Dummy Titlepage

Boye Gravningen Sjo

Autumn 2022

TMA4500

Industrial Mathematics Specialisation Project

Department of Mathematical Sciences

Faculty of Information Technology and Electrical Engineering

Norwegian University of Science and Technology

Contents

Chapter 1

Introduction

1.1 Relevant work

1.1.1 Cerezo et al. (2021)

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

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

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

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

Still, there are many difficulties when applying VQAs. Barren plateaus are a common occurrence, making the optimisation futile. The choosing of the ansatz determines the performance and feasibility of the algorithms, and there are many strategies and options. Some rely on ex-

ploiting the specific quantum hardware's properties, while some use the specifics of the problem at hand. Finally, the inherent noise and errors on near-term hardware will still be a problem and limit circuit depths.

1.1.2 Moll et al. (2018)

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

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

1.1.3 Torlai et al. (2020)

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

1.1.4 Schuld et al. (2019)

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

1.1.5 Pesah et al. (2021)

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

1.1.6 Farhi et al. (2018)

Quantum neural networks (QNNs) are simply an abstraction of parametrised quantum circuits with some sort of data encoding. As with classical neural networks or supervised learning in general, the parameters are optimised by minimising a cost function. For QNNs, the output can be a single designated read-out qubit, where the states are interpreted as classes in a binary classification problem. This was shown to indeed be feasible for handwritten digit recognition, using downsampled MNIST data. With the qubit count on current quantum devices and the

amount that can be easily simulated, the dimensionality of the data can not be much more than a dozen.

1.1.7 Abbas et al. (2021)

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

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

Chapter 2

Machine learning

2.1 Supervised learning

2.2 Optimisation

2.3 Bias-variance trade-off

2.4 Neural networks

2.4.1 Convolutional neural networks

2.4.2 Backpropagation

2.4.3 Universal approximation theorem

Chapter 3

Quantum computing

Chapter 4

Variational quantum algorithms

4.1 Ansätze

4.2 Optimisation

4.2.1 Parameter-shift

4.2.2 Barren plateaus

4.3 Applications

Chapter 5

Quantum machine learning

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

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

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

Classical data being processed on classical computers is classical machine learning. Though not explicitly linked to quantum computing, there are some ways in which quantum computing influence classical machine learning. For example, the quantum inspired application of tensor networks in [**felser2020**].

Using classical machine learning for quantum computing is used to improve quantum computers general performance. For example, with machine learning algorithms, the variance of the measurements can be reduced, as shown in [**torlai2020**]. Alternatively, advanced machine learning models like neural networks can be employed to describe quantum states more efficiently.

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

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

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

5.1 Data encoding

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

5.1.1 Basis encoding

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

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

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

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

5.1.2 Amplitude encoding

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

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

where x_i is the 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

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

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

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

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

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

5.1.3 Angle encoding

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

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

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

or

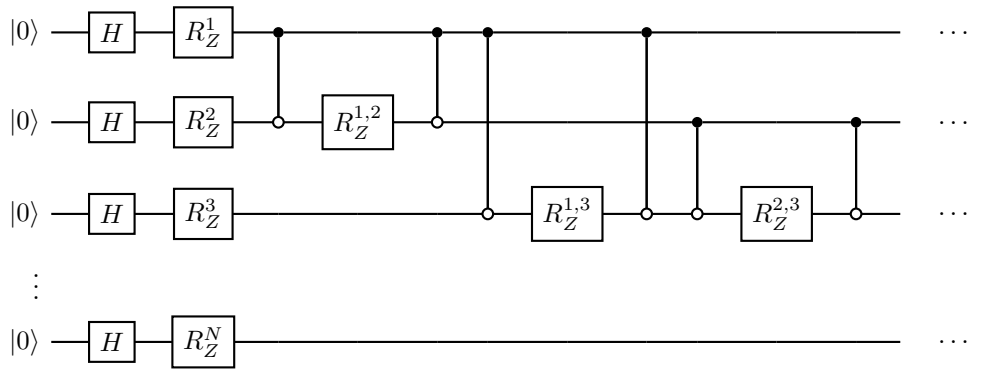
$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_Z(x) H |0\rangle , \tag{5.7}$$

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

5.1.4 Second order angle encoding

Havlicek et al. propose in [**havlicek2018**] a second-order angle encoding, which they conjecture to be hard to simulate classically. First, angles are encoded as above, but then the qubits are

entangled and rotated further based on second order terms. In circuit notation, such an encoding with Z-rotations reads



(5.8)

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

5.1.5 Repeats

The expressive power of models heavily rely on the encoding strategy. For instance, a single qubit rotation only allows the model to learn sine functions, where the frequency is determined by the scaling of the data. Generally, quantum models will learn periodic functions, and thus Fourier analysis is a useful tool. Schuld et al. study in [**schuld2021**] the implications of this, and they show that simply repeating basic encoding blocks allows for learning of more frequencies and thus more complex functions. Asymptotically, such repeats lets a quantum model learn arbitrary functions.

