# Dummy Titlepage

**Boye Gravningen Sjo**

Autumn 2022

*TMA4500*

Industrial Mathematics Specialisation Project

Department of Mathematical Sciences

Faculty of Information Technology and Electrical Engineering

Norwegian University of Science and Technology

# Contents

# Chapter 1

# Introduction

Quantum computing offers a new paradigm for computation. Using quantum properties such as superposition and entanglement, quantum computers can solve problems that are intractable for classical computers. While they were first conceived of by the famous Richard Feynman in 1982 [1] to simulate difficult quantum mechanical problems it was only really with the discovery of Shor's algorithm [2] in 1994 that the potential of quantum computers gained widespread attention. Shor's algorithm is a quantum algorithm that can factor large numbers in polynomial time, a problem that is believed to be exponentially hard and therefore intractable for classical computers. Since then, the search has continued for what other problems can be solved more efficiently on quantum computers.

How actually to construct a quantum computer is still an open question, and perhaps a more pressing one. There are several types of hardware being developed and researched, such as superconducting circuits used by IBM, Google and more, trapped ions used by IonQ and Honeywell, photonic quantum computers developed by Xanadu and Psi Quantum in addition to many other types. Common for all current approaches are problems of noise and decoherence. Theoretically, with computers of great enough scale, errors can be mitigated, but for the near future, the systematic errors of quantum computers will be a limiting factor and something to be taken into account when designing algorithms. Hence, the focus of much current quantum computing research considers noisy intermediate scale quantum (NISQ) devices. How to overcome the difficulties of NISQ hardware and be able to extract the full potential of quantum computers is a current research topic.

Variational quantum algorithms (VQAs) have been seen as a promising approach to achieve quantum advantage on NISQ devices. Such algorithms use a classical optimiser to find the best parameters for a general algorithm. In this way, the quantum hardware is only used for a small part of the algorithm, and the rest is done classically. Consequently, there is less time for noise and decoherence to compound, ruining the computation, and the efficiency of classical hardware can still be utilised. VQAs are in a sense a very natural approach, as most quantum hardware are inherently parametrised; microwave pulses for superconducting circuits, laser pulses for trapped ions and so on have to use a particular pulse length, frequency and so on. Applications of VQAs are numerous. A typical example is finding the ground state of a Hamiltonian for a molecule. Such problems are exponential in the particle count, and thus intractable on classical hardware for larger molecules, while the problem of evaluating the Hamiltonian on quantum hardware is typically polynomial. VQAs are also well suited for general mathematical problems and optimisation.

Machine learning (ML) and VQAs are a natural fit, as the optimisation of parameters is a common task in machine learning. With some way of encoding data into quantum hardware, VQAs are easily interpreted as a machine learning models. The output of the quantum algorithm can be seen as a prediction in a supervised learning problem, and so the quantum model can be trained to predict. With gate based quantum computers, the quantum model can show some similarities to classical neural networks, bringing forth the notion of quantum neural nets, while the idea of encoding data into to high-dimensional quantum state is reminiscent of classical kernel methods. Research indicate some advantages of using quantum machine learning models over classical ones, such as requiring fewer training iterations, but the field of quantum machine learning (QML) is still in its infancy and much work remains to be done. In particular, whether any quantum advantage for ML can be achieved with NISQ devices is not certain.

## 1.1 Relevant work

### 1.1.1 Cerezo et al. (2021)

Variational quantum algorithms (VQAs) are envisioned as the most likely candidate for quantum advantage to be achieved. By optimising a set of parameters that describe the quantum circuit, classical optimisation techniques are applicable, and only using the quantum hardware for what can be interpreted as function calls, limits the circuit depths needed. Running the same circuit many times with slightly different parameters and inputs in a classical-quantum-hybrid fashion, rather than a complete quantum implementation, means that the quantum operations can be simple enough for the noise and decoherence to be manageable.

Generally, VQAs start with defining a cost function, depending on some input data (states) and the parametrised circuit, to be minimised with respect to the parameters of the quantum circuit. For example, the cost function for the variational quantum eigensolver (VQE) is the expectation value of some Hamiltonian, which is the energy of a system. The cost function should be meaningful in the sense that the minimum coincides with the optimal solution to the problem, and that lower values generally implies better solutions. Additionally, the cost function should be complicated enough to warrant quantum computation by not being easily calculated on classical hardware, while still having few enough parameters to be efficiently optimised.

The optimisation of the cost function is often done with gradient descent methods. To evaluate the gradient of the quantum circuit w.r.t. the parameters, the very convenient parameter shift rule is often used. Though appearing almost as a finite difference scheme, relying on evaluating the circuit with slightly shifted parameters, it is indeed and exact formula. Furthermore, it may be used recursively to evaluate higher order derivatives, which is useful for optimisation methods that require the Hessian.

VQA's applications are numerous. The archetypical example is finding the ground state of a Hamiltonian for a molecule. Such problems are exponential in the particle count, and thus intractable on classical hardware for larger molecules, while the problem of evaluating the Hamiltonian on quantum hardware is typically polynomial. VQAs are also well suited for general mathematical problems and optimisation, even machine learning, another common example being QAOA for the max-cut problem.

Still, there are many difficulties when applying VQAs. Barren plateaus are a common occurrence, making the optimisation futile. The choosing of the ansatz determines the performance and feasibility of the algorithms, and there are many strategies and options. Some rely on exploiting the specific quantum hardware's properties, while some use the specifics of the problem at hand. Finally, the inherent noise and errors on near-term hardware will still be a problem and limit circuit depths.

### 1.1.2 Moll et al. (2018)

The computational performance of quantum computers is decided by five main factors. Naturally, the total qubit count is important, but also their connectivity (if they are not connected, intermediate operations like swapping is needed). How many gates/operations can be used before decoherence, noise and errors ruins the result also determines what programmes are feasible. Furthermore, which physical gates are available also matters, as transpiling to native gates will increase the circuit depth. Lastly, the degree of gate parallelisation can allow for shallower circuits and increased performance.

With all these factors in mind, the metric of quantum volume is defined, giving a single number describing the performance. It is effectively defined as the largest rectangular circuit of two-qubits a quantum computer may execute.

### 1.1.3 Torlai et al. (2020)

Due to the probabilistic nature of quantum computers and their exponentially great number of states, measuring complex observables accurately requires many samples. By post-processing the measurements using an artificial neural network, the variance of the samples are significantly reduced, though at the cost of some increased bias.

### 1.1.4 Schuld et al. (2019)

In optimising the parameters of variational circuits, having access to the gradient of the cost function (with respect to the parameters) is beneficial. The individual measurements are probabilistic, but the expectation is a deterministic value whose gradient can be calculated. Often, this is possible exactly using the parameter shift rule, allowing for evaluating the gradient using the same circuit with changed parameters. For circuits containing gates whose derivatives are not as nice, a method of linear combination of unities can be used. This method requires an extended circuit including an ancillary qubit.

### 1.1.5 Pesah et al. (2021)

The problem of barren plateaus plagues the optimisation of variational circuits and quantum neural network; for randomly initialised ansätze, the gradient of the cost function may exhibit exponentially small gradients, prohibiting gradient based optimisation. Under certain assumptions, it is shown that for quantum convolutional neural networks, the gradient of the cost function is no worse than polynomially small, such that the networks can be trainable.

