Quantum Neural Networks

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

Preface

This report concludes the course *TMA4500 Industrial mathematics, specialisation project* whose purpose is two-fold. First, it is to acquaint the student to a specific topic, which in this case is quantum machine learning or more generally quantum computing. Secondly, it is to make the student familiar with the process of writing a semester-long thesis under the supervision of an expert, and so make them ready for the master's thesis next semester.

The initial aim of the project was to learn about variational quantum algorithms and the application thereof to machine learning. This lead to the investigation of quantum neural networks, which are mostly just interpretation of variational quantum algorithms through the lens of standard, classical neural network design. With the limited scope of this pre-master thesis, it was decided only to implement and test some simple models based on recent research papers.

I would like to thank my de-facto supervisor Franz Fuchs for not only taking time off his important work to supervise me, but also for showing enthusiasm and inviting me into this brave new world of quantum computing when I did not really know what I wanted to do. I also want to thank my co-supervisor, Alexander Stasik, for his help and guidance in the process of writing this thesis. Finally, I would like to thank my official supervisor, Gunnar Taraldsen, for allowing me to work with something I find truly exciting, for still providing feedback while I worked with something out of the ordinary and for answering e-mails within minutes.

All code used in this report will be made available on GitHub via `https://github.com/boyesjo/tma4500`.

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. Their first conception goes back to the early 80s, often being accredited to Richard Feynman's 1982 paper [1], with the goal of simulating difficult quantum mechanical problems. It was however only really with the discovery of Shor's algorithm in 1994 [2] that the potential of quantum computers gained widespread attention. Shor's is a quantum algorithm that can factor large numbers in polynomial time, a problem that is believed to be practically exponentially hard and therefore intractable for classical computers. Since then, the search has continued for what other tasks can be solved more efficiently on quantum hardware.

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. Although some of these have already claimed quantum supremacy, that is they have solved a problem believed to be intractable for classical computers, supremacy has only been shown in very particular settings with no practical use. Common for all current approaches are difficulties with noise and decoherence. Theoretically, with computers of great enough scale, errors can be mitigated, but in 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 use NISQ devices to solve problems with actual practical use. 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, which could ruin 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 must use a particular pulse length, frequency et cetera. Instead of calibrating the hardware to follow a Platonic algorithm, it could instead be better to calibrate it to map the input to the desired output.

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 process of evaluating the Hamiltonian on quantum hardware is typically polynomial. VQAs are also well suited for general mathematical tasks such as optimisation.

Machine learning (ML) and VQAs are a good fit, as the optimisation of parameters is how many machine learning models are trained. With some way of encoding data into quantum hardware, VQAs are easily interpreted as machine learning models. The output of the quantum algorithm can be seen as a prediction in supervised learning, and so the quantum model can be trained to predict. Though less efficient than classical backpropagation, gradients can be calculated for VQAs, and thus optimisation can be accomplished using classical optimisers. With gate-based quantum computers, the quantum model can show some similarities to classical neural networks, bringing forth the notion of quantum neural nets (QNNs). Moreover, 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, and no exponential speed-up has yet been demonstrated.

The study of quantum machine learning is not an easy one. A lot of machine learning's success is anecdotal; it is often difficult to prove correctness and convergence, so some trust of ML relies on empirical testing. Obviously, quantum computing does not simplify matters. Being restrained to current day hardware, there is little chance that any advantage for quantum hardware can be shown empirically, and so arguments for quantum advantage must be made theoretically or by extrapolation of simple tests. Most research thus far can only be seen as proof of concepts, showing some benefits on trivial problems.

This thesis serves as an introduction to the field of quantum machine learning. The two next chapters give a theoretical background for QML. Chapter 2 reviews the basics of machine learning and neural networks in particular. Thereafter, chapter 3 introduces the basics of quantum computing and the limitations of NISQ devices in addition to an introduction to variational quantum algorithms. In chapter 4, how data is encoded into quantum hardware is first discussed. Next, quantum neural networks are introduced. The penultimate chapter 5 presents various simulations of quantum classical neural networks and a comparison with classical neural networks. Finally, chapter 6 concludes the thesis and discusses future work.

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. With ever more data available and with ever more powerful computers, machine learning has become a powerful tool in many fields, solving problems previously thought intractable. Such algorithms or models can be used for a plethora of tasks, which are mainly divided into three main categories:

- *Supervised learning:* Given data with corresponding labels, find the relationship and try to assign correct labels to new, unseen data.
- *Unsupervised learning:* Given data, find some underlying structure, patterns, properties or relationships, such as clusters or outliers.
- *Reinforcement learning:* Given a set of rules or environment, such as a game, try different strategies and determine one that optimises 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. This chapter is hardly a comprehensive introduction to machine learning, but rather a brief overview of the most important concepts and techniques, serving as a reference point for the later endeavours into quantum machine learning. Some familiarity with basic probability theory is assumed.

2.1 Supervised learning

2.1.1 Data

For supervised learning, some data is assumed given. Mathematically, this data is described by a set of input-output pairs $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, where $\boldsymbol{x}^{(i)} \in \mathcal{X}$ is a vector of features in the input domain $\mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ is the corresponding label in the output domain $\mathcal{Y}$. The data is assumed to be drawn from a joint distribution $p(\boldsymbol{x}, y)$. Given a new, unseen input $\boldsymbol{x}$, the goal predict its corresponding label y.

The input domain $\mathcal{X}$ is usually assumed to be the vector space $\mathbb{R}^N$ for some integer N, though some dimensions may be discrete and even binary. Some preprocessing is often done to the data, such rescaling the features or using the data to calculate new features. While preprocessing can be essential to the performance of a machine learning model, it does not matter for this theoretical discussion; any preprocessed data can be assumed to be drawn from the distribution of the transformed variable.

The output domain $\mathcal{Y}$ is assumed to be single-dimensional, as a multidimensional prediction can be decomposed into multiple one-dimensional prediction problems. It may be finite, in which case the problem is called classification, or continuous, in which case it is called regression. Regression methods may easily be applied to classification problems through simple thresholds.

Example: Fisher's *Iris* data set

To have a concrete example, consider Fisher's *Iris* data set [3] containing samples of three different species of the flowering plant in the genus *Iris*. There are 50 samples of each species, giving a total of 150 samples, and each sample has four features: sepal length, sepal width, petal length and petal width. Table 2.1 lists some samples. The goal is to predict the species of a new sample given its features, where the species is usually encoded ordinally as

$$\begin{aligned} \mathit{Setosa} &\mapsto 0, \\ \mathit{Versicolor} &\mapsto 1, \\ \mathit{Virginica} &\mapsto 2, \end{aligned} \tag{2.1}$$

or as a one-hot vector by

$$\begin{aligned} \mathit{Setosa} &\mapsto (1,0,0)^\top, \\ \mathit{Versicolor} &\mapsto (0,1,0)^\top, \\ \mathit{Virginica} &\mapsto (0,0,1)^\top. \end{aligned} \tag{2.2}$$

One-hot encoding is better suited for multi-class classification problems like this, allowing probabilistic predictions of the respective classes, but for binary classification problems, ordinal encoding is more convenient, needing only a single to describe both probabilities. In Fisher's original paper, linear discriminant analysis was used to classify the flowers. Since then, the data set has become a standard benchmark for statistical classification techniques and machine learning methods like support vector machines.

2.1.2 Models

Deterministic models

In machine learning, models are typically deterministic and thus can be seen as functions $f : \mathcal{X} \to \mathcal{Y}$ that map inputs to outputs,

$$\hat{y} = f(\boldsymbol{x}), \tag{2.3}$$

Table 2.1: Fisher's *Iris* data set. The premier column, species, is the class label. The latter four columns are the features, all measured in centimetres.

Species	**Sepal length**	**Sepal width**	**Petal length**	**Petal width**
Setosa	5.1 cm	3.5 cm	1.4 cm	0.2 cm
Setosa	4.9 cm	3.0 cm	1.4 cm	0.2 cm
...	...	...	...	...
Versicolor	7.0 cm	3.2 cm	4.7 cm	1.4 cm
...	...	...	...	...
Virginica	6.3 cm	3.3 cm	6.0 cm	2.5 cm
...	...	...	...	...

where $\hat{y}$ is the predicted output. For instance, a deterministic model for Fisher's *Iris* data set would take in some $\boldsymbol{x} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ and output a species $\hat{y} \in \{0, 1, 2\}$ as per eq. (2.1).

A model belongs to some family, a set of models with similar properties, where families often are parametric. Parametric models are defined by a finite set of parameters $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_k\}$, where each parameter θ_i is a real number. Linear regression, for example, is a parametric model with parameters $\boldsymbol{\theta} = \{w_1, \ldots, w_N, b\}$, defining a linear function $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$.

Probabilistic models

With probabilistic models, the model instead attempts to reconstruct the conditional probability distribution $p(y|\boldsymbol{x})$. In statistics, this distribution will often be of a known type which is parametrised by a function of the data and some parameters. For example, a linear regression model can be used as a probabilistic model with a Gaussian distribution for the output, such that the conditional distribution is given by $p(y|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x} + b, \sigma^2)$, where $\boldsymbol{w}$, b and σ are the parameters of the model. A probabilistic model for Fisher's *Iris* data set would take in some $\boldsymbol{x} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ and output a probability vector $p(y|\boldsymbol{x}) \in [0, 1]^3$. Classification models are often probabilistic, predicting the probability of each class. In that way, the model output is continuous and potentially differentiable.

Alternatively, the conditional distributions may be thought of as being defined by

$$p(y|\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon, \tag{2.4}$$

where ε is a random variable and f a deterministic function. The ε can be seen as some inherent randomness or noise in the data, or as a result of lacking information. It is often assumed to be independent of the input.

Probabilistic models can be converted to deterministic models using a method to

extract a single output from the distribution, such as the maximum a posteriori,

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} p(y|\boldsymbol{x}), \tag{2.5}$$

or the mean,

$$\hat{y} = \int_{\mathcal{Y}} y\, p(y|\boldsymbol{x}) \mathrm{d}y, \tag{2.6}$$

for regression problems.

Probabilistic models have the benefit of being able to provide a measure of uncertainty in the prediction, which is often useful. With simpler settings, statistical arguments can be made about the relationship between the input and output, which can be used to make inferences about the data. However, the simpler design of deterministic models makes them more suitable for computationally demanding tasks, such as deep learning, and in automated settings, where decisions need to be made quickly and uncertainty is not a concern.

2.1.3 Measuring and optimising performance

A loss function, $L(y', \hat{y})$ for deterministic models or $L(y', p(y|\boldsymbol{x}))$ for probabilistic models, is used to optimise a model by comparing its predictions to the true labels y'. By convention, the loss should be positive, and lower values indicate better predictions, zero indicating exact agreement. Its main purpose is to optimise the model, so it need not be the most useful metric of performance. It is usually chosen to be differentiable, as this allows for the use of gradient-based optimisation. Therefore, something like $I(\hat{y} = y')$ is not used as a loss (though often as a performance metric). Instead, losses like the square loss,

$$L(y', \hat{y}) = (y' - \hat{y})^2, \tag{2.7}$$

is often used for regression problems.

