional neural network**.	[25]
Meta-Learning for QAOA	Use of **recurrent neural networks** to find initialization parameters for the QAOA.	[47]
Binary classification of quantum states	**Classifying elements** derived from two different quantum data sources.	[48]
MNIST classification	Using a hybrid quantum-classical neural network to **classify classical data**.	[49]
Parameterized Quantum Circuits for Reinforcement Learning	Implementation of **quantum reinforcement learning** algorithm for the cart-pole balancing problem [50].	[51], [52]
Entangling Quantum Generative Adversarial Networks	Implementation of a new architecture for **generative adversarial networks**.	[53]
Barren plateaus	Examining the problem of barren plateaus which refers to the situation when the gradient of quantum neural network becomes extremely small during the training of quantum neural network.	[54]
Layerwise learning for quantum neural networks	Investigating the results of the **layerwise learning** technique for improving the results of training quantum neural networks and avoiding barren plateaus.	[15]

Considering the collection of available implementations of research papers, as of June 2021, TFQ has a much larger amount of tutorials than Qiskit ML[1], and similar number of tutorials to PennyLane[2].

4.3 Detailed assessment plan

Based on the above analysis of the TFQ features, we propose a plan for evaluating this library. In order to assess the actual possibilities of TensorFlow Quantum, the aim of investigating the implementation of the Quantum Approximate Algorithm was chosen. First of all, the QAOA is considered one of the most prospective quantum algorithms for near-term quantum devices [55]. It is believed that the advancements provided with this algorithm can lead to achieving quantum supremacy [56], so it is beneficial to evaluate whether TFQ is suitable for implementing this algorithm. Secondly, there are out-of-the-box methods that help with the construction of the QAOA developed in other frameworks, e.g., Qiskit[3] (albeit they do not utilize quantum machine learning features) and PennyLane[4], but there are not any in Google Cirq or TFQ.

What is worth noting is that Cirq provides a tutorial on implementing the QAOA[5]. Nevertheless, this implementation lacks the use of a more advanced classical optimizer (optimizers are not natively included in Cirq) — it just performs a grid search over specified parameter values. Similarly, the existing implementation of the QAOA in TFQ[6] introduces only a basic quantum neural network that cannot solve complex problems. For this reason, Chapter 5 describes how to express the combinatorial optimization problem of the Traveling Salesman to solve it with the QAOA, and Chapter 6 provides an investigation of the feasibility of performing the QAOA in TFQ. The Traveling Salesman Problem was chosen because this problem is often used as a benchmark for optimization methods [57].

4.4 Summary

TensorFlow Quantum is a hybrid quantum-classical machine learning library that enables the creation of models composed of both quantum and classical components. This chapter provided a discussion of this library in terms of its design and core features. Additionally, a detailed plan of assessing TFQ was described, and the following chapters are dedicated to its realization.

[1]`https://qiskit.org/documentation/machine-learning/`

[2]`https://pennylane.ai/qml/demos_research.html`

[3]`https://qiskit.org/documentation/stubs/qiskit.aqua.algorithms.QAOA.html`

[4]`https://pennylane.readthedocs.io/en/stable/code/qml_qaoa.html`

[5]`https://quantumai.google/cirq/tutorials/qaoa`

[6]`https://github.com/tensorflow/quantum/tree/research/qaoa`

Chapter 5

Traveling Salesman Problem

This first part of this chapter presents the definition of the Traveling Salesman Problem — one of the NP-hard combinatorial optimization problems. The second part of this chapter describes how to create a parameterized quantum circuit that will be utilized as one of the layers in the hybrid quantum-classical neural network model used for experiments in Chapter 6.

5.1 Problem formulation

For the purpose of presenting a solution based on the QAOA, an NP-hard combinatorial optimization traveling salesman problem (TSP) was chosen. The TSP is defined as follows ”find a path through a weighted graph that starts and ends at the same vertex, includes every other vertex exactly once, and minimizes the total cost of edges” [58]. To solve this problem with the QAOA, it is necessary to encode it into a Hamiltonian. However, before the problem can be presented in the form of the Hamiltonian, the intermediate, auxiliary step is to express it in the form of QUBO (cf. Sec. 2.3 and Eq. 2.3.12 – 2.3.13).

Assume that (cf. Eq. 2.3.1 – 2.3.4)

- N — cities amount;
- $X = [x_{i,t}]_{NxN}$, — Boolean matrix, where

$$x_{i,t} = \begin{cases} 1, & \text{if the salesman visits the city } i \text{ at the timestamp } t, \\ 0, & \text{otherwise;} \end{cases}$$

- $D = [d_{i,j}]_{NxN}$ — symmetric matrix of distances between cities, if $i = j$ then $d_{i,j} = 0$.

1. The cost function B expresses the sum of costs for each taken path. The number of cities is equal to the number of timestamps.

$$B = \sum_{\substack{i,j=0\\ i\neq j}}^{N-1} d_{i,j} \sum_{t=0}^{N-1} x_{i,t}x_{j,t+1}. \tag{5.1.1}$$

2. Constraints A_1 and A_2 ensure the solution's correctness. If the solution is incorrect a suitable penalty is added.
   (a) Each city i should be visited exactly once.

$$A_1 = \sum_{i=0}^{N-1}(1 - \sum_{t=0}^{N-1} x_{i,t})^2 \tag{5.1.2}$$

   (b) At each timestamp t the salesman should be in exactly one city.

$$A_2 = \sum_{t=0}^{N-1}(1 - \sum_{i=0}^{N-1} x_{i,t})^2 \tag{5.1.3}$$

The task is to minimize the weighted sum [59]

$$\begin{aligned} QUBO_C &= a \cdot A_1 + a \cdot A_2 + b \cdot B \\ &= a\sum_{i=0}^{N-1}(1 - \sum_{t=0}^{N-1} x_{i,t})^2 + a\sum_{t=0}^{N-1}(1 - \sum_{i=0}^{N-1} x_{i,t})^2 + b\sum_{\substack{i,j=0\\ i\neq j}}^{N-1} d_{i,j} \sum_{t=0}^{N-1} x_{i,t}x_{j,t+1}, \end{aligned} \tag{5.1.4}$$

where $0 < b \cdot \max_{i\neq j}(d_{i,j}) < a$.
The weights a, b enable balancing the importance of particular components. Also, to simplify operations on indices, it is assumed that if $t = N-1$ then $t+1 \to 0$.

To translate the $QUBO_C$ into the Hamiltonian it is necessary to express the city indices by a single number instead of a pair of numbers, and substitute all $x_{i,j}$ values according to Property 2.3.11

$$x_{i,t} \longleftrightarrow \frac{I - Z_k}{2}, \tag{5.1.5}$$

where $k = i + t \cdot N$ is the qubit index.
Additionally,

$$1 \longleftrightarrow I. \tag{5.1.6}$$

Therefore, the Hamiltonian requires N^2 qubits and has the following form

$$\begin{aligned} H_C = a\sum_{i=0}^{N-1}\left(I - \sum_{t=0}^{N-1}\frac{I - Z_k}{2}\right)^2 + a\sum_{t=0}^{N-1}\left(I - \sum_{i=0}^{N-1}\frac{I - Z_k}{2}\right)^2 + \\ + b\sum_{\substack{i,j=0\\ i\neq j}}^{N-1} d_{i,j}\sum_{t=0}^{N-1}\frac{I - Z_k}{2}\cdot\frac{I - Z_{k'}}{2}. \end{aligned} \tag{5.1.7}$$

5.2 Parameterized quantum circuit

With the aim of understanding how a parameterized quantum circuit is generated, below is presented an example of translating the TSP Hamiltonian into a circuit. To make it concise, a trivial instance of the problem is taken — the case of only two cities. Nevertheless, the method for translating cases of larger problems remains analogous.