### 1.1.6 Farhi et al. (2018)

Quantum neural networks (QNNs) are simply an abstraction of parametrised quantum circuits with some sort of data encoding. As with classical neural networks or supervised learning in general, the parameters are optimised by minimising a cost function. For QNNs, the output can be a single designated read-out qubit, where the states are interpreted as classes in a binary classification problem. This was shown to indeed be feasible for handwritten digit recognition, using downsampled MNIST data. With the qubit count on current quantum devices and the amount that can be easily simulated, the dimensionality of the data can not be much more than a dozen.

### 1.1.7 Abbas et al. (2021)

Whether quantum neural networks have inherent advantages is still an open question. Using the Fisher information of models, the authors calculate the effective dimension as a measure of expressibility. For models comparable in their input, output and parameter count, the effective dimension of particular quantum neural networks can be significantly higher. This advantage is empirically shown to be useful with a particular model on real quantum hardware, showing convergence in fewer steps than a similarly specced classical network.

The importance of feature maps is remarked upon, affecting both the expressibility of the model and the risk of barren plateaus, which in turn determines trainability.

# Chapter 2

# Machine learning

Machine learning lies at the intersection of statistics, computer science and optimisation. The central idea is to design an algorithm that uses data to solve a problem, and in so avoid explicitly programming a solution. Such algorithms or models can be used for a plethora of tasks, which is mainly divided into three major categories:

- **Supervised learning**: Given data and labels, find the relationship and try to be able to assign correct labels to new data.
- **Unsupervised learning**: Given data, find some underlying structure, patterns, properties or relationships.
- **Reinforcement learning**: Given some rules (e.g. a game), find a strategy to maximise some reward.

Only the first thereof will be explicitly considered in this thesis, though much of the theory and results can be extended to the latter two.

## 2.1 Supervised learning

Supervised learning is the most common and well-studied form of machine learning. It has the benefit of easily being mathematically formulated, and it can apply statistical methods to solve the problem. Given a data set $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}),\ \ldots,\ (\boldsymbol{x}^{(n)}, y^{(n)})\}$ of $n$ samples, where $\boldsymbol{x}^{(i)}$ is a vector of features and $y^{(i)}$ is the corresponding label, the goal is to find a function $f$ that maps $\boldsymbol{x}$ to $y$. In statistical terms, supervised learning can be thought of as having samples from a joint distribution $p(\boldsymbol{x}, y)$ with the goal of to find a conditional distribution $p(y|\boldsymbol{x})$, or at least the expectation thereof. The labels $y^{(i)}$ are usually assumed to be single-dimensional. They may be categorical, in which case the problem is called classification, or continuous, in which case it is called regression.

The marginal distribution $p(y|\boldsymbol{x})$ if often thought of as decomposed into a known deterministic function $f(\boldsymbol{x})$ and a random noise term $\varepsilon$ such that the labels are given by

$$y = f(\boldsymbol{x}) + \varepsilon \tag{2.1}$$

where $\varepsilon$ is assumed to be independent of $\boldsymbol{x}$. This simplifies the problem to approximating the function $f(\boldsymbol{x})$. Namely, find a function $\hat{f}$ such that the predictions $\hat{y} = \hat{f}(\boldsymbol{x})$ are good approximations of $y$. The *loss* (or cost) is a measure of how well the model fits the data. In statistics, the (log-) likelihood is often used, while in machine learning, simpler, more naïve functions like mean square error (MSE) are often used.

### 2.1.1 Parametric models

Parametric models are a subclass of supervised learning models that are defined by a finite set of parameters $\boldsymbol{\theta}$. This means that the model is fully defined by the parameters, and the data is only used to estimate the parameters.

### 2.1.2 Training

Since the model is defined by the parameters, the loss function is a function of the parameters. The supervised learning problem with a parametric model is rephrased into a standard optimisation problem:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta}; \boldsymbol{\mathcal{D}}) \tag{2.2}$$

where $L$ is the loss as a function of the parameters and given the data. $\boldsymbol{\theta}^*$ is then the optimal set of parameters. This optimisation usually done using gradient descent methods, which means that the loss function should be differentiable with respect to the parameters.

### 2.1.3 Bias-variance trade-off

In machine learning, there is a constant struggle between having models with lots of parameters and great expressive power versus simpler models with fewer parameters. The former are more likely to overfit the data, while the latter are more likely to underfit the data. This is known as the bias-variance trade-off. The main goal is of course to *generalise*, that is to have a model that truly captures the underlying properties of the data and subsequently performs well on data that it has not seen before.

Intrinsically, with an assumption like that of eq. (2.1), there is some uncertainty or noise that is inherent to the data. Consequently, one must choose a model that is flexible enough to capture the underlying structure of the data, but not so flexible that it captures the noise. When a model is too simple to capture the underlying structure, it is said to have high bias or be underfitted, while a model complex enough to capture the noise is said to have high variance or be overfitted. An overfitted model will have a low or zero errors on the data used for training, but may be wildly inaccurate on new data. This is captured in fig. 2.1.

## 2.2 Neural networks

Modern machine learning owes much of its popularity to the success of artificial neural networks, or if the context is clear, just neural networks (NNs). With easier access to larger datasets, more powerful hardware (in particular GPUs or even dedicated TPUs) and the backpropagation algorithm, NNs have become able to solve problems far too complicated for traditional methods.

Though state-of-the neural networks can contain billions of parameters, training them remains feasible. Modern hardware is of course paramount, but also backpropagation is crucial. Neural networks are trained using gradient methods, and with backpropagation, the gradient can be computed efficiently.

With the great size and complexity is interpretability sacrificed. The models are often black boxes, and it is difficult to understand why they make the predictions they do. Luckily, there have been some developments, perhaps most notably the universal approximation theorem. It

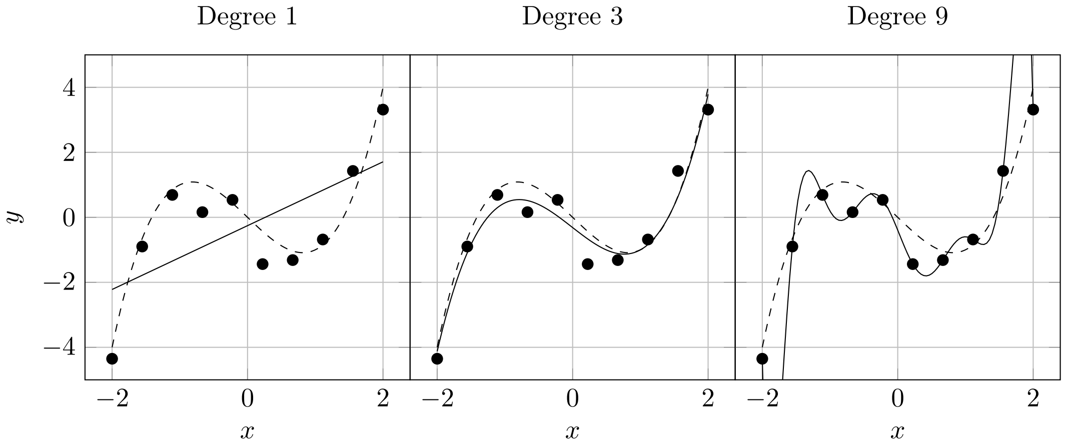


**Figure 2.1:** A simple example of overfitting and underfitting. 10 data points were generated by $x^3 - 2x$ plus some Gaussian noise with 0.4 standard deviation, shown in the figure as dots. The solid lines denote the fitted models which are polynomials of degree 1, 3 and 9, while the dashed line is the true function. The model with degree 1 is underfitted, while the model with degree 9 is overfitted – it perfectly fits all samples, but greatly deviates from the true function elsewhere. However, the 'correct' cubic polynomial lies much closer to the true function.

states that a neural network with a single hidden layer can approximate any continuous function to arbitrary precision, given enough neurons[1]. This gives some credence to the idea that NNs of other structures could be used to approximate complex functions.

