ZURICH UNIVERSITY OF APPLIED SCIENCES

RESEARCH PROJECT

Implementing Neural Networks In A Quantum Computer

Authors:
Ricardo Daniel MONTEIRO SIMÕES, Patrick HUBER

Supervisors:
Prof. Dr. Kurt STOCKINGER
Prof. Dr. Rudolf Marcel FÜCHSLIN

December 27, 2021

Zurich University
of Applied Sciences

School of
Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirm that this work is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the project work have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Zürich, 27.12.2021

Signature:

.....

.....

.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all project works submitted.

Zurich University
of Applied Sciences

ZURICH UNIVERSITY OF APPLIED SCIENCES

Abstract

Institute of Applied Information Technology

Research Project

Implementing Neural Networks In A Quantum Computer

by Ricardo Daniel MONTEIRO SIMÕES, Patrick HUBER

One important part of neural networks are computational neurons and their highly adapted derivatives. This allows neural networks to offer solutions to otherwise hard to solve problems. Theoretical expectations and current advancements in the field of quantum computation allow the development and evaluation of self-contained quantum neuron designs. Through various experiments, data is collected and assessed. The results indicate the existence of general quantum classifiers that offer results comparable to the classical variant. At the same time, the variance when it comes to achievable training accuracy dampens any possible impact. Another issue arises, as running quantum classifiers on real hardware leads to vastly inferior results when compared to simulator runs, due to induced noise. Nevertheless, quantum classifiers appear to be a promising, future alternative for classical classifiers.

Contents

Chapter 1

Introduction

For a long time, the ever occurring question 'Can quantum x beat the classical variant?' was always met with the answer 'theoretically, yes'. During this time, all possible gains remained in the theoretical realm[1], but recent advances in the field of quantum computing have brought out solutions and proposals[2–4] that can be programmed and used *today*. At the same time, quantum computing is getting more and more available to non-researchers as IBM, Microsoft, Google and co. fight for dominance in this new segment of computation.

The training of neural networks to solve the hardest challenges demands an ever-increasing amount of computational power[5]. The goal of this paper is to evaluate current quantum offerings to solve classification problems, as well as analyse their performance. In a first step, a broad overview of quantum operations is created, as well as an assessment about the application of features and weights onto quantum gates. Subsequently, two possible designs are evaluated, the first being a solution that replicates the basic arithmetic operations computational neurons use, and the second one which uses quantum specific superpositions and entanglement. Due to the necessary amount of qubits for the arithmetic solution, as well as the absence of quantum specific operations which make quantum supremacy possible, only the second solution further developed and evaluated.

Using a variety of different quantum circuits, as well as three different datasets, training is done on the simulator to determine the viability of these designs as well as to compare them to a basic Multi-layer Perceptron classifier. The results show that the right combination of quantum gates leads to multiple classifiers that can be optimized for different problem spaces. One issue that accompanies these design is the discrepancy when it comes to achieved accuracy. Under reproducible circumstances, no two training results are equal. This issue comes directly from the vast expressibility such a circuit offers - by making the problem space much wider, it gets harder to traverse and therefore, find a viable solution.

When running the trained circuit on real hardware for a given dataset, another issue arises as the noise affecting the real devices disrupts the operations and measurements, which in turn leads to vastly inferior accuracy. Nevertheless, our results suggest that classification with quantum circuits is not only possible, but also a viable alternative.

Chapter 2

Quantum Operations

2.1 Basic operations

Before creating a quantum circuit, a basic understanding of quantum operations and gates is needed. Comparable to the classical bit, qubits are the more powerful quantum equivalent. Whilst classical bits can only assume a state of 1 and 0, a qubit can be 1, 0 or anything in between. The state of a single, non-entangled qubit can be visualized with the Bloch sphere[6]. Figure 2.1 shows a simple circuit containing one qubit, annotated with q_0, and figure 2.2 shows the Bloch sphere for it.

