Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Quantum-enhanced Machine Learning in the NISQ era

Marco Radic

Course of Study:	Informatik
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Daniel Vietz, M.Sc.
Commenced:	April 29, 2019
Completed:	October 29, 2019

Abstract

Quantum computation technologies have reached a new level of sophistication with the release of the first commercial offerings. Likewise, Machine Learning is popular for use-cases in both industry and research. With Quantum Machine Learning, one hopes to combine both areas in a symbiotic relationship to achieve an advantage in artificial intelligence with the use of quantum technologies. Recently presented approaches make use of quantum technologies in combination with classical hardware resources in order to mitigate the problems imposed by shortcomings of quantum computers of the current generation. Some of these approaches use quantum circuits with free parameters, which are optimized to solve problems and objectives in Machine Learning. This work presents a concept for automated modelling of these quantum circuits, with the goal of constructing suitable circuits for the task of classification. The concept is implemented in a prototype and validated in experiments.

Kurzfassung

Mit der Veröffentlichung der ersten kommerziellen Angebote haben Quantencomputer-Technologien einen neuen Reifegrad erreicht. Ebenso erfreuen sich Machine Learning-Verfahren großem Interesse aus Forschung und Industrie. Mit Quantum Machine Learning erhofft man sich, diese beiden Felder symbiotisch zu vereinen, um einen Vorteil im Bereich künstlicher Intelligenz durch den Einsatz von Quantentechnologien zu erreichen. Kürzlich vorgestellte Ansätze behandeln den Einsatz von Quantencomputern in Kombination mit klassischen Hardwareressourcen, um Nachteilen der aktuellen Generation von Quantencomputern entgegenzuwirken. Einige dieser Verfahren nutzen Quantenschaltkreise mit freien Parametern, welche mit klassischen Optimierungsverfahren auf Machine Learning-Probleme abgestimmt werden. In dieser Arbeit wird ein Ansatz vorgestellt, welcher die Modellierung dieser Quantenschaltkreise dynamisch übernimmt, mit dem Ziel, geeignete Schaltkreise für die Klassifikations-Aufgabe zu konstruieren. Der Ansatz wird prototypisch implementiert und mit Experimenten validiert.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

When the idea of using quantum systems for computation was first introduced by Feynman [Fey82], it was merely a theoretical concept to improve the simulation of physical systems. In recent years, the research field has seen a jump in activity and interest with the first commercial offerings of quantum computers reaching the market [LMR+17]. Although this marks a milestone for the field, existing devices have several shortcomings that separate them from fully utilizing their capabilities for a quantum advantage. It is hypothesized that it will take at least several years for the hardware to overcome these shortcomings. Quantum devices of this generation are labeled as *noisy intermediate-scale quantum* (NISQ) [Pre18]. In order to suit the capabilities of NISQ devices, a new group of quantum algorithms have been explored in research. By combining computational advantages of quantum hardware along with classical computation in specialized hybrid-algorithms, shortcomings of either computing paradigm can be mitigated and quantum computers have the potential to become of interest for industry applications in the near future [MRN+17].

Especially Machine Learning, which experiences popularity in research and industry, is considered to be an area where specialized quantum hardware could soon lead to an advantage. In recent times, several algorithms were proposed, which rely on incorporating quantum computation in a hybrid setting, combined with classical optimization to realize Machine Learning models using quantum resources of the current generation. To train the models, parametrized quantum circuits are constructed and then optimized by incorporating classical hardware and optimization methods to suit a particular use-case, for example performing classification on a dataset. When designing quantum circuits, special considerations have to be made in order suit to currently available NISQ devices. As such, circuits should be performing as few operations as possible, in order to guarantee the most reliable computation that is currently possible. Furthermore, these short, low-depth circuits are also best suited for error correction schemes [TBG17]. The search for a suitable circuit structure is mostly manual, time consuming and does not always follow systematic procedures. This is similar to the shortcomings of finding neural network architectures in Deep Learning. Oftentimes, better performing network architectures are found by manually tweaking existing architectures, running time consuming training and comparing the models. In order to systematically automate this search procedure for Deep Learning, but also other fields of Machine Learning, the research field for automated Machine Learning gained traction in the past years [THHL13]. By leaving the search for suitable architectures, settings and hyperparameters up to an automated procedure, non-domain experts can focus on problem solving. This is desirable especially in the context of the complexity quantum computing theory.

To this end, we present an approach that applies concepts from automated Machine Learning to parametrized quantum circuits used for classification tasks. Instead of using fixed circuits structures, different structures are systematically explored with the goal of generating the most-suitable quantum circuit. It eliminates the complexity of manually designing quantum circuits to realize Machine

Learning use-cases with quantum hardware and hopefully can accelerate the adoption of quantum computers for real-world applications. We validate our approach in experiments that were conducted using a prototypical implementation.

Outline

The remainder of this work is divided into the following Chapters:

Chapter 2 – Foundations: Here, necessary foundational knowledge for quantum computation and Machine Learning is introduced.

Chapter 3 – Related Work: Related research for dynamic quantum circuit generation and automated Machine Learning is presented and differences to our work are discussed.

Chapter 4 – Approach: This Chapter presents our approach and constitutes the main part of this work.

Chapter 5 – Validation: The prototypical implementation is presented. Furthermore, it is used to validate the approach in an experimental setting.

Chapter 6 – Conclusion and Future Work: Provides a summary of the work and gives conclusive remarks, along with an outlook to future work.

2 Foundations

This Chapter establishes the foundations necessary to follow the remainder of this work. The basics of quantum computation are introduced, followed by a general overview of Machine Learning, where Supervised Learning and Reinforcement Learning are touched upon followed by a brief introduction into methods of automated Machine Learning. Eventually, the emerging discipline of Quantum Machine Learning is introduced.

2.1 Quantum Computation

The following section aims to describes the basics of quantum computation, starting with the definition of a qubit, up to performing arbitrary computations using a system of qubits and manipulations of said system. Unless noted otherwise, the content of this section is based on the standard textbook on the topic by Chuang and Nielsen [NC00].

2.1.1 Introduction

Quantum computation describes a computational paradigm where phenomena from quantum mechanics are used to perform arbitrary computations on so-called qubits. When this idea was first introduced [Fey82], no actual physical realization was in sight. Moreover, for years it was not clear whether a quantum computer would ever be built. Nonetheless, research sparked many subfields of quantum computation and quantum information. With recent progress in research and development, the first commercially available quantum computing systems became available.

2.1.2 Qubits

A *qubit*, or *quantum bit*, mathematically describes the state of a quantum system. In comparison, a classical bit also describes the state of a system. The states of a classical bit are limited to 0 and 1. A qubit can represent these states, as vectors in Dirac notation, as $|0\rangle$ and $|1\rangle$ respectively. In the standard basis, we can write out these vectors as

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.1}$$

In the following, we will assume the $|0\rangle$ and $|1\rangle$ vectors to be in standard basis at any time. In addition to the $|0\rangle$ and $|1\rangle$ state, a qubit can be in a *superposition* of these states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \tag{2.2}$$

where $\alpha, \beta \in \mathbb{C}$ with respect to the constraint $\big\| |\psi\rangle \big\| = 1$, which implies $|\alpha|^2 + |\beta|^2 = 1$. The values α and β are called the *amplitudes* of the $|0\rangle$ and $|1\rangle$ state, respectively. An alternative representation is given by

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle\right) \tag{2.3}$$

where $\varphi \in [0, 2\pi], \theta \in [0, \pi]$. Because global phase factors of a quantum system are not measurable, $e^{i\gamma}$ and subsequently γ can be dropped. Equation 2.3 can therefore be simplified to

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle \tag{2.4}$$

With the two remaining degrees of freedom φ and θ, the state of a single qubit can be visualized as a point on the *Bloch sphere*, which is depicted in Figure 2.1. It is considered particularly useful for human interpretation of the state of a single qubit, as a state that changes over time can be observed as a moving point on the sphere.

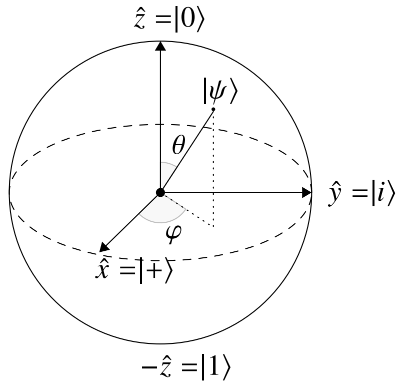


Figure 2.1: Illustration of the Bloch sphere, adapted from [NC00]. A quantum state $|\psi\rangle$ is a point on the depicted sphere

Systems with Multiple Qubits

Multiple systems of single qubits $\{|\psi_1\rangle \ldots |\psi_n\rangle\}$ can be combined into a system of multiple qubits $|\Psi\rangle$, where

$$|\Psi\rangle = |\psi_1\rangle \otimes \ldots \otimes |\psi_n\rangle = |\psi_1\rangle \ldots |\psi_n\rangle = |\psi_1 \ldots \psi_n\rangle \tag{2.5}$$

$$= \alpha_1 |0 \ldots 0\rangle + \cdots + \alpha_{2^n} |1 \ldots 1\rangle \tag{2.6}$$

which we call a *qubit register* of size n. The symbol '$\otimes$' describes the *Kronecker product*, or *tensor product*, defined as

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}B & A_{m2}B & \cdots & A_{mn}B \end{bmatrix} \tag{2.7}$$

where $A \in \mathbb{C}^{m \times n}, B \in \mathbb{C}^{p \times q}$.

We note that the resulting qubit register $|\Psi\rangle$ now has 2^n amplitude entries and still satisfies $\big\| |\Psi\rangle \big\| = 1$, that is, the squared amplitudes of the system sum up to one. Every amplitude α_k is associated with a bitstring $b_k \in \{0, 1\}^n$, representing a number between 0 and $2^n - 1$ in the decimal system. The value $|\alpha_k|^2$ can be interpreted as the probability of measuring b_k and therefore $|\Psi\rangle$ lends itself to the interpretation of describing a probability distribution over bitstrings of length n. The notion of measurement will be touched upon in the following section. From this point of view, quantum computation can be understood as manipulating the probabilities of measuring a certain bitstring, which can for example encode a certain result. This core idea is used for many algorithms, especially the family of algorithms that rely on performing *amplitude amplification*, such as for example the Grover algorithm [Gro96] for search in an unstructured set.

Measurement

While qubits can be in a linear combination of states, the so-called *superposition* of states, these states cannot by observed. To observe a qubit, a *measurement* of the qubit has to be performed. While the set of possible states of a qubit is infinite, the set of measurable states is finite. As previously mentioned, amplitudes can be associated with the probability of observing a basis state when measuring in an appropriate basis. This leads to the observation that a single measurement in general leads to a result that is subject to stochastic variance. In practice, computation followed by measurement is therefore repeated multiple times, resulting in a probability distribution of measurement results. The most probable result can then be selected from the distribution following the maximum likelihood principle.

Entanglement

Up to now, a qubit register can be viewed as the tensor product of single-qubit systems, which is called *separable*. A characteristic property of quantum systems is the ability for the system to evolve into a state that is not separable into its single-qubit systems. We call this state *entangled*. An example is the Bell state $|\Phi^+\rangle$ with

$$\left|\Phi^+\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.8}$$

There exist no valid single qubit states $|x\rangle$, $|y\rangle$, such that $|\Phi^+\rangle = |x\rangle \otimes |y\rangle$. Therefore, $|\Phi^+\rangle$ is an entangled state. We note that this state has interesting and illustrative properties for measurement. Let the outcome of the measurement of the first qubit of $|\Phi^+\rangle$ be 0. Then, with certainty, the second qubit can subsequently only be measured in the 0-state too. By measuring one qubit, the probabilities for measuring the other in a certain state are affected. Entanglement is a characteristic property of a quantum system and necessary for quantum speedup [JL03]. That is, a quantum algorithm on a quantum computer can only gain relevant speedup over a classical computer if entanglement is used as a resource in the algorithm.

2.1.3 Gate-based computing model

The previous section assumed a qubit or multiple qubits to be in a certain state. Quantum computation includes manipulating this state in order to perform computation. This is accomplished by applying transformations to the state. Physically, this is realized by, for example, laser beams. Omitting further details for the physical realization of quantum computers, we are concerned with the mathematical abstractions and introduce a standard method of composing, structuring and visualizing quantum algorithms.

Quantum states are manipulated using *unitary transformations*. An operator U is *unitary* iff it preserves the inner product in a complex Hilbert space $\mathcal{H}$. The operator U is therefore length preserving and implies the existence of an inverse. This means that every step in a quantum algorithm and therefore the entire algorithm is reversible, which is another characteristic of quantum computation.

A general framework for building quantum algorithms is the gate-based circuit model by Deutsch [Deu89]. One or multiple qubits are manipulated by applying unitary operations in the form of *gates* over time. The most commonly used gates in quantum algorithms are listed in Figure 2.2. The entirety of gates applied over time is called a *circuit*.

While the unitary operators for single qubits are 2×2 complex matrices, the unitary transformation U_k can be applied to the k-th of n qubits in a register by constructing the unitary transformation

$$U = \mathbb{I}_1 \otimes \cdots \otimes U_k \otimes \cdots \otimes \mathbb{I}_n \tag{2.9}$$

where $\mathbb{I}_i$ is the 2×2 identity matrix. Similar conditions hold for multiple-qubit-gates, such as the *conditional* NOT-gate (CNOT) acting on two qubits.