### 2.2.1 Types of neural networks

**Dense feed-forward neural networks**

**Convolutional neural networks**

[1] In addition to some requirements regarding the activation function.

# Chapter 3

# Quantum computing

## 3.1 The qubit

The quantum bit, the qubit, is the building block of quantum computing. Like the classical binary digit it can be either 0 or 1. But being quantum, these are quantum states, $|0\rangle$ and $|1\rangle$, and the qubit can be in any superposition of these states. The state of the qubit lies in a two-dimensional vector space, and the states $|0\rangle$ and $|1\rangle$ are basis vectors, known as the computational basis states. Thus, the state of a qubit can be expressed as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{3.1}$$

where $\alpha$ and $\beta$ are any numbers, even complex ones. The only requirement is that the state is normalised, i.e. $|\alpha|^2 + |\beta|^2 = 1$. In particular, the qubit state lies in the Hilbert space $\mathcal{H} = \mathbb{C}^2$.

### 3.1.1 The Bloch sphere

A useful tool for visualising the state of a qubit is the Bloch sphere. First, it should be noted for states on the form eq. (3.1) are not unique, only the relative complex phase matters. There is a global phase which is not measurable, and thus not relevant for the state of the qubit. Therefore, taking also the normalisation requirement into account, the state of the qubit can be expressed as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{3.2}$$

where $\theta, \phi \in \mathbb{R}$. Interpreting $\theta$ as the polar angle and $\phi$ the azimuthal angle, the state of the qubit can be identified with a point a sphere, the Bloch sphere. There, the state $|0\rangle$ is typically thought of as the north pole, and $|1\rangle$ as the south pole. Figure 3.1 shows the Bloch sphere, and the state of the qubit in eq. (3.2).

### 3.1.2 Multiple qubits

Although the continuous nature of the qubit is indeed useful, the true power of quantum computers lie in how multiple qubits interact. Having multiple qubits allows for the creation of entanglement, which is a key feature of quantum computing. The state of multiple qubits can be expressed using the tensor product as

$$|\psi_1\psi_2\cdots\psi_n\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle\,. \tag{3.3}$$

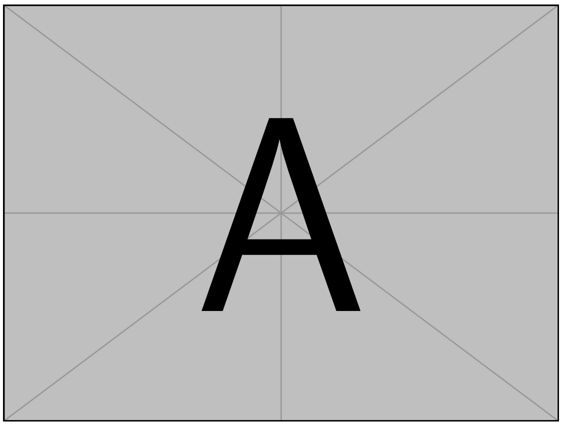


**Figure 3.1:** The Bloch sphere figure placeholder.

What makes this so powerful is that the state of a multi-qubit system does not have to be a product state. In can be anything on the form

$$|\psi_1\psi_2\cdots\psi_n\rangle = c_1\,|0\ldots00\rangle + c_2\,|0\ldots01\rangle + \cdots + c_{2^n}\,|1\ldots11\rangle = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{2^n} \end{pmatrix} \in \mathbb{C}^{2^n}, \tag{3.4}$$

which means that with $n$ qubits, the system can be in any superposition of the $2^n$ basis states. Operating on several qubits then, one can do linear algebra in an exponentially large space.

## 3.2 Operating with qubits

### 3.2.1 Single-qubit gates

To do an operation on one or more qubits, a unitary matrix is applied to the state, where the unitarity is needed for states to remain normalised. These operations are often thought of as gates, paralleling the classical gates in digital logic. The most basic gates are the Pauli gates, which are the $X$, $Y$ and $Z$ gates:

$$X = |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{3.5}$$

$$Y = |0\rangle\langle 1| - |1\rangle\langle 0| = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \tag{3.6}$$

$$Z = |0\rangle\langle 0| - |1\rangle\langle 1| = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{3.7}$$

These gates are half turns around the $x$, $y$ and $z$ axes of the Bloch sphere, respectively. The $X$ gate is also known as the NOT gate, as it mirrors the classical NOT gate by mapping $|0\rangle$ to $|1\rangle$ and vice versa.

The Hadamard gate

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.8}$$

is a rotation around the $x$-axis by $\pi/2$. It may be the most important gate in quantum computing, and is used to create superpositions of the computational basis states.

The $R_X$, $R_Y$ and $R_Z$ gates are rotations around the $x$, $y$ and $z$ axes, respectively, by an arbitrary angle $\theta$:

$$\begin{aligned} R_X(\theta) &= \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \\ R_Y(\theta) &= \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \\ R_Z(\theta) &= \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}. \end{aligned}$$

These parametrised gates will be useful in chapter 4.

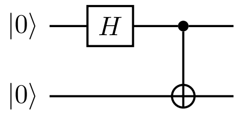
### 3.2.2 Multi-qubit gates

The most important multi-qubit gate is the controlled-$X$ gate, also known as the CNOT, which is a controlled version of the $X$ gate. Being controlled means that it only acts on the second qubit if the first qubit is in the state $|1\rangle$. Of course, the first qubit may be in a superposition, and the CNOT this way allows for the creation of entanglement between the two qubits. The CNOT gate is defined as

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.9}$$

### 3.2.3 Quantum circuits

The operations on qubits are often described using quantum circuits, which are a graphical representation of the operations on the qubits, the quantum algorithms. They are read from left to right. It is standard procedure to assume all qubits start in the state $|0\rangle$. For instance, using an $H$-gate to create a superposition before applying a CNOT gate, can be expressed as



(3.10)

## 3.3 Limitations of NISQ hardware

Quantum hardware have been physically realised and even outperforms classical computers in very contrived situations, but the hardware is still very limited. The hardware is limited in the number of qubits, the connectivity between the qubits, and the noise and decoherence of the qubits. It is believed that quantum hardware will continue to improve and eventually perform demanding algorithms like Shor's for large numbers. Still, the era dubbed NISQ (Noisy Intermediate-Scale Quantum) is the first step, and to make use of the hardware, algorithms must take these limitations into consideration.

Noise and decoherence severely limits how large circuits can be run on the hardware. Decoherence refers to the fact that the qubits are not isolated from the environment, and may be ruined by the environment, e.g. electrical noise from the control electronics. Furthermore, with the continuous nature of quantum states, minor errors can compound. If for instance a qubit is to be rotated many times, a small error in the rotation may cause the qubit to be rotated by a large angle. Because of this, NISQ algorithms must be shallow, meaning that the amount of gates applied before measurement is small.