Based on Equations 2.3.15, 2.3.16 that define the layers of the QAOA circuit ansatz, it is essential to evaluate the expressions $U_C(\gamma)$ and $U_M(\gamma)$ for the proper Hamiltonians. At first, we will focus on obtaining $U_C(\gamma)$. The cost Hamiltonian expression for the case of two cities (considering the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$) is derived in Appendix A (Sec. A.1), and its final form is

$$\begin{aligned} H_C = {} & (2a + b \cdot d_{0,1})I + \\ & + a\left(\frac{Z_0Z_2}{2} + \frac{Z_1Z_3}{2} + \frac{Z_0Z_1}{2} + \frac{Z_2Z_3}{2}\right) + \\ & + b \cdot d_{0,1}\left(-\frac{Z_0}{2} - \frac{Z_1}{2} - \frac{Z_2}{2} - \frac{Z_3}{2} + \frac{Z_0Z_3}{2} + \frac{Z_1Z_2}{2}\right). \end{aligned} \tag{5.2.1}$$

The parameters a, b correspond to the weights from Equation 5.1.4 and $d_{0,1} = d_{1,0}$ is the distance between the cities.

Based on the Hamiltonian H_C, $U_C(\gamma)$ can be expressed as an expression of commutable terms

$$\begin{aligned} U_C(\gamma) &= e^{-i\gamma H_C} \\ &= exp\Big(-i\gamma\left((2a + b \cdot d_{0,1})I + a\frac{Z_0Z_2}{2} + \cdots + b \cdot d_{0,1}\frac{-Z_0}{2} + \cdots + b \cdot d_{0,1}\frac{Z_1Z_2}{2}\right)\Big) \\ &= exp(-i\gamma(2a + b \cdot d_{0,1})I) \cdot exp\Big(-i\gamma a\frac{Z_0Z_2}{2}\Big) \cdot \ldots \cdot exp\Big(-i\gamma b d_{0,1}\frac{-Z_0}{2}\Big) \cdot \\ &\quad \cdot exp\Big(-i\gamma b d_{0,1}\frac{Z_1Z_2}{2}\Big). \end{aligned} \tag{5.2.2}$$

Dropping the coefficients and qubit indices, it can be observed that $U_C(\gamma)$ consists of three types of components. Each of them can be translated into a set o basic quantum gates acting on proper qubits.

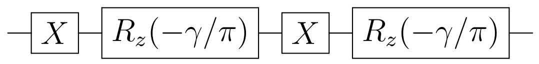
The first component $e^{-i\gamma I}$ can be decomposed as follows

$$\begin{aligned} e^{-i\gamma I} &= exp\left[-i\gamma\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right] \\ &= e^{-i\gamma}I \end{aligned} \tag{5.2.3}$$

and omitting the global phase its circuit equivalent is

$$e^{-i\gamma I} \equiv -\boxed{I}- \quad = —— \quad .$$

However, the automatic decomposition available in TensorFlow Quantum results in:

$e^{-i\gamma I} \equiv$ 

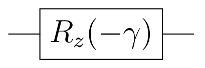
always acting on the first qubit. This decomposition seems to not be necessary since is equal to performing the Identity gate

$$\begin{aligned} R_z\left(\frac{-\gamma}{\pi}\right) X R_z\left(\frac{-\gamma}{\pi}\right) X &= \begin{bmatrix} e^{-i\frac{\gamma}{2\pi}} & 0 \\ 0 & e^{i\frac{\gamma}{2\pi}} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e^{-i\frac{\gamma}{2\pi}} & 0 \\ 0 & e^{i\frac{\gamma}{2\pi}} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I. \end{aligned} \tag{5.2.4}$$

The second component $e^{-i\gamma\frac{-Z}{2}}$ can be expressed as

$$\begin{aligned} e^{-i\gamma\frac{-Z}{2}} &= exp\left[i\frac{\gamma}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right] \\ &= \begin{bmatrix} e^{i\frac{\gamma}{2}} & 0 \\ 0 & e^{-i\frac{\gamma}{2}} \end{bmatrix} \\ &= R_z(-\gamma) \end{aligned} \tag{5.2.5}$$

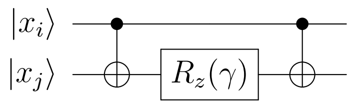
and its circuit representation is

$e^{-i\gamma\frac{-Z}{2}} \equiv$  .

The third component $e^{-i\gamma\frac{ZZ}{2}}$ can be represented as

$$\begin{aligned} e^{-i\gamma\frac{ZZ}{2}} &= exp\left[-i\frac{\gamma}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right] \\ &= exp\left[-i\frac{\gamma}{2}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\right] \\ &= \begin{bmatrix} e^{-i\frac{\gamma}{2}} & 0 & 0 & 0 \\ 0 & e^{i\frac{\gamma}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{\gamma}{2}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{\gamma}{2}} \end{bmatrix} \\ &= CNOT \cdot (I \otimes R_z(\gamma)) \cdot CNOT \end{aligned} \tag{5.2.6}$$

and its circuit equivalent is (cf. Sec. A.2.1)

$e^{-i\gamma\frac{Z_iZ_j}{2}} \equiv$ 

Additionally, the standard mixing Hamiltonian $U_M(\gamma)$ is Rotation-X gate

$$U_M(\gamma) = e^{-i\gamma X} = R_x(2\gamma) \tag{5.2.7}$$

with the circuit representation

$$e^{-i\gamma X} \quad \equiv \quad -\boxed{R_x(2\gamma)}- \quad .$$

In TensorFlow Quantum the mixing Hamiltonian is decomposed into an equivalent set of gates (cf. A.2.3)

$$e^{-i\gamma X} \quad \equiv \quad -\boxed{H}-\boxed{R_z(2\gamma)}-\boxed{H}- \quad .$$

Based on all of the above derivations of $U_C(\gamma)$ and $U_M(\beta)$, Figure 5.1 presents the QAOA circuit ansatz for the Traveling Salesman Problem.

5.3 Summary

This chapter presented how to translate the QUBO definition of the Traveling Salesman Problem into the Hamiltonian form and then into the parameterized quantum circuit that can be used as the Quantum Approximate Optimization Algorithm ansatz. In Chapter 6, the prepared parameterized quantum circuit will be used for creating the core layer for the hybrid quantum-classical neural network model.