If the model is probabilistic and assumed to be a particular type of distribution, its structure can be used to define a loss. Maximum likelihood estimation can be used, switching only the sign to conform with machine learning conventions, leading to the negative log-likelihood (NLL) loss,

$$L(y', p(y|\boldsymbol{x})) = -\log p(y'|\boldsymbol{x}). \tag{2.8}$$

This loss is often used for classification problems, where the output is a probability vector $(p_0, p_1, \ldots, p_N)^\top$, and the true label is encoded as a one-hot vector $y' = (0, 0, \ldots, 1, \ldots, 0)^\top$. It then reads

$$L(y', p(y|\boldsymbol{x})) = \sum_{i=1}^{k} y'_i \log p_i, \tag{2.9}$$

where k is the number of classes. This formulation is referred to as cross-entropy loss in machine learning.

With some loss, one can consider the risk, defined as the expected loss over the joint distribution of the data,

$$R(f) = \mathrm{E}(L(y', f(\boldsymbol{x}))) = \iint_{\mathcal{X}\times\mathcal{Y}} L(y', f(\boldsymbol{x}))\, p(\boldsymbol{x}, y)\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}y, \tag{2.10}$$

and wish to minimise it. Of course, this is only possible if the joint distribution $p(\boldsymbol{x}, y)$ is known, so it is in practice replaced by the empirical risk, assuming independent and equally likely samples,

$$\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, f(\boldsymbol{x}^{(i)})). \tag{2.11}$$

This can be assumed to converge to the true risk as $n \to \infty$ by the law of large numbers. More data is certainly better, but it may be expensive to collect, and the deviance between the empirical and true risk may not decrease monotonically with n. It is therefore desirable to find models and training methods that can minimise the true risk with as few samples as possible.

These empirical risks that depend on the whole data set defines cost functions that are used to train the model. For instance, square loss leads to the mean squared error (MSE),

$$C(f; \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} (y'^{(i)} - f(\boldsymbol{x}^{(i)}))^2, \tag{2.12}$$

while a likelihood based loss leads to

$$C(p; \mathcal{D}) = -\frac{1}{n}\sum_{i=1}^{n} \log p(y'^{(i)} | \boldsymbol{x}^{(i)}). \tag{2.13}$$

For a chosen parametric model family, the supervised learning problem can be formulated as an optimisation problem to find the best parameters $\boldsymbol{\theta}^*$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}; \boldsymbol{\mathcal{D}}), \tag{2.14}$$

where the cost function is parametrised by the model parameters $\boldsymbol{\theta}$. Since the loss function and models are often designed to be differentiable, giving differentiable costs, this can be solved using gradient descent methods. In the simplest case, that means that the parameters are updated until some convergence criterion is met by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}^{(t)}, \mathcal{D}), \tag{2.15}$$

where $\eta > 0$ is a learning rate and $\boldsymbol{\theta}^{(t)}$ the parameters at iteration t. In practice, more sophisticated gradient methods are often used, such as the Adam optimiser [4].

The process of optimising the cost function is called training. As the model improves, it is said to learn.

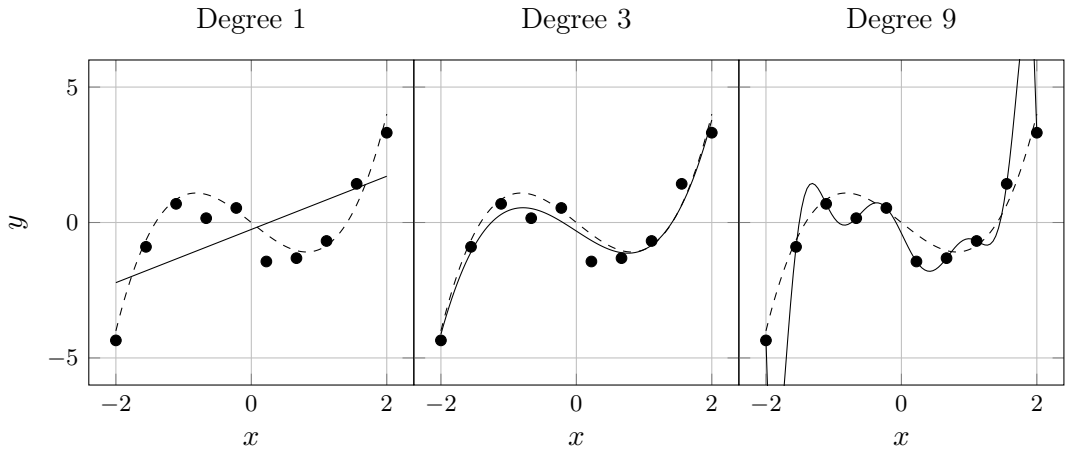


Figure 2.1: A simple example of overfitting and underfitting. Ten data points were generated by $x^3 - 2x$ plus some Gaussian noise with 0.4 standard deviation, shown in the figure as dots. The solid lines denote models fitted using squared loss, being respectively polynomials of degree 1, 3 and 9, while 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.

2.1.4 Generalisation: the bias-variance trade-off

The goal of supervised learning is to find a model that performs well on unseen data. Optimising the cost function on the training data will lead to a model that performs well on the training data, but may not generalise well to unseen data. Therefore, the model is evaluated on a test set, which is not used for training.

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.

There may be mismatch between the true risk in eq. (2.10) and the empirical risk in eq. (2.11). Minimising the empirical risk may not be the same as minimising the true risk. Generally, 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 small or no 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-art 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. By cleverly storing intermediate results, the gradients for all parameters can be computed in a single backward pass through the network.

How such large models avoid overfitting is not entirely clear. Seemingly contradicting the bias-variance trade-off, the double descent phenomenon appears as model complexity increases or as more gradient descent iterations are done. This is a phenomenon where, after some point, the variance no longer increase with model complexity. Instead, it decreases and converges toward some value. This can be seen in fig. 2.2, where convolutional networks[1] were tested with different layer sizes. Unlike statistical methods, once past this complexity threshold, there is little reason, other than the increased computational cost, not to increase the model complexity and thereby the performance. Double descent have been shown to appear for a variety of models [5].

With the great size and complexity, interpretability is sacrificed. The models are deterministic and often black boxes; 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 states that a neural network with a single hidden layer can approximate any continuous function to arbitrary precision, given enough neurons[2]. This gives some credence to the idea that NNs of other structures could be used to approximate intricate functions.

2.2.1 Architectures

Dense feed-forward neural networks

The basic neural network is a dense feed-forward neural network, with a typical structure shown in fig. 2.3. There can be more or fewer nodes in each layer, and as many or few hidden layers as desired.

In the input layer, the data is fed in with each input node corresponding to a feature. Then, in the hidden layers, the data is transformed by a series of linear transformations

[1] See section 2.2.1.

[2] In addition to some easily met requirements regarding the activation function.

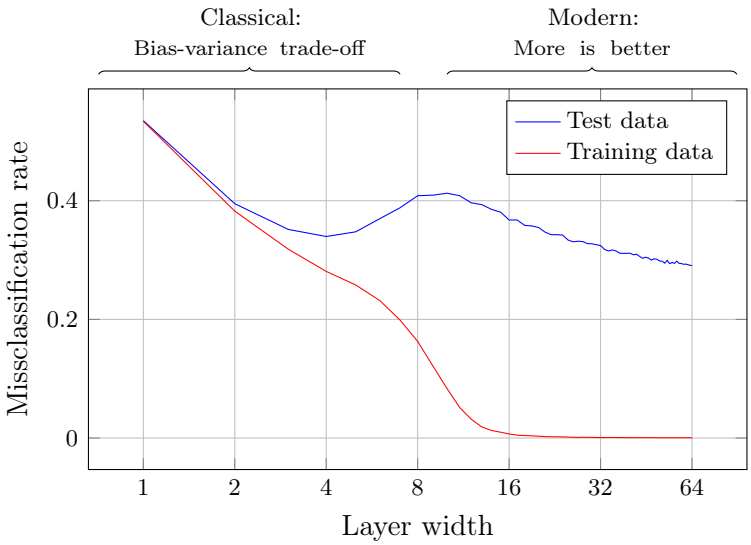


Figure 2.2: Double descent phenomenon. As model complexity increases, the train error decreases steadily, but the test error reaches a minimum and after which the test error starts to increase again. However, past a certain point, the test error decreases again. This defines two regimes: the first classical regime, where the bias-variance trade-off applies, and model complexity must remain low to avoid overfitting, compared to the modern ML regime, where more complexity is always better. The data is from [5], where ResNet18 models were used for image classification with added label noise.

and non-linear activation functions. These can be expressed as

$$a_i^{(j)} = \sigma\left(b_i^{(j)} + \sum_{n=1}^{m} w_{i,n}^{(j)} a_n^{(j-1)}\right), \tag{2.16}$$

where j is the layer number, i is the node number and w and b are the weights and biases of the network — parameters to be optimised.

The activation function σ is a non-linear function, such as the sigmoid function, the hyperbolic tangent or the rectified linear unit (ReLU). Non-linearity is needed for the network not to collapse to one great linear transformation. Some commonly used activation functions are listed in table 2.2.

The output layer is similar to the hidden layers, though perhaps with different activation functions and fewer nodes. For instance, if the goal is to classify the data into k classes, the output layer could have k nodes with an activation function that ensures the sum of the outputs is one. In that way, the output can be interpreted as probabilities of the data belonging to the respective classes.

Models being dense mean that each node in one layer depends on all nodes in the previous layer. It is feed-forward, because the data flows in one direction; the perceptrons in a layer depends only on those in the former.

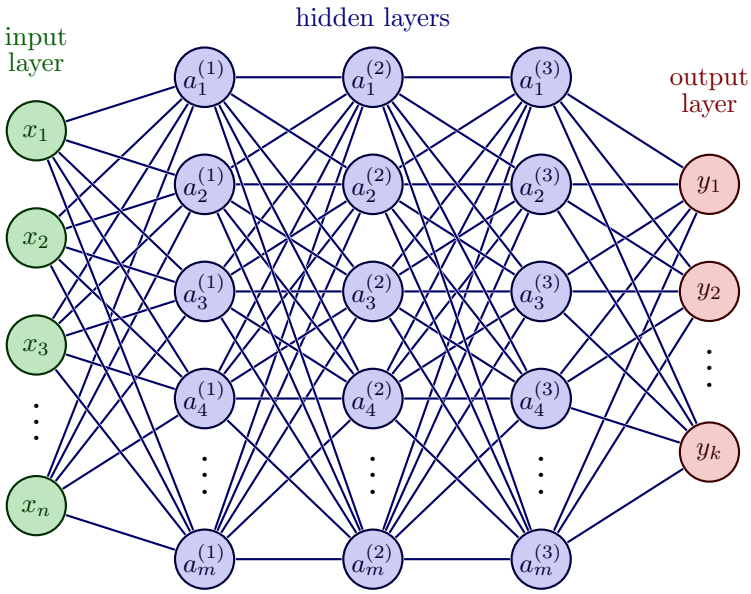


Figure 2.3: Typical structure of a dense feed-forward neural network. Here it has three hidden layers of constant size m, but the number of layers and the size of each layer can be chosen arbitrarily. Being dense means that each node in one layer is connected to all nodes in the next layer, while being feed-forward means that the connections are unidirectional. From [6].