An example for a circuit is depicted in Figure 2.3. From the initial ground state $|\psi_0\rangle = |00\rangle$, an application of the Hadamard gate on the first qubit leads to

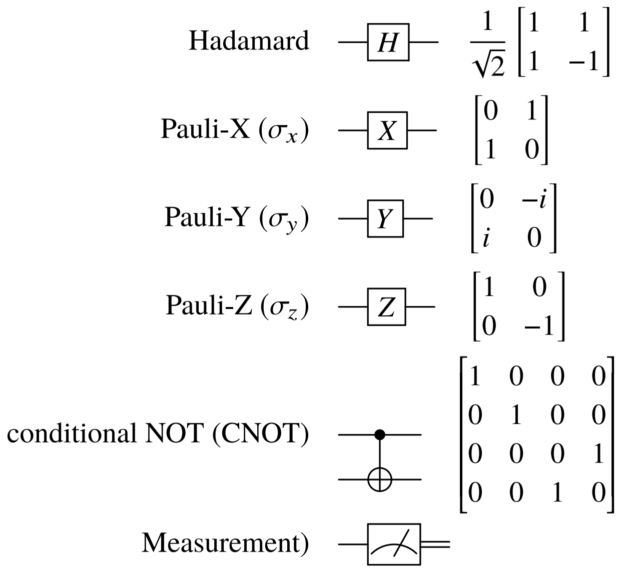

Hadamard
H
$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$
Pauli-X (σ_x)
X
$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$
Pauli-Y (σ_y)
Y
$\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}$
Pauli-Z (σ_z)
Z
$\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$
conditional NOT (CNOT)
$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$
Measurement)

Figure 2.2: Common gates used in quantum circuits, listed with their description, gate symbol and unitary transformation that they implement

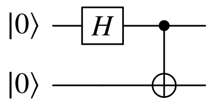


Figure 2.3: Circuit that prepares the Bell state $|\Phi^+\rangle$ in Eq. 2.8 from the ground state $|00\rangle$

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \tag{2.10}$$

After applying the CNOT gate to the qubits, which is the quantum equivalent to a classical `XOR`-operation of two bits, we get the desired state

$$|\psi_2\rangle = \left|\Phi^+\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.11}$$

In addition to the transformations listed above, a generalized gate can be described by the following matrix, which performs a rotation around an arbitrary axis:

$$R(\alpha, \beta, \gamma, \phi) = e^{i\phi}\begin{bmatrix} e^{i\beta}\cos\alpha & e^{i\gamma}\sin\alpha \\ -e^{-i\gamma}\sin\alpha & e^{-i\beta}\cos\alpha \end{bmatrix} \tag{2.12}$$

The $e^{i\phi}$ and subsequently ϕ can be dropped because its a global phase factor that cannot be measured.

The rotation operators $R_x(\theta), R_y(\theta), R_z(\theta)$ around the main axes are given by:

$$R_x(\theta) \equiv e^{i\theta X/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \tag{2.13}$$

$$R_y(\theta) \equiv e^{i\theta Y/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \tag{2.14}$$

$$R_z(\theta) \equiv e^{i\theta Z/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{-\theta/2} \end{bmatrix} \tag{2.15}$$

2.1.4 Physical Realization in the NISQ-era

DiVincenzo [DiV00] names the following five necessary conditions for the successful implementation of quantum computation:

1. A scalable physical system with well characterized qubits
2. The ability to initialize the state of the qubits to a simple fiducial/ground state
3. Long relevant decoherence times, much longer than the gate operation time
4. A “universal” set of quantum gates
5. A qubit-specific measurement capability

Realizations of quantum computers are currently for example performed using superconducting circuits, ion traps or quantum dots. While first commercial offerings are available, they do have shortcomings, not necessarily fulfilling all of the stated necessary conditions. Quantum computation devices of this era are therefore in summary characterized as *noisy intermediate-scale quantum* (NISQ) devices [Pre18]. Characteristical for offerings of existing quantum computers or devices in the near future are noise, connectivity issues and limited decoherence time [Pre18], which affects the third necessary condition. Here, noise refers to errors that manipulate or corrupt the quantum state over time or with the application of gates. As a result, a gate in a quantum circuit might not perform the expected transformation. The noise properties of a qubit might change over execution time and vary for every qubit. Decoherence refers to the loss of the ability to steadily remain in an arbitrary, possibly superposition-, state over time. Current devices have limited decoherence times. This effectively limits the number of gates that can be applied and therefore the length of the algorithms that can be executed on the device, before calculations become unreliable. In addition, the number of qubits made available by the underlying quantum system of a quantum computer is limited, as devices in the near future are not able to scale well, perturbing the first condition. Connectivity of qubits, or topology, refers to the connections between physical qubits, which are needed to apply multi-qubit gates to them. Currently, devices oftentimes do not offer full connectivity, which means that intermediary qubit swap routines have to be inserted during the compilation step of a quantum circuit to run on actual hardware. Specialized efforts towards error mitigation and NISQ-ready compilation have been made [PZW] [AAA+19] [MBJ+19] in addition to benchmarking methods [CBS+18], which hopefully accelerate the quality of quantum computation systems and research for NISQ devices.

2.2 Machine Learning

This section briefly introduces Machine Learning and its different subfields. Unless noted otherwise, the content is based on [HTFF05].

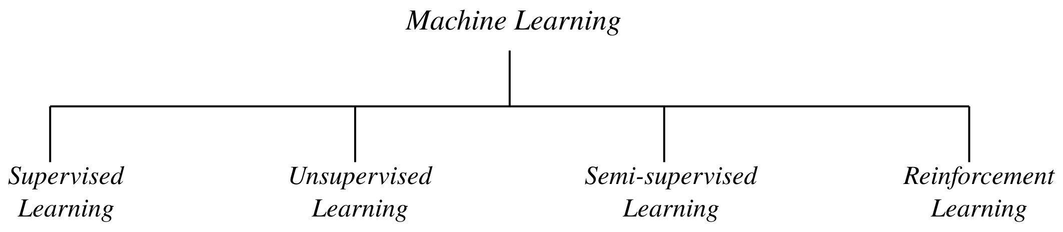


Figure 2.4: Hierarchical ontology of Machine Learning and its main subfields

Machine Learning describes a subfield of artificial intelligence, where generally data is used to make decisions and find patterns in unstructured environments. The field is further divided into four main subfields, as shown in Figure 2.4. *Supervised Learning* refers to learning from labeled datasets for tasks such as classification and regression and needs labeled examples, such as a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where each datapoint x_i is assigned a label y_i. *Unsupervised Learning* describes learning methods that are suited for datasets that do not have a label available. Example tasks include dimensionality reduction and clustering. *Semi-supervised Learning* is mostly concerned with learning with data that is only partially labeled. Oftentimes, the goals are the same as in supervised learning, but labelling the entire dataset might not be possible or economically feasible. In *Reinforcement Learning*, one or multiple agents act in an environment and learn through active exploration and exploitation inside the environment from the feedback it provides. Active areas of research for Reinforcement Learning include robotics and solving complex games [SB98].

2.2.1 Neural Networks

While a throughout description of all important aspects of neural networks and Deep Learning is out of the scope of this work, we still want to briefly discuss the two relevant core concepts: layer-based architectures and gradient-based optimization, as they share similarities and partially acted as an inspiration for variational circuit classifiers, which are the focus of this work. For an extensive introduction into the field we refer to the textbook by Goodfellow et al. [GBC16]. The following content is based on [GBC16] and [LBH15].

Firstly, neural networks are composed by neurons and weights which are combined into various layers. These layers can be of different size and implement different functions, but they have to be compatible with each other for chaining them together to compute a result. As such, a neural network can be seen as a stacked sequence of individual layers, without being explicitly concerned with every single weight that is part of it. In Deep Learning, stacking a high number of layers, along with nonlinear transformations between them, is successfully used for state-of-the-art methods in the fields of Computer Vision and Natural Language Processing (NLP).

The second import concept we want to mention is gradient-based optimization. Neural networks are optimized by minimizing a cost function C. This cost function usually measures how well the networks predictions on a dataset compare to the true labels. If the cost function is differentiable, its gradient with respect to the weights can be used to update the weights $\boldsymbol{\theta}$ in such a way, that the updated values minimize the cost function. This can be summarized in the gradient descent update rule, which updates the weights $\theta_i \in \boldsymbol{\theta}$ in iteration $t+1$ using the weight values from the previous iteration t, and the loss gradient $\nabla_{\theta_i} C(\boldsymbol{\theta})$:

$$\theta_i^{(t+1)} \leftarrow \theta_i^{(t)} - \eta \nabla_{\theta_i} C(\boldsymbol{\theta}) \tag{2.16}$$

where $\eta \in [0, 1]$ is the *learning rate*. The loss gradient can be computed using the backpropagation algorithm. As we will see later, this useful optimization can be achieved with quantum circuits, as the gradient of the circuit with respect to inputs can be evaluated on quantum hardware [SBG+19].

2.2.2 Reinforcement Learning

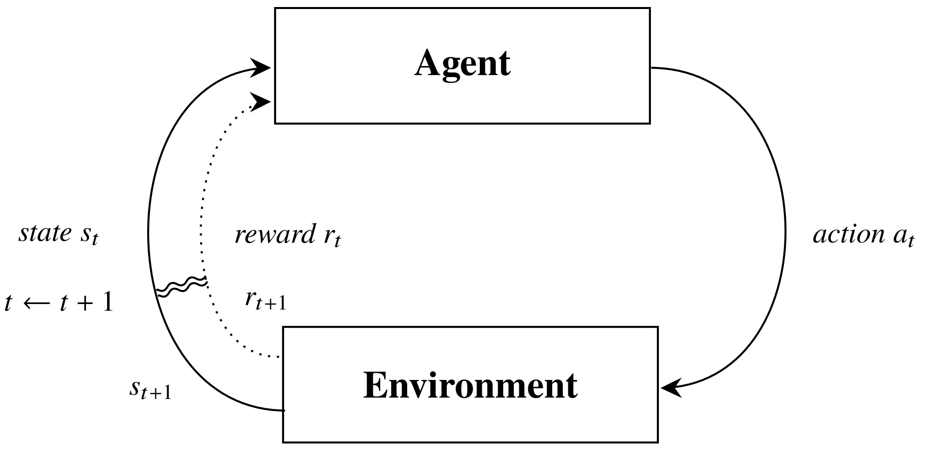


Figure 2.5: Generic Reinforcement Learning Agent-Environment-loop, adapted from [SB98]

The term Reinforcement Learning describes a class of methods where one or multiple agents are placed in an environment for interactive task solving. The research field is influenced by artificial intelligence, especially robotics, and classical control theory. In the following, we follow the formalism from the textbook by Sutton et al. [SB98]. First, we introduce the terminology of Reinforcement Learning:

Environment and Agent The environment describes the space that a learning agent is placed in to solve tasks. While the environment can be either simulated or real, the agent is usually following a strategy that computationally defines how to explore and 'act' in the environment. The goal of an agent in an environment is to learn to solve the specified task in the environment. During the learning phase, it follows a search strategy to explore and learn a strategy to exploit the experienced knowledge in order to solve the task.

States A state s describes an observable part of the environment that the agent is in. That description might not necessarily be complete, as the environment might only be partially observable. For example, a robot tasked with navigating in a room might only have access to a camera image representing the room and the robots position. *Terminal* states denote a goal state, after which the environment usually experiences some kind of reset to an initial state. A terminal state also marks the end of an *episode*. The State space $\mathcal{S}$ describes the space of all possible states.

Actions When in a state s, an agent can perform an action a from an action space $\mathcal{A}$. This results in a new state $s' \in \mathcal{S}$ and a reward signal. From then on, the agent is again tasked with performing an action, given the state s'. This loop is also illustrated in Figure 2.5. An agents interaction in an episode with the environment gives rise to a sequence

$$T = ((S_0, A_0, R_1), (S_1, A_1, R_2), \ldots \tag{2.17}$$

which is also referred to as a *trajectory*. For example, a robot tasked with navigating from point A to B might get a small negative reward of -1 for every part of the trajectory it takes towards point B, but a huge positive reward of 100 for actually stepping into B, and thus reaching a terminal state that finishes the trajectory. The rewards therefore encourage the robot to reach B with as few, well thought-out actions as possible, taking the direct route, as to minimize the potential negative reward that accumulates from taking detours.

Policy A policy is a learned function that the agent uses to navigate in the environment. For example, after every step, the previously mentioned robot directing towards a goal point is tasked with deciding in which direction to navigate next. A policy accomplishes this task. Usually, the policy is essentially random at the beginning of the learning phase. The agent uses interactions with the environment in order to estimate a policy that maximizes its expected rewards over time. Formally, we define a policy π to be a function that, conditioned on a state $s \in \mathcal{S}$, outputs the probability of choosing an action $a \in \mathcal{A}$.

$$\forall a \in \mathcal{A}, s \in \mathcal{S} : \pi(a|s) \in [0, 1] \quad \forall s \in \mathcal{S} : \sum_{a \in \mathcal{A}} \pi(a|s) = 1 \tag{2.18}$$

The agents goal is to approximate the optimal policy π^* that, starting from any state s_t, chooses an action a_t that maximizes the *expected discounted return* G_t it can expect when following the optimal policy afterwards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} y^k R_{t+k+1} \tag{2.19}$$

The parameter $\gamma \in [0, 1]$ is called *discount rate* and refers to a measure of how important future rewards are for the current timestep t. Usually, rewards that are farther away in the future are discounted, as to put more emphasis on rewards that can be collected in the short-term. In order to construct a policy it can use to base its decisions on, an agent usually keeps track of experienced rewards for the actions it performs in certain states. Oftentimes, this is accomplished by keeping track of value estimates, which capture the expected discounted returns for every state-action pair that is encountered. The *state-action value function* $q_\pi(s, a)$ formalizes these value estimates for choosing action a in state s. It is defined as the expected value of returns that can be achieved, when following a policy π, after choosing an action a in state s:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{2.20}$$

Using these state-action value estimates, a policy can be derived. For example, a simple greedy policy can constructed as follows: for every state s that is encountered, choose the action $a \in \mathcal{A}$ which maximizes the state-action value function $q(s, a)$.