q_0 : ——

FIGURE 2.1: Quantum circuit with a single qubit and no quantum gates

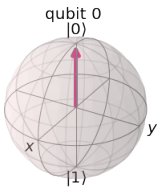


FIGURE 2.2: Bloch sphere with the state $|0\rangle$

The visible arrow shows the current state, which lies directly on the z axis. The state of the qubit is described as a vector in equation 2.1.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.1}$$

The classical NOT gate, that inverts the state of a chosen bit, also exists as a quantum gate. The matrix definition of the Pauli X[7] gate is represented in figure 2.3. When applying it to a qubit, a multiplication of the state vector of q_0 and the matrix X results in the inverted state $|1\rangle$, as demonstrated in equation 2.2. The circuit for this example is shown in figure 2.4. Figure 2.5 shows that our state now points in the opposite direction when compared to figure 2.2.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

FIGURE 2.3: Matrix that defines the Pauli X gate

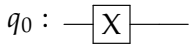


FIGURE 2.4: Quantum circuit with a single qubit and X gate

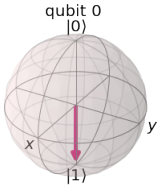


FIGURE 2.5: Bloch sphere with the state $|1\rangle$

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{2.2}$$

A superposition is any state that is neither $|0\rangle$ and $|1\rangle$, but a combination of both. Such a state is defined as $\alpha|0\rangle + \beta|1\rangle$, where the quadratic value of α and β[1] corresponds to the probability of the given state, and therefore $\alpha^2 + \beta^2 = 1$. When building a quantum circuit, the Hadamard, or H[8], gate can be used to put a qubit in an *equal* superposition, that is equal probability for $|0\rangle$ and $|1\rangle$. Figure 2.6 shows the matrix definition of the Hadamard gate, figure 2.7 a simple single qubit circuit with the Hadamard gate, and equation 2.3 the calculation of said circuit. The Bloch sphere in figure 2.8 shows the attained superposition, which is orthogonal to the Z axis.

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

FIGURE 2.6: Matrix that defines the Hadamard gate

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{2.3}$$

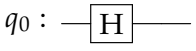


FIGURE 2.7: Quantum circuit with a single qubit and Hadamard gate

[1] Gross simplification, see chapter 2.2.1

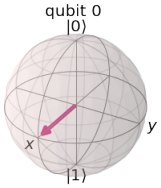


FIGURE 2.8: Bloch sphere with the equal superposition state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

2.2 Rotations

As seen in chapter 2.1, quantum gates apply rotations to the state inside the Bloch sphere. There exist parametrizable, rotational gates that rotate the state around a given axis, by the supplied value. One of those gates is the RY(θ)[9] gate. The RY gate is constructed from the Pauli Y gate as shown in equation 2.4. Note that the resulting power series is equal to the ones defined for cos and sin in equation 2.5[10]. As measurements in a quantum circuit are, to put it simply, collapsing the state vector onto the Z axis[11], rotation around the Y axis have a direct impact on the probabilities of the measurement.

$$
\begin{aligned}
\mathrm{RY}(\theta) &= e^{-i\frac{\theta}{2}\mathrm{Y}} = e^{\frac{-i\theta}{2}\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}} \\
\mathrm{Y}' &= -i\frac{\theta}{2}\mathrm{Y} = \begin{pmatrix}0 & \frac{-\theta}{2}\\ \frac{\theta}{2} & 0\end{pmatrix} \\
e^{\mathrm{Y}'} &= I^{2\times 2} + \mathrm{Y}' + \frac{\mathrm{Y}'^2}{2!} + \frac{\mathrm{Y}'^3}{3!} + \frac{\mathrm{Y}'^4}{4!} + \cdots \\
\mathrm{RY}(\theta) &= \begin{pmatrix}1 - \frac{\theta^2}{8} + \frac{\theta^4}{384} + \cdots & -\frac{\theta}{2} + \frac{\theta^3}{48} - \frac{\theta^5}{3840} + \cdots\\ \frac{\theta}{2} - \frac{\theta^3}{48} + \frac{\theta^5}{3840} + \cdots & 1 - \frac{\theta^2}{8} + \frac{\theta^4}{384} + \cdots\end{pmatrix} \\
&= \begin{pmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}
\end{aligned}
\tag{2.4