Table 2.2: Common activation functions. Usually, they are applied element-wise to the output of a linear transformation. However, in the case of the softmax function, it depends on the whole layer.

Name	**Definition**	**Typical use case**
Identity	$\sigma(x_i) = x_i$	Regression output
Sigmoid	$\sigma(x_i) = 1/(1 + e^{-x_i})$	Hidden layers, binary classification output
Hyperbolic tangent	$\sigma(x_i) = \tanh(x_i)$	Hidden layers, binary classification output
ReLU	$\sigma(x_i) = \begin{cases} x_i, & x \geq 0 \\ 0 & x < 0 \end{cases}$	Hidden layers
Leaky ReLU	$\sigma(x_i) = \begin{cases} x_i, & x \geq 0 \\ 0.01x_i, & x < 0 \end{cases}$	Hidden layers
Softmax	$\sigma(x_i) = e_i^x / \sum_{j=1}^{k} e^{x_j}$	Multi-class classification output

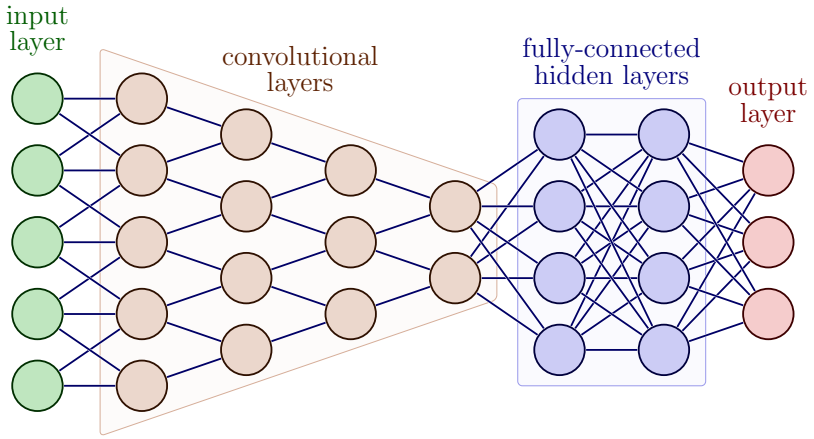


Figure 2.4: The basic structure of a convolutional neural network. Data is input before conventional layers are applied, in which perceptrons are only connected to small regions of the previous layer. After sufficient dimensionality reduction, regular dense layers can be used. From [6].

Convolutional neural networks

Convolutional neural networks (CNNs) are a special type of neural network that are particularly suited for certain tasks like image processing. A greatly simplified CNN is shown in fig. 2.4.

The basic component of the CNN is the convolutional layer. In it, a kernel or filter is applied to the input data, which often is as simple as a 3×3 matrix. It is applied to only parts of the input, and thus only extracts local properties. Typically, pooling layers complement the convolutions by reducing the dimensionality through some simple non-parametrised operation, such as taking the maximum value in a 2×2 matrix. CNNs generally finish with one or more dense layers, which then operate on a significantly reduced number of features. The reduction of dimensionality is important because it reduces the number of parameters in the model and the risk of overfitting. Furthermore, the convolutional approach forces the model to learn local features. This is very beneficial for tasks such as image classification, where local features, such as edges, are more important than global ones, such as the position of the subject.

Generative adversarial networks

Generative adversarial networks (GANs) refer to special kind of design where two different networks are trained by competing against each other. With an unlabelled dataset, the goal is to generate data that is indistinguishable from the real data. The first model, the generator, attempts to generate samples from the underlying distribution. On the other hand, the discriminator is tasked with distinguishing between data produced by the generator and the real samples from the data set. Accordingly, the discrimination is a

supervised problem. Both models are trained simultaneously by first sampling from the generator and the data set, using supervised methods to update the discriminator, and then using the discriminators predictions to update the generator.

GANs are mainly used for unsupervised learning, having had great success in generating random images. They have also demonstrated success in more abstract tasks, such as translating text prompts to images or predicting what will happen next in a video.

Chapter 3

Quantum computing

The field of quantum computing is split into two main branches: the development of quantum hardware and the study of algorithms that use such hardware. Only the second branch is relevant for this thesis, and even so only a brief explanation is offered here. This chapter is primarily based on the Qiskit textbook [7], *Machine Learning with Quantum Computers* by Schuld and Petruccione [8] and *Quantum Computation and Quantum Information* by Nielsen and Chuang [9]. The discussion of variational quantum algorithms in section 3.4 is based on the review by Cerezo et al. [10]. Any reader should have a basic understanding of linear algebra and classical computing. Knowledge of quantum mechanics is not assumed, albeit certainly helpful.

3.1 Quantum states

3.1.1 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 $|1\rangle$[1], and the qubit can be in any superposition of these states. The state of the qubit lies in a two-dimensional Hilbert 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 α and β are any numbers, even complex. The only requirement is that the state is normalised, $|\alpha|^2 + |\beta|^2 = 1$.

Normalisation is required due to the Born rule, as the absolute square of the coefficients is the probability of measuring the qubit in the corresponding basis state. It is one of

[1]The $|\cdot\rangle$ notation is known as a ket and is used in quantum mechanics to denote a quantum state. It is effectively a column vector in a Hilbert space, whose inner product may be taken with a bra, $\langle\cdot|$, to give a scalar. These inner products are then denoted by $\langle\cdot|\cdot\rangle$. Similarly, outer products are well-defined and denoted by $|\cdot\rangle\langle\cdot|$.

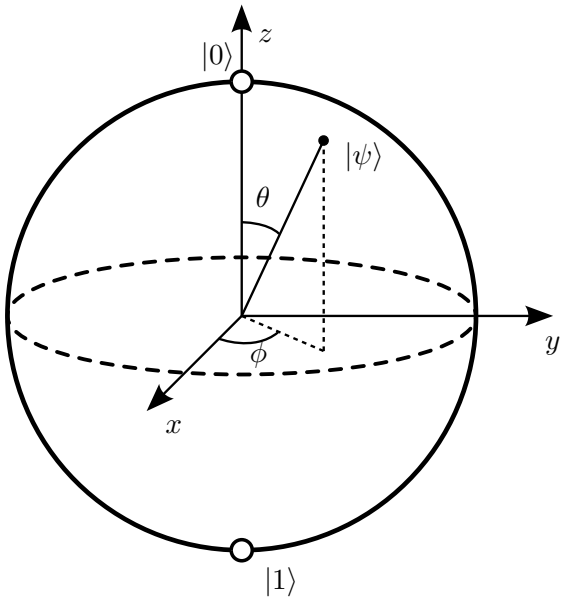


Figure 3.1: The Bloch sphere. On it, the state of a single qubit state is represented by a point. The state $|0\rangle$ is the north pole, and $|1\rangle$ is the south pole. The latitudinal angle θ determines the probability of measuring the qubit in the state $|0\rangle$, while the longitudinal angle ϕ corresponds to the complex phase between the two basis states. From [11].

Nature's great mysteries what exactly a measurement is, but in the quantum computational setting, it can be thought of as taking a random sample with the probabilities given by the coefficients, which can be done systematically.

3.1.2 The Bloch sphere

A useful tool for visualising the state of a qubit is the Bloch sphere. First, it should be noted that 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 to 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 θ as the polar angle and ϕ the azimuthal angle, the state of the qubit can be identified with a point a sphere, known as 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 with the state of the qubit in eq. (3.2).

3.1.3 Mixed states and density operators

It is not only the superpositions of states that are important in quantum computing, but also the mixed states in which classical uncertainties manifest. For the description of mixed states, the formalism of density operators is more useful than the state vector formalism. In a mixed state, some classical probabilities p_i are associated with the different states $|\psi_i\rangle$, and the state of the system is described by the density operator

$$\rho = \sum_{i=1}^{n} p_i \, |\psi_i\rangle\langle\psi_i| \tag{3.3}$$

where $|\psi_i\rangle$ are the states of the system, and $\langle\psi_i|$ are the corresponding dual vectors. Naturally, the p_i must be non-negative and sum to one. Furthermore, the density operator is positive semidefinite. If there is no classical uncertainties, the state is called pure, and the density operator can be expressed a single ket-bra, $\rho = |\psi\rangle\langle\psi|$.

For a density operator, the diagonal elements are the probabilities of measuring the system in the corresponding basis states, and hence they are positive and the trace one. Ergo, for a single qubit, the density operator can be expressed as

$$\rho = \frac{1}{2}\left(I + x\sigma_x + y\sigma_y + z\sigma_z\right). \tag{3.4}$$

Here, $x, y, z \in \mathbb{R}$, and $\sigma_x, \sigma_y, \sigma_z$ are the Pauli matrices,

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{3.5}$$

which together with the identity matrix serve as a basis for Hermitian 2×2 matrices. Being positive semidefinite, the determinant should be non-negative, and thus it can be showed that $x^2 + y^2 + z^2 \leq 1$. This allows density operators to be interpreted as points on the Bloch sphere or indeed within it. Notably, pure states lie on the surface, while mixed states lie within the sphere (or rather, the Bloch ball). A pure quantum superposition of $|0\rangle$ and $|1\rangle$ with equal probabilities would have a complex phase and lie somewhere on the equator, while a statistical mixture with equal classical probabilities of being $|0\rangle$ and $|1\rangle$ would lie in its centre.

3.1.4 Systems of 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.

With just two qubits, there are four possible states, $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. Each of these four have their own probability amplitude, and thus their own probability of being measured. A two-qubit system will therefore operate with four complex numbers.

Generally, the state of multiple qubits can be expressed using the tensor product as

$$|\psi_1\psi_2\cdots\psi_n\rangle = |\psi_1\rangle\,|\psi_2\rangle\cdots|\psi_n\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle. \tag{3.6}$$

What makes this so powerful is that the state of a multi-qubit system can be anything on the form

$$|\psi_1\psi_2\cdots\psi_n\rangle = c_1\,|0\ldots00\rangle + c_2\,|0\ldots01\rangle + \cdots + c_{2^n}\,|1\ldots11\rangle = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{2^n} \end{pmatrix} \in \mathbb{C}^{2^n}, \quad (3.7)$$

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. This is the origin of the exponential speed-up of quantum computers.