5.2 Quantum neural networks

Quantum neural networks (QNNs) are simply an abstraction of parametrised quantum circuits with some sort of data encoding. As classical artificial neural networks have made classical machine learning into a powerful tool, QNNs are envisioned as a quantum counterpart, inheriting 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. **??** shows the general circuit layout. In the first step or layer, data is encoded into the qubits, typically using a method discussed in **??**. Next, the data is passed through a sequence of parametrised quantum gates which often can be interpreted as layers. Lastly, an output is produced, which is typically a measurement of some observable. Usually, the observable is a combination of Pauli operators on every qubit. Thence, a cost function can be evaluated, and the variational circuit can be optimised.

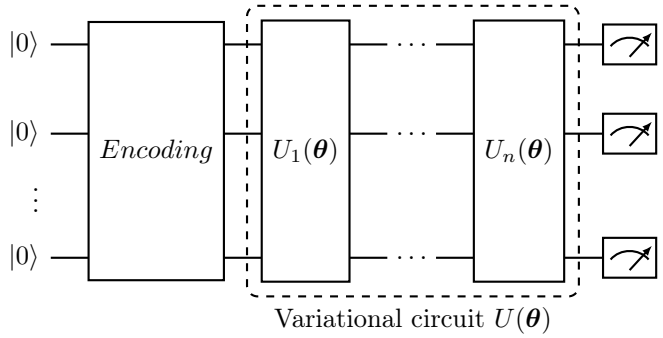


Figure 5.1: The structure of a quantum neural network. First, the data $\boldsymbol{x}$ is encoded into a state $|\psi(\boldsymbol{x})\rangle$ using some encoding strategy. Then, the state is transformed by a parametrised quantum circuit $U(\theta)$. This variational circuit can often be decomposed into a sequence of gates $U_1, \ldots, U_n$, making the QNN structure more akin to the layered classical neural networks. Finally, measurements are made and used to calculate the model output.

5.2.1 Architectures and their applications

Quantum convolutional neural networks

Originally introduced by Cong et al. in [**cong2019**], quantum convolutional neural networks (QCNNs) take inspiration from classical convolutional neural networks in that a sequence of convolutional and pooling layers are used to extract features and reduce the dimension before the output is made. In the quantum convolutional layers, neighbouring qubits are entangled with some parametrised gates. After that, pooling layers halve the active qubit count by yet a parametrised gate. When pooling, the qubits to be discarded could be measured and used to determine the operations on the still active qubits. Otherwise, the unused qubits are simply ignored. After several iterations, a gate can be employed on the remaining qubits, analogous to a fully connected layer in classical CNNs, before the final measurement and output.

Because of the constant reduction of layer sizes in (Q)CNNs, the total parameter count is only of order logarithm of the network depth, making them easier to train than dense networks of similar input size. In [**cong2019**], 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 [**oh2020**]. They have shown desirable properties with regard to avoiding barren plateaus [**pesah2021**].

Quantum generative adversarial networks

Quantum generative models have been shown to potentially have an exponential advantage over their peers [**gao2018**]. Due to the inherent probabilistic nature of quantum machines, it should not be surprising that they more naturally learn difficult distributions than classical computers do. For instance, real quantum hardware has been used to generate (admittedly low-resolution) images of handwritten images [**huang2021**].

Hybrid quantum-classical neural networks

Another option is to include a quantum layer or node is some larger pipeline or even non-linear graph structure. Because parametrised quantum circuits are differentiable, they can easily be handled using the chain rule when back propagating a hybrid model.

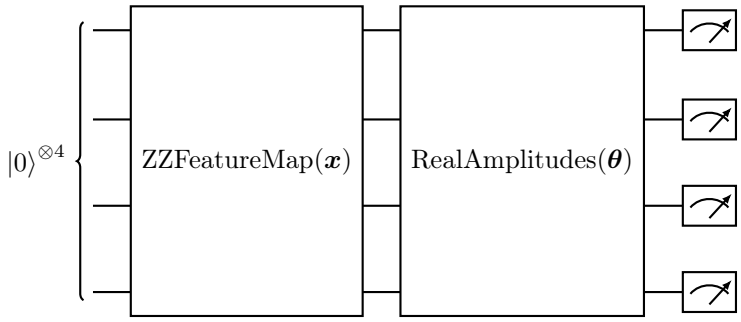


Figure 5.2: Structure of the QNN used for classification of the iris dataset. The first block maps the input data $\boldsymbol{x}$ to the quantum state $|\psi(\boldsymbol{x})\rangle$. The second block is the variational circuit, parametrised by $\boldsymbol{\theta}$, a vector with eight components. Finally,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

5.3 Benchmarks

5.3.1 QNNs vs NNs

In order to compare the performance of the QNN and the NN, architectures suited for binary classification with exactly 8 parameters are used. The QNN structure is shown in **??**. The data used is Fisher's iris dataset, perhaps the most used dataset for studying classification in statistics, containing samples of three different species of iris flowers. For each species, there are 50 samples, each with four features: sepal length, sepal width, petal length, and petal width. Like in [**abbas2021**], only the two first species are considered, which happen to be linearly separable in the feature space.