Another limiting factor is the amount of qubits. Current hardware has around 10-100 qubits, which though still may be enough to express states too large to be expressed on classical computers, is not enough to perform the most demanding algorithms. With more qubits, error correction could be used to mitigate the effects of noise and decoherence, but this would require many more qubits than are currently available. Another current limitation is the connectivity between the qubits. Not all qubits are directly linked, which means that applying a multi-qubit gate may require intermediate swapping of qubits. This increases circuit depth which in turn increases the error rate.

# Chapter 4

# Variational quantum algorithms

## 4.1 Ansätze

## 4.2 Optimisation

### 4.2.1 Parameter-shift

### 4.2.2 Barren plateaus

## 4.3 Applications

# Chapter 5

# Quantum machine learning

How to combine quantum computing and machine learning is not easily answered. In the discussion of quantum machine learning, it is standard practice to reference the four quadrants of table 5.1, first described by Schuld & Petruccione [3].

Classical data being processed on classical computers is classical machine learning. Though not explicitly linked to quantum computing, there are some ways in which quantum computing influence classical machine learning, such as the quantum-inspired application of tensor networks in [4].

Using classical machine learning for quantum computing is used to improve quantum computers general performance. For example, with machine learning algorithms, the variance of the measurements can be reduced, as shown in [5]. Alternatively, advanced machine learning models like neural networks can be employed to describe quantum states more efficiently.

How to use quantum algorithms to solve machine learning problems is the main topic of this thesis and is what will be meant when quantum machine learning (QML) is mentioned. QML concerns itself with how better to do what classical machine learning already does. Quantum algorithms are most often advertised with speed-ups contra classical algorithms, often exponentially so as with Shor's algorithm. While this is true, there are major difficulties in achieving these speed-ups. However, there may be other advantages to be had, in terms of the amount of data needed to how much training has to be done.

The last quadrant of quantum computing handling quantum data includes quantum machine learning from for example quantum experiments or machine learning when the data is inherently quantum states. With NISQ hardware, fully quantum procedures are difficult, so this field is not of immediate interest. There is obviously much overlap with CQ as the data is quantum once

| | | **Computer** | |
|---|---|---|---|
| | | *Classical* | *Quantum* |
| **Data** | *Classical* | CC | CQ |
| | *Quantum* | QC | QQ |

**Table 5.1:** The four fundamental ways in which quantum computing and machine learning can be combined. CC: classical computer and classical data. CQ: classical computer and quantum data. QC: quantum computer and classical data. QQ: quantum computer and quantum data. Lifted from [3].

| | Qubits needed | Circuit depth | Hard to simulate classically |
|---|---|---|---|
| Basis encoding | $b(N)$ | $\mathcal{O}(b(N))$ | No |
| Amplitude encoding | $\lceil \log_2 N \rceil$ | $\mathcal{O}(N)$ | Yes |
| Angle encoding | $N$ | $\mathcal{O}(N)$ | No |
| Second order angle encoding | $N$ | $\mathcal{O}(N^2)$ | Yes[1] |

**Table 5.2:** Properties of different data encodings for an $N$-dimensional data set of $M$ data points. $b(N) > N$ is the number of bits needed to represent an $N$-dimensional data point.

encoded into the quantum computer, but as will be made clear, the encoding is such a big part of CQ that results thence are not necessarily applicable QQ.

## 5.1 Data encoding

In order for quantum computers to use classical data, it must first be encoded in a way that is compatible with the quantum hardware. How this is done has major implications on both the computational performance and the model expressibility. While naive techniques like basis encoding are possible and easy to understand, more complex procedures are often needed to achieve good performance. The four methods that will be discussed in this section are summarised in table 5.2.

### 5.1.1 Basis encoding

The perhaps simplest way to encode data is to use the computational basis states of the qubits. This is done in much the way that classical computers use binary numbers. For example, some data $x$ can be expressed as a bit-string $x = \{x_1, x_2, \ldots, x_n\}$, where each $x_i$ is either 0 or 1, where any continuous variables are encoded as floating point numbers. For multidimensional data, the bit-strings are simply concatenated.

If for instance the data point 010101 is to be encoded in a quantum computer, it is simply mapped to the computational basis state $|010101\rangle$. This allows for multiple data points to be encoded in parallel as

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left| \boldsymbol{x}^{(m)} \right\rangle, \tag{5.1}$$

where $\mathcal{D}$ is the data set, $M$ the total number of data points and $\boldsymbol{x}^{(m)}$ the $m$-th binarised data point. This is a simple encoding and has some significant disadvantages. There must be at least as many qubits as there are bits in the binarised data. For $N$ bits, there are $2^N$ possible states, but at most $M$ are used, which means that the embedding will be sparse. This means that the computational resources required to encode the data will in some sense wasted, and that the quantum computer will not be able to exploit the full power of the quantum hardware. To utilise the entire Hilbert space, amplitude encoding is better suited.

[1] Conjectured.

### 5.1.2 Amplitude encoding

A more efficient way to encode data is to use amplitude encoding, exploiting the exponentially large Hilbert space of quantum computers. This is done by mapping the bits in the bit-string to individual qubits, but to individual amplitudes in the exponentially large Hilbert space. Mathematically, for some $N$-dimensional data point $\boldsymbol{x}$, this reads

$$|\psi(\boldsymbol{x})\rangle = \sum_{i=1}^{N} x_i \, |i\rangle \,, \tag{5.2}$$

where $x_i$ is the $i$th component of the data point and $|i\rangle$ is the $i$th computational basis state. This has the advantage of being able to encode any numeric type natively, and perhaps more importantly, only needing logarithmically many qubits. For $N$-dimensional data points, only $\lceil \log_2 N \rceil$ qubits are needed. This is a significant improvement over the basis encoding, which requires $N$ qubits (or more if integers and floats are to be binarised).

An insignificant drawback is that the data must be normalised, which can be done without loss of information by requiring an additional bit to encode the normalisation constant. Also, some padding may be needed if the number of qubits is not a power of two.

Furthermore, amplitude encoding can easily be extended to cover the entire dataset. This is done by concatenating the data points, and then normalising the resulting state at the low cost of a single additional bit. Then, the data set $\mathcal{D}$ with $M$ data points can be encoded as

$$|\mathcal{D}\rangle = \sum_{m=1}^{M} \sum_{i=1}^{N} x_i^{(m)} \, |i\rangle \, |m\rangle \,, \tag{5.3}$$

where $x_i^{(m)}$ is the $i$-th component of the $m$-th data point. For such encodings, only $\lceil \log_2(NM) \rceil$ qubits are needed.

The main drawback of amplitude encoding is the practical difficulties of preparing such states. Any state of the form

$$|\psi\rangle = \sum_{i} a_i \, |i\rangle \tag{5.4}$$

must be efficiently and correctly prepared, which is not trivial. Unless some very specific assumptions are made, this is not possible in polynomial time (as a function of the number of qubits), which limits the potential for exponential speed-ups [3]. In general, for classical data, circuits must be linearly deep in the size of the data and ergo exponentially deep in the amount of qubits, which makes it beyond the reach of NISQ hardware.