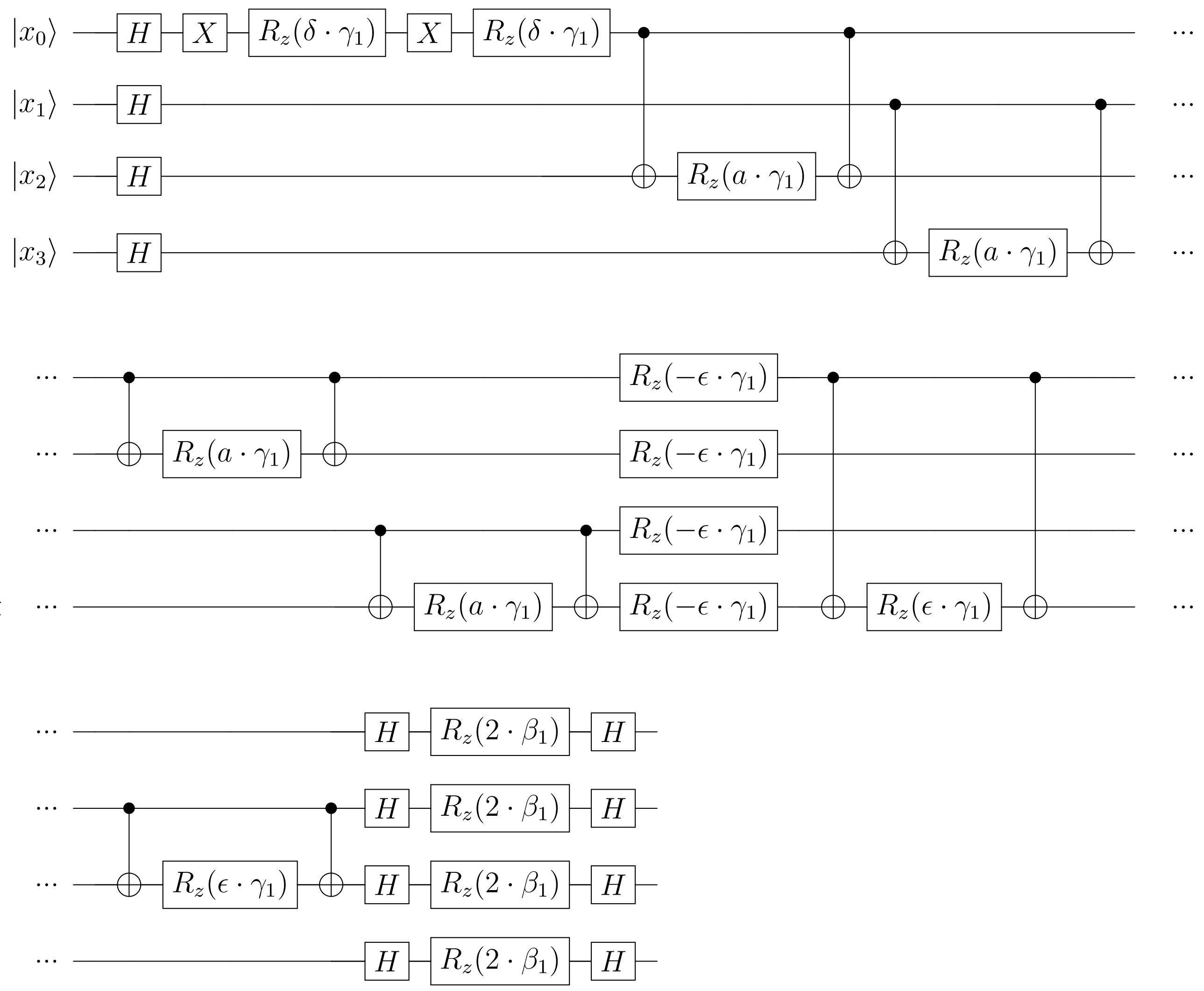


Figure 5.1: The parametrized quantum circuit for solving the TSP with two cities using the QAOA. The circuit ansatz was created based on on Eq. 5.2.2 and Eq. 5.2.7. The coefficient δ is equal to $-(2a + b \cdot d_{0,1})/\pi$ and ϵ is $b \cdot d_{0,1}$.

Chapter 6

Experiments

In this chapter, two experiments based on a hybrid quantum-classical neural network are presented. The first experiment examined the possibility of minimizing the expectation value for a single Traveling Salesman Problem instance. The second experiment investigated the generalization abilities of a hybrid neural network. The experiments were executed on the supercomputer Prometheus[1] (via access provided by the Academic Computer Centre Cyfronet AGH).

6.1 Optimization task

This experiment was aimed at obtaining correct solutions for a single instance of the Traveling Salesman Problem using the Quantum Approximate Optimization Algorithm. The main task of the QAOA is finding optimal values of the variational parameters (angles $\gamma_1, \ldots, \gamma_p$ and $\beta_1, \ldots, \beta_p$) that minimize the expectation value of the problem's Hamiltonian, cf. Equation 5.1.7. The minimal working example of the QAOA implementation provided by the Google team (cf. Tab. 4.3) was proposed for an easier problem — maximum cut and was not powerful enough to solve the TSP (unless nearly optimal values of variational parameters are provided as initial values, but it was almost unfeasible to guess them). The suggested neural network model consisted of a single Parameterized Quantum Circuit Layer and, as a result, had only $2p$ trainable parameters (p — the number of layers in the QAOA ansatz), which turned out not enough for optimizing more complex problems.

To solve the TSP with the QAOA, we proposed a hybrid quantum-classical neural network model that minimized the expectation value of the problem's Hamiltonian by optimizing the parameters of the TSP's parameterized quantum circuit, cf. Eq. 2.3.19 – 2.3.20. Below are presented steps needed to achieve this task.

[1]`https://www.cyfronet.pl/en/computers/15226,artykul,prometheus.html`

6.1.1 Data preparation and encoding

The first step was to create a single instance of the TSP problem. Since the algorithm was executed on a quantum simulator, its complexity was exponential ($O(2^{N^2})$). For this reason, a feasible example to solve consisted of four cities ($N = 4$). The matrix of distances between cities was determined as the Euclidean distance between the city coordinates. The distance value was used as a coefficient in the Hamiltonian thus had a significant impact on the expectation value. From the perspective of optimizing the expectation value, it was beneficial to limit the range of distances between cities. The normalization prevented having very small values of distances (a case of close cities). What could have been normalized were either the city coordinates or distances between the cities. The option of normalizing the city coordinates was selected. Assuming that $x_{\max}$ — maximum value of the x coordinate from all the selected city coordinates (x, y) $y_{\max}$ — maximum value of the y coordinate from all the selected city coordinates (x, y), the normalized coordinates were

$$(x', y') = (x/x_{\max}, y/y_{\max}). \tag{6.1.1}$$

Table 6.1 contains four pairs of randomly chosen city coordinates $(x, y), 0 \leq x, y \leq 10000$ and its normalized coordinates.

Table 6.1: Coordinates of the four cities randomly selected for the TSP.

City identifier	**City coordinates** (x, y)	**Normalized city coordinates** (x', y')
0	(8105, 6018)	(1.0000 , 0.6053)
1	(2151, 3824)	(0.2654, 0.3846)
2	(359, 5493)	(0.0443, 0.5525)
3	(1935, 9942)	(0.2387, 1.0000)

This normalization enabled to have the matrix of distances with different values in range $(0, \sqrt{2})$.

After performing the city coordinates normalization, the distance matrix was created. The elements of Table 6.2 were used for calculating the cost Hamiltonian.

Table 6.2: Distances between normalized city coordinates, where i, j are city identifiers.

$i \backslash j$	0	1	2	3
0	0.0	0.7670	0.9572	0.8575
1	0.7670	0.0	0.2776	0.6159
2	0.9572	0.2776	0.0	0.4879
3	0.8575	0.6159	0.4879	0.0

To convert the integer representation of the solution to binary representation, one-hot encoding of the selected city in each timestamp was used. For example, one-hot encoding for the cycle $2 \to 3 \to 0 \to 1 \to 2$ was as follows

0 0 1 0	0 0 0 1	1 0 0 0	0 1 0 0
timestamp 0	timestamp 1	timestamp 2	timestamp 3

Table 6.3 presents mapping of timestamp, city identifiers to qubit indices in accordance with Property 5.1.5.

Table 6.3: Translation of timestamp-city index to qubit index.

timestamp	0				1				2				3			
city	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
qubit index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

6.1.2 Quantum neural network

To create the parameterized quantum circuit that was used in the hybrid quantum-classical neural network model, it was necessary to construct the cost and mixer Hamiltonians (cf. Sec. 5.2). One of the challenges was determining the cost Hamiltonian's coefficients a and b (Eq. 5.1.7). The condition for Eq. 5.1.4 states that $0 < b \cdot \max_{i \neq j}(d_{i,j}) < a$ (cf. Table 6.2). Assuming that $b = 1.0$ was fixed, then $b \cdot \max_{i \neq j}(d_{i,j}) \approx 0.9571 \Rightarrow 0.9571 < a$. Considering only the correct solutions, the upper bound of the cost function value B (Eq. 5.1.1) was $2 + 2\sqrt{2} \approx 4.82$ (the maximum value of the cycle length between four cities selected from the square of length 1) . As a result, we assumed that $0.9571 < a \leq 4.82$. To make a reasonable choice of the parameter a, an external tool [60] for calculating the eigenvalues of the cost Hamiltonian was used. The aim of using this tool was to calculate the energies of all the possible solution states for the specified matrix of city distances. Using the grid search method it was determined that $a = 4$ provides a significant gap between energies of correct and incorrect solution states. When it comes to the mixing Hamiltonian, the simplest version that consists of a single Pauli-X gate applied on each qubit was used (cf. Eq. 2.3.14).

Having determined the cost Hamiltonian H_C and the mixing Hamiltonian H_M, the operators $U_C(\gamma)$ and $U_M(\beta)$ (cf. Eq. 2.3.15, 2.3.16) were automatically translated into the parameterized quantum circuit by TFQ. The obtained circuit had to be prepended with the layer of the Hadamard gates to initialize the superpositions of all computational basis states. After all these operations, this circuit was used as the QAOA ansatz, cf. Figure 2.6.

When it comes to the neural network model, we proposed a feedforward quantum neural network (cf. Sec. 3.1.3) that consisted of an input layer with $2p$ neurons, a hidden layer with $2p$ neurons, and an expectation layer, cf. Figure 6.1. The design of this network was based on the fact that the input (initial values of the angles $\gamma_1, \beta_1, \ldots, \gamma_p, \beta_p$) had $2p$ parameters and the expectation layer also needed to receive $2p$ parameters. The intermediate hidden layer extended the number of trainable parameters thus the capacity of this network was increased.