Q-Learning

Building on the state-action value function q, the *Q-Learning* [WD92] algorithm can be used as a learning agent. As a temporal difference control algorithm, it can be used to iteratively estimate the state-action action value function Q by modifying previous estimates. When action A_t is performed in state S_t, a reward R_{t+1} is received along with a successor state S_{t+1}. Then, the update to the value function Q follows the Bellman equation update:

$$Q(S_t, A_t) \leftarrow \underbrace{Q(S_t, A_t)}_{\text{current estimate}} + \underbrace{\alpha \left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right]}_{\text{weighted update of estimate}} \tag{2.21}$$

where α is the step size and γ the discount factor with $\alpha, \gamma \in [0, 1]$. Using the value update in Equation 2.21, the Q-Learning agent strategy can be constructed. The procedure is shown in Algorithm 2.1. The ε-greedy policy acts as follows: with probability ε, a random action is chosen, with probability $(1 - \varepsilon)$, action $\operatorname{argmax}_a Q(s, a)$ is chosen for a state s.

Algorithm 2.1 Q-Learning

```
procedure QLearning($\alpha \in (0,1]$, $\gamma \in (0,1]$, $\varepsilon > 0$)
    $Q(s,a) \leftarrow 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
    for each episode do
        $s \leftarrow$ initial state of environment
        for each step in episode do
            $a \leftarrow$ action following policy derived from current $Q$, e.g. using $\varepsilon$-greedy strategy
            $r, s' \leftarrow$ take action $a$ in environment    // taking action $a$ led to state $s'$ and reward $r$
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$
            $s \leftarrow s'$                                    // update current state
        end for
        if $s$ is terminal then
            finish episode
        end if
    end for
end procedure
```

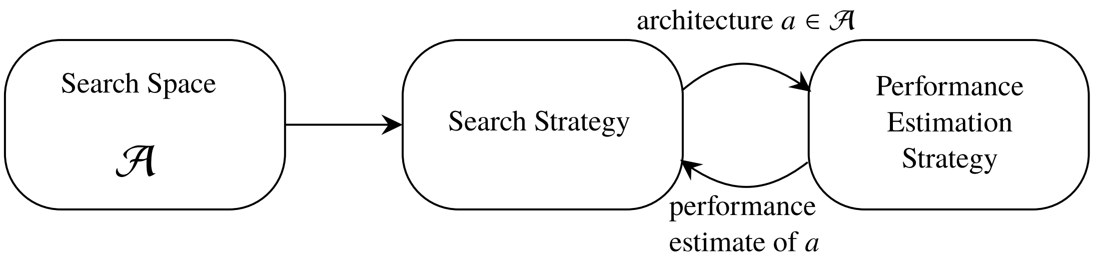

architecture $a \in \mathcal{A}$
Search Space
$\mathcal{A}$
Search Strategy
Performance Estimation Strategy
performance estimate of a

Figure 2.6: General Neural Architecture Search method illustrated with the main tasks, adapted from [EMH19]

2.2.3 Automated Machine Learning

The term *AutoML*, or automated Machine Learning, describes a variety of meta-modelling algorithms that are concerned with automatic optimization and structure search for hyper- and metaparameters of Machine Learning models [THHL13]. Depending on the Machine Learning algorithm, different aspects of the model are subject to manual design and hyperparameters, such as for example threshold values, learning rates or number of clusters that are predetermined.

Neural Architecture Search (NAS) is a family of AutoML techniques for the automated modelling neural network structures and hyperparameters, which mainly consists of choosing and appropriate sequence of neural network layers. When first introduced, the focus was on using evolutionary algorithms to design neural network architectures [SWE92] [SM02], while more recent work is mostly based on Reinforcement Learning [BGRN17][ZL16][LDY]. The process of searching for a suitable neural network structure for a given dataset in NAS methods can be generally divided into three components following Elsken et al. [EMH19]:

Search Space The search space is the space of valid network architectures than can be found. It contains all structures that can be constructed using the available building blocks, rules and constraints. Oftentimes, the search space is induced by defining a set of valid parts, so-called *layers* or *cells* alongside a set of rules, both of which are usually defined manually and stem from experience and the observation of previously proposed architectures.

Search Strategy The search strategy defines how the space of network architectures is explored with the goal of finding well-performing architectures according to measures of performance. Special considerations in the strategy have to be made to prevent problems such as early convergence, as the space is usually high dimensional and might have multiple nonoptimal local maxima.

Performance Estimation Strategy The best network architectures are usually those with high performance and generalization capabilities on test data that was not part of the dataset that was used for training. While this type of performance evaluation is common, it might impose a burden if the training process takes too long. Performance estimation allows for the formulation of methods to evaluate the architectures performance consuming less resources, such as by using parameter sharing schemes or training on smaller subsets of the data to achieve lower fidelity estimates of the actual performance.

2.3 Quantum Machine Learning

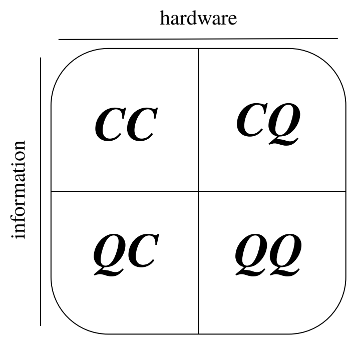


Figure 2.7: Quantum Machine Learning typology table, adapted from [ABG06]

The term *Quantum Machine Learning* (QML) generally describes the involvement of quantum computation or quantum information in Machine Learning methods [SP18]. It is considered a promising area of research in the NISQ-era [PBRB18]. Early work was concerned with translating classical Machine Learning methods to run entirely or mostly on quantum hardware [HHL09] [RML14]. The practicality of those algorithms, especially for near-term devices, is debatable [Aar14]. While this inspired algorithms on classical hardware to adopt techniques proposed in quantum-related research [Tan18a] [GLT18] [Tan18b] [CLW], the current focus of research lies on using the inherent capabilities of quantum computers to an advantage [PBRB18][SFP17], rather than translating classical work. The involvement of quantum computation and quantum information in Machine Learning can be categorized in four types according to Aïmeur et al. [ABG06], as displayed in Figure 2.7. Generally, one differentiates between the involvement of quantum and classical computing resources, and whether the data comes in the form of quantum information or classical information. This leads to a distinction of four types of Quantum Machine Learning:

CC - Classical Information on Classical Hardware This describes the conventional case, where classically encoded data, for example in the form of vectors or images, is used to train a model using exclusively classical hardware. Most research and development in the field of Machine Learning is focused on this case.

QC - Quantum Information on Classical Hardware Here, the focus is on classical Machine Learning models that run on conventional hardware, which are fed with quantum or quantum-related information. Examples include learning from the measurement data of quantum computers, or performing prediction tasks of physical behaviour of quantum systems [MNK+18]. We note that the dataset must be made available in data structures on classical hardware, which makes the difference between *QC* and *CC* opaque, at least for the Machine Learning model.

CQ - Classical Information on Quantum Hardware When denoting hardware as *quantum*, we do not restrict ourselves to exclusive usage of quantum computers. As described later, the two types of computing resources can be used in conjunction. Generally, quantum hardware refers to the involvement of quantum computation resources in the computation or learning process. Much like in the *QC*-type, classical information also has to undergo a transformation to be available in a quantum computer. In theory, all conventional Machine Learning tasks can also be accomplished with the involvement of quantum hardware, provided that the quantum hardware can scale along with classical hardware.

QQ - Quantum Information on Quantum Hardware In the special case of quantum-related datasets that use quantum hardware for learning, a possibly time-consuming quantum encoding procedure of classical data might be prevented. This leads to different possible scenarios, such as for example placing a QML procedure that directly receives inputs from physical experiments in a superposition state.

We note that this work focuses on the types *CQ* and to a degree *QQ*. That is, we do not assume the data to be originating from either a classical or quantum source. Furthermore, we are concerned and therefore focus on supervised learning, especially the task of classification.

2.3.1 Data Encoding

In the most general case, classical information in the form of data samples is provided to Machine Learning models. In the case of classification, a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{l}$ is provided, where $x \in \mathbb{R}^d$ as well as $y \in \mathbb{R}$ are samples of classical information. We explicitly state here that we assume datapoints to be vectors over real numbers. As previously mentioned, special consideration has to be paid to the encoding and representation of data in a quantum computer.

Amplitude Encoding

In this work, we rely on encoding a datapoint into the amplitudes of quantum registers. This approach of data encoding is called *Amplitude Encoding* [SBSW18]. In order to properly represent a datapoint in the state of a quantum registers, consideration has to be paid to the constraints that a quantum state imposes. A datapoint thus first has to undergo a preprocessing procedure, in which its dimensions are padded to the nearest higher power of 2, which requires $\lceil \log_2 d \rceil$ qubits. Here, uninformative padding is preferred, where every datapoint is padded equally. This can be accomplished using either zero-padding, or padding using a constant $c \in \mathbb{R}$. Secondly, a quantum state has to be of length 1, the padded datapoint therefore has to be normalized. The resulting vector can then be represented as a quantum state, which needs to be prepared first. Assuming an initial ground state of $|0 \ldots 0\rangle$, there exist procedures on quantum computers to prepare an arbitrary state [MVBS04] [SP18]. In the remainder of this work, we consider the state preparation-, or amplitude encoding-procedure that encodes the i-th dimension of the preprocessed datapoint x into the i-th amplitude of a quantum register to be provided and denote it as $\mathcal{S}_x$. The visual notation of the procedure in a quantum circuit is shown in Figure 2.8.

Feature Maps

In addition to necessary preprocessing for amplitude encoding, datapoints can be manipulated before they are encoded for classification. This includes embedding datapoints into different vector spaces using *feature maps*. For example, these mappings allow linear models to linearly separate data in a higher-dimensional space, which may not be possible in the *original* space.

Schuld et al. [SBSW18] propose to use a *tensorial* feature map in the context of Quantum Machine Learning. This feature mapping maps a quantum state to a new state that results from applying the tensor product to multiple copies of the input state:

$$\varphi' : |\psi\rangle \mapsto |\psi\rangle^{\otimes n} \quad n \in \mathbb{N}^+ \tag{2.22}$$

A popular example for feature maps in conventional Machine Learning models is the *polynomial* feature map, which is comparable to tensorial feature maps [HTFF05]. Feature maps have a strong relation to so-called *kernels*, which provide explicit formulas to compute inner products in feature spaces, without actually performing the feature mapping. This makes them popular in algorithms such as Support Vector Machines or Gaussian Processes [HTFF05].

Data Dimensionality

In the context of NISQ devices, scalability imposes a constraint, especially for Machine Learning models. Because there is only a limited number of qubits available, datapoints cannot exceed a certain dimension. To address this issue, there exist several approaches for *dimensionality reduction*, such as principal component analysis, neural network autoencoders, slow feature analysis or or feature hashing [GBC16][KL18][HTFF05]. While the remainder of this work does not make use

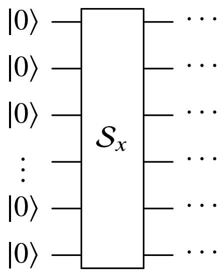


Figure 2.8: State preparation circuit that encodes a datapoint $x \in \mathbb{R}^d$ into the amplitudes of the $|0\rangle^{\otimes n}$ state

of dimensionality reduction techniques, we argue that special consideration has to be paid to the scalability, as the dimensionality does play a role in the feasibility of QML algorithms for real-world use-cases.

2.3.2 Hybrid Quantum-Classical Algorithms

With the limitations and constraints of NISQ devices, real-world quantum speedup and scalability are not to be expected in the near future when relying exclusively on quantum computers, or more generally Quantum Processing Units (QPU). A promising group of algorithms and methods that overcome at least some limitations in the NISQ-era are the so-called *hybrid Quantum-Classical Algorithms*, or *variational* quantum algorithms. In general, these quantum algorithms have free parameters and other tuneable parts that run on quantum hardware, but are (partially) controlled using classical computation, thus the term *hybrid* is used. Comparable to other specialized hardware such as Graphical Processing Units (GPU), in this setting, a QPU is considered as a computational resource that can be leveraged for certain parts of an algorithm that benefit from the potential speedup or resource efficiency, while topics such as the main control flow and persistence are handled by classical hardware. This hybrid setting can therefore mitigate limitations of quantum hardware, such as decoherence time or scalability. On the other hand, due to the overhead of communicating and encoding classical to quantum information and vice versa, pure quantum speedup might be lost due to long processing times.

Another characteristic of variational quantum algorithms are free parameters, such as rotational degrees of rotation gates, which are optimized towards an objective. This optimization is then usually carried out on classical hardware and can leverage the full theoretical and practical support in the form of research and high-efficiency libraries for optimization. Furthermore, due to dynamic adaptation of the parameters for a specific QPU, the characteristics of the topology and properties of individual qubits are implicitly taken into consideration during the optimization process. This leads to more robustness and device-tailored algorithms that, in contrast to standard textbook algorithms, take into consideration the characteristics of the quantum device that algorithms run on.

An impactful variational quantum algorithm presented is the *Variational Quantum Eigensolver* (VQE) [PMS+14]. It can be used to compute the smallest eigenvalue of a Hamiltonian describing a system, building on the *variational principle* [SC95]. Another example is the *Quantum Approximate Optimization Algorithm* (QAOA) [FGG14], which can be used for combinatorial optimization, such as for computing the `MaxCut` of a graph.