3.2 Quantum operations

3.2.1 Single-qubit gates

To operate on one or more qubits, a unitary operation is applied to the state, where the unitarity is needed for states to remain normalised. As they are unitary, any purity is preserved. With the finite number of qubits, these operations can be expressed as matrices. These operations are often thought of as gates, paralleling the classical gates in digital logic. Mathematically, a unitary gate U can be expressed as matrices acting on the state vector, $|\psi\rangle$, as

$$|\psi'\rangle = U\,|\psi\rangle\,, \quad (3.8)$$

where $|\psi'\rangle$ is the resulting state.

The most basic gates are the Pauli gates, which are applications of the Pauli matrices from eq. (3.5) and are as gates simply denoted as X, Y, and Z respectively. These gates can be seen as half turns around the x-, y- and z-axes, respectively, of the Bloch sphere. 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. It is however more general, being also applicable to superposition states.

The Hadamard gate,

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad (3.9)$$

is a rotation around the line between the x- and z-axes by $\pi/2$. It is an important gate in quantum computing, as it is used to create superpositions of the computational basis states. Applied on the initial $|0\rangle$ state, it creates the entangled state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Two consecutive applications thereof is a no-op, the second returns the state to the initial state, as can be seen from the matrix squaring to the identity.

The R_X-, R_Y- and R_Z-gates are rotations around the x-, y- and z-axes, respectively,

by an arbitrary angle θ:

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

These parametrised gates will be useful in section 3.4.

3.2.2 Multi-qubit gates

The most used multi-qubit gate is the controlled X-gate, also known as the CNOT. 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. If the first qubit has probability amplitude α of being in the state $|1\rangle$, the second qubit will have probability amplitude α of being flipped. The CNOT gate can be expressed in matrix form as

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.10}$$

In theory, any unitary single-qubit operation can be controlled. Another interesting two-qubit gate is the controlled Z-gate, CZ, expressible as the matrix

$$\text{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{3.11}$$

Because it only alters the amplitude of $|11\rangle$, it does not actually matter which qubit is the control and which is the target.

3.2.3 Observables and measurements

For an output to be obtained from a quantum computer, a measurement must be performed. This is typically done at the end of all operations and of all qubits, thus yielding a single output of zeros and ones, a bit-string. It is important to note that the measurement is not a deterministic process, but rather a probabilistic one. Often, the underlying probabilities are what is of interest. Therefore, many measurements are performed. Usually, these results are averaged to obtain an estimate, but more complicated post-processing methods

are also possible. For instance, neural networks have shown useful properties in regard of reducing variance in the estimates, though at the cost of some bias [12].

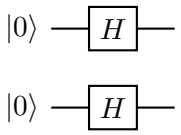
The Z-basis is the canonical basis for measurements, but any other basis can be used, at least in theory. Often, only canonical basis measurements are implemented in the hardware. Using another basis can be done by properly modifying the state before the measurement; a change of basis can simply be implemented by a unitary operation.

Measurements may be done in the middle of operations and be used to control the operations. If the qubits are entangled, measuring one will affect the measurement probabilities of others. Using such intermediate measurements is a way of introducing non-linearities in the otherwise unitary nature of the quantum world.

3.2.4 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 $|0\rangle$. Gates are generally written as boxes with the name of the gate inside.

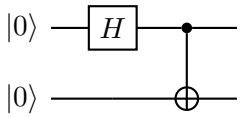
A simple example is the circuit



, (3.12)

which prepares the state $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. This is a pure state with no entanglement, and so the measurement probabilities of the two qubits are independent.

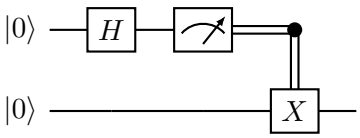
Slightly more interesting is the circuit



(3.13)

in which the first qubit is put into a superposition using the Hadamard gate before a CNOT gate is applied to the second, controlled on the first. This creates the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. The measurement probabilities of the two qubits are now correlated; if the first qubit is measured to be $|1\rangle$, the second will always be $|1\rangle$ and vice versa. The probability of measuring the qubits to be different is nil.

To create a mixed state, an intermediate measurement can be used to control a gate. For instance, the circuit



(3.14)

places the second qubit in the mixed state $\frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$. If it were immediately to be measured, it would have a 50% chance of being $|0\rangle$ and a 50% chance of being $|1\rangle$. The

uncertainty is only classical, and it could therefore not be used to create entanglement or for any other quantum spookiness.

3.2.5 Quantum supremacy

Exponential speed-ups do not come for free. Quantum computers do only solve certain problems more efficiently than classical computers, and finding the algorithms to do so is not trivial. Shor's algorithm has time complexity $O((\log N)^3)$ while the most efficient known classical algorithm has time complexity, the general number field sieve, is sub-exponential with a time complexity on the form $O(\exp(C(\log N)^{1/3}))$ for some constant C. To solve linear system, there is the HHL algorithm with time complexity $O(\log(N)\kappa^2)$, where κ is the condition number. This is an exponential speed-up over the fastest known classical algorithm, which has time complexity $O(N\kappa)$. Still, these are non-trivial algorithms, not yet usable in practice and that were not easily found.

Polynomial speed-ups are more easily found. For example, the Grover algorithm which is used to search for an element in an unsorted list has time complexity $O(\sqrt{N})$. Classically, this can not be done in less than $O(N)$ time. This algorithm, or the more general amplitude amplification on which it builds, solves a very general problem and is often used a subroutine to achieve quadratic speed-ups in other algorithms. Being only a quadratic speed-up, it is not as impressive as the exponential speed-ups, and achieving quantum supremacy in that manner would require larger quantum computers than if the speed-up were exponential.

It is proven that the class of problems quantum computers can solve in polynomial time (with high probability), BQP, contains the complexity class P. This follows from the fact than quantum computers can do any classical algorithm. Since quantum computers can solve problems like integer factorisation and discrete logarithms efficiently, it is believed that BQP is strictly greater than P, but as whether P = NP remains unknown, these problems could actually be in P. In a similar vein, NP-complete problems are believed to lie outside BQP.

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. Once enough qubits are available, error correction can be implemented and truly enable quantum supremacy. Still, the era dubbed NISQ (Noisy Intermediate-Scale Quantum) is the first step, and to make use of the hardware, the inherent noise and its consequences must be understood, and algorithms must take the following limitations into consideration.

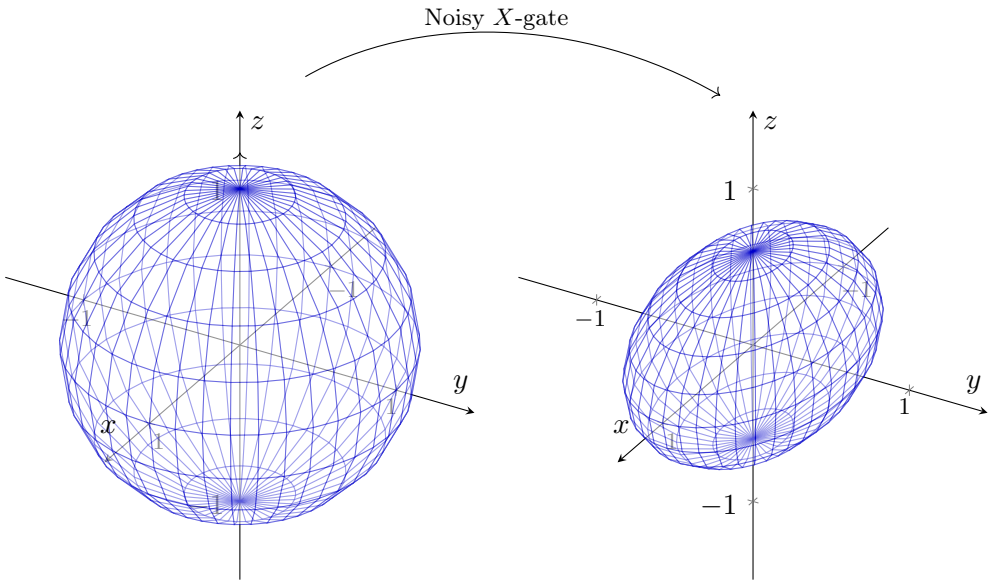


Figure 3.2: Illustration of applying a noisy X-gate on the Bloch sphere. The left plot shows the set of all pure states, the Bloch sphere. On the right, the same set of states is shown after the application of a noisy X-gate with probability $p = 0.2$ of failing. There, the y- and z-axes are contracted towards the centre by a factor $1 - 2p = 0.6$, introducing mixing.

3.3.1 Quantum channels and decoherent noise

The unitary gates from section 3.2 are an idealisation, and in reality, the gates are not perfect. A more realistic description of the gates is given by quantum channels, which can take noise into account. In an unmeasured, ideal quantum system, the state evolution is veritably unitary, but as the system in practice does interact with the environment, this can be modelled as a quantum channel in which traditionally probabilistic noise is introduced.

Take the X-gate as an example. If the fails to apply with probability p, the resulting density operator ρ' can be expressed as

$$\rho' = p\rho + (1-p)X\rho X^\dagger, \tag{3.15}$$

where ρ is the initial density. Such a channel has eigenvalues p and $1-2p$, where I and X share the former and $Y$and Z the latter. Consequently, states with Y- or Z-components will have mixing introduced. Geometrically, this can be interpreted as contraction of states on the Bloch sphere towards the centre, illustrated in fig. 3.2.

Any physical gate will suffer some such noise, and so the tendency will be for states to degenerate towards the mixed centre of the Bloch sphere (or its higher-dimensional analogue). Furthermore, measurements, letting a qubit idle when operating on others and even the preparation of the initial state $|0\rangle^{\otimes n}$ will suffer from decoherence.

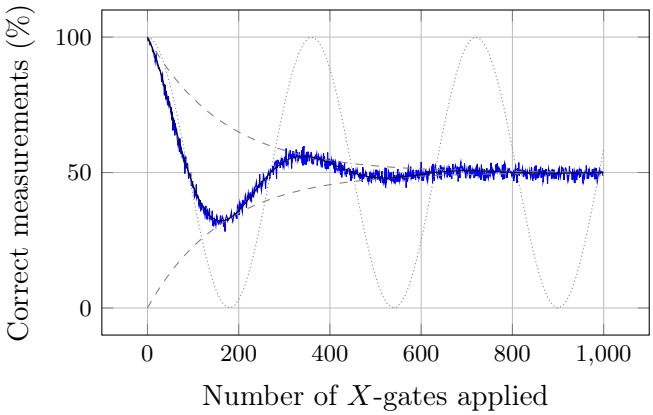


Figure 3.3: Proportion of correct measurements of a qubit after applying several noisy X-gates. The error in the rotation is 1°, while the probability of failing to apply the gate is $p = 0.3\%$. For each number of gates, the measurement is repeated 1000 times. The coherent error causes a sinusoidal behaviour, in which 180 rotations causes the overall rotation to be completely opposite. The decoherent error causes an exponential dampening, as the state becomes more and more mixed. The wiggly pattern is simply classical randomness stemming from the probabilistic measurement with a finite number of samples.