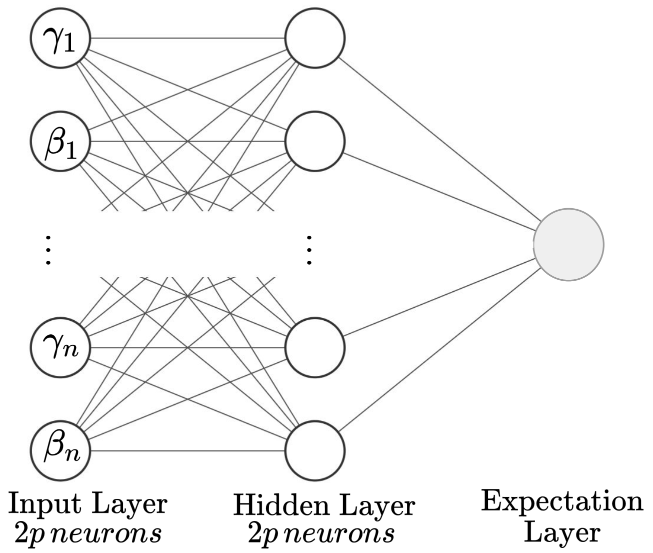
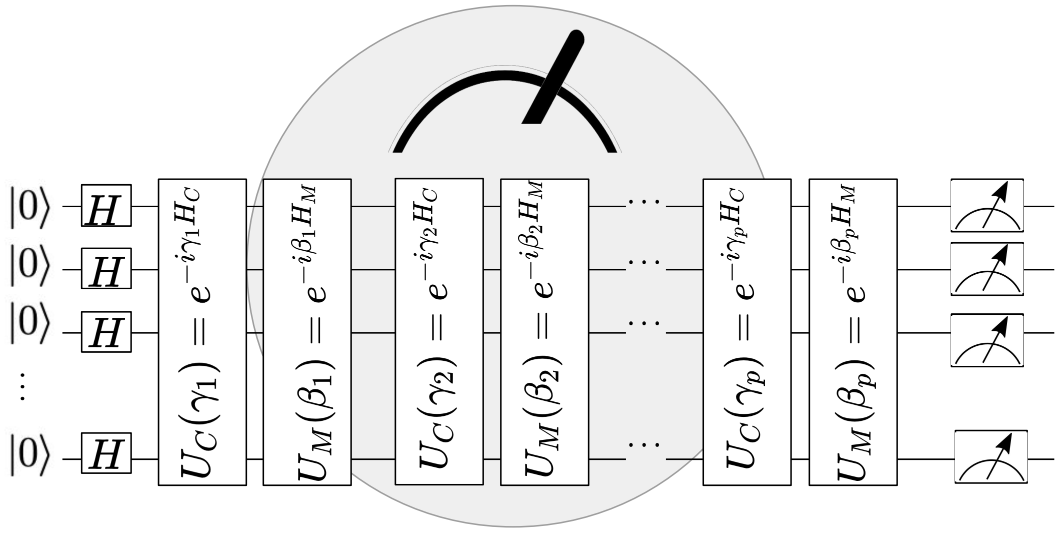


Figure 6.1: Feedforward neural network for optimizing the QAOA's input parameters.

The expectation layer is a TFQ layer that outputs classical information based on the measurement of a quantum state. This layer as input receives a parameterized quantum circuit, the problem's Hamiltonian, and trainable values of the circuit's parameters. Supplied with this data, it then prepares a quantum state and outputs its expectation value. Figure 6.2 visualizes the components of an expectation layer.



$|0\rangle$
H
$U_C(\gamma_1) = e^{-i\gamma_1 H_C}$
$U_M(\beta_1) = e^{-i\beta_1 H_M}$
$U_C(\gamma_2) = e^{-i\gamma_2 H_C}$
$U_M(\beta_2) = e^{-i\beta_2 H_M}$
$\cdots$
$U_C(\gamma_p) = e^{-i\gamma_p H_C}$
$U_M(\beta_p) = e^{-i\beta_p H_M}$

Figure 6.2: Expectation layer with a parameterized quantum circuit.

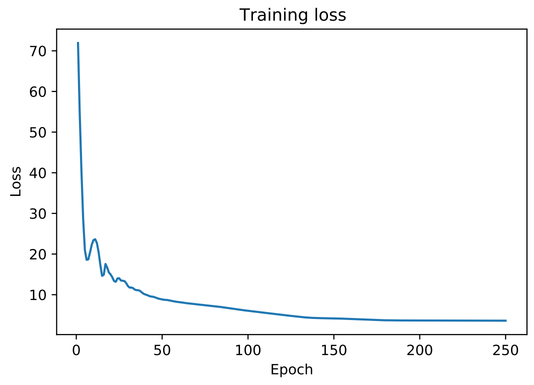
The power of the QNN optimization feature lies in the choice of the optimization routine. We chose a very efficient, gradient-based optimization algorithm — Adam [61] with the learning rate value 0.01. The role of the optimizer is to minimize the loss value (output of the neural network), which in the case of our quantum neural network was the expectation value.

The initial values (γ, β) should be chosen from the following ranges $\gamma \in [0, 2\pi]^p, \beta \in [0, \pi]^p$. Using the trial-and-error method, it was determined that a good guess of initial values is zeros.

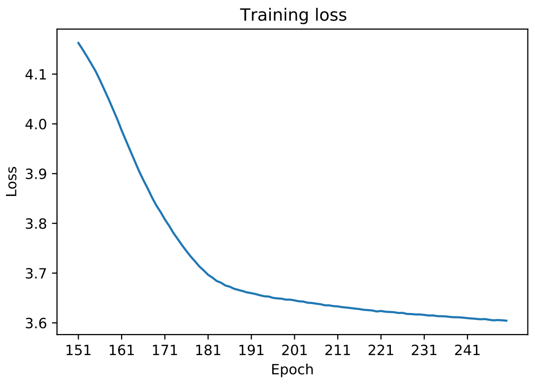
6.1.3 Results

What we wanted to observe was the impact of changing the value of p on the expectation value and the number of correct solutions for a single instance of the TSP problem. It is presumed that with the increase of the value of p, the expectation value gets closer to the optimal value [56]. For this purpose, we conducted experiments with the value of $p \in \{1, \dots, 10\}$.

Based on the observation of the training process, it was determined that it is sufficient to execute the optimization algorithm 250 times (the optimization process consisted of 250 epochs). Figure 6.3 presents the changes of loss/expectation value for the case of the QAOA circuit ansatz for $p = 10$ (10 layers). The attempt to minimize the expectation value was successful since its value dropped from above 70 to around 3.6.



(a) Training loss for the entire training consisting of 250 epochs.



(b) Zoom on the training loss for the last 100 training epochs.

Figure 6.3: Training loss for the optimizing quantum neural network for $p = 10$.

After the optimization process, the obtained values of the angles $\gamma_1, \dots, \gamma_p$ and $\beta_1, \dots, \beta_p$ were supplied to the parameterized quantum circuit and then the circuit was sampled for the solution binary strings 2^{16} times. The results are presented in Table 6.4.

Table 6.4: Results of sampling binary strings from the circuit with $p = 10$. In total, there were returned 1741 solutions out of the space of 2^{16} possible solutions. A significant gap between the number of occurrences of correct and incorrect solutions could be observed.