### 5.1.3 Angle encoding

A third option is angle encoding. Here, the potentially continuous components of the data are mapped to rotations of the qubits. For the rotations to be meaningful angles and not loop around, the data needs be normalised. An $N$-dimensional data point $\boldsymbol{x}$ is then encoded as

$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_X(x) \, |0\rangle \,, \tag{5.5}$$

$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_Y(x) \, |0\rangle \tag{5.6}$$

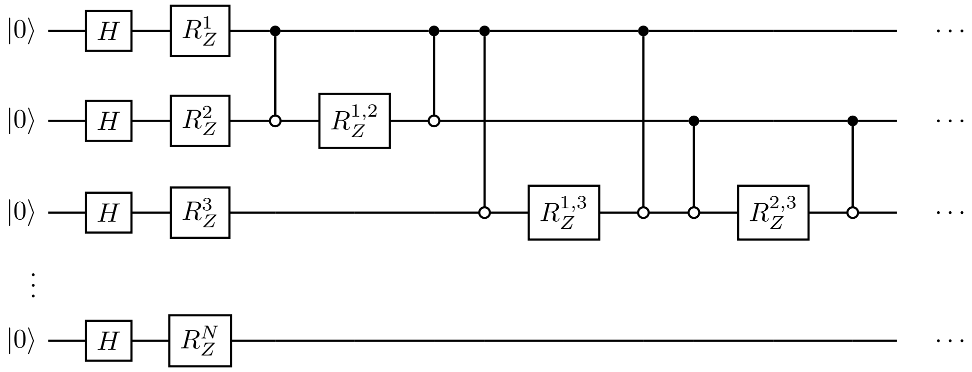
or

$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_Z(x) H \, |0\rangle \,, \tag{5.7}$$

depending on which rotation is used. For Z-rotations, a Hadamard gate is needed for the operation to do something. $N$ qubits are still required, but with native support for continuous variables, angle encoding can be more efficient than basis encoding. A constant number of gates are needed to prepare the state, which is a significant advantage over amplitude encoding. Still, being a product state, it offers no inherent quantum advantage.

### 5.1.4 Second order angle encoding

Havlicek *et al.* [6] propose a second-order angle encoding, which they conjecture to be hard to simulate classically. First, angles are encoded as above, but then the qubits are entangled and rotated further based on second order terms. In circuit notation, such an encoding with Z-rotations reads



(5.8)

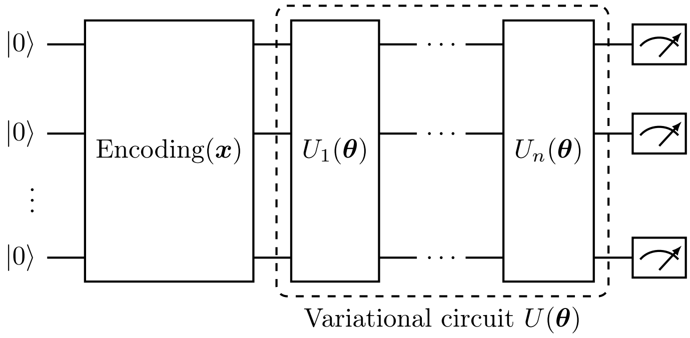
where $R_Z^i = R_Z(x_i)$ and $R_Z^{i,j} = R_Z((\pi - x_i)(\pi - x_j))$ and with the entanglements and second-order rotations being applied pairwise for all $N$ qubits. This increases the circuit depth to order $N^2$ and full connectivity is needed. Nonetheless, it may be feasible for data of moderate dimensionality on NISQ hardware, and were it indeed classically hard to simulate, it could provide quantum advantage.

### 5.1.5 Repeats

The expressive power of models heavily rely on the encoding strategy. For instance, a single qubit rotation only allows the model to learn sine functions, where the frequency is determined by the scaling of the data. Generally, quantum models will learn periodic functions, and thus Fourier analysis is a useful tool. Schuld *et al.* [7] study the implications of this, and they show that simply repeating basic encoding blocks allows for learning of more frequencies and thus more complex functions. Asymptotically, such repeats lets a quantum model learn arbitrary functions.

## 5.2 Quantum neural networks

Quantum neural networks (QNNs) are simply an abstraction of parametrised quantum circuits with some sort of data encoding. As classical artificial neural networks have made classical machine learning into a powerful tool, QNNs are envisioned as a quantum counterpart, inheriting



$|0\rangle$
$|0\rangle$
$\vdots$
$|0\rangle$
Encoding($\boldsymbol{x}$)
$U_1(\boldsymbol{\theta})$
$\cdots$
$U_n(\boldsymbol{\theta})$
Variational circuit $U(\boldsymbol{\theta})$

**Figure 5.1:** The structure of a quantum neural network. First, the data $\boldsymbol{x}$ is encoded into a state $|\psi(\boldsymbol{x})\rangle$ using some encoding strategy. Then, the state is transformed by a parametrised quantum circuit $U(\theta)$. This variational circuit can often be decomposed into a sequence of gates $U_1, \ldots, U_n$, making the QNN structure more akin to the layered classical neural networks. These gates or layers do not need to use all qubits, but can be restricted to a subset, mimicking the classical concept of differently sized hidden layers. Finally, measurements are made and used to calculate the model output.

some classical theory, nomenclature and perhaps unfounded hype. The main goal of QNNs is to do what classical NNs do, but with some quantum advantage, be it in terms of generalisability, training required or something else.

The structure of most quantum neural networks follow classical feed-forward networks. Figure 5.1 shows the general circuit layout. In the first step or layer, data is encoded into the qubits, typically using a method discussed in section 5.1. Next, the data is passed through a sequence of parametrised quantum gates which often can be interpreted as layers. Lastly, an output is produced, which is typically a measurement of some observable. Usually, the observable is a combination of Pauli operators on every qubit. Thence, a cost function can be evaluated. Often, methods like parameter-shift allows for computation of gradients, which makes it possible to train the network using classical methods.

### 5.2.1 Architectures and their applications

**Quantum convolutional neural networks**

Originally introduced by Cong *et al.* [8], quantum convolutional neural networks (QCNNs) take inspiration from classical convolutional neural networks in that a sequence of convolutional and pooling layers are used to extract features and reduce the dimension before the output is made. In the quantum convolutional layers, neighbouring qubits are entangled with some parametrised gates. After that, pooling layers halve the active qubit count by yet a parametrised gate. When pooling, the qubits to be discarded could be measured and used to determine the operations on the still active qubits. Otherwise, the unused qubits are simply ignored. After several iterations, a gate can be employed on the remaining qubits, analogous to a fully connected layer in classical CNNs, before the final measurement and output.

Because of the constant reduction of layer sizes in (Q)CNNs, the total parameter count is only of order logarithm of the network depth, making them easier to train than dense networks of similar input size. In [8], QCNNs were shown to be able to classify topological phases of matter, and since then, it has been shown that they have inherited they classical counterparts' ability to classify images [9]. They have shown desirable properties with regard to avoiding barren plateaus [10], which could be essential in training at for problems of interesting size.

**Quantum generative adversarial networks**

Quantum generative models have been shown to potentially have an exponential advantage over their peers [11]. Due to the inherent probabilistic nature of quantum machines, it should not be surprising that they more naturally learn difficult distributions than classical computers do. For instance, real quantum hardware has been used to generate (admittedly low-resolution) images of handwritten images [12].