The four dimensional input data $\boldsymbol{x}$ is mapped to some quantum state $|\psi(\boldsymbol{x})\rangle$ using the ZZFeatureMap from Qiskit with two repetitions, which is a second-order Pauli-Z evolution circuit. This feature map is essentially a mix of rotation around the Z axis parametrised by the input features or functions thereof and CNOT and Hadamard gates. Thus, the features get highly entangled and embedded in some higher-dimensional space. The quantum state is then evolved by the RealAmplitudes ansatz. All four qubits a rotated in the Y direction by some parameter before CNOT-gates are applied pairwise, for a final parametrised rotation in the Y direction, for a total of 8 parameters. For details on the circuit components, see Qiskit's documentation [**qiskit**] or the original paper [**abbas2021**]. The parity of the output 4-bit string is interpreted as the prediction, and with 100 shots used, the model gave a probability for each of the two iris classes. Both exact simulations and noisy simulations were performed, with the latter using noise modelled after the 27-qubit IBM Montreal architecture, the actual hardware used in the original paper.

The classical neural network was a standard dense feed-forward model, with a 4-1-1-1-2 layered structure without biases, giving a total of 8 parameters. The activation functions were leaky ReLUs,

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

and the output layer used a softmax activation function.

Both models were implemented using PyTorch, with the QNN being implemented using Qiskit's PyTorch interface. Consequently, the models could be trained in the exact same manner,

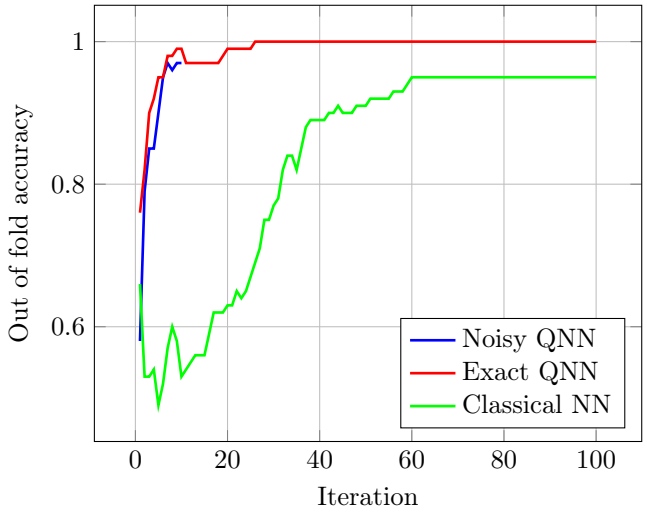


Figure 5.3: Mean accuracy during training for the iris dataset using 10-fold cross validation. All models have 8 parameters and are trained using the Adam optimiser with a learning rate of 0.1, using cross-entropy as the loss function. Due to the computational cost, the noisy (simulated IBM Montreal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

using the Adam optimiser with a learning rate of 0,1. The models were trained for 100 epochs, with the loss function being cross-entropy.

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

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 NN, on the other hand, requires more iterations to converge and does not always do so. In some cases, the model did not converge, only predicting one class, which is why the out-of-fold accuracy was not 100% for all folds.