This noise is very hard to avoid, as controlling the qubits necessarily requires some interaction with the environment. Additionally, to operate on multiple qubits, a connection between them is required, which too will suffer from noise. Nature tends to work against large quantum systems, made evident by the absence of quantum effects in regular life.

Due to the multiplicative effect of noise, the overall degeneracy will increase exponentially with the number of operations. This means that there will be limits to the number of operations that can be performed before the system becomes unusable, assuming no error-correction is done.

3.3.2 Coherent noise

There may also be systematic errors in the unitary gates applied. This kind of noise, known as coherent noise does not need quantum channels to be modelled, but may rather be modelled simply by slightly different unitary gates. Here too, the X-gate can serve as an example. Such a gate is typically implemented by applying a Hamiltonian for some time, requiring calibration of said time.

If it is not calibrated correctly, the effective operation will be a rotation that slightly deviates from the intended half turn. When a gate like the X-is applied many times, even small errors will add up, which could cause an overall rotation of the state by a significant angle.

Figure 3.3 shows the effects of both coherent and decoherent noise on the measurement of a qubit after applying several noisy X-gates. Clearly, information is quickly lost as the

quantum state ends up different from what is expected and as the system becomes more and more mixed, losing quantum properties and instead obeying classical probabilities. Even though current hardware suffers much less noise than the figure shows, NISQ algorithms must nonetheless be shallow, meaning that the amount of gates applied before measurement is small. There are more sources of noise than the two included in the figure, and having to deal with multiple qubits certainly does not help.

3.3.3 Qubit counts and connectivity

Another limiting factor is the amount of qubits. Current hardware has around 10 to 100, which though still may be enough to express states too large for classical computers, is not enough to perform the most demanding algorithms. Recent estimates require millions of noisy qubits would be needed to break RSA encryption [13]. Another current limitation is the connectivity between the qubits. Not all of them are directly linked, which means that applying a multi-qubit gate may require intermediate swapping. This increases circuit depth, which in turn increases error rates.

3.4 Variational quantum algorithms

Variational quantum algorithms (VQAs) are envisioned as the most likely candidate for quantum advantage to be achieved in the NISQ-era. 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. This loop of running the circuit, measuring the output to evaluate a cost, optimising and updating the parameters is pictured in fig. 3.4.

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 imply 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.

Important to the success of a VQA is the choice of the quantum circuit. The circuit selected is known as the ansatz, and there are a plethora of different ansätze to choose from. An example are hardware-efficient ansätze, which, as the name implies, is a general term used for ansätze designed in accord with the hardware properties of a given quantum computer, taking for instance the physical gates available into account and minimising

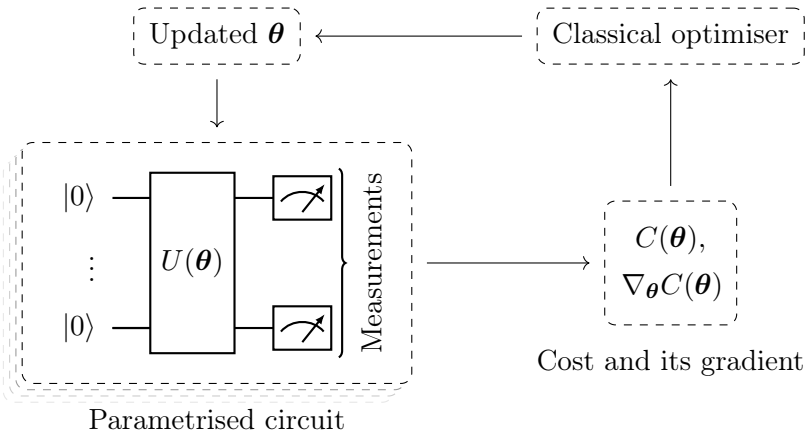


Figure 3.4: The general structure of a VQA. Given some initial parameters $\boldsymbol{\theta}$, a parametrised quantum circuit is run multiple times on quantum hardware, after which a cost $C(\boldsymbol{\theta})$ function is calculated based on the measurements. Additionally, using e.g., the parameter-shift rule, the gradient of the cost function with respect to the parameters can also be calculated. The parameters of the quantum circuit are subsequently updated by a classical optimiser, and the process is repeated until some convergence criterion is met.

the circuit depth. Other ansätze may be problem-specific, like the quantum alternating operator ansatz used for the quantum approximate optimisation algorithm (QAOA) (both sharing the QAOA acronym, confusingly), while others are more general and can be used to solve a variety of problems.

The optimisation of the cost function is often done with gradient descent methods. To evaluate the gradient of the quantum circuit with respect to the parameters, the very convenient parameter shift rule is commonly used. Though appearing almost as a finite difference scheme, relying on evaluating the circuit with slightly shifted parameters, it is indeed an exact formula. Not having to rely on finite differences is a major advantage, as the effects of small changes in parameters would quickly be drowned out in noise. Furthermore, it may be used recursively to evaluate higher order derivatives, which allows the usage of more advanced optimisation methods like the Newton method that requires the Hessian.

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, with another common example being QAOA for the max-cut problem.

Still, there are many difficulties when applying VQAs. Exponentially vanishing gradients, known as barren plateaus, are a common occurrence, making 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 others 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.

Chapter 4

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 4.1, first described by Schuld and Petruccione [14].

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 influences classical machine learning, such as the quantum-inspired application of tensor networks in [15].

Using classical machine learning for quantum data includes improving quantum computers' general performance. For example, with machine learning algorithms, the variance of the measurements can be reduced, as shown in [12]. 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

Table 4.1: 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. Adapted from [14].

		Computing device	
		Classical	*Quantum*
Data	*Classical*	CC	CQ
	Quantum	QC	QQ

Table 4.2: Properties of different data encodings. Given N-dimensional data set of M data points, qubits needed is a lower bound for qubits required to represent the data, and circuit depth is the number of gates needed for the encoding algorithm. For basis encoding, $b(N) \geq N$ is the number of bits needed to represent an N-dimensional data point, for instance by using floating point representations of continuous data.

Encoding strategy	**Qubits needed**	**Circuit depth**	**Hard to simulate classically**
Basis encoding	$b(N)$	$O(b(N))$	No
Amplitude encoding	$\lceil \log_2 N \rceil$	$O(N)$	Yes
Angle encoding	N	$O(N)$	No
Second order angle encoding	N	$O(N^2)$	Yes? (Conjectured)

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 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.

4.1 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 naïve 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 4.2.

4.1.1 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

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left| \boldsymbol{x}^{(m)} \right\rangle , \tag{4.1}$$

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.

4.1.2 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 not to individual qubits, but to individual amplitudes in the exponentially large Hilbert space. Mathematically, for some N-dimensional data point $\boldsymbol{x}$, this reads

$$|\psi(\boldsymbol{x})\rangle = \sum_{i=1}^{N} x_i \, |i\rangle , \tag{4.2}$$

where x_i is the ith component of the data point and $|i\rangle$ is the ith 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

$$|\mathcal{D}\rangle = \sum_{m=1}^{M} \sum_{i=1}^{N} x_i^{(m)} \, |i\rangle \, |m\rangle \tag{4.3}$$

where $x_i^{(m)}$ is the ith component of the mth 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

$$|\psi\rangle = \sum_i a_i |i\rangle , \tag{4.4}$$

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 [14]. 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.

4.1.3 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

$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_X(x) |0\rangle , \tag{4.5}$$

$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_Y(x) |0\rangle \tag{4.6}$$

or

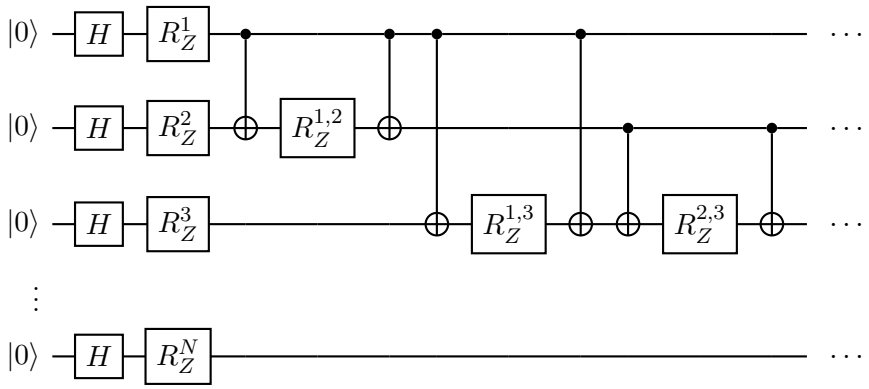
$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_Z(x) H |0\rangle , \tag{4.7}$$

depending on which rotation is used. For Z-rotations, a Hadamard gate is needed for the operation to have an effect. 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.

4.1.4 Second order angle encoding

Havlicek et al. [16] 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



(4.8)

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.

4.1.5 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, Sweke and Meyer [17] study the implications of this, and they show that simply repeating simple encoding blocks allows for learning of more frequencies and thus more complex functions. Asymptotically, such repeats lets a quantum model learn arbitrary functions.

4.2 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 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 4.1 shows the general circuit layout. In the first step or layer, data is encoded into the qubits, typically using a method discussed in section 4.1. 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.

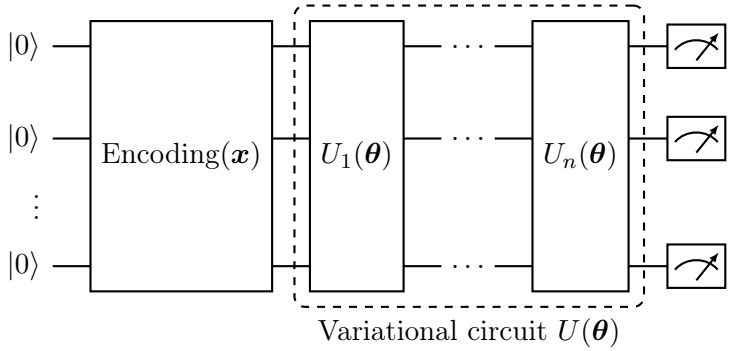


Figure 4.1: General structure of quantum neural networks. First, some data $\boldsymbol{x}$ is encoded into a state $|\psi(\boldsymbol{x})\rangle$ using some encoding strategy. Then, the state is transformed by a parametrised quantum circuit $U(\theta)$. This variational circuit can often be decomposed 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.

Usually, the observable is a combination of Pauli operators on every qubit. Thence, a cost function can be evaluated.