**Hybrid quantum-classical neural networks**

Another option is to include a quantum layer or node is some larger pipeline or even non-linear graph structure. Because parametrised quantum circuits are differentiable, they can easily be handled using the chain rule when back propagating a hybrid model. Killoran *et al.* [13] describe and test several such models. They note that for the NISQ era, limiting quantum components of models to very particular tasks to which they are especially suited should be beneficial. As quantum hardware develops, they can take over more and more of the hybrid models.

More recently, Zeng *et al.* [14] have explored using a hybrid model to multi-class classification on real world data sets using a CNN-inspired structure. There it is shown that the hybrid model outperforms a classical CNN of similar parameter size.

## 5.3 Comparisons

### 5.3.1 QNNs vs NNs

In order to compare the performance of the QNN and the NN, architectures suited for binary classification with exactly 8 parameters are used. The QNN structure is shown in fig. 5.2. The data used is Fisher's iris dataset, perhaps the most used dataset for studying classification in statistics, containing samples of three different species of iris flowers. For each species, there are 50 samples, each with four features: sepal length, sepal width, petal length, and petal width. Like in [15], only the two first species are considered, which happen to be linearly separable in the feature space.

The four-dimensional input data is first scaled to have zero mean and unit variance. Then, it is encoded into a quantum state a second order angle encoding with Z rotations, discussed in section 5.1.4, with two repetitions. In the Qiskit framework, this is implemented in the `ZZFeatureMap` class. This entangles the qubits and embeds them in higher dimensional space.

Next, the state is evolved by the parametrised circuit. It consists of initial parametrised Y-rotations, then full entanglement using controlled not-gates, and lastly final parametrised Y-rotations. The different rotation direction ensures the gates do not commute. There are in total 8 parameters. This is implemented in Qiskit as the `RealAmplitudes` ansatz.

Finally, all four qubits are measured and the parity of the four bit output is interpreted as the prediction of the class label. Figure 5.2 shows the structure of the QNN and how the parameters are used.

Both exact simulations and noisy simulations were performed, with the latter using noise modelled after the 27-qubit IBM Montreal architecture, the actual hardware used in the original paper.

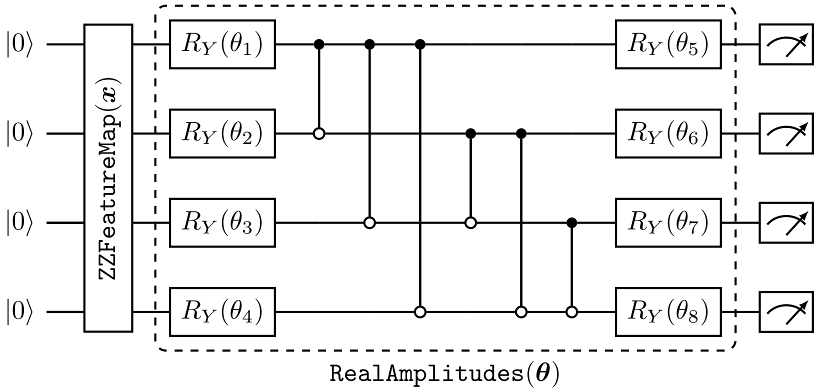


**Figure 5.2:** Structure of the QNN used for classification of the iris dataset. The first block maps the input data $\boldsymbol{x}$ to the quantum state $|\psi(\boldsymbol{x})\rangle$ using a second order Z rotation feature map. The second block is the variational circuit, parametrised by $\boldsymbol{\theta}$, a vector with eight components. Finally, all qubits are measured, where the parity is interpreted as the prediction.

The classical neural network was a standard dense feed-forward model. To make in comparable to the QNN, it used a 4-1-1-1-2 layered structure without biases, giving a total of 8 parameters. The activation functions were leaky ReLUs,

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}, \tag{5.9}$$

and the output layer used a softmax activation function.

Both models were implemented using PyTorch, with code partly taken from the original paper[2]. The QNN was adapted to use Qiskit's PyTorch interface. Consequently, the models could be trained in the exact same manner, using the Adam optimiser with a learning rate of 0.1 and cross-entropy loss. The classical and noiseless models were trained for 100 epochs, while the noisy model was only trained for 10, as simulating the noise severely impacted training time.

For validation, 10-fold cross-validation was used. That is, the dataset was split into 10 equal parts or *folds*. Each fold us used as the validation set once, their accuracies being recorded during the training with the other nine folds. The mean accuracy over the 10 folds was used for the final performance metric, shown in fig. 5.3.

As in the original paper, the QNN converges much quicker and more consistently, with an out-of-fold accuracy of 100% for all ten folds. The classical network, on the other hand, requires more iterations to converge and does not always do so. In some cases, the model did not converge, only predicting one class, which is why the out-of-fold accuracy was not 100% for all folds. This is in line with the original paper, underlining the potential advantage of quantum neural networks.

### 5.3.2 Quantum convolutional neural networks

To implement and test a quantum CNN, Qiskit's online tutorials were closely followed [16]. Being limited to few qubits, images with resolution 2 × 4 were generated, containing either vertical or horizontal with some Gaussian noise. Figure 5.4 shows examples thereof. The task of the QCNN was to classify the images as either vertical or horizontal lines.

[2] Available at `https://github.com/amyami187/effective_dimension`.

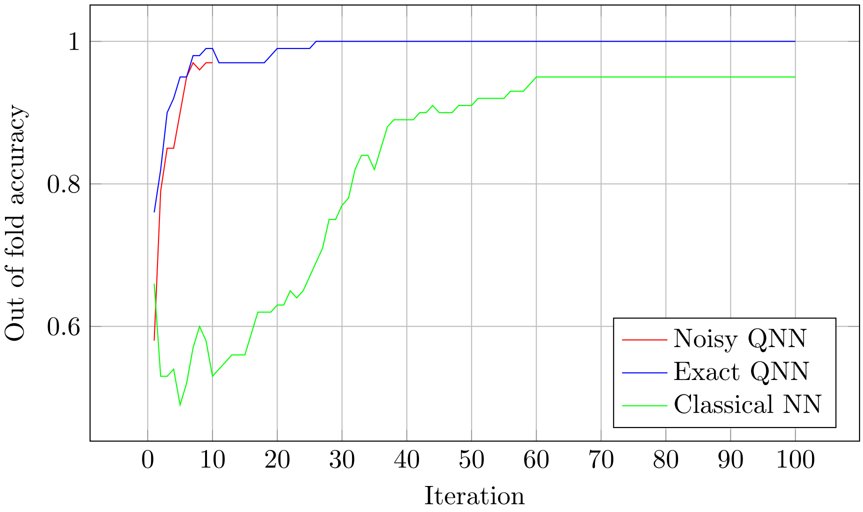


**Figure 5.3:** Mean accuracy during training for the iris dataset using 10-fold cross validation. All models have 8 parameters and are trained using the Adam optimiser with a learning rate of 0.1, using cross-entropy as the loss function. Due to the computational cost, the noisy (simulated IBM Montreal backend) QNN was only trained for 10 epochs.