2.3.3 Variational Circuit Classifier

With the recent shift in focus for the research community towards *"NISQ-ready"* quantum algorithms, several proposals for the usage of parametrized quantum circuits for classification have been made [SBSW18] [FN18] [HCT+19] [MNKF18]. The general approach is generally referred to as a *Variational Circuit Classifier*, *Quantum Neural Networks* or *Quantum Circuit Learning*. An overview for various approaches and techniques is provided in [BLS19]. We follow the nomenclature and framework presented by Schuld et al. [SBSW18] in the following description. Furthermore, we limit the scope of this work to binary classification, which discriminates data between two classes.

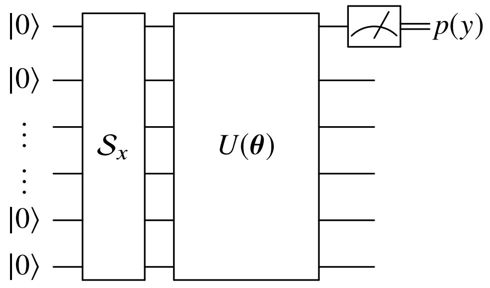


Figure 2.9: Generic variational circuit for classification consisting of a state preparation layer, a parametrized model circuit structure and a measurement for a classification result

The variational circuit classifier describes a parametric quantum-classical hybrid computation model, in which the free parameters of a parametrized circuit are optimized to minimize an object function using gradient-based optimization on classical hardware in order to solve a classification task using labeled samples of an underlying distribution.

Classification

An illustration for the general structure of a variational circuit is depicted in Figure 2.9. In order to classify a datapoint $x \in \mathbb{R}^d$ using the classifier, it is preprocessed and encoded in the quantum circuit, for example using amplitude encoding. The resulting quantum state is $\varphi(x)$. The remainder of the quantum circuit contains gates, of which a nonempty subset contains free parameters $\theta_i \in \boldsymbol{\theta}$. Common parametrized gates are rotation and controlled-rotation gates, which perform a rotation operation on the target qubit iff the control-qubit is set. The entirety of the parametrized unitary $U(\boldsymbol{\theta})$ acts on $\varphi(x)$, resulting in the state $\varphi'(x) = U(\boldsymbol{\theta})\varphi(x)$. In the end, a measurement on $\varphi'(x)$ is performed to obtain classical information. In the case of binary classification, without loss of generality, the classification can be realized by measuring the first qubit q_1 in the σ_z measurement basis. The probability of observing the first qubit in state $|1\rangle$ is then interpreted as the probability $p(q_1 = 1, x; \theta)$ of datapoint x belonging to a class y, which is the sum of squared amplitudes of the states in which the first qubit is set to 1:

$$p(q_1 = 1, x; \theta) = \sum_{k=2^{n-1}+1}^{2^n} |(\varphi'(x))_k|^2 \tag{2.23}$$

Together with a bias parameter $b \in \mathbb{R}$ that is added during classical postprocessing

$$\pi(x; \theta, b) = p(q_1 = 1, x; \theta) + b \tag{2.24}$$

the result can then be thresholded by a function f that assigns a class label to x_i:

$$f(x; \theta) = \begin{cases} 1 & \text{if } \pi(x; \theta, b) > 0.5 \\ 0 & \text{else} \end{cases} \tag{2.25}$$

Optimization

Optimization of the classifier happens in an iterative hybrid-setting, illustrated in Figure 2.10, as the parameters $\boldsymbol{\theta}$, including the bias parameter b, are controlled by classical hardware. The goal of the training process is to optimize values for the parameters that minimize the classification error for a dataset $\mathcal{D}$. Comparable to training the weights of deep neural networks, the free parameters start out with random values or are determined by educated guesses [GB10]. The performance using the current parameter values on the data is then assessed by calculating the error of the results compared to the labels. This is achieved by using a so-called loss function, for example the mean-squared error function (MSE). Using gradient information of the circuit, the parameters are updated with the aim of minimizing the loss function for the next iteration. Gradients of quantum circuits can be computed according to a differentiation scheme presented by Schuld et al. [SBG+19]. Because parameters are stored and optimized on classical hardware, they do not have to be held in a quantum state. Problems with decoherence times are therefore mitigated, as information is held in classical storage. Furthermore, the optimization uses gradient information that is evaluated on the quantum device, which leads to device-specific optimization that takes properties of the individual qubits such as fidelity of operations and noise behaviour into account.

Circuit Structures

While a variational quantum circuit contains trainable parameters, the general structure, also called *ansatz*, of the circuit is predetermined. In their work, Schuld et al. present a circuit structure for n qubits that consists of repeated placements of the unitary

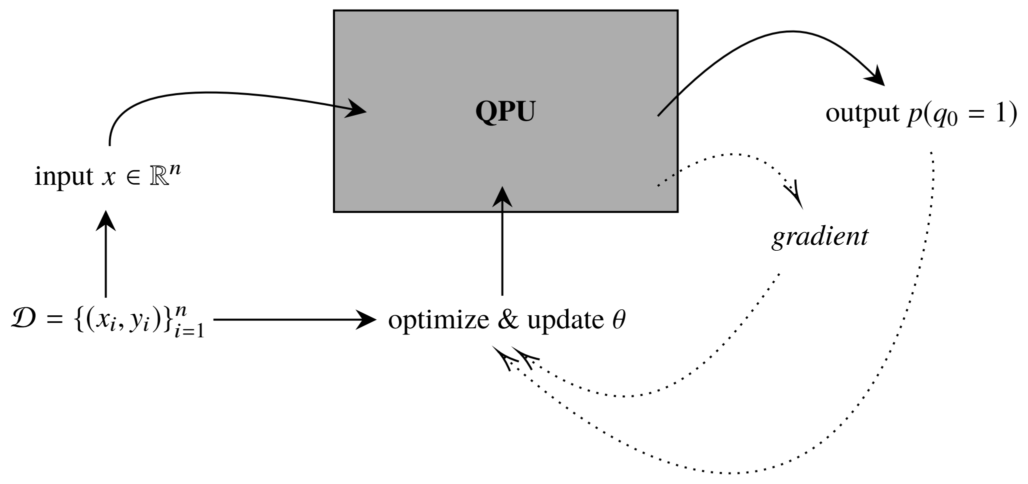

QPU
output $p(q_0 = 1)$
input $x \in \mathbb{R}^n$
gradient
$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
optimize & update θ

Figure 2.10: Hybrid training scheme for variational quantum circuits adapted from [SBSW18]. Here, *QPU* denotes a Quantum Processing Unit, which describes any quantum computation resource that can execute quantum circuits, including simulations.

$$U_i = \text{CNOT}^{\otimes n} \circ R_z(\theta)^{\otimes n} \circ R_y(\theta)^{\otimes n} \circ R_x(\theta)^{\otimes n} \tag{2.26}$$

after the state preparation procedure to construct $U(\boldsymbol{\theta})$. Here, n refers to the number of qubits and $\text{CNOT}^{\otimes n}$ refers to a layer of cascading CNOT gates, defined as:

$$\text{CNOT}^{\otimes n} = \prod_{i=0}^{n-1} \text{CNOT}((i+1), ((i+1) \bmod n) + 1) \tag{2.27}$$

$\text{CNOT}((i+1), ((i+1) \bmod n) + 1)$ denotes the controlled NOT operation acting on the
$(((i+1) \bmod n) + 1)$-th qubit conditioned on the $(i+1)$-th qubit. It is used to generate entangled quantum states. A visual depiction of the resulting circuit representation of U_i can be seen in Figure 2.11. To construct $U(\boldsymbol{\theta})$, U_i is concatenated d times after a state preparation circuit to construct the parametrized circuit.

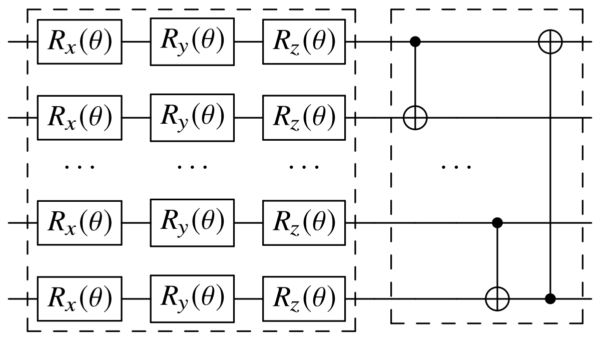


Figure 2.11: Layer cells building blocks proposed by Schuld et al. [SBSW18]. These layers are repeatedly placed to construct a circuit structure with free parameters

3 Related Work

Chapter 2 was concerned with providing foundations necessary for this work. In the following Chapter we present related work in the field of automated quantum circuit generation. Especially, approaches for specialized NISQ-ready device-specific quantum circuit compilation are examined. Furthermore, in order to provide a wider context for existing approaches in Machine Learning research, several Neural Architecture Search methods are briefly outlined.

3.1 Quantum Circuit Structure

Grimsley et al. [GEBM] propose the *Adaptive Derivative-Assembled Pseudo-Trotter ansatz Variational Quantum Eigensolver* (ADAPT-VQE) procedure to dynamically find a shallow circuit with a preferably small number of parameters by iteratively growing an initially empty circuit structure and optimizing the *Variational Quantum Eigensolver* [PMS+14] procedure in a joint manner. In every iteration, candidate operators from a predefined set of available operators are placed in the ansatz. Then, the operator with the largest gradient of the energy function with respect to each candidate operator is selected in a greedy manner until a convergence criterion is met.

Khatri et al. [KLP+18] describe *Quantum-assisted Quantum Compiling* (QAQC), a variational hybrid algorithm for dynamically finding a device-specific quantum circuit using a trainable unitary V for implementing a known optimal target unitary U. They propose to differentiate between the continuous free parameters of individual quantum gates and the discrete set of gates that make up the circuit structure. For continuous parameters, gradient-free as well as gradient-based optimization is used. In order to evaluate the corresponding cost function on the quantum device, they propose to incorporate the *Hilbert-Schmidt-Test* (HST). Structural changes of the circuit ansatz are performed iteratively in a pertubative fashion based on simulated annealing, where an existing circuit is modified by randomly replacing a subset of gates with new candidate gate structures from a set of gate structures. If the cost of the newly proposed circuit decreases after optimization of the continuous parameters, the changes to the circuit are accepted, otherwise they are rejected. This procedure of structural optimization is performed iteratively until convergence. We note that this approach attempts to optimize the circuit structure in a greedy manner, without keeping track of a measure of *value* or *contribution* for the subsets of gates. Additionally, the algorithm needs knowledge a known *optimal* unitary on an optimal quantum computer, which would paradoxically imply knowledge about an optimal classification model in the context of variational quantum circuit classifiers.

The work of Cincio et al. [CSSC18] similarly uses random changes in the gate sequence with continuous parameter optimization to optimize the circuit structure. Different from QAQC, in their problem setting, they do not require a known target unitary and rely on labelled samples that the targeted unitary or circuit would produce instead. This essentially comes down to learning a circuit structure that solves a regression problem for a given dataset $\mathcal{D} = \{x_i, f(x_i)\}_{i=1}^{n}$, where f is the function to implement on the quantum computer. We want to point out the similarity to our approach, in that it aims to construct circuits that solve a supervised learning task. While we focus on the task of classification in our approach, an extension to the task regression is possible with modifications.

Ostaszewski et al. [OGB19] present an approach for circuit structure learning that also jointly optimizes circuit structure as well as rotation angle parameters. Their approach relies on properties of the expectation value of the Hermitian of a circuit.

The general idea of using Reinforcement Learning for dynamically learning circuit structures has already been explored by McKiernan et al. [MDAR19] in the context of learning a circuit structure for solving the `MaxCut` problem for graphs. Instead of Q-Learning, they propose to use *Proximal Policy Optimization* (PPO) [SWD+17], a policy-gradient-based agent strategy using deep neural networks for function approximation. Furthermore, in order for the agent to generalize to unseen graph instances, the authors propose to encode a representation of the entire graph into the state representation of the environment during training. This inherently leads to longer training time for the agent, because in addition to solving the `MaxCut` problem using the state representation, it has to find a generalized representation of the problem representation in the state. The PPO-agent was trained for approximately 1.7 million episodes on graphs to solve the `MaxCut`-problem and observed better performance on average than with a generic QAOA ansatz with $p = 1$. We want to add to this that policy-gradient based methods in practice, while considered powerful, are considered to have a high sample complexity. Additionally, encoding the problem instance into the state of an environment is not a feasible option for supervised learning, as the model performance highly depends on the dataset that is provided.

Evolutionary algorithms for circuit modification share characteristics with Reinforcement Learning approaches. Both have to balance between exploration and exploitation [Dav91] [SB98]. Different from Reinforcement Learning, evolutionary approaches use a different search strategy, made up of operations that are inspired by evolution observed in nature. The recent work of Potoček et al. [PRFC] explores a multi-objective evolutionary approach, applying standard genetic operators such as but not limited to the insertion, replacement, swapping or deletion of random subsequences of quantum circuits and applies it to Grover and Fourier Transform.

3.2 Neural Architecture Search for Deep Learning

Baker et al. [BGNR16] use Q-Learning to optimize the structure of image classifiers based on Convolutional Neural Networks (CNN). In their experiments, tabular Q-Learning compares favourably to model-free policy-gradient methods when comparing training time in order to generate high-performing architectures [Bak17].

Zoph et al. [ZL16] [ZVSL18] similarly generate architectures for CNNs, but generate architecture sequences using Recurrent Neural Networks (RNN). In their proposed framework, the policy gradient-method PPO is used for training this sequence generator to output high performing architectures.