Though methods like the parameter-shift allows for computation of gradients, which makes it possible to train the network using classical methods, it is not as efficient as classical backpropagation. This is because the parameter-shift method requires a separate evaluation of the circuit, including numerous shots, for each parameter, whereas backpropagation is easily done after a forward pass through the network. Add to this the problems of noise and vanishing gradients, and it is clear that QNNs will not scale in the same manner as classical networks do.

4.2.1 Architectures and their applications

Quantum convolutional neural networks

Originally introduced by Cong, Choi and Lukin [18], 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 an output is made. In the quantum convolutional layers, neighbouring qubits are entangled with some parametrised gates. After that, pooling layers reduce (usually by half) the active qubit count. The pooling may also be parametrised. 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, gates can be employed on the remaining qubits, analogous to finishing 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 [18], 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 [19]. They have shown desirable properties with regard to avoiding barren plateaus [20], 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 [21]. 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. Moreover, leveraging a classical model as the adversary ensures that the quantum model can be of reasonable scale. Real quantum hardware has been used to generate (admittedly low-resolution) images of handwritten images [22].

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. As parametrised quantum circuits are differentiable in their parameters, they can easily be handled using the chain rule when backpropagating a hybrid model. Killoran et al. [23] 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.

Quanvolutional neural networks, proposed by Henderson et al. [24], are a hybrid model in which the convolutional layers are replaced by quantum layers. As the quantum part is restrained to a single layer and a small convolutional kernel, the design can be implemented with small quantum circuits with little requirements for error-mitigation, still being able to process high-dimensional data, thereby making it a good candidate for NISQ-era hardware.

More recently, Zeng et al. [25] have explored using a hybrid model to multi-class classification on real world data sets using a CNN-inspired structure. In their model, the quantum part is placed in the middle, after classical convolutions and pooling and before a classical fully connected layer. There, it is shown that the hybrid model outperforms a classical CNN of similar parameter size.

Chapter 5

Simulated comparisons of QNNs

To see if quantum machine learning is indeed a viable alternative to classical machine learning, some empirical comparisons are in order. In this chapter, three quantum neural networks are implemented and tested. First, the convergence of a quantum neural network is compared to a classical neural network. Second, a quantum convolutional neural network is implemented and tested. Lastly, the convolutional network is expanded to include mid-circuit measurements.

All simulations were performed on an Apple M1 Pro processor, with the Qiskit [26] and PennyLane [27] frameworks for quantum computing and PyTorch [28] for classical machine learning and optimisation.

Simulating quantum systems is exponentially hard (after all, it prompted the conception of quantum computing in the first place), so the number of qubits is limited. Only systems of up to around a dozen qubits are easily simulated on consumer-grade hardware. To what extent the results hence applies to larger systems is hard to say.

5.1 Quantum NNs versus classical NNs

In order to compare the performance of the QNN and the NN, architectures suited for binary classification with exactly 8 parameters are used, following approach used by Abbas et al. [29] in 'The Power of Quantum Neural Networks'. In their paper, they study the effective dimension of QNNs and NNs, and find that the effective dimension of QNNs is much higher than that of NNs, which they argue leads to better trainability of QNNs. The goal of this section is to reproduce their results, but also to shed some light on whether the NN used is a fair comparison to the QNN.

5.1.1 Data

The models are tested on Fisher's *Iris* data set, introduced in section 2.1.1. Like in [29], only the two first species are considered, reducing the task to a binary classification problem. These are linearly separable, so the models should be able to learn to classify them.

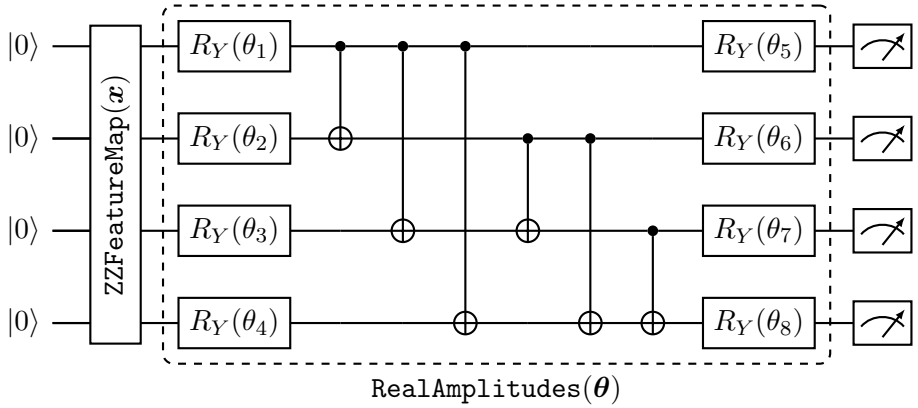


Figure 5.1: Structure of the QNN used for classification of the *Iris* data set. The first block maps the input data $\boldsymbol{x}$ to the quantum state $|\psi(\boldsymbol{x})\rangle$ using a second order Z-rotation feature map. The second block is the variational circuit, parametrised by $\boldsymbol{\theta}$, a vector with eight components. Finally, all qubits are measured, where the parity is interpreted as the prediction.

5.1.2 Quantum model

First, the four-dimensional input data is first scaled to have zero mean and unit variance. Some scaling is necessary for the encoding to work properly, and standardising the data is a natural choice, common in machine learning. Then, it is encoded into a quantum state a second order angle encoding with Z-rotations, discussed in section 4.1.4, 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 5.1 shows the structure of the QNN and how the parameters are used.

Both exact simulations and noisy simulations were performed. To model noise, the Qiskit simulator was set to simulate the 27-qubit IBM Montreal architecture, which was the actual hardware used in [29].

5.1.3 Classical model

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, and the output layer used a softmax activation function. Whether such an architecture makes sense is arguable,

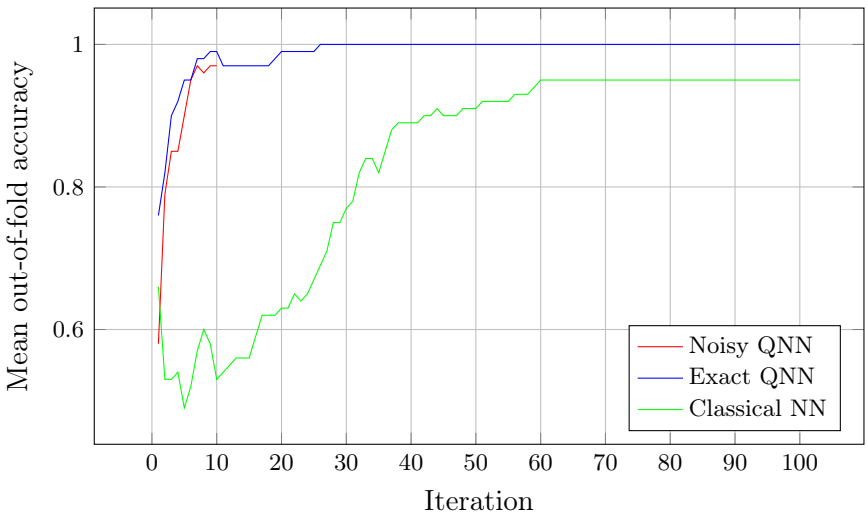


Figure 5.2: Mean accuracies during training for the *Iris* data set. Ten-fold cross-validation was used, with the mean out-of-fold accuracy being used as the final performance metric. All models have 8 parameters and are trained using the Adam optimiser with a learning rate of 0.1, using cross-entropy as the loss function. Due to the computational cost, the noisy (simulated IBM Montreal backend) QNN was only trained for ten epochs.

and it was likely made so mainly to ensure that the number of parameters would be the same as the QNN.

5.1.4 Implementation and training

Both models were implemented using PyTorch, with code partly taken from the original paper[1]. The QNN was adapted to use Qiskit's PyTorch interface. It automatically applies parameter-shift to evaluate gradients. 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.

5.1.5 Results

For validation, 10-fold cross-validation was used. That is, the data set was split into 10 equal parts (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. 5.2.

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

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

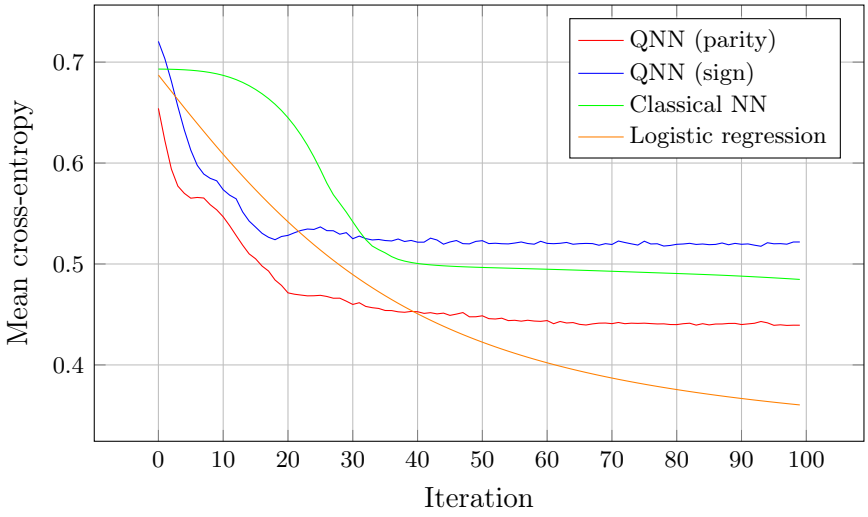


Figure 5.3: Mean cross-entropy loss during training for the *Iris* data set. The QNN with parity interpretation has eight parameters, while the QNN with sign interpretation has effectively five. The classical neural network has eight and the logistic regression has five parameters.

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. For such a simple case, statistical methods like logistic regression or discriminant analysis are better suited than neural networks, but simulating quantum hardware limits the problem size that can be solved. Hopefully, the results extrapolate to more complex problems.

Why parity was chosen as output interpretation is not clear, other than being cited as a 'standard in practice'. However, it is not the only possible interpretation, and a perhaps more intuitive one would be to use the sign of just one of the qubits. Doing so effectively ignores three of the qubits, giving a parameter count of five. Five is coincidentally also the parameter choice of a logistic regression scheme, which could be a more natural choice for a classical classification algorithm than the neural network whose architecture appears mostly based on the goal of having eight parameters.

Looking at the testing visualised in fig. 5.3 reveals that parity indeed performs better than the sign of a single qubit, though not by much. The bigger revelation is the performance of logistic regression, which greatly outperforms the neural network, and while slightly slower to converge than the quantum models, does approach an overall smaller loss value.