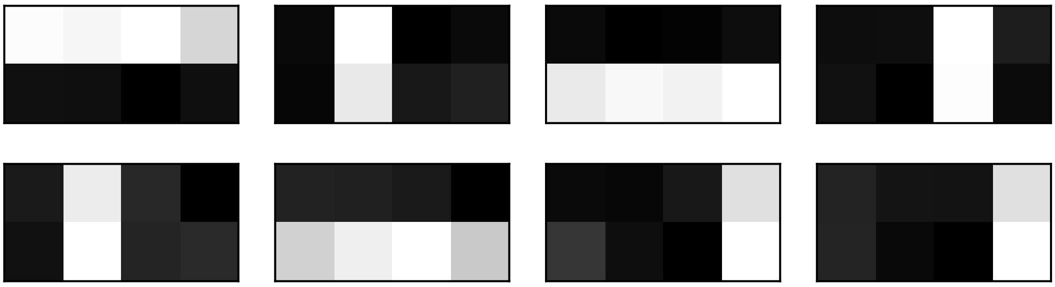
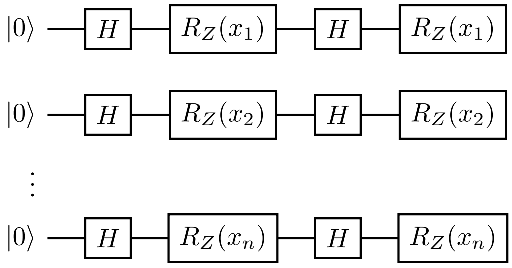
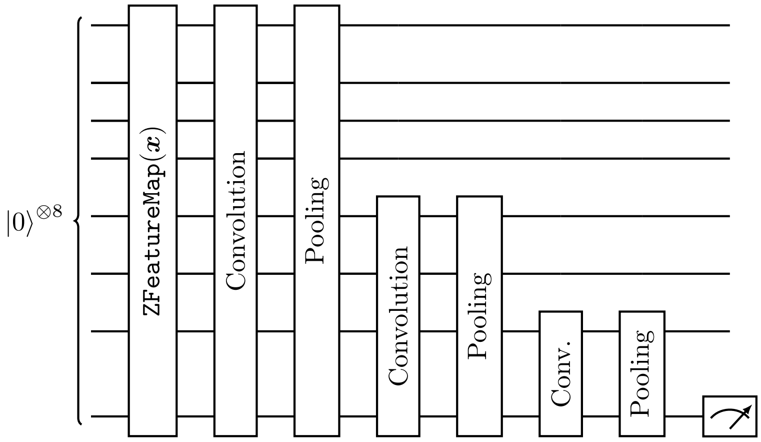


**Figure 5.4:** Data for the QCNN. With a total of 64 training images and 16 for testing, they form balanced dataset of $2 \times 4$ pixels, with either a vertical or horizontal line encoded as 1 and $-1$. The images are generated with some Gaussian noise.

First, data is encoded using Qiskit's ZFeatureMap; each of the eight pixels of the image is mapped to a qubit through two repetitions of the Hadamard gate and Z-rotations parametrised by the pixel value being applied, in circuit notation:



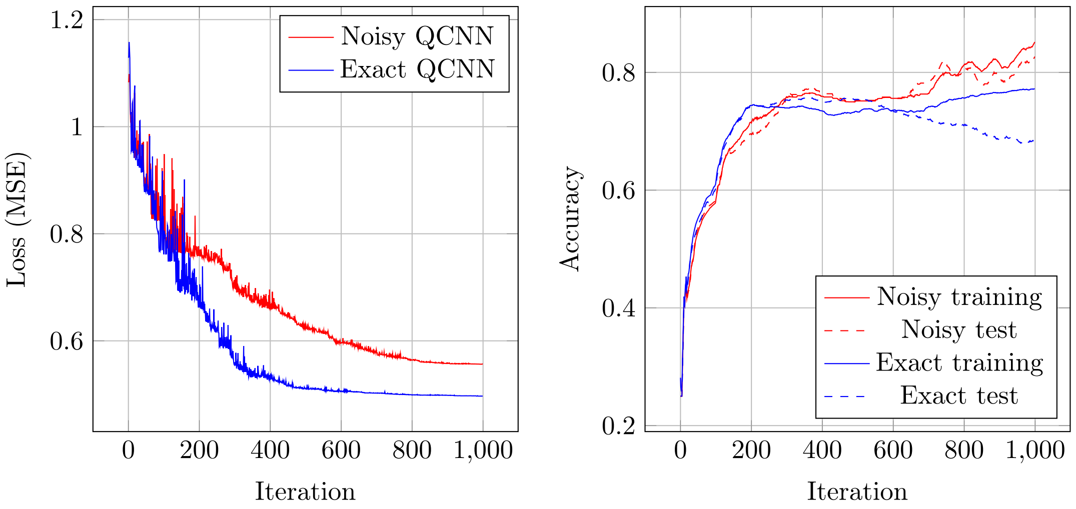
The convolution layers act with pairwise parametrised rotations of neighbouring qubits, also wrapping around, entangling the first and last qubits through various CNOT gates and both parametrised and fixed Z and Y rotations. Thereafter, pooling layers halve the active qubit counts by parametrised rotations and CNOT gates. For the final layer, the sole remaining qubit is measured, and the result is interpreted as the prediction. In total, the circuit appears as



with a total of 63 parameters.

As in Qiskit's guide, training was done using the COBYLA optimiser[3] which does not use gradients. Why this optimiser was chosen is not clear, but testing shows that simulations using gradient based methods such as Adam or simple gradient descent is significantly slower. The accuracies and loss (mean square error) during training is shown in fig. 5.5. Like in section 5.3.1, noise is modelled after the IBM Montreal hardware. The networks were trained for 1000 epochs, and while neither reached full accuracy, the losses shrunk, indicating at least increased certainty in the predictions. Interestingly, the noisy simulation appears to yield better predictions, despite suffering from higher losses during training. It seems that the noiseless QCNN is overfitting to the training data, while the noisy QCNN generalises better.

[3] Constrained Optimisation BY Linear Approximation.



**(a)** **(b)**

**Figure 5.5:** Training of the QCNN. The red curves are for the noisy model (modelled after IBM's Montreal hardware), while the blue curves are for the exact model. The dashed curves are for the test set. (a) loss (mean square error) during training. (b) accuracy on the training and test sets (running mean with a 100 iteration window).

### 5.3.3 QCNN with intermediate measurements

A more complex QCNN structure was described by Pesah *et al.* [10], where the pooling modules measure a qubit and use the result to control a unitary gate on its neighbour. The use of mid-circuit measurements complicates the circuit and its implementation, but allows for a non-linear and potentially more powerful model.

Following the description in [10], a QCNN was implemented with structure as shown in fig. 5.6. First, the data was encoded using angle encoding in the $X$ direction. The convolutional layers consisted of pairwise $W$ gates, a mix of parametrised rotations and CNOTs, with total of 15 parameters per gate. The pooling layers consisted of a measurement and a conditional single-qubit unitary gate. Without any particular recommendation in the original paper, a simple $X$-gate was used. Like in section 5.3.2, the network used 8 qubits to handle the 8-dimensional data. Three convolutional and pooling layers were used, reducing the data from 8 to 2 dimensions Lastly, a final general two-qubit ansatz was used, and a single qubit was measured and interpreted as a prediction. This was a simple parametrised entangler, similar to the `RealAmplitudes` in section 5.3.1, but with $X$ rotations. In total, the network had 154 parameters.