In a recent approach by Liu et al. [LDY], a framework for differentiable architecture search called *DARTS* is proposed. In their formulation, the discrete space of architectures becomes continuous and differentiable, which allows for gradient-based optimization towards higher performing network architectures.

3.3 Summary

While device-specific circuit structure generation approaches exist, they mainly rely on essentially random modifications to existing circuit structures. Such approaches arguably do not scale well to larger circuits, as they do not memorize the effect of placements of subsets of circuit structures over time. Our approach systematically links building blocks of a circuit to a *value*, enabling to first follow a similar approach to random replacement in an initial, explorative phase, but to later make use of these memorized value estimates for proposing more suitable circuit structures.

4 Approach

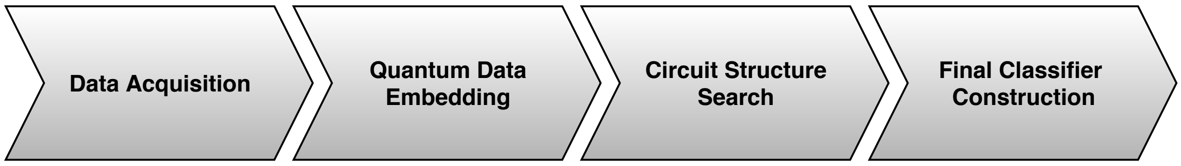


Figure 4.1: Sequence diagram for the general workflow of the end-to-end approach

In this Chapter, we present our approach for the automated generation of variational quantum circuit structures in an end-to-end manner for supervised learning tasks. We divide the proposed workflow into four main tasks, as can be seen in Figure 4.1. The tasks range from collecting a dataset, appropriately preparing the data for use with a variational quantum circuit classifier, to dynamically exploring circuit structures in order to construct and finally output a well-performing classification model. The novelty in our approach lies in the fact that circuit structures are systematically explored using Reinforcement Learning methods following key concepts of Neural Architecture Search, which aim to estimate the value of parts of the circuit structure. We also show a way to regulate the depth of explored circuits, which reduces the number of gates and takes decoherence time of qubits into consideration. This allows for the proposal of shallow-depth circuit structures, which is highly desirable for near-term devices. The following sections describe every task of the workflow in detail.

4.1 Data Acquisition

While the performance of any supervised learning task is data-driven and therefore influenced by the quality and properties of the data that is available, desirable properties for a dataset are difficult to assess generally. As datapoints are samples of an underlying distribution, in general, enough of these samples have to be available in order to represent the distribution, as new samples that have to be classified in the future also originate from it. A dataset $\mathcal{D}$ can also undergo transformations before it is made available to any learning procedure in order to enrich it. For example, domain expertise can be applied and be used to determine new or enhanced features. Furthermore, in the context of NISQ devices, the dimensionality of features of a datapoint have to be taken into account. Because quantum computational systems having scaling limitations in terms of available qubits in the near term, the datasets that are being considered for use with them have to reflect that. To mitigate this problem, there exist techniques for dimensionality reduction that can be applied here. Finally it is to be noted that, while for the most part we assume data to be available as classical information, it can also originate from quantum experiments. In this case, data is already quantum encoded, the next step may be skipped in case no further transformations are required.

4.2 Data Quantum Embedding

In the case of data that is in the form of classical information, it has to undergo a preprocessing in order to be ready for use with a quantum computation device. To this end, we aim encode datapoints according to the Amplitude Encoding scheme outlined in Chapter 2. Following a standard normalization procedure, which applies uninformative padding and normalizes the length, feature maps can be applied to further enrich the classifiers capabilities to find linearly separate the dataset in the mapped feature space. The resulting datapoints are then ready to be encoded into the amplitudes of a quantum state. In the case of data that is already available as quantum information, different possibilities arise. For example, a more complex preprocessing or embedding can take place. An interesting area of research is finding feature maps or transformations for datapoints that are considered difficult to compute classically, which can lead to a quantum speedup. The output of the embedding task is a dataset $\mathcal{D}_\phi$, to which feature maps have already been applied. The labels y_i are unaffected by transformations and do not require a quantum embedding, as they are only classically processed and compared with the output of measurements, which is also presented to the optimizer in the form of classical information.

4.3 Circuit Structure Search

With having a 'quantum-ready' preprocessed dataset $\mathcal{D}_\phi$ available, a suitable circuit for the classification task has to be found. As with the overwhelming majority of conventional machine learning models, certain aspects of the model are tuneable or subject to predefined hyperparameters. For instance, convolutional neural networks are usually composed by a stack of layers of different types, such as convolutional layers, pooling layers and fully-connected layers [GBC16]. Each of these types has different hyperparameters at different stages of the network that are fixed for the network architecture during training and inference, such as the size of the convolution- and pooling operators, the layer width or activation function. In the case of the variational quantum circuit model, the circuit architecture acts as a metamodel, or blueprint, for the trained circuit. The difference lies only in the values of the free parameters $\boldsymbol{\theta}$, as the structure itself is fixed. Section 2.3.3 presented a proposed circuit structure that is constructed using repeated building blocks for rotations and entanglement of qubit registers. While they provide the parametrized model with many degrees of freedom, the number of repetitions of these building blocks still has to be manually determined. Furthermore the fully flexible rotations with three degrees of freedom in addition to cascading CNOT gates imply a relatively high resource usage and do not encourage shallow-depth circuits. In order to eliminate the need for a human in-the-loop to propose and define circuit structures, we aim to automatically find resource-efficient and well-performing circuit structures by active exploration. We propose to frame the problem of circuit structure search as an instance of Neural Architecture Search that we solve using a tabular Q-Learning agent for Reinforcement Learning. Figure 4.2 provides a detailed depiction for the task. We use a learning agent to construct quantum circuits, which are evaluated by a training procedure and then used to estimate a value function that can be used to propose well-performing circuits. Although neural networks and variational quantum circuit classifiers do have some fundamentally differentiating characteristics, we hypothesize that, along with an appropriate problem formulation, core concepts from Neural Architecture Search can be transferred and accelerate research in the specialized QML domain. In the following, we describe a way to compose quantum circuits, formalize the problem and present a way to solve it.

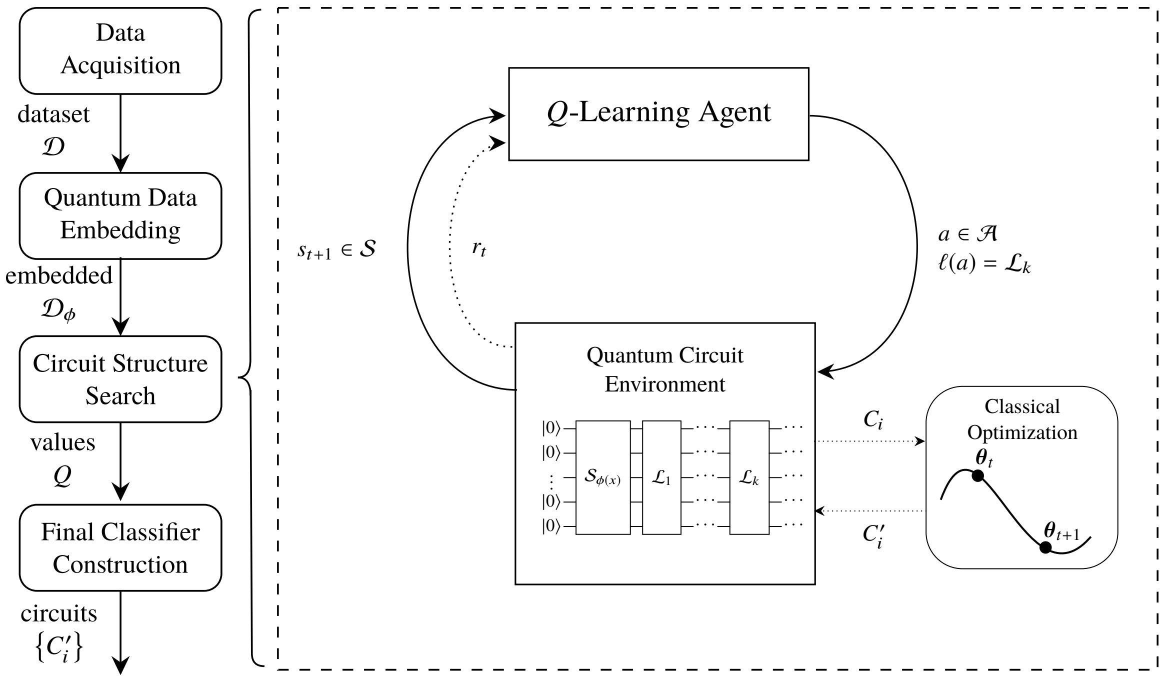


Figure 4.2: Workflow of the approach (left) with their respective outputs and detailed illustration of the Circuit Structure Search step

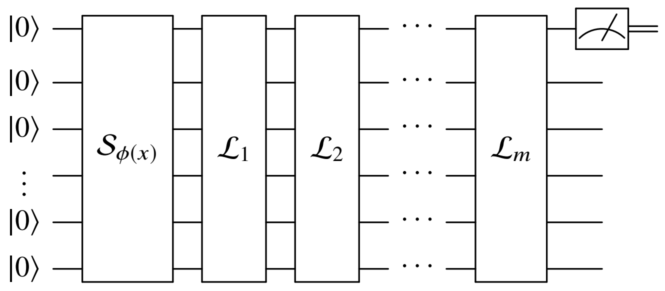


Figure 4.3: Generic quantum circuit for classification composed by a state preparation circuit $\mathcal{S}_{\phi(x)}$, a stack of layer blocks $\mathcal{L}_1, \ldots, \mathcal{L}_m$ and a measurement operation.

4.3.1 Circuit Layers

In order to abstract the placement of individual gates in quantum circuits to facilitate the automatic generation, we view circuits as a sequence of individual unitaries, or *layers*, as illustrated in Figure 4.3. A layer is therefore defined as a unitary acting on the entire qubit register at a specific timestep t. From an algorithmic viewpoint, each layer acts as steps of an algorithm at time t, manipulating the current quantum state $|\psi_{t-1}\rangle$, resulting in the state $|\psi_t\rangle$, where $|\psi_0\rangle$ is the initial state:

$$|\psi_m\rangle = \mathcal{L}_m \circ \cdots \circ \mathcal{L}_1 |\psi_0\rangle \tag{4.1}$$

As mentioned, Schuld et al. also proposed to compose circuits for the variational quantum circuit classifier using layer, or blocks of unitaries [SBSW18]. In order to provide more flexibility for the construction, and with the goal of constructing short-depth circuits we propose to use a set of layers $\mathbb{L}_n$ that splits the fully rotational layer into their 'primitive' unitaries:

$$\mathbb{L}_n = \left\{R_x(\theta)^{\otimes n}, R_y(\theta)^{\otimes n}, R_z(\theta)^{\otimes n}, \mathrm{CNOT}^{\otimes n}, \mathcal{L}^\tau\right\} \tag{4.2}$$

Here, n refers to the number of qubits and $\mathrm{CNOT}^{\otimes n}$ refers to a layer of cascading CNOT gates. $\mathcal{L}^\tau$ is a *terminal* layer and indicates the measurement operation on the first qubit. We note that more and modified layers can be added to the proposed set $\mathbb{L}_n$, such as more high-level building blocks that can compose algorithms. Furthermore, the layers can be of different varying granularity.

4.3.2 Quantum Circuit Environment

In order to solve the Circuit Structure Search problem using Reinforcement Learning methods, we propose the definition of an environment in which quantum circuits can be iteratively constructed by agents using layers as building blocks. We define the state and action space of the environment, discuss reward signals and outline the relation to the main components of Neural Architecture Search.

Action Space

In order to define the action space $\mathcal{A}$ of the environment, we first define a bijective function $\ell : \mathcal{A} \rightarrow \mathbb{L}_n$ that maps every action from the action space to a specific layer. The action space $\mathcal{A}$ is then induced by the image of ℓ^{-1}. Upon choosing an action $a_t \in \mathcal{A}$ in state s_t, the layer associated with the action is appended to the circuit.The agent can notably only perform modifications to the existing circuit in the environment by appending a layer to the current structure. While more flexible formulations of the action space could have been made, such as allowing for insertions between any pair of previously placed layers, they generate an exponentially larger set of state-action pairs, increasing the training time. Here, a tradeoff between accuracy and complexity has to be made. Choosing the terminal action corresponding to the layer $\mathcal{L}^\tau$ marks a circuit as *complete*. Any subsequent action will not result in any change of the state until the environment is reset, which marks a new *episode* for an agent. Every circuit C_i that can be constructed by the agent is part a set $\mathbf{C}$. It is finite and forms the search space in the Neural Architecture Search framework.

State Space

Every state $s_t \in \mathcal{S}$ is defined as a tuple of the most recently placed layer and the current circuit depth, which is equal to the number of layers placed at timestep t:

$$s_t = (\ell(a_{t-1}), t) \tag{4.3}$$

The state of the environment is therefore only described by the most recently added layer along with the current number of layers. A similar state space was also used by Baker et al. [BGNR16] for Neural Architecture Search for CNNs. In order to define a finite state space, we limit the number of layers that can be placed in a circuit. Although imposing a circuit depth limit is not necessary by construction, it mitigates issues, such as overly long optimization times if the number of free parameters becomes too large, as the analytical differentiation scheme used requires several evaluations on the quantum device for each parameter. Furthermore, different than simulations, real-world NISQ devices imply a depth limit for the circuit with their relatively short decoherence times.