No.	**Sampled binary string**	**Order of visiting cities**	**Number of occurrences**	**Correct solution?**	**Optimal solution?**
1	0010 0100 1000 0001	$2 \to 1 \to 0 \to 3 \to 2$	2754	Yes	Yes
2	0100 1000 0001 0010	$1 \to 0 \to 3 \to 2 \to 1$	2717	Yes	Yes
3	1000 0001 0010 0100	$0 \to 3 \to 2 \to 1 \to 0$	2709	Yes	Yes
4	0100 0010 0001 1000	$1 \to 2 \to 3 \to 0 \to 1$	2664	Yes	Yes
5	1000 0100 0010 0001	$0 \to 1 \to 2 \to 3 \to 0$	2660	Yes	Yes
6	0001 1000 0100 0010	$3 \to 0 \to 1 \to 2 \to 3$	2623	Yes	Yes
7	0010 0001 1000 0100	$2 \to 3 \to 0 \to 1 \to 2$	2584	Yes	Yes
8	0010 1000 0001 0100	$2 \to 0 \to 3 \to 1 \to 2$	2574	Yes	No
9	0001 0010 0100 1000	$3 \to 2 \to 1 \to 0 \to 3$	2570	Yes	Yes
10	0100 0001 1000 0010	$1 \to 3 \to 0 \to 2 \to 1$	2569	Yes	No
11	1000 0010 0001 0100	$0 \to 2 \to 3 \to 1 \to 0$	2537	Yes	No
12	0001 0100 1000 0010	$3 \to 1 \to 0 \to 2 \to 3$	2531	Yes	No
13	0100 0001 0010 1000	$1 \to 3 \to 2 \to 0 \to 1$	2531	Yes	No
14	1000 0010 0100 0001	$0 \to 2 \to 1 \to 3 \to 0$	2492	Yes	No
15	0001 0100 0010 1000	$3 \to 1 \to 2 \to 0 \to 3$	2474	Yes	No
16	0100 1000 0010 0001	$1 \to 0 \to 2 \to 3 \to 1$	2467	Yes	No
17	0001 0010 1000 0100	$3 \to 2 \to 0 \to 1 \to 3$	2461	Yes	No
18	0001 1000 0010 0100	$3 \to 0 \to 2 \to 1 \to 3$	2441	Yes	No
19	1000 0001 0100 0010	$0 \to 3 \to 1 \to 2 \to 0$	2418	Yes	No
20	0100 0010 1000 0001	$1 \to 2 \to 0 \to 3 \to 1$	2400	Yes	No
21	1000 0100 0001 0010	$0 \to 1 \to 3 \to 2 \to 0$	2389	Yes	No
22	0010 0001 0100 1000	$2 \to 3 \to 1 \to 0 \to 2$	2369	Yes	No
23	0010 0100 0001 1000	$2 \to 1 \to 3 \to 0 \to 2$	2368	Yes	No
24	0010 1000 0100 0001	$2 \to 0 \to 1 \to 3 \to 2$	2346	Yes	No
25	0000 1000 0100 0001	constraints violated	41	No	No
26	0000 0100 1000 0010	constraints violated	36	No	No
...	...	...	...	...	...
1740	0000 0101 1000 0011	constraints violated	1	No	No
1741	0111 1000 0001 0001	constraints violated	1	No	No

Table 6.5 presents results for different number of variational layers p. With the increasing value of p, the number of correct solutions came closer to 100%, and the loss/expectation value that was minimized approached the optimal value of 2.39 (the shortest cycle length). The closer the loss value to the optimal value, the higher the number of correct solutions.

Table 6.5: Result of optimizing the cost of the QAOA with different numbers of variational layers for the optimization experiment.

Number of variational layers p	**Loss value**	**Correct solutions** [%]
1	17.0728	2.42
2	14.0526	5.63
3	10.4777	16.6
4	9.3789	24.47
5	8.6068	31.58
6	7.1554	55.14
7	5.9443	70.41
8	4.5044	86.01
9	3.7726	91.67
10	3.6044	92.54

6.2 Generalization task

One of the major assets of a successfully trained neural network is its ability to generalize well to unseen data. In this experiment, we checked whether it is possible to obtain a hybrid quantum-classical neural network that, after training on many instances of the Traveling Salesman Problem, will yield proper angles (γ, β) for an unseen instance of the TSP.

To create training data, 1000 different sets of four pairs of city coordinates were created. As in the optimization experiment (cf. Sec. 6.1), these coordinates sets were normalized, and then 1000 matrices of distances between normalized city coordinates were obtained. Next, this collection of distance matrices was split into training data (800 samples), validation data (100 samples), and test data (100 samples). The training data was used for training the network, the validation data for diagnosing the learning performance during training, and the test data for evaluating the model after training. To enable comparing the performance of the approaches from optimization and generalization experiments, these experiment results were obtained separately for different depths ($p \in \{1, \ldots, 10\}$) of the QAOA ansatz.

The first steps of preparing this experiment — data preparation and encoding, and the quantum neural network creation — were very similar to the ones in the optimization experiment (cf. Sec. 6.1.1 – 6.1.2). The only difference was that here, for each value of $p \in \{1, \ldots, 10\}$ the training set consisted of 1000 input samples instead of one input sample. E.g., for $p = 1$, there were 1000 cost Hamiltonians and associated with them 1000 parameterized quantum circuits of depth 1 (one variational layer), for $p = 2$, different 1000 cost Hamiltonians and associated with them 1000 parameterized quantum circuits of depth 2 (two variational layers), etc.

When it comes to the hybrid quantum-classical neural network model, it consisted of five layers: an input layer with $2p$ neurons, two hidden layers with 32 neurons each, a hidden layer with $2p$ neurons, and an expectation layer, cf. Figure 6.4. Based on an educated guess, it was determined that adding two hidden layers with 32 neurons to the model presented in Sec. 6.1.2 would create a new model suitable to learn to identify the patterns hidden in the data. To prevent the problem of *overfitting* (performing very well on training data and poorly on testing data), the regularization technique called *dropout* [62] (ignoring randomly picked neurons in the training phase) was used. The dropout rate was 0.2, and it was applied after the first and second hidden layers. Additionally, in the hidden layers number 1 and 2 the Rectified Linear Unit (ReLU) [63] *activation function* (which specifies the way of transforming neuron inputs into a neuron output) was applied.

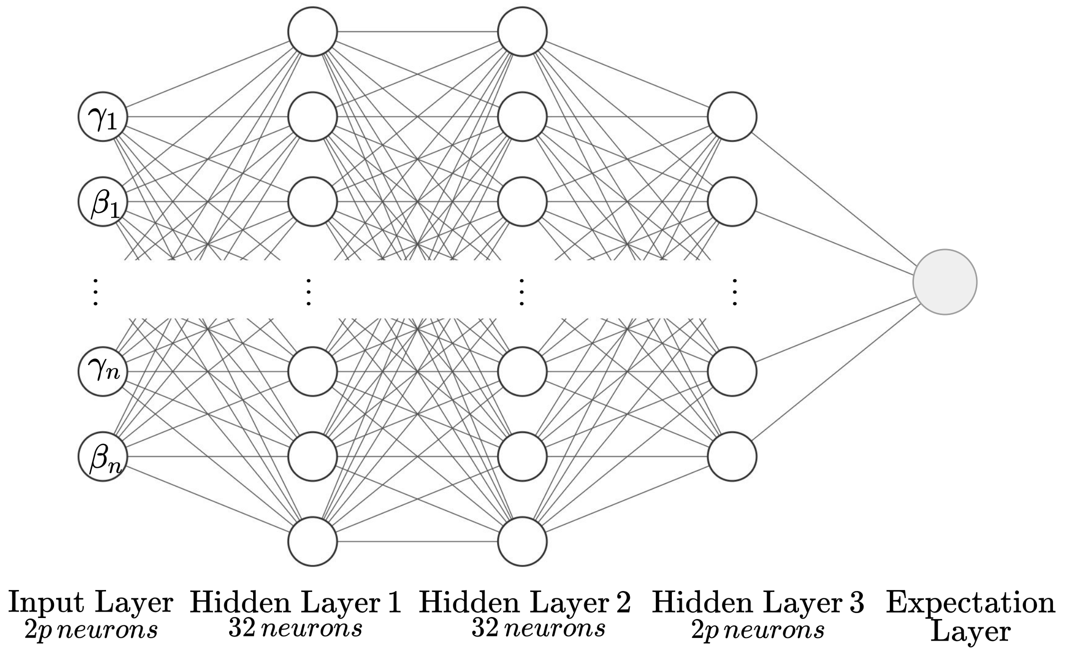


Figure 6.4: Feedforward neural network used in the generalization experiment.

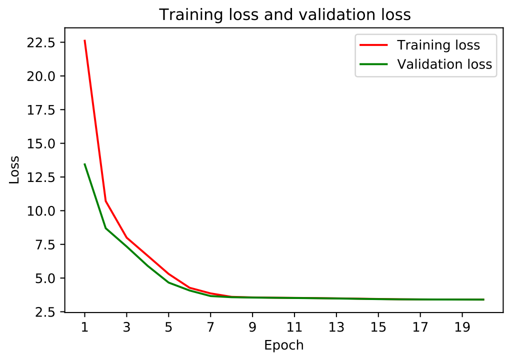
The training algorithm consisted of 60 epochs (every training data instance is processed once in an epoch). Moreover, to enable using mini-batch updates (updating the network's parameters after seeing a small subset of the data set) the training data was divided into 25 mini-batches of size 32. As in the previous experiment, the Adam optimization method was used with the learning rate 0.01. Initially, all input angles were set to zeros.