The QCNN was implemented using the PennyLane framework and trained with the Adam optimiser with a learning rate of 0.01. With the PennyLane implementation, there were no problems using gradient based optimisation. The results are shown in fig. 5.7. The training loss was lower than for the QCNN in section 5.3.2, despite the fewer iterations. Furthermore, this model achieves a perfect accuracy on both the training and test sets. Better performance is to be expected with the higher-order information used in the optimisation and the higher parameter count.

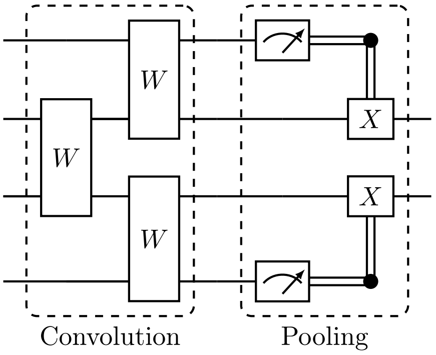


**Figure 5.6:** QCNN convolution and pooling layer structure with intermediate measurements. Some encoded data or already pooled data enter the convolution layer where parametrised gates $W$ entangle the qubits. The pooling modules measure a qubit and use the result to control a unitary gate on a neighbour (here the $X$ gate).

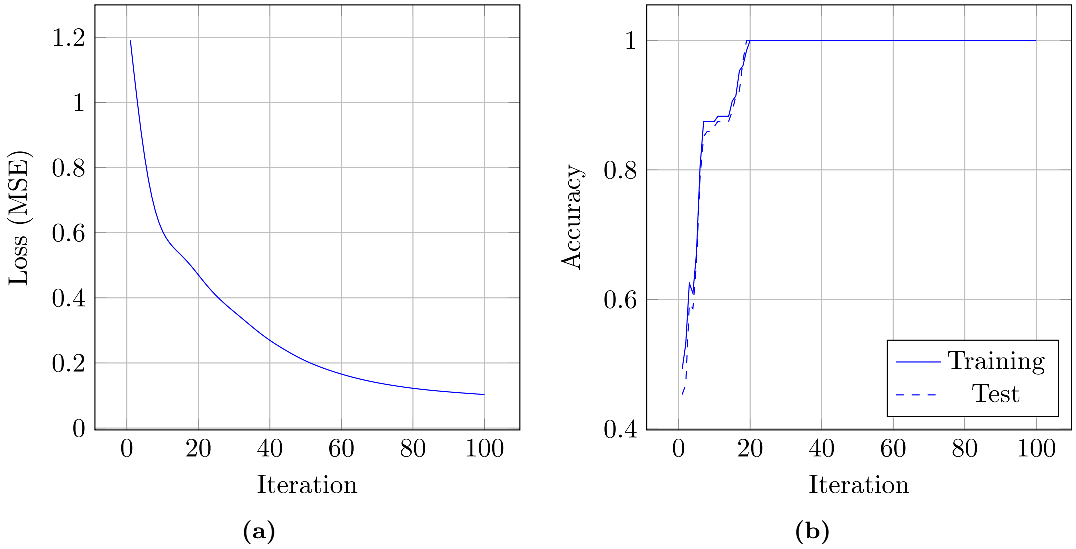


**Figure 5.7:** Training of the QCNN with intermediate measurements. (a) loss (mean square error) during training. (b) accuracy on the training and test sets.

# References

1. Feynman, R. P. Simulating physics with computers. *International Journal of Theoretical Physics* **21,** 467–488. ISSN: 0020-7748, 1572-9575. http://link.springer.com/10.1007/BF02650179 (2022) (June 1982).

2. Shor, P. *Algorithms for quantum computation: discrete logarithms and factoring* in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), 124–134.

3. Schuld, M. & Petruccione, F. *Supervised Learning with Quantum Computers* ISBN: 9783319964249 (Springer International Publishing, 2018).

4. Felser, T. *et al.* Quantum-inspired machine learning on high-energy physics data. *npj Quantum Information* **7,** 111. ISSN: 2056-6387. http://www.nature.com/articles/s41534-021-00443-w (2022) (Dec. 2021).

5. Torlai, G., Mazzola, G., Carleo, G. & Mezzacapo, A. Precise measurement of quantum observables with neural-network estimators. *Phys. Rev. Research* **2,** 022060. https://link.aps.org/doi/10.1103/PhysRevResearch.2.022060 (2 June 2020).

6. Havlicek, V. *et al.* Supervised learning with quantum-enhanced feature spaces. *Nature* **567,** 209–212. https://doi.org/10.1038%2Fs41586-019-0980-2 (Mar. 2019).

7. Schuld, M., Sweke, R. & Meyer, J. J. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A* **103.** https://doi.org/10.1103%2Fphysreva.103.032430 (Mar. 2021).

8. Cong, I., Choi, S. & Lukin, M. D. Quantum convolutional neural networks. *Nature Physics* **15,** 1273–1278. https://doi.org/10.1038%2Fs41567-019-0648-8 (Aug. 2019).

9. Oh, S., Choi, J. & Kim, J. *A Tutorial on Quantum Convolutional Neural Networks (QCNN)* in *2020 International Conference on Information and Communication Technology Convergence (ICTC)* (2020), 236–239.

10. Pesah, A. *et al.* Absence of Barren Plateaus in Quantum Convolutional Neural Networks. *Physical Review X* **11.** https://doi.org/10.1103%2Fphysrevx.11.041011 (Oct. 2021).

11. Gao, X., Zhang, Z.-Y. & Duan, L.-M. A quantum machine learning algorithm based on generative models. *Science Advances* **4,** eaat9004. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.aat9004. https://www.science.org/doi/abs/10.1126/sciadv.aat9004 (2018).

12. Huang, H.-L. *et al.* Experimental Quantum Generative Adversarial Networks for Image Generation. *Physical Review Applied* **16.** https://doi.org/10.1103%2Fphysrevapplied.16.024051 (Aug. 2021).

13. Killoran, N. *et al.* Continuous-variable quantum neural networks. *Physical Review Research* **1.** https://doi.org/10.1103%2Fphysrevresearch.1.033063 (Oct. 2019).

14. Zeng, Y., Wang, H., He, J., Huang, Q. & Chang, S. A Multi-Classification Hybrid Quantum Neural Network Using an All-Qubit Multi-Observable Measurement Strategy. *Entropy* **24.** ISSN: 1099-4300. https://www.mdpi.com/1099-4300/24/3/394 (2022).

15. Abbas, A. *et al.* The power of quantum neural networks. *Nature Computational Science* **1.** Code available at https://github.com/amyami187/effective_dimension, 403–409. ISSN: 2662-8457. http://www.nature.com/articles/s43588-021-00084-1 (2022) (June 2021).

16. IBM. *The Quantum Convolution Neural Network* https://qiskit.org/documentation/machine-learning/tutorials/11_quantum_convolutional_neural_networks.html.

17. Treinish, M. *et al. Qiskit: Qiskit 0.37.2* version 0.37.2. Aug. 2022. https://doi.org/10.5281/zenodo.7017746.

18. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2014. https://arxiv.org/abs/1412.6980.

19. Bergholm, V. *et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations* 2018. https://arxiv.org/abs/1811.04968.

20. Paszke, A. *et al.* in *Advances in Neural Information Processing Systems 32* (eds Wallach, H. *et al.*) 8024–8035 (Curran Associates, Inc., 2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.