Reward Signals

After every step that an agent takes in an environment, a new state along with a reward signal is returned to the agent. In our approach, after the placement of every layer a reward of 0 is rewarded to the agent, except for placing the terminal layer $\mathcal{L}^\tau$. The terminal layer is also placed automatically if the predefined depth limit for a circuit is exceeded. After the terminal layer is placed, the reward signal is determined by a training procedure, which optimizes the parameters of the circuit C_i using optimization on classical hardware as described in Section 2.3.3. The resulting circuit with optimized parameters C_i' is then used to evaluate the performance of the circuit structure for the variational quantum circuit classifier for the dataset $\mathcal{D}_\phi$. The resulting accuracy is then fed back to the agent as a reward. This sparse reward setting, where for example an informative reward is only given for reaching the goal state, is not uncommon in Reinforcement Learning tasks and emphasizes the requirement for agents to be able to learn from long-term dependencies and propagate reward information along trajectories.

4.3.3 Q-Learning Agent for Quantum Circuit Construction

This section briefly outlines properties of the Q-Learning algorithm formulation that are useful in the context of our problem at hand. First, the design of the reward function is discussed, followed by a method for regulating the depths of generated circuits. Then, initial value estimates are examined. The section is concluded with a discussion on scheduling the exploration in the space of circuit structures.

Reward Shaping

As described, we define the environment to only give out non-null rewards when a circuit is completed. Using domain knowledge, especially about the quantum computer that the classifier should be executed with, a more complex reward function can be designed. This way, rewards can be used to signal the agent to favour circuit with certain properties over other, an can be ’nudged’ to create more suitable circuits, especially in the context of near-term devices. In the following, we aim to provide pointers to domain-specific considerations for the reward function with regards to the construction of quantum circuits:

Classifier Accuracy The classifiers accuracy is vital to assess the actual performance of the classifier and therefore considered the dominant reward signal. A reward function should therefore always refer to this signal in some way.

Circuit Depth As low-depth circuits are considered preferable in the NISQ-era, because of decoherence properties of the qubits, the reward function can reflect that. For example, instead of outright not allowing circuits to exceed a certain depth, small or, depending on the design, negative reward signals can be introduced, which can discourage the agent from exploring deep circuit structures to instead focus on shorter ones.

Gate Cost Similarly to circuits regarding their depth, the execution of certain gates can be more costly than others on some quantum computer implementations. Especially multi-qubit gates can impose a higher execution time than others. If the qubit-topology is not fully connected, a multi-qubit gate affecting qubits that are not connected in the topology can lead to a sequence of CNOT gates that have to be inserted into the circuit during the transpilation stage to swap qubit states in order to map the intended operation to the qubits. The cost of added operations and the potential introduction of noise can be reflected in the reward function, encouraging the agent to only place expensive gates or layers when they are necessary.

Regulating Circuit Depth

A common shortcoming of NISQ devices is their limited decoherence time and therefore limited number of operations that can be executed with a certain level of reliability [Pre18]. The formulation of the value estimate update in the Q-Learning algorithm allows to use the parameter γ, called the *discount factor*, for regulation of circuit depths. Depending on the value of γ, rewards that are propagated backwards during the value update are discounted along the along the trajectory. Suppose the value of γ to be smaller than 1, and two constructed circuit structures with depths d_1 and d_2, where $d_1 < d_2$. If both structures received the same reward for the placement of the terminal layer, and a reward of 0 otherwise, the trajectory for the circuit with depth d_1 yields a higher expected discounted return for choosing its initial action when used repeatedly for value updates until convergence.

Value Initialization

Before an untrained agent starts interacting with an environment, the value function has to be initialized. In the case of tabular Q-Learning, this means that every state-action value $Q(s,a), s \in \mathcal{S}, a \in \mathcal{A}$ is assigned with a default value. We resort to the commonly found initialization [SB98] with

$$Q(s,a) = 0 \qquad s \in \mathcal{S}, a \in \mathcal{A} \tag{4.4}$$

but want to point out the possibility of biased initial estimates [SB98]. In this setting, some states, or state-action pairs, might be heuristically preferred over others, and therefore can be initialized with a higher initial value than others. In relation to our work, this means that choosing certain layers at certain depths might be favoured. For example, a previously proposed circuit can act as a heuristic, resulting in slightly higher initial values for the layers found in its structure, which leads the agent to at least initially prefer choosing these layers for circuit construction. Over time, these initial estimates are either further enforced, or discarded as better performing circuit structures are consistently found.

Exploration versus Exploitation

Another parameter that can be actively used to tune the exploration of circuit structures is the value of ε, which is used in the ε-greedy strategy when deriving a policy from the value function Q. Usually, ε starts out with a high initial value, for example 1, which is then decayed over time to a value closer to 0. When ε is is initially deliberately set to a high value, this leads the agent to *explore* the space of circuits it can generate, because most actions will be chosen at random with probability ε. Doing so is important for the agent, as otherwise states and actions encountered early on during the learning phase are biased, which can lead to unfavourable early convergence and the exploration of only a small subspace. When the value is decayed, the agent can use its value estimates it constructed from experience to act more greedily and *exploit* the structures more to maximize the expected return.

4.4 Final Classifier Construction

The final step of the workflow is to provide a suitable circuit structure for use with a variational quantum circuit classifier. After the training procedure, the agent has accumulated value estimates for every state-action pair by interacting with the environment. At this point, using this information, we have different options to construct quantum circuits.

One option is to use the highest-performing classifier that was encountered during the training phase, which is the circuit that yielded the highest accuracy in the variational quantum classifier setting. This has the advantage that trained parameters are already available and the classifier can be used without performing optimization of parameters again. On the other hand, if the performance estimation strategy for example only uses a subset of the data for training during the structure search loop, this results in lower fidelity estimates. Retraining the circuit is advisable in this case, taking advantage of the entire dataset that is available.

The other option is proposing new circuit structures using a policy derived from the value function that was learned during circuit structure search. A deterministic policy proposes exactly one action, in our case a layer, for every state it encounters:

$$\pi(s) = a \quad a \in \mathcal{A}, s \in \mathcal{S} \tag{4.5}$$

During the Q-Learning procedure, the ε-greedy policy was used, building an estimate of the policy, which does not yield one action but a probability distribution over actions. To sample a circuit from the subspace of circuits, that can be constructed using a combination of layers from the defined set of layers, then means sampling from the policy by sampling from the probability distributions. The distribution has probability $1 - \varepsilon$ for sampling the layer that yields the highest expected return

$$\operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a) \tag{4.6}$$

and probability

$$\frac{\varepsilon}{|\mathcal{A}|} \tag{4.7}$$

for choosing a random layer.

5 Validation

This Chapter describes the prototypical implementation of the ideas presented in the approach detailed in Chapter 4. The prototype was developed in Python[1] and aims for a modular and reusable design. In addition, the approach is validated in an experimental setting.

5.1 Implementation

In the following we describe the components of our prototypical implementation. It is structured according to the architecture depicted in Figure 5.1. In addition to interactions between the agent and environment, additional components have been introduced in the implementation to achieve higher modularity, reusability and a separation of concerns.

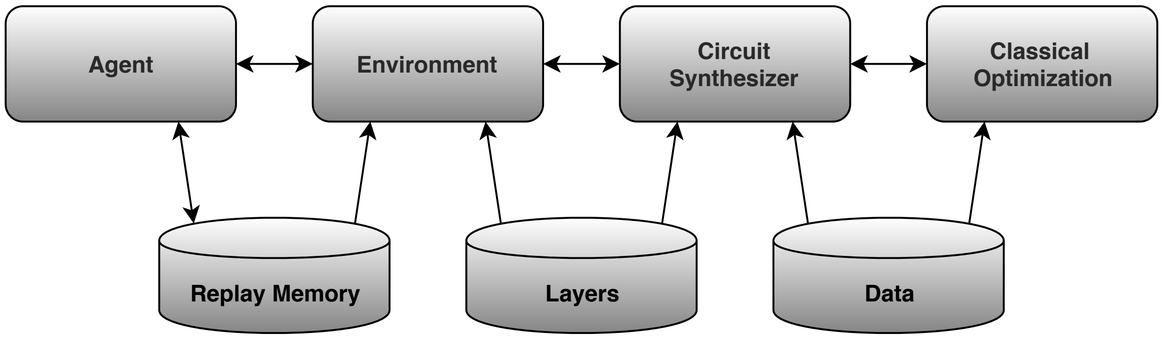


Figure 5.1: Conceptual architecture for the prototype

5.1.1 Q-Learning Agent

We modified the Q-Learning procedure in the implementation by adding a replay memory for experience replay [Lin93] [MKS+15]. This memory buffer stores the resulting trajectories of finished episodes and has the benefit of decorrelating the trajectories from their temporal order. The training procedure of the agent is split into two parts because of it, as indicated in Figure 5.2. In the first part, episodes are played out regularly using the current value estimates from Q. The trajectories of the episodes are then stored and persisted in the replay memory. The memory conveniently also doubles as a cache, storing all rewards, including the accuracy of the optimized circuit. If an identical circuit is constructed again by the agent, the stored rewards are reused and the circuit parameters are not optimized again from scratch, which increases the performance of

[1] `https://www.python.org/`

the entire search process. The update procedure, where the Q function is updated according to the experiences from the trajectories, is decoupled from the episode loop and is only performed in a fixed interval, happening after every k episodes have been played out. In order to update the Q function, a fixed number of trajectories, in our case this corresponds to the constructed circuits along with the received rewards, is sampled from the replay memory and used to update the value estimates, similarly to the procedure described in Algorithm 2.1, with a slight modification. Instead of using a sampled trajectory for updates unmodified, we reverse the order in which the state-action pairs are updated, which empirically results in faster convergence, as the value of all state-action pairs involved in a trajectory are updated at once.

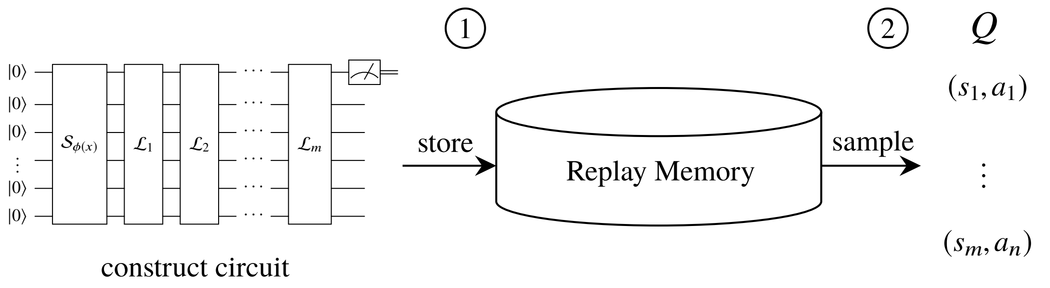


Figure 5.2: Conceptual illustration of the modified Q-Learning update procedure using a replay memory. **1**: Trajectories are generated using the current Q-value estimates and stored in a replay memory. **2**: Trajectories are randomly sampled from the replay memory for updates of the value function

5.1.2 Environment

The environment implements an interface very similar to the standardized interfaces in the `gym`[2] library, a popular framework for benchmarking Reinforcement Learning implementations. This achieves a decoupling from the search strategy that the agent component uses. Furthermore, the environment is decoupled from the task of actual circuit construction and execution, as it only exposes an abstracted view of actions and states to the agent.

5.1.3 Circuit Synthesizer

The environment externalizes the task of handling quantum circuits directly by introducing a *Circuit Synthesizer* in order to achieve higher modularity and separation of concerns. The synthesizer manages the construction and implementation of quantum circuits based on the available layers. It furthermore manages what backend device is used for the execution of a quantum circuit. The actual construction of quantum circuits is implemented using *PennyLane* [BIS+18], which is a framework for gradient-based optimization of quantum circuits and mainly developed by *Xanadu*[3]. We consider the framework to be suitable for our use-case because of the following capabilities:

[2] `https://gym.openai.com/`

[3] `https://www.xanadu.ai/`

Computational Graph Paradigm The paradigm of structuring computations in the form of a directed graph, as described by Abadi et al. [ABC+16] and Murray et al. [MMI+13] is popular for Machine Learning, especially Deep Learning, with support from major libraries. Individual *computational nodes* have a defined input and output format and are connected with *directed edges*, forming a computational graph. This allows for effective and clean distribution of computations, their respective resources, internal states and coordination of distributed systems. *PennyLane* supports this paradigm by providing nodes labeled `QNodes`. `QNodes` provide an abstraction for how computations are actually executed on a quantum device or simulation. For the global view on the graph, the definition of the input and output format suffices. *PennyLane* manages the calculations of gradients of `QNodes` with a differentiation scheme described in [SBG+19][BIS+18].

External Library Support *PennyLane* supports integration with the existing popular Machine Learning libraries as backends. Currently *TensorFlow* [ABC+16] and *PyTorch* [PGC+17] are officially supported. When using either backend, computations of classical and quantum type can be combined into a single machine learning model. The hybrid quantum-classical optimization part of the circuits is also realized in a computational graph setting, where the optimizer is a classical computation node. The extensive support of these major libraries allows the usage of their optimized routines such as built-in optimizers and integrating quantum computational parts into existing, classical models.

Open Source Software The *PennyLane* development repository is publicly available[4] and is licensed under the Apache License 2.0 [AL2] license. This allows for contributions by community members and grants the rights to modify and extend the framework. Nowadays, Open Source Software (OSS) is considered a driving force of innovation in many areas including Machine Learning research and development. We argue that the same holds true for Quantum Machine Learning research.