5.2 Quantum convolutional neural networks

As quantum CNNs have been studied and shown to have favourable properties (as discussed in section 4.2.1), they are a reasonable choice to implement and test. To do so,

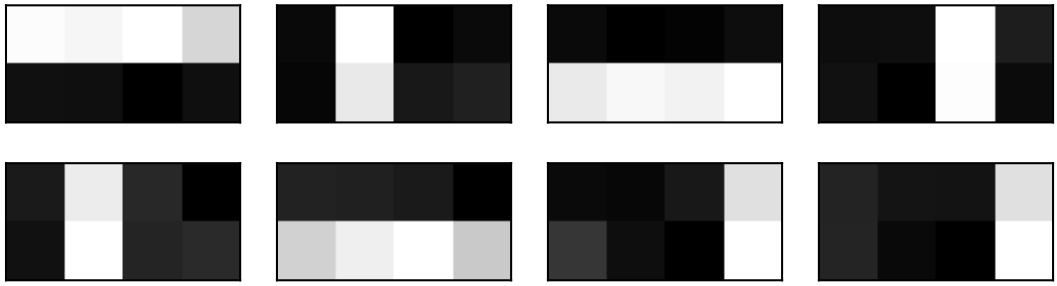


Figure 5.4: Data for the QCNN. With a total of 64 training images and 16 for testing, they form balanced data set of 2×4 pixels, with either a vertical or horizontal line encoded as 1 and -1. The images are generated with some Gaussian noise.

Qiskit's online tutorials [30] were closely followed.

5.2.1 Data

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.

5.2.2 Model

First, data is encoded using two repetitions of Z-angle encoding, implemented in Qiskit as the `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:

$$
\begin{array}{l}
|0\rangle \text{ — } \boxed{H} \text{ — } \boxed{R_Z(x_1)} \text{ — } \boxed{H} \text{ — } \boxed{R_Z(x_1)} \\
|0\rangle \text{ — } \boxed{H} \text{ — } \boxed{R_Z(x_2)} \text{ — } \boxed{H} \text{ — } \boxed{R_Z(x_2)} \\
\vdots \\
|0\rangle \text{ — } \boxed{H} \text{ — } \boxed{R_Z(x_n)} \text{ — } \boxed{H} \text{ — } \boxed{R_Z(x_n)}
\end{array}
. \tag{5.1}
$$

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. Effectively, each consecutive pair of

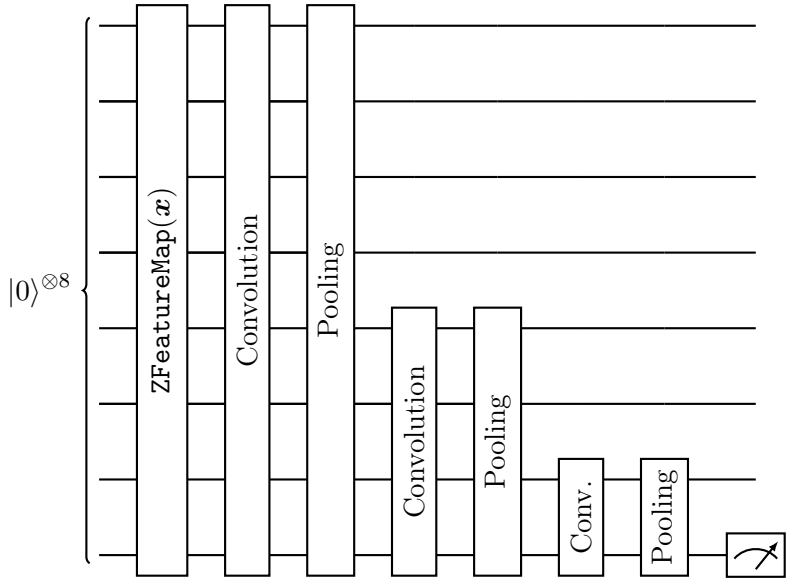
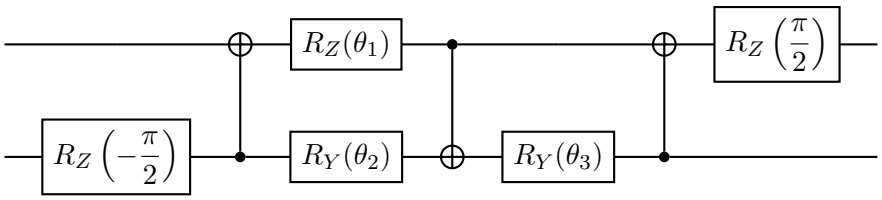


Figure 5.5: Circuit diagram of the QCNN. Data is encoded using Z-angle encoding, before convolutional and pooling layers are applied. The final layer is a single qubit measurement.

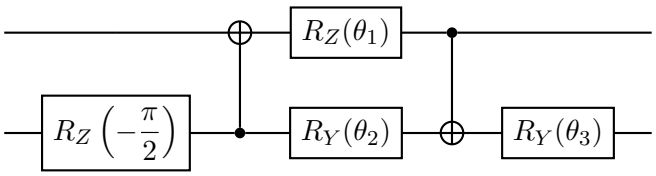
qubits were entangled by



, (5.2)

giving $3n$ parameters for each convolution layer of n qubits.

Thereafter, pooling layers halve the active qubit counts by parametrised rotations and CNOT gates. This is done by acting on the first and fifth, second and sixth et cetera with the following circuit:



, (5.3)

giving $3n/2$ parameters for each pooling layer of n qubits.

For the final layer, the sole remaining qubit is measured, and the result is interpreted as the prediction. In total, the circuit appears as in fig. 5.5 with a total of 63 parameters.

5.2.3 Implementation

As in Qiskit's guide, training was done using the COBYLA optimiser[2] 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 was significantly slower. However, as will be seen in section 5.3, that seems to have more to do with Qiskit's implementation than the optimiser itself.

5.2.4 Results

The accuracies and mean square error during training are shown in fig. 5.6. As in section 5.1, noise is modelled after the 27-qubit 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.

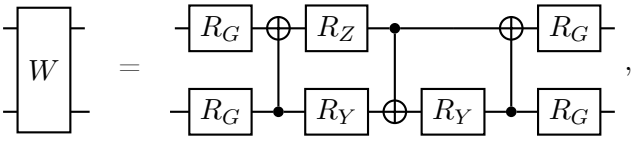
Although both the exact and noisy simulations do converge and yield reasonable results, for such an easy problem, one would expect a 63-parameter network to be able to achieve perfect accuracy. This is not the case, and the reason for this is not clear. A more expressive feature map could improve matters, and the COBYLA optimiser is certainly suboptimal.

5.3 QCNN with mid-circuit measurements

A more intricate QCNN structure was described by Pesah et al. [20], 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. With the aim of seeing if this provides any benefit, such a QCNN was implemented and compared with that of section 5.2 on the data thence.

5.3.1 Model

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 As circuits, they appear as



(5.4)

[2]Constrained Optimisation BY Linear Approximation.

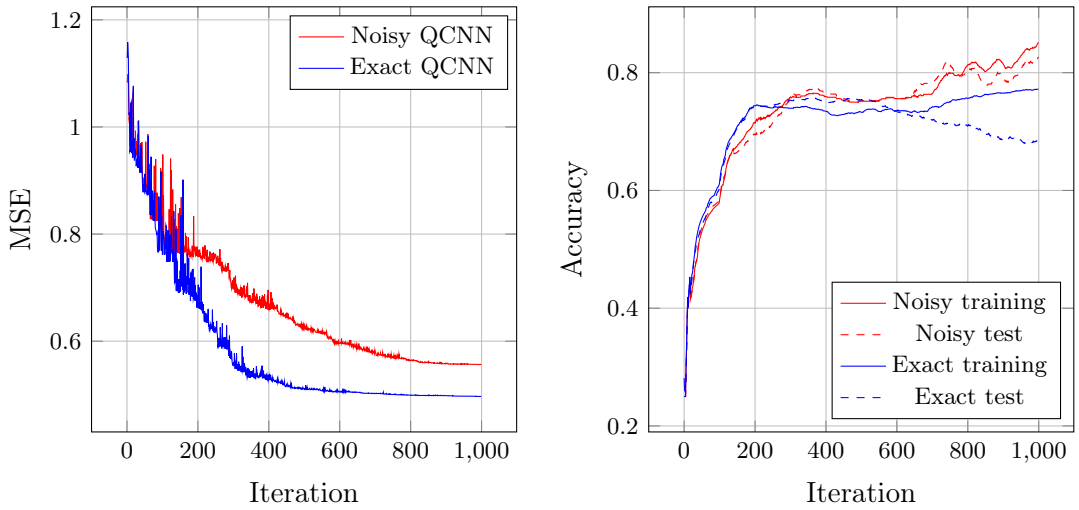


Figure 5.6: Training of the basic QCNN. The red curves are for the noisy model (modelled after IBM's Montreal hardware), while the blue curves are for the exact model. The dashed curves are for the test set. (a) loss (mean square error) during training. (b) accuracy on the training and test sets (running mean with a 100 iteration window).

where R_G are general rotations around all three axes and so have three parameters.

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. Combined, the convolution and pooling modules appear as in fig. 5.7.

Like in section 4.2.1, 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 5.1, but with X-rotations. In total, the network had 154 parameters.

5.3.2 Implementation

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, likely due to PennyLane defaulting to analytic evaluation of the gradients. This prompted the reimplementation of the former QCNN, which was here also trained with the same Adam optimiser.

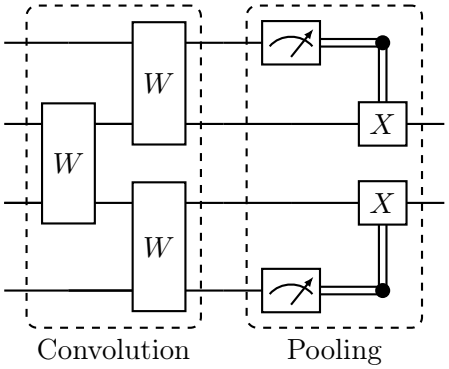


Figure 5.7: QCNN convolution and pooling layer structure with intermediate measurements. Some encoded data or already pooled data enter the convolution layer where parametrised gates W entangle the qubits. The pooling modules measure a qubit and use the result to control a unitary gate on a neighbour (here the X-gate).

5.3.3 Results

The results are shown in fig. 5.8. The training loss was lower than for the QCNN in section 4.2.1, despite the fewer iterations, showing expected advantages of using gradient based optimisation. Furthermore, the new model with intermediate measurements achieves a perfect accuracy on both the training and test sets in only 10 iterations. Looking at the loss curves, the model with intermediate measurements seems only to converge slightly faster than the model without intermediate measurements.

However, by extending the model without mid-circuit measurements to have a more comparable parameter count of 231, it overtakes the model with intermediate measurements in terms of training loss. This extension was done by switching out the convolutional gates from eq. (5.2) with the W-gates from eq. (5.4).

Evidently, gradient-based optimisation, at least Adam, makes the training much more stable and fast than the previous implementation. Despite large parameter counts, all three models perform just as well on the training data as on the test data, showing that the models are not overfitting. Is it clear that model like these, given enough parameters and an apt optimising algorithm, can be trained to perform well on unseen data. Still, to what extent this scales to larger data sets and more complex models is unclear. Furthermore, any significant advantage over classical models remains elusive.