Figure 6.5 presents the learning curves for the case of $p = 10$. The training/expectation loss has decreased from around the value of 22.5 to 3 and reached a stage of stability. The validation loss reflects the network's performance during training. Based on these curves, it was determined that 60 epochs of training were sufficient (the validation loss also reached a stage of stability, and the gap between the training loss and validation loss was minimal).

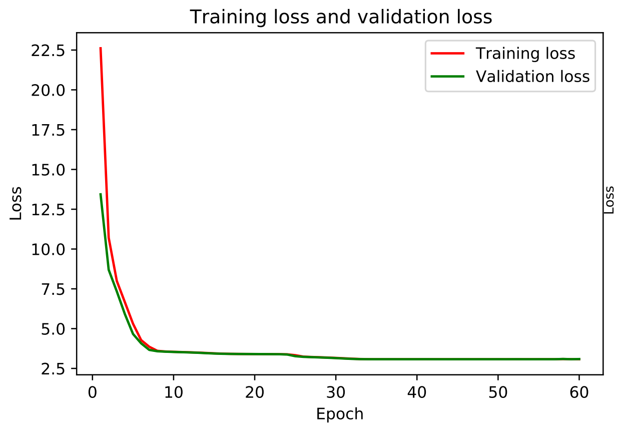
Table 6.6 presents the results of training the network. The training loss estimates the QAOA's expectation value on the training data, and the test loss estimates the expectation value on testing data. Both of the losses are of similar value, so it means that the network performed well on new data. The last column represents the percent of correct solutions for a selected new test case with the same city coordinates as in the first experiment (it was ensured that this test case was not in the training data).

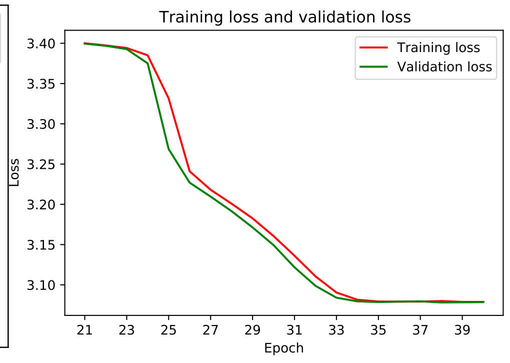
Table 6.6: Result of optimizing the cost of the QAOA with different numbers of variational layers for the generalization experiment.

Number of variational layers p	**Training loss value**	**Test loss value**	**Correct solutions for the selected test-case** [%]
1	17.0211	17.0446	2.31
2	13.9815	14.0100	5.98
3	10.4129	10.4418	16.57
4	9.0603	9.0859	27.04
5	7.5826	7.6183	48.99
6	5.4289	5.4602	75.44
7	4.3094	4.3414	87.09
8	3.9094	4.3414	90.57
9	3.4846	3.5141	92.56
10	3.0803	3.1221	96.22

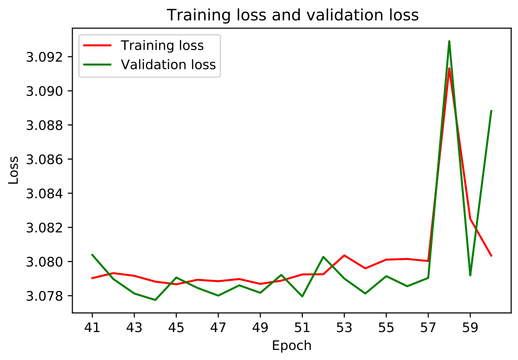


(a) Zoom on epochs 1–20.





(b) Zoom on epochs 21–40.

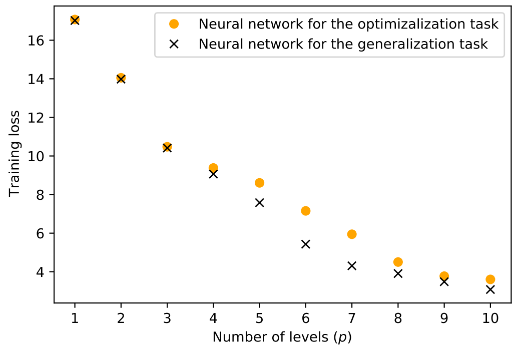


(c) Zoom on epochs 41–60.

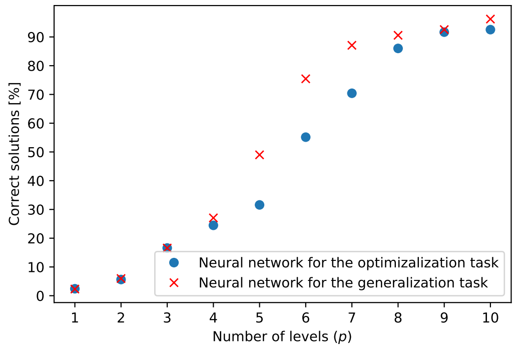
Figure 6.5: Learning curves for the QAOA circuit with $p = 10$.

6.3 Summary

All in all, the conducted experiments demonstrated that TensorFlow Quantum could be successfully used for minimizing the cost function of a parameterized quantum circuit for a combinatorial-optimization problem. Furthermore, the results of both experiments, when tested on the same problem instance, yield similar results, cf. Figure 6.6. However, the advantage of the network from the generalization experiment is that after it is trained, it immediately returns a result for new data (while in the model from the optimization experiment, the training process needs to be executed anew for each new data instance).



(a) Training losses with respect to the QAOA circuit's depth.



(b) Percent of correct solutions with respect to the QAOA circuit's depth.

Figure 6.6: Comparison of the optimization and generalization experiments results. The network from the generalization experiment yields considerably better outcomes for the $p \in \{5, 6, 7\}$, but for the other values of p no significant difference can be observed.

As expected, the results indicate that the greater the value of p, the lower the training loss and the higher the number of correct solutions.

Chapter 7

Conclusion

In this chapter, we summarize the thesis. At first the overview of the achieved goals is provided. The last section outlines the possible future research directions.

7.1 Summary

It is thought that TensorFlow has accelerated the research in the field of classical machine learning. For this reason, the authors of TensorFlow Quantum believe that the developed by them library will bring more attention to the field of hybrid quantum-classical machine learning. Therefore, the main purpose of this work was to provide an assessment of the capabilities of the TensorFlow Quantum library, especially in the context of the objectives defined in Section 1.2.

TensorFlow Quantum outline. Before introducing TensorFlow Quantum, Google has been offering the TensorFlow library for machine learning and Cirq for quantum computing. However, it was impossible to create hybrid quantum-classical machine learning models using only these libraries. A software solution that bridges the capabilities of TensorFlow and Cirq was needed. As a consequence, TensorFlow Quantum emerged as the missing link.

First of all, TensorFlow Quantum seamlessly integrates with TensorFlow and Cirq, which significantly accelerates the creation of both classical and quantum parts of the machine learning model. TFQ offers access to quantum devices and simulators. However, the publicly available paid quantum device has only 11 qubits, so to solve the problems that require a larger number of qubits, it might be necessary to use high-performance computing systems for running simulations. When it comes to the programming experience, TFQ can be perceived as relatively easy to use since it provides a high-level API, has comprehensible documentation, and is open-source.

Investigation of quantum algorithms feasible to implement. Since TFQ allows automatic differentiation for hybrid quantum-classical models, many quantum neural network types inspired by classical neural networks can be implemented. TFQ offers a repository of tutorials that contains a variety of minimal working implementations for many cutting-edge algorithms, incl. quantum classifiers, quantum convolutional neural networks, quantum generative adversarial networks, and quantum reinforcement learning. Based on the analysis of existing implementations, TFQ features, and the most prospective NISQ era algorithms, it was determined that the Quantum Approximate Optimization Algorithm is a good choice for investigating the actual capabilities of TFQ.