Device agnostic Quantum devices in *PennyLane* are entirely plugin-based. Every `QNode` is associated with such a `device`-instance that it is executed on. Therefore, multiple devices can be used for multiple different `QNodes` in composition. New devices can be connected by implementing a standardized interface. Devices expose their supported gatesets and operations and provide means to provide an input to the device and receive an output. This makes it trivial to switch out devices and train and test circuits on different devices. For example, initial parameter optimization for a parametrized circuit can be performed using a simulated quantum device for efficiency, while fine-tuning for specific hardware can be accomplished using further training of the circuit on real quantum hardware.

5.1.4 Classical Optimization

When a previously untrained, completed circuit is encountered in the environment, a training procedure for the variational quantum circuit classifier is triggered. In the beginning, every free parameter $\theta_i \in \boldsymbol{\theta}$ of the circuit is uniformly random initialized with a value in $[-\pi, \pi)$. The dataset

[4] `https://github.com/XanaduAI/pennylane`

episodes	ε
1-300	1.0
301-600	0.8
601-960	0.5
961-1080	0.2
1081-1200	0.05

Table 5.1: Schedule for the value of ε over the entire training procedure

$\mathcal{D}_\phi$ is shuffled and split into sets for training and testing. Then, the parameters $\boldsymbol{\theta}$ are optimized in a hybrid setting, as described in Chapter 2, using the Adam optimizer [KB14], a variant of the stochastic gradient descent optimizer with an adaptive stepsize for better convergence properties.

5.2 Experiments

In order to validate our approach, we conduct experiments where the goal is to find a classifier for a synthetic dataset. Quantum computation was performed on a simulated quantum computer using four qubits. We perform circuit structure search for 1200 episodes with a Q-Learning agent with $\alpha = 0.2$ and a scheduled value of ε. The schedule is listed in Table 5.1. For the parameter optimization of variational quantum circuit classifiers, we use the Adam optimizer [KB14] with a mean-squared error loss for 10 epochs, a learning rate of 0.1 and a batch size of 4. Additionally, we limit the number of layers that can be placed by an agent to 10. This setup is executed multiple times for different values of the discount rate γ.

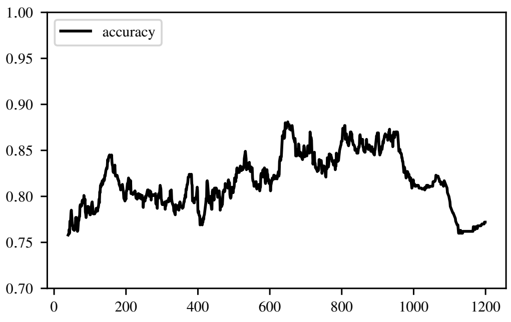
5.2.1 Dataset

We used a two-dimensional, balanced, synthetically generated 'moons' dataset $\mathcal{D}$ with 100 datapoints, where two halfmoon-shaped point clouds, each belonging to one class, interleave. It was generated using the `make_moons` function from the *scikit-learn* Python library[5]. The data was preprocessed with a tensorial feature map ϕ:

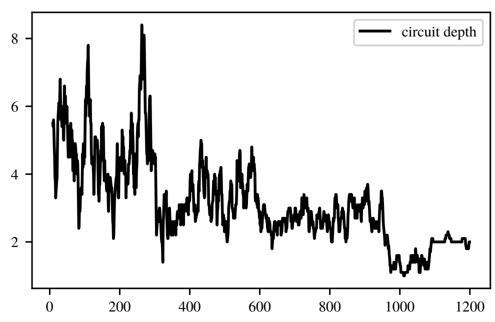
$$\phi : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \otimes \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2 x_1 \\ x_2^2 \end{bmatrix} \tag{5.1}$$

and normalized, such that $\left\|\phi(x_i)\right\| = 1$ for every $(x_i, y_i) \in \mathcal{D}$.

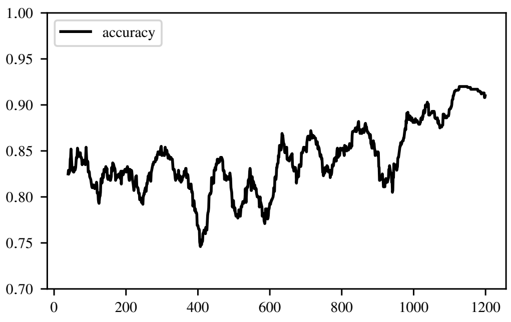
[5] https://scikit-learn.org/



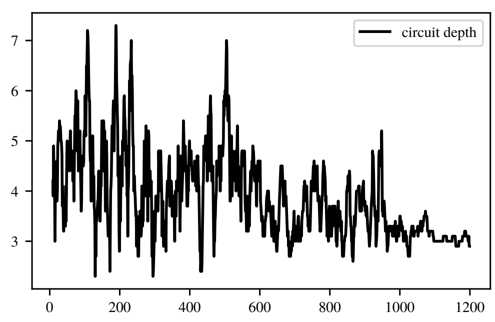
(a) $\gamma = 0.6$



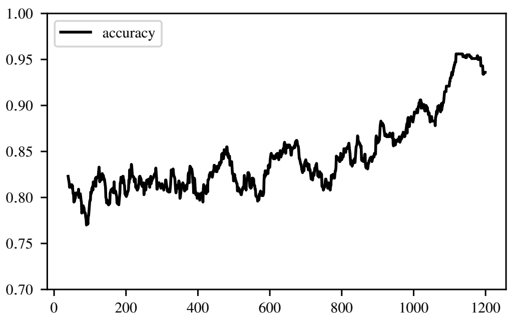
(b) $\gamma = 0.6$



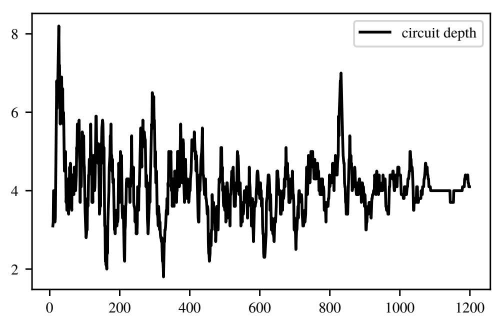
(c) $\gamma = 0.9$



(d) $\gamma = 0.9$



(e) $\gamma = 1.0$



(f) $\gamma = 1.0$

Figure 5.3: Accuracy (left) and circuit depth (right) plotted over time for different values of γ. The plots are smoothened using a moving average with window size 40 for accuracy and window size 10 for circuit depth.

5.2.2 Results

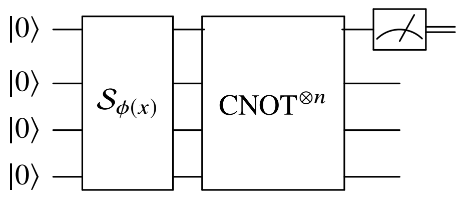
Figure 5.3 shows the results for γ at 0.6, 0.9 and 1.0. As previously described, this parameter acts as a circuit depth regularizer, as smaller values encourage the agent to exploit the rewards from shorter circuits. Over time, the accuracy tends to rise after the initial exploration phase, where ε has a value of 1 and uniformly random circuits are generated.

For $\gamma = 0.6$, we can observe that the average performance decreases towards the end (Figure 5.3 (a)). At the same time, we can see that the average circuit depth makes stepwise jumps when the value of ε is decreased (Figure 5.3 (b)). After ε is scheduled to a value of 0.2, model accuracy drastically suffers, as choosing a lower depth circuit becomes a dominating force. This can be interpreted as a tradeoff between accuracy and complexity of the model. The staircasing effect of the average circuit depth along with the suffering average accuracy leads to our interpretation that the choice of γ is too aggressive at 0.6. This is confirmed when observing the accuracy curves in the Figures 5.3 (c) and (e). Here, the choice of γ, while providing an incentive for the agent to construct shorter circuits, does not sacrifice performance as much. As such, we can observe that the agent produces the best results on average in the last episodes (Figure 5.3 (e)), with an average circuit depth of around 4. The choice of γ at 0.9 seems like a reasonable compromise, regulating the average circuit depth to around 3 in the last few episodes(Figure 5.3 (f)), with a steadily rising average accuracy towards the end of training.

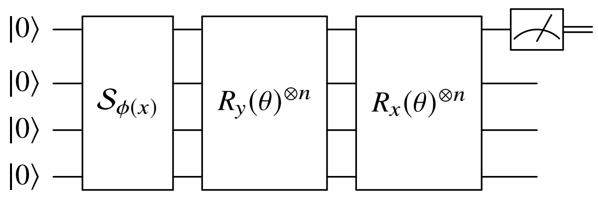
Interestingly, when observing the most common structures that the agent proposes for various values of γ after ε was decayed to 0.05, as depicted in Figure 5.4, it becomes apparent that the circuit structures in (b) and (c) are very similar. The two subsequent $R_y(\theta)^{\otimes n}$ layers in (c) can be merged into one rotational unitary, leaving two rotations, similarly to the structure seen in (b). Another interesting observation is that, for our dataset, no entangling layers were used for construction of the final classifiers, except for the structure proposed by the agent with $\gamma = 0.6$. Because a measurement only occurs on the first qubit, this means that the agents with γ chosen at 0.9 and 1.0 only make use of one qubit in their proposed computations. The gates performing operations on the other qubits can therefore be dropped, which can easily be detected and transpiled in a compilation step.

5.2.3 Summary

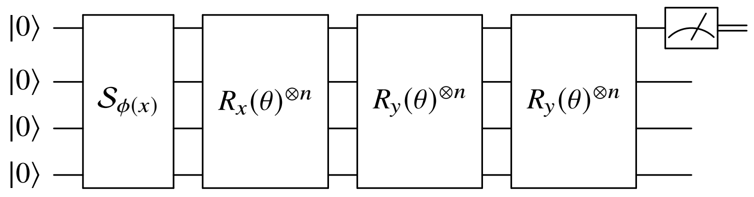
This Chapter outlined our implementation and presented experiments which validate our general approach for finding circuit structures. We observed that the learning agent can be trained propose structures that maximize the expected return over time. Furthermore, we can manually encourage the agent to favour certain structures over others, as demonstrated with the depth of the proposed circuits. The experiments additionally showed an important insight which should be reflected in the environment. As duplicate layers following each other immediately in the circuit can, for our proposed set of layers, be merged and are therefore redundant, the repeated placement could for example be punished with a negative reward.



(a) $\gamma = 0.6$



(b) $\gamma = 0.9$


$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$\mathcal{S}_{\phi(x)}$
$R_x(\theta)^{\otimes n}$
$R_y(\theta)^{\otimes n}$
$R_y(\theta)^{\otimes n}$

(c) $\gamma = 1.0$

Figure 5.4: Most common circuit structures generated by the agent after ε was set to 0.05 for various values of γ

6 Conclusion and Future Work

This Chapter briefly recapitulates the the presented work with a summary as well as providing an outlook for Quantum Machine Learning research in the near future. First, foundations of quantum computation and Machine Learning lead up to an introduction of Quantum Machine Learning, where the variational quantum circuit classifier was outlined. We then presented related work in the domains of automated quantum circuit generation and Neural Architecture Search. This is followed by a detailed description of our approach, which dynamically searches for quantum circuit structures that perform well on a classification task for a dataset. The approach is then validated in a prototypical implementation and experiments using a simulated quantum computer.

6.1 Conclusion

Work on variational quantum algorithms forms one of the most promising areas of research in quantum computation for applications in the near-term, as typical paradigms of Machine Learning, such as the effect of noise to generelization properties, can be mapped to NISQ devices. We base the motivation for our approach on the desire of non-experts in the field of Quantum Machine Learning to make use of this research by introducing self-supervised metamodelling capabilities with our approach based on Reinforcement Learning. The validation indicates that automatically constructing circuit structures that optimize an objective function using our approach is possible. In contrast to previous work, information and feedback that is received during explorative phases is used to make more informative guesses for future actions. Additionally, the approach fits well into the NISQ-era, as special considerations have been actively made towards handling the constraints that the properties of near-term devices imply.

6.2 Future Work

The flexible problem formulation extends beyond the specialized research area of Quantum Machine Learning and can also be incorporated generally for algorithms running on quantum hardware, as the design of quantum algorithms is oftentimes considered to be unintuitive for humans [Sho03], which we leave for future work. Furthermore, the design of more complex layers that involve more powerful operations can be explored. The general formulation of Neural Architecture Search also leaves room for further refinements. For example, the performance estimation strategy can be optimized for large datasets by making use of lower fidelity estimates, as listed in [EMH19], or by extrapolating the learning curve of the classifier as proposed in [BGRN17]. As for the search strategy, trying to use agent strategies beyond Q-Learning can open new possibilities. Especially when distributed Reinforcement Learning is introduced, which can construct and search for quantum circuits on many devices in parallel. The optimization process of the classifier can be further modified, leaving room for further research on hyperparameters of the learning process, or benchmarking the effect of using different optimizers, such as the recently presented Quantum Natural Gradient [SIKC19]. After circuit structures are found, additional techniques for transpilation, compilation and further optimization could be used [NRS+18].

In the domain of QML, further research can be conducted on the possibility of learning circuit structures for algorithms other than variational quantum circuit classifiers, such as quantum kernel computing circuits for the Quantum Support Vector Machine [HCT+19].

Bibliography

[AAA+19] H. Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: `10.5281/zenodo.2562110` (cit. on p. 21).

[Aar14] S. Aaronson. “Quantum Machine Learning Algorithms: Read the Fine Print”. In: *Nature Physics* (2014), p. 5. ISSN: 1745-2473. DOI: `doi:10.1038/nphys3272`. URL: `https://www.scottaaronson.com/papers/qml.pdf%0Ahttp://www.scottaaronson.com/blog/?p=2196` (cit. on p. 26).

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. “TensorFlow: A system for large-scale machine learning”. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283 (cit. on p. 49).