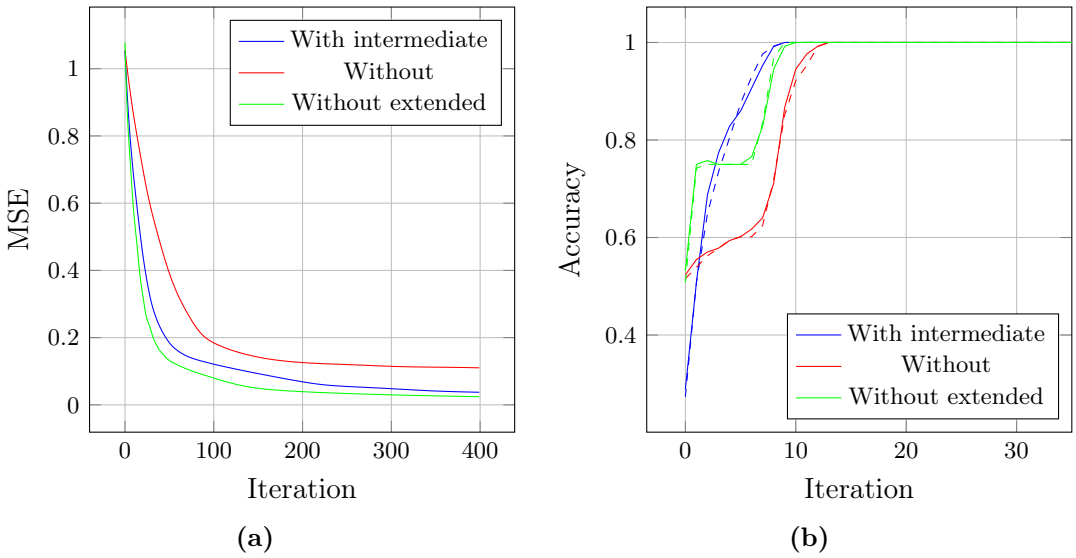


Figure 5.8: Training of QCNNs with and without intermediate measurements. With intermediate refers to the QCNN with intermediate measurements of [20]. Without refers to the QCNN without intermediate measurements from Section 4.2.1, while without extended is the same with a more expressive convolutional layer. (a) Mean square errors during training. (b) Accuracies on the training and test sets. Solid lines correspond to the training set and dashed lines to the test set. Note the reduced x-axis range; all models achieve perfect accuracy quickly.

Chapter 6

Conclusion

In this thesis, quantum machine learning has been discussed, coming from the perspective of classical machine learning and neural networks. Focus has been paid to various strategies of encoding classical data into quantum states and how to do so in a way that is not easily simulated classically while still being feasible for NISQ hardware. Particularly second-order rotations appear to be a good choice for encoding data, potentially with multiple repetitions, but it being hard to simulate classically is so far only conjecture.

Designs for VQAs or QNNs inspired by classical NNs were discussed, and several have been explored by simulations. The results of [29], in which a simple QNN was shown to converge significantly quicker than classical NN of similar parameter count, were reproduced. Whether the NN used was a fair comparison is debatable; a simpler logistic regression outperforms it.

Quantum convolutional networks were also studied, where one model included intermediate measurements. To what degree the mid-circuit measurements bettered the results is not clear. Overall, being limited to models simple enough to be simulated on classical computers, it is hard to draw any conclusions about the performance of quantum neural networks At least they seem able to learn.

Quantum neural networks can still be studied in much more detail, and there are more models to explore. How data encoding affects the performance of quantum neural networks is also an interesting question, and what type of encoding strategy goes best with which model could be studied. Classical networks are also suited for more than just supervised learning; they have successfully been used as generative models and for reinforcement learning, so it could be worth looking more into how quantum neural networks perform at these tasks. As quantum networks are necessarily probabilistic, they could have interesting properties that classical networks do not, which could prove useful.

Finally, it is important to remember that quantum neural networks are not the only way to combine quantum computing and machine learning. Interpreting quantum data encoding instead as kernel methods could be a fruitful avenue to explore.

References

[1] Richard P. Feynman. 'Simulating Physics with Computers'. In: *International Journal of Theoretical Physics* 21.6-7 (June 1982), pp. 467–488. ISSN: 0020-7748, 1572-9575. DOI: 10.1007/BF02650179.

[2] Peter W. Shor. 'Algorithms for Quantum Computation: Discrete Logarithms and Factoring'. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 35th Annual Symposium on Foundations of Computer Science. Santa Fe, NM, USA: IEEE Comput. Soc. Press, 1994, pp. 124–134. ISBN: 978-0-8186-6580-6. DOI: 10.1109/SFCS.1994.365700.

[3] Ronald A. Fisher. 'The Use of Multiple Measurements in Taxonomic Problems'. In: *Annals of Eugenics* 7.2 (1936), pp. 179–188. ISSN: 2050-1439. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[4] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 29th Jan. 2017. DOI: 10.48550/arXiv.1412.6980.

[5] Preetum Nakkiran et al. 'Deep Double Descent: Where Bigger Models and More Data Hurt'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (1st Dec. 2021), p. 124003. ISSN: 1742-5468. DOI: 10.1088/1742-5468/ac3a74.

[6] Izaak Neutelings. *Neural networks*. Licensed under a Creative Commons Attribution-ShareAlike 4.0 International License with © Copyright 2021 – TikZ.net. Sept. 2021. URL: https://tikz.net/neural_networks/ (visited on 24/10/2022).

[7] Amira Abbas et al. *Learn Quantum Computation Using Qiskit*. 2020. URL: https://qiskit.org/textbook/.

[8] Maria Schuld and Francesco Petruccione. *Machine Learning with Quantum Computers*. Quantum Science and Technology. Cham: Springer International Publishing, 2021. ISBN: 978-3-030-83097-7. DOI: 10.1007/978-3-030-83098-4.

[9] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 1st ed. Cambridge University Press, 5th June 2012. ISBN: 978-1-107-00217-3 978-0-511-97666-7. DOI: 10.1017/CBO9780511976667.

[10] Marco Cerezo et al. 'Variational Quantum Algorithms'. In: *Nature Reviews Physics* 3.9 (Sept. 2021), pp. 625–644. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00348-9.

[11] Smite Meister. *Bloch sphere*. This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license https://creativecommons.org/licenses/by-sa/3.0/deed.en. In this thesis it is rendered using Inkscape with LaTeX text. 30th Jan. 2009. URL: https://upload.wikimedia.org/wikipedia/commons/6/6b/Bloch_sphere.svg (visited on 13/10/2022).

[12] Giacomo Torlai et al. 'Precise Measurement of Quantum Observables with Neural-Network Estimators'. In: *Physical Review Research* 2.2 (17th June 2020), p. 022060. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.2.022060.

[13] Craig Gidney and Martin Ekerå. 'How to Factor 2048 Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits'. In: *Quantum* 5 (15th Apr. 2021), p. 433. ISSN: 2521-327X. DOI: 10.22331/q-2021-04-15-433.

[14] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-96423-2. DOI: 10.1007/978-3-319-96424-9.

[15] Timo Felser et al. 'Quantum-Inspired Machine Learning on High-Energy Physics Data'. In: *npj Quantum Information* 7.1 (Dec. 2021), p. 111. ISSN: 2056-6387. DOI: 10.1038/s41534-021-00443-w.

[16] Vojtech Havlicek et al. 'Supervised Learning with Quantum Enhanced Feature Spaces'. In: *Nature* 567.7747 (Mar. 2019), pp. 209–212. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-0980-2.

[17] Maria Schuld, Ryan Sweke and Johannes Jakob Meyer. 'The Effect of Data Encoding on the Expressive Power of Variational Quantum Machine Learning Models'. In: *Physical Review A* 103.3 (24th Mar. 2021), p. 032430. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.103.032430.

[18] Iris Cong, Soonwon Choi and Mikhail D. Lukin. 'Quantum Convolutional Neural Networks'. In: *Nature Physics* 15.12 (Dec. 2019), pp. 1273–1278. ISSN: 1745-2473, 1745-2481. DOI: 10.1038/s41567-019-0648-8.

[19] Seunghyeok Oh, Jaeho Choi and Joongheon Kim. 'A Tutorial on Quantum Convolutional Neural Networks (QCNN)'. In: 2020 International Conference on Information and Communication Technology Convergence (ICTC). Jeju Island, Korea (South): IEEE, 21st Oct. 2020, pp. 236–239. ISBN: 978-1-72816-758-9. DOI: 10.1109/ICTC49870.2020.9289439.

[20] Arthur Pesah et al. 'Absence of Barren Plateaus in Quantum Convolutional Neural Networks'. In: *Physical Review X* 11.4 (15th Oct. 2021), p. 041011. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.11.041011.

[21] Xun Gao, Z.-Y. Zhang and Luming Duan. 'A Quantum Machine Learning Algorithm Based on Generative Models'. In: *Science Advances* 4.12 (7th Dec. 2018), eaat9004. ISSN: 2375-2548. DOI: 10.1126/sciadv.aat9004.

[22] He-Liang Huang et al. 'Experimental Quantum Generative Adversarial Networks for Image Generation'. In: *Physical Review Applied* 16.2 (27th Aug. 2021), p. 024051. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.16.024051.

[23] Nathan Killoran et al. 'Continuous-Variable Quantum Neural Networks'. In: *Physical Review Research* 1.3 (31st Oct. 2019), p. 033063. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.1.033063.

[24] Maxwell Henderson et al. 'Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits'. In: *Quantum Machine Intelligence* 2.1 (June 2020), p. 2. ISSN: 2524-4906, 2524-4914. DOI: 10.1007/s42484-020-00012-y.

[25] Yi Zeng et al. 'A Multi-Classification Hybrid Quantum Neural Network Using an All-Qubit Multi-Observable Measurement Strategy'. In: *Entropy* 24.3 (11th Mar. 2022), p. 394. ISSN: 1099-4300. DOI: 10.3390/e24030394.

[26] Matthew Treinish et al. *Qiskit/Qiskit: Qiskit 0.39.2*. Version 0.39.2. Zenodo, 4th Nov. 2022. DOI: 10.5281/ZENODO.2573505.

[27] Ville Bergholm et al. *PennyLane: Automatic Differentiation of Hybrid Quantum-Classical Computations*. 29th July 2022. DOI: 10.48550/arXiv.1811.04968.

[28] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html (visited on 13/11/2022).

[29] Amira Abbas et al. 'The Power of Quantum Neural Networks'. In: *Nature Computational Science* 1.6 (June 2021), pp. 403–409. ISSN: 2662-8457. DOI: 10.1038/s43588-021-00084-1.

[30] IBM. *The Quantum Convolution Neural Network*. 2022. URL: https://qiskit.org/documentation/machine-learning/tutorials/11_quantum_convolutional_neural_networks.html.