Assessment of the TFQ capabilities. To evaluate the TFQ, we performed two experiments based on solving the Traveling Salesman Problem with the use of the Quantum Approximate Optimization Algorithm and feedforward quantum neural networks (Sec. 6). The first experiment was focused on finding solutions for a single instance of the TSP using the optimization feature of the neural networks. The second experiment was aimed at training the hybrid neural network on many examples of the TSP to find correct solutions for unseen instances of the TSP. The computational steps that were required to conduct these experiments and correlated with them features of the tested library are presented in Table 7.1.

Table 7.1: Computational steps and features provided by the TFQ needed to execute hybrid quantum-classical machine learning models.

Experiment step	**Utilized framework features**
Dataset preparation	Converting Hamiltonian operators and quantum circuits into TF tensors.
Quantum model evaluation	Extracting classical information from the quantum elements of the neural network model through the measurement (calculating the expectation value).
Classical model evaluation	Evaluation of the model that has only classical data (since quantum information is always extracted before this step via measurement).
Training process	Estimating quantum gradients and performing automatic differentiation. Using gradient-based optimizers provided by TF.
Parameters update	Using batched circuit execution that enables updating the model's parameters after processing a mini-batch of data.

7.2 Future work

The future research directions might include:

- verifying the simulation results on real hardware when the appropriate number of qubits will be available;
- modifying the quantum circuit:
  - using another type of the mixer Hamiltonian in the parameterized quantum circuit ansatz [64];
  - reducing the number of needed qubits by changing the problem's encoding [65];
- trying to solve a different combinatorial optimization problem, e.g., the workflow scheduling problem [66], using the proposed neural network models;
- exploring new architectures of hybrid quantum-classical neural networks.

Further exploration of quantum machine learning, accelerated by software such as TensorFlow Quantum, can lead to exceptional discoveries in this field and integration of quantum computing with the mainstream computing paradigms.

Appendix A

Appendix to Chapter 5

A.1 Hamiltonian derivation for the Traveling Salesman Problem

First of all, we will expand the sum of Equation 5.1.2 for $N = 2$.

$$\begin{aligned} A_1 &= \sum_{i=0}^{1}(1 - \sum_{t=0}^{1} x_{i,t})^2 \\ &= \sum_{i=0}^{1}(1 - x_{i,0} - x_{i,1})^2 \\ &= (1 - x_{0,0} - x_{0,1})^2 + (1 - x_{1,0} - x_{1,1})^2 \\ &= 1 + x_{0,0}^2 + x_{0,1}^2 - 2x_{0,0} - 2x_{0,1} + 2x_{0,0}x_{0,1} + 1 + x_{1,0}^2 + x_{1,1}^2 - 2x_{1,0} - 2x_{1,1} + 2x_{1,0}x_{1,1} \end{aligned} \tag{A.1.1}$$

Using the Property 5.1.5

$$x_{i,t} \longleftrightarrow \frac{I - Z_k}{2}$$

and based on the fact that the square of the Identity gate and the square of Pauli-Z gate are the Identity gate

$$I^2 = Z^2 = I \tag{A.1.2}$$

the Hamiltonian expression H_{A_1} can be simplified as follows

$$\begin{aligned}
H_{A_1} &= I + \left(\frac{I-Z_0}{2}\right)^2 + \left(\frac{I-Z_2}{2}\right)^2 - 2\left(\frac{I-Z_0}{2}\right) - 2\left(\frac{I-Z_2}{2}\right) + 2\left(\frac{I-Z_0}{2}\cdot\frac{I-Z_2}{2}\right) + \\
&\quad + I + \left(\frac{I-Z_1}{2}\right)^2 + \left(\frac{I-Z_3}{2}\right)^2 - 2\left(\frac{I-Z_1}{2}\right) - 2\left(\frac{I-Z_3}{2}\right) + 2\left(\frac{I-Z_1}{2}\cdot\frac{I-Z_3}{2}\right) \\
&= I + \frac{I^2+Z_0^2-2Z_0}{4} + \frac{I^2+Z_2^2-2Z_2}{4} - I + Z_0 - I + Z_2 + 2\frac{I^2-Z_0-Z_2+Z_0Z_2}{4} + \\
&\quad + I + \frac{I^2+Z_1^2-2Z_1}{4} + \frac{I^2+Z_3^2-2Z_3}{4} - I + Z_1 - I + Z_3 + 2\frac{I^2-Z_1-Z_3+Z_1Z_3}{4} \\
&= I + \frac{I}{2} - \frac{Z_0}{2} + \frac{I}{2} - \frac{Z_2}{2} - 2I + Z_0 + Z_2 + \frac{I}{2} - \frac{Z_0}{2} - \frac{Z_2}{2} + \frac{Z_0Z_2}{2} + \\
&\quad + I + \frac{I}{2} - \frac{Z_1}{2} + \frac{I}{2} - \frac{Z_3}{2} - 2I + Z_1 + Z_3 + \frac{I}{2} - \frac{Z_1}{2} - \frac{Z_3}{2} + \frac{Z_1Z_3}{2} \\
&= I + \frac{Z_0Z_2}{2} + \frac{Z_1Z_3}{2}.
\end{aligned} \tag{A.1.3}$$

Analogically, the Equation 5.1.3 for $N = 2$

$$A_2 = \sum_{t=0}^{1}(1 - \sum_{i=0}^{1} x_{i,t})^2$$

corresponds to

$$H_{A_2} = I + \frac{Z_0Z_1}{2} + \frac{Z_2Z_3}{2}. \tag{A.1.4}$$

The cost function Equation 5.1.1 expanded for $N = 2$ is

$$\begin{aligned}
B &= \sum_{i=0}^{1}\sum_{j=0}^{1} d_{i,j} \sum_{t=0}^{1} x_{i,t}x_{j,t+1} \qquad \text{(A.1.5)} \\
&= \sum_{i=0}^{1}\sum_{j=0}^{1} d_{i,j}(x_{i,0}x_{j,1} + x_{i,1}x_{j,0}) \\
&= d_{0,1}(x_{0,0}x_{1,1} + x_{0,1}x_{1,0}) + d_{1,0}(x_{1,0}x_{0,1} + x_{1,1}x_{0,0}) \\
&= (d_{0,1} + d_{1,0})(x_{0,0}x_{1,1} + x_{0,1}x_{1,0}).
\end{aligned}$$

Based on the problem definition ($d_{i,j} = d_{j,i}$), the Hamiltonian corresponding to B is

$$\begin{aligned} H_B &= 2d_{0,1}\left(\frac{I-Z_0}{2}\cdot\frac{I-Z_3}{2}+\frac{I-Z_1}{2}\cdot\frac{I-Z_2}{2}\right) \\ &= d_{0,1}\left(\frac{I^2-Z_0-Z_3+Z_0Z_3}{2}+\frac{I^2-Z_1-Z_2+Z_1Z_2}{2}\right) \\ &= d_{0,1}\left(I-\frac{Z_0}{2}-\frac{Z_1}{2}-\frac{Z_2}{2}-\frac{Z_3}{2}+\frac{Z_0Z_3}{2}+\frac{Z_1Z_2}{2}\right). \end{aligned} \tag{A.1.6}$$

A.2 Quantum gates as matrices

Evaluation of $CNOT \cdot R_z(\gamma) \cdot CNOT$.

$$\begin{aligned} CNOT\cdot(I\otimes R_z(\gamma))\cdot CNOT &= \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}e^{-i\frac{\gamma}{2}}&0&0&0\\0&e^{i\frac{\gamma}{2}}&0&0\\0&0&e^{-i\frac{\gamma}{2}}&0\\0&0&0&e^{i\frac{\gamma}{2}}\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix} \\ &= \begin{bmatrix}e^{-i\frac{\gamma}{2}}&0&0&0\\0&e^{i\frac{\gamma}{2}}&0&0\\0&0&e^{i\frac{\gamma}{2}}&0\\0&0&0&e^{-i\frac{\gamma}{2}}\end{bmatrix} \end{aligned} \tag{A.2.1}$$

Evaluation of $R_x(\gamma)$.
Using

$$sin(\frac{\gamma}{2}) = \frac{ie^{-i\frac{\gamma}{2}}-ie^{i\frac{\gamma}{2}}}{2}, \quad cos(\frac{\gamma}{2}) = \frac{e^{-i\frac{\gamma}{2}}+e^{i\frac{\gamma}{2}}}{2} \tag{A.2.2}$$

$R_x(\gamma)$ can be expressed as

$$\begin{aligned} R_x(\gamma) &= \begin{bmatrix}cos\frac{\gamma}{2}&-isin\frac{\gamma}{2}\\-isin\frac{\gamma}{2}&cos\frac{\gamma}{2}\end{bmatrix} \\ &= \begin{bmatrix}e^{-i\frac{\gamma}{2}}+e^{i\frac{\gamma}{2}}&e^{-i\frac{\gamma}{2}}-e^{i\frac{\gamma}{2}}\\e^{-i\frac{\gamma}{2}}-e^{i\frac{\gamma}{2}}&e^{-i\frac{\gamma}{2}}+e^{i\frac{\gamma}{2}}\end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}e^{-i\frac{\gamma}{2}}&1\\1&e^{i\frac{\gamma}{2}}\end{bmatrix}\begin{bmatrix}1&1\\1&-1\end{bmatrix} \\ &= HR_Z(\gamma)H. \end{aligned} \tag{A.2.3}$$

List of Figures

List of Tables

Bibliography

[1] G. Brassard, I. Chuang, S. Lloyd, and C. Monroe. Quantum computing. *Proceedings of the National Academy of Sciences*, 95(19):11032–11033, 1998.