5.3.2 Quantum convolutional neural networks

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

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

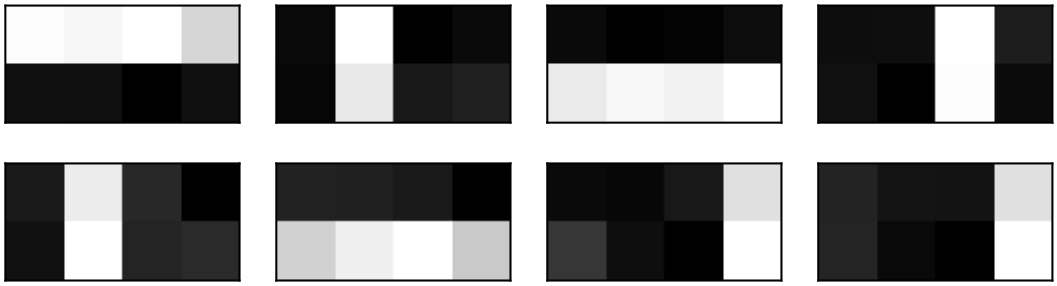
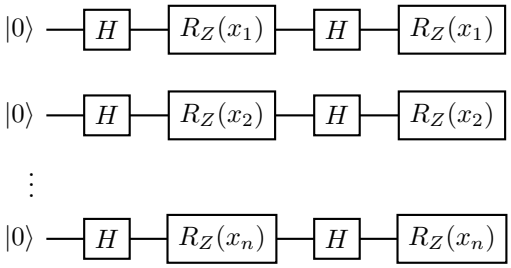
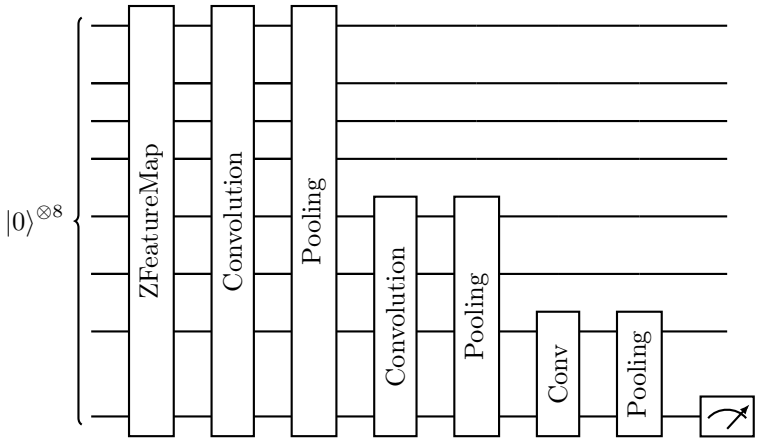


Figure 5.4: Data for the QCNN. With a total of 64 training images and 16 for testing, they form balanced dataset of 2×4 pixels, with either a vertical or horizontal line encoded as 1 and -1. The images are generated with some Gaussian noise.



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



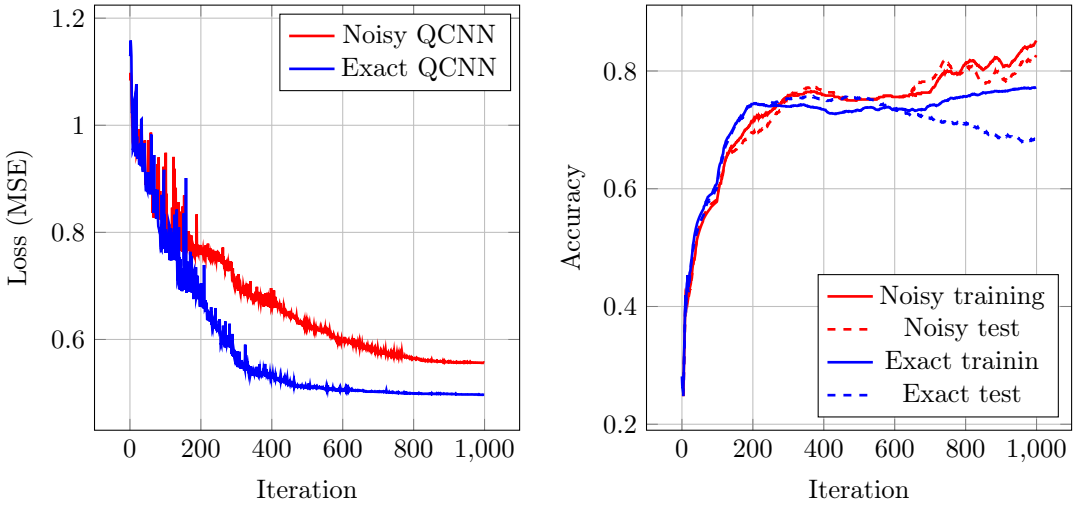


Figure 5.5: Training of the QCNN. The red curves are for the noisy model (modelled after IBM's Montreal hardware), while the blue curves are for the exact model. The dashed curves are for the test set. (a) loss (mean square error) during training. (b) accuracy on the training and test sets (running mean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

As in Qiskit's guide, training was done using thee COBYLA optimiser[1] which does not use gradients. The accuracies and loss (mean square error) during training is shown in **??**. Like in **??**, noise is modelled after the IBM Montreal hardware. The networks were trained for 1000 epochs, and while neither reached full accuracy, the losses shrunk, indicating at least increased certainty in the predictions. Interestingly, the noisy simulation appears to yield better predictions, despite suffering from higher losses during training. It seems that the noiseless QCNN is overfitting to the training data, while the noisy QCNN generalises better.

[1]Constrained Optimisation BY Linear Approximation.