[ABG06] E. Aimeur, G. Brassard, S. Gambs. “Machine Learning in a Quantum World”. In: *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer. 2006, pp. 431–442 (cit. on p. 26).

[AL2] *APACHE LICENSE, VERSION 2.0*. Version 2. Apache Software Foundation, Jan. 1, 2004. URL: `https://www.apache.org/licenses/LICENSE-2.0` (cit. on p. 49).

[Bak17] B. Baker. *Towards Practical Neural Network Meta-Modeling*. MIT, 2017 (cit. on p. 36).

[BGNR16] B. Baker, O. Gupta, N. Naik, R. Raskar. “Designing Neural Network Architectures using Reinforcement Learning”. In: (2016), pp. 1–18. URL: `http://arxiv.org/abs/1611.02167` (cit. on pp. 36, 43).

[BGRN17] B. Baker, O. Gupta, R. Raskar, N. Naik. “Accelerating neural architecture search using performance prediction”. In: *arXiv preprint arXiv:1705.10823* (2017) (cit. on pp. 25, 55).

[BIS+18] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, C. Blank, K. McKiernan, N. Killoran. “PennyLane: Automatic differentiation of hybrid quantum-classical computations”. In: (2018), pp. 1–12. URL: `http://arxiv.org/abs/1811.04968` (cit. on pp. 48, 49).

[BLS19] M. Benedetti, E. Lloyd, S. Sack. “Parameterized quantum circuits as machine learning models”. In: *arXiv preprint arXiv:1906.07682* (2019) (cit. on p. 30).

[CBS+18] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, J. M. Gambetta. *Validating quantum computers using randomized model circuits*. Tech. rep. 2018. arXiv: `1811.12926v1`. URL: `https://arxiv.org/pdf/1811.12926.pdf` (cit. on p. 21).

[CLW] N.-H. Chia, H.-H. Lin, C. Wang. *Quantum-inspired sublinear classical algorithms for solving low-rank linear systems*. Tech. rep. arXiv: `1811.04852v1`. URL: `https://arxiv.org/pdf/1811.04852.pdf` (cit. on p. 26).

[CSSC18] L. Cincio, Y. Subaşı, A. T. Sornborger, P. J. Coles. “Learning the quantum algorithm for state overlap”. In: *New Journal of Physics* 20.11 (2018), p. 113022 (cit. on p. 35).

[Dav91] L. Davis. “Handbook of genetic algorithms”. In: (1991) (cit. on p. 36).

[Deu89] D. E. Deutsch. “Quantum computational networks”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 425.1868 (1989), pp. 73–90 (cit. on p. 18).

[DiV00] D. P. DiVincenzo. “The physical implementation of quantum computation”. In: *Fortschritte der Physik: Progress of Physics* 48.9-11 (2000), pp. 771–783 (cit. on p. 20).

[EMH19] T. Elsken, J. H. Metzen, F. Hutter. *Neural Architecture Search: A Survey*. Tech. rep. 2019, pp. 1–21. URL: http://jmlr.org/papers/volume20/18-598/18-598.pdf (cit. on pp. 25, 55).

[Fey82] R. P. Feynman. “Simulating physics with computers”. In: *International journal of theoretical physics* 21.6 (1982), pp. 467–488 (cit. on pp. 13, 15).

[FGG14] E. Farhi, J. Goldstone, S. Gutmann. “A quantum approximate optimization algorithm”. In: *arXiv preprint arXiv:1411.4028* (2014) (cit. on p. 30).

[FN18] E. Farhi, H. Neven. “Classification with Quantum Neural Networks on Near Term Processors”. In: (2018), pp. 1–21. URL: http://arxiv.org/abs/1802.06002 (cit. on p. 30).

[GB10] X. Glorot, Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256 (cit. on p. 31).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 21, 28, 40).

[GEBM] H. R. Grimsley, S. E. Economou, E. Barnes, N. J. Mayhall. *An adaptive variational algorithm for exact molecular simulations on a quantum computer*. Tech. rep. arXiv: 1812.11173v2. URL: https://arxiv.org/pdf/1812.11173.pdf (cit. on p. 35).

[GLT18] A. Gilyén, S. Lloyd, E. Tang. *Quantum-inspired low-rank stochastic regression with logarithmic dependence on the dimension*. Tech. rep. 2018. arXiv: 1811.04909v1. URL: https://arxiv.org/pdf/1811.04909.pdf (cit. on p. 26).

[Gro96] L. K. Grover. “A fast quantum mechanical algorithm for database search”. In: *arXiv preprint quant-ph/9605043* (1996) (cit. on p. 17).

[HCT+19] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, J. M. Gambetta. “Supervised learning with quantum-enhanced feature spaces”. In: *Nature* 567.7747 (2019), pp. 209–212. ISSN: 14764687. DOI: 10.1038/s41586-019-0980-2 (cit. on pp. 30, 56).

[HHL09] A. W. Harrow, A. Hassidim, S. Lloyd. “Quantum algorithm for linear systems of equations”. In: *Physical Review Letters* 103.15 (2009), pp. 1–15. ISSN: 00319007. DOI: 10.1103/PhysRevLett.103.150502 (cit. on p. 26).

[HTFF05] T. Hastie, R. Tibshirani, J. Friedman, J. Franklin. “The elements of statistical learning: data mining, inference and prediction”. In: *The Mathematical Intelligencer* 27.2 (2005), pp. 83–85 (cit. on pp. 21, 28).

[JL03] R. Jozsa, N. Linden. “On the role of entanglement in quantum-computational speed-up”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 459.2036 (2003), pp. 2011–2032 (cit. on p. 18).

[KB14] D. P. Kingma, J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 50).

[KL18] I. Kerenidis, A. Luongo. “Quantum classification of the MNIST dataset via Slow Feature Analysis”. In: (2018), pp. 1–25. URL: `http://arxiv.org/abs/1805.08837` (cit. on p. 28).

[KLP+18] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, P. J. Coles. “Quantum-assisted quantum compiling”. In: (2018). URL: `http://arxiv.org/abs/1807.00800` (cit. on p. 35).

[LBH15] Y. LeCun, Y. Bengio, G. Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436 (cit. on p. 21).

[LDY] H. Liu, K. S. Deepmind, Y. Yang. *DARTS: Differentiable Architecture Search*. Tech. rep. arXiv: `1806.09055v2`. URL: `https://github.com/quark0/darts` (cit. on pp. 25, 37).

[Lin93] L.-J. Lin. *Reinforcement learning for robots using neural networks*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993 (cit. on p. 47).

[LMR+17] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, C. Monroe. “Experimental comparison of two quantum computing architectures”. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3305–3310 (cit. on p. 13).

[MBJ+19] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, M. Martonosi. “Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM. 2019, pp. 1015–1029 (cit. on p. 21).

[MDAR19] K. A. McKiernan, E. Davis, M. S. Alam, C. Rigetti. “Automated quantum programming via reinforcement learning for combinatorial optimization”. In: *arXiv preprint arXiv:1908.08054* (2019) (cit. on p. 36).

[MKS+15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529 (cit. on p. 47).

[MMI+13] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, M. Abadi. “Naiad: a timely dataflow system”. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM. 2013, pp. 439–455 (cit. on p. 49).

[MNK+18] A. A. Melnikov, H. P. Nautrup, M. Krenn, V. Dunjko, M. Tiersch, A. Zeilinger, H. J. Briegel. “Active learning machine learns to create new quantum experiments”. In: *Proceedings of the National Academy of Sciences* 115.6 (2018), pp. 1221–1226 (cit. on p. 27).

[MNKF18] K. Mitarai, M. Negoro, M. Kitagawa, K. Fujii. “Quantum circuit learning”. In: *Physical Review A* 98.3 (2018), pp. 1–3. ISSN: 24699934. DOI: `10.1103/PhysRevA.98.032309` (cit. on p. 30).

[MRN+17] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, J. Martinis. “Commercialize Quantum technologies in five years”. In: *Nature News* 543.7644 (2017), p. 171 (cit. on p. 13).

[MVBS04] M. Möttönen, J. J. Vartiainen, V. Bergholm, M. M. Salomaa. “Quantum circuits for general multiqubit gates”. In: *Physical review letters* 93.13 (2004), p. 130502 (cit. on p. 28).

[NC00] M. Nielsen, I. Chuang. *Quantum Computation and Quantum Information*. 2000. URL: `https://aapt.scitation.org/doi/pdf/10.1119/1.1463744` (cit. on pp. 15, 16).

[NRS+18] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, D. Maslov. “Automated optimization of large quantum circuits with continuous parameters”. In: *npj Quantum Information* 4.1 (2018), p. 23 (cit. on p. 56).

[OGB19] M. Ostaszewski, E. Grant, M. Benedetti. *Quantum circuit structure learning*. Tech. rep. 2019. arXiv: `1905.09692v1`. URL: `https://arxiv.org/pdf/1905.09692v1.pdf` (cit. on p. 36).

[PBRB18] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, R. Biswas. “Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers”. In: *Quantum Science and Technology* 3.3 (2018), pp. 1–13. ISSN: 20589565. DOI: `10.1088/2058-9565/aab859` (cit. on p. 26).

[PGC+17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer. “Automatic differentiation in PyTorch”. In: (2017) (cit. on p. 49).

[PMS+14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O’brien. “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature communications* 5 (2014), p. 4213 (cit. on pp. 30, 35).

[Pre18] J. Preskill. *Quantum Computing in the NISQ era and beyond*. Tech. rep. 2018. DOI: `10.22331/q-2018-08-06-79`. arXiv: `1801.00862v3`. URL: `https://quantum-journal.org/papers/q-2018-08-06-79/pdf/?` (cit. on pp. 13, 20, 44).

[PRFC] V. Potoček, A. P. Reynolds, A. Fedrizzi, D. W. Corne. *Multi-objective evolutionary algorithms for quantum circuit discovery*. Tech. rep. arXiv: `1812.04458v1`. URL: `https://github.com/vasekp/quantum-ga` (cit. on p. 36).

[PZW] A. Paler, A. Zulehner, R. Wille. *NISQ circuit compilers: search space structure and heuristics*. Tech. rep. arXiv: `1806.07241v1`. URL: `https://arxiv.org/pdf/1806.07241.pdf` (cit. on p. 21).

[RML14] P. Rebentrost, M. Mohseni, S. Lloyd. “Quantum support vector machine for big data classification”. In: *Physical Review Letters* 113.3 (2014), pp. 1–5. ISSN: 10797114. DOI: `10.1103/PhysRevLett.113.130503` (cit. on p. 26).

[SB98] R. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning Series)*. 1998, p. 338. ISBN: 0262193981 (cit. on pp. 21, 22, 36, 45).

[SBG+19] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran. “Evaluating analytic gradients on quantum hardware”. In: *Physical Review A* 99.3 (2019), pp. 1–7. ISSN: 24699934. DOI: 10.1103/PhysRevA.99.032331 (cit. on pp. 22, 31, 49).

[SBSW18] M. Schuld, A. Bocharov, K. Svore, N. Wiebe. “Circuit-centric quantum classifiers”. In: (2018), pp. 1–17. URL: http://arxiv.org/abs/1804.00633 (cit. on pp. 28, 30, 32, 33, 42).

[SC95] J. J. Sakurai, E. D. Commins. *Modern quantum mechanics, revised edition*. 1995 (cit. on p. 30).

[SFP17] M. Schuld, M. Fingerhuth, F. Petruccione. “Implementing a distance-based classifier with a quantum interference circuit”. In: *arXiv preprint arXiv:1703.10793* (2017) (cit. on p. 26).

[Sho03] P. W. Shor. “Why haven’t more quantum algorithms been found?” In: *Journal of the ACM (JACM)* 50.1 (2003), pp. 87–90 (cit. on p. 55).

[SIKC19] J. Stokes, J. Izaac, N. Killoran, G. Carleo. “Quantum natural gradient”. In: *arXiv preprint arXiv:1909.02108* (2019) (cit. on p. 56).

[SM02] K. O. Stanley, R. Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127 (cit. on p. 25).

[SP18] M. Schuld, F. Petruccione. *Supervised Learning with Quantum Computers*. Vol. 17. Springer, 2018 (cit. on pp. 26, 28).

[SWD+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on p. 36).

[SWE92] J. D. Schaffer, D. Whitley, L. J. Eshelman. “Combinations of genetic algorithms and neural networks: A survey of the state of the art”. In: *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE. 1992, pp. 1–37 (cit. on p. 25).

[Tan18a] E. Tang. “A quantum-inspired classical algorithm for recommendation systems”. In: *arXiv preprint arXiv:1807.04271* (2018) (cit. on p. 26).

[Tan18b] E. Tang. *Quantum-inspired classical algorithms for principal component analysis and supervised clustering*. Tech. rep. 2018. arXiv: 1811.00414v1. URL: https://arxiv.org/pdf/1811.00414.pdf (cit. on p. 26).

[TBG17] K. Temme, S. Bravyi, J. M. Gambetta. “Error mitigation for short-depth quantum circuits”. In: *Physical review letters* 119.18 (2017), p. 180509 (cit. on p. 13).

[THHL13] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown. “Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 847–855 (cit. on pp. 13, 25).

[WD92] C. J. Watkins, P. Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292 (cit. on p. 24).

[ZL16] B. Zoph, Q. V. Le. “Neural Architecture Search with Reinforcement Learning”. In: (2016), pp. 1–16. URL: http://arxiv.org/abs/1611.01578 (cit. on pp. 25, 36).

[ZVSL18] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le. “Learning Transferable Architectures for Scalable Image Recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018), pp. 8697–8710. ISSN: 10636919. DOI: `10.1109/CVPR.2018.00907` (cit. on p. 36).

All links were last followed on October 24, 2019.

Declaration