[2] F. Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[3] E. Pednault et al. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits. *arXiv preprint*, 2019.

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

[5] J. Preskill. Quantum computing and the entanglement frontier. *Rapporteur Talk at the 25th Solvay Conference on Physics, Brussels*, 2012.

[6] M. Cerezo, A. Arrasmith, R. Babbush, et al. Variational quantum algorithms. *arXiv preprint*, 2020.

[7] A. Peruzzo, J. McClean, P. Shadbolt, et al. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), 2014.

[8] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. *arXiv preprint*, 2014.

[9] M. Broughton, G. Verdon, T. McCourt, et al. Tensorflow quantum: A software framework for quantum machine learning. *arXiv preprint*, 2020.

[10] M. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.

[11] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4), 2019.

[12] S. Sim, P. D. Johnson, and A. Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum–classical algorithms. *Advanced Quantum Technologies*, 2(12), 2019.

[13] M. Medvidović and G. Carleo. Classical variational simulation of the quantum approximate optimization algorithm. *npj Quantum Information*, 7(1), 2021.

[14] S. Khairy, R. Shaydulin, L. Cincio, Y. Alexeev, and P. Balaprakash. Learning to optimize variational quantum circuits to solve combinatorial problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2367–2375, 2020.

[15] A. Skolik, J. R. McClean, M. Mohseni, P. van der Smagt, and M. Leib. Layerwise learning for quantum neural networks. *Quantum Machine Intelligence*, 3(1), 2021.

[16] C. C. McGeoch. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing*, 5(2):1–93, 2014.

[17] S. Hadfield. On the representation of boolean and real functions as hamiltonians for quantum computing. *arXiv preprint*, 2018.

[18] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[19] A. LeNail. NN-SVG: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[20] F. Tacchino, P. Barkoutsos, C. Macchiavello, I. Tavernelli, D. Gerace, and D. Bajoni. Quantum implementation of an artificial feed-forward neural network. *Quantum Science and Technology*, 5(4):044010, 2020.

[21] S. Yan, H. Qi, and W. Cui. Nonlinear quantum neuron: A fundamental building block for quantum neural networks. *Physical Review A*, 102(5), 2020.

[22] Y. LeCun, Haffner P., L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–345. Springer Berlin Heidelberg, 1999.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS'12*, page 1097–1105, 2012.

[24] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

[25] I. Cong, S. Choi, and M. D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

[26] M. Henderson, S. Shakya, S. Pradhan, and T. Cook. Quanvolutional neural networks: Powering image recognition with quantum circuits. *arXiv preprint*, 2019.

[27] S. Oh, J. Choi, and J. Kim. A tutorial on quantum convolutional neural networks (qcnn). *arXiv preprint*, 2020.

[28] Y. LeCun, C. Cortes, and C. Burges. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`. Accessed: 2021-05.

[29] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, et al. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680. MIT Press, 2014.

[30] T. Salimans, I. Goodfellow, W. Zaremba, et al. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[31] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[32] R. A. Yeh, C. Chen, T. Y. Lim, and others. Semantic image inpainting with deep generative models. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6882–6890, 2017.

[33] S. Lloyd and C. Weedbrook. Quantum generative adversarial learning. *Physical Review Letters*, 121(4), 2018.

[34] P. Dallaire-Demers and N. Killoran. Quantum generative adversarial networks. *Physical Review A*, 98(1), 2018.

[35] A. Baydin, B. Pearlmutter, A. Radul, and J. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[36] M. Abadi, P. Barham, J. Chen, 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.

[37] A. Paszke, S. Gross, F. Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, pages 8024–8035. Curran Associates, Inc., 2019.

[38] F. Bastien, P. Lamblin, R. Pascanu, et al. Theano: new features and speed improvements. *arXiv preprint*, 2012.

[39] V. Bergholm, J. Izaac, M. Schuld, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint*, 2020.

[40] Jax: Autograd and xla documentation. `https://jax.readthedocs.io/en/latest/`. Accessed: 2021-05.

[41] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, et al. Qiskit: An open-source framework for quantum computing. *Zenodo*, 2019.

[42] W. Jiang, J. Xiong, and Y. Shi. A co-design framework of neural networks and quantum circuits towards quantum advantage. *Nature Communications*, 12(1), 2021.

[43] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang. Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design. *Quantum*, 4:341, 2020.

[44] Zygote julia package for differentiable programming. `https://fluxml.ai/Zygote.jl/latest/`. Accessed: 2021-05.

[45] Cirq Developers (see full list of authors on Github: `https://github.com/quantumlib/Cirq/graphs/contributors`). Cirq. *Zenodo*, 2018.

[46] G. E. Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *arXiv preprint*, 2019.

[47] G. Verdon, M. Broughton, J. R. McClean, et al. Learning to learn with quantum neural networks via classical neural networks. *arXiv preprint*, 2019.

[48] H. Chen, L. Wossnig, S. Severini, H. Neven, and M. Mohseni. Universal discriminative quantum neural networks. *Quantum Machine Intelligence*, 3(1), 2020.

[49] E. Farhi and H. Neven. Classification with quantum neural networks on near term processors. *arXiv preprint*, 2018.

[50] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[51] S. Jerbi, C. Gyurik, S. Marshall, H. J. Briegel, and V. Dunjko. Variational quantum policies for reinforcement learning. *arXiv preprint*, 2021.

[52] A. Skolik, S. Jerbi, and V. Dunjko. Quantum agents in the gym: a variational quantum algorithm for deep q-learning. *arXiv preprint*, 2021.

[53] M. Y. Niu, A. Zlokapa, M. Broughton, et al. Entangling quantum generative adversarial networks. *arXiv preprint*, 2021.

[54] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), 2018.

[55] M. E. S. Morales, J. D. Biamonte, and Z. Zimborás. On the universality of the quantum approximate optimization algorithm. *Quantum Information Processing*, 19(9), 2020.

[56] E. Farhi and Harrow A. W. Quantum supremacy through the quantum approximate optimization algorithm. *arXiv preprint*, 2019.

[57] A. Glos, A. Krawiec, and Z. Zimborás. Space-efficient binary optimization for variational computing. *arXiv preprint*, 2020.

[58] P. E. Black. Traveling Salesman, in dictionary of algorithms and data structures. `https://www.nist.gov/dads/HTML/travelingSalesman.html`. Accessed: 2021-05.

[59] A. Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2, 2014.

[60] J. Plewa and A. Sieńko. Hybrid algorithms for workflow scheduling problem in quantum devices based on gate model. Master's thesis, AGH University of Science and Technology in Krakow, 2021 [in preparation].

[61] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*, 2015.

[62] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[63] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, pages 315–323. PMLR, 2011.

[64] S. Hadfield, Z. Wang, B. O'Gorman, et al. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2), 2019.

[65] N. Chancellor. Domain wall encoding of discrete variables for quantum annealing and qaoa. *Quantum Science and Technology*, 4(4), 2019.

[66] D. Tomasiewicz, M. Pawlik, M. Malawski, and K. Rycerz. Foundations for workflow application scheduling on d-wave system. In *Lecture Notes in Computer Science*, pages 516–530. Springer International Publishing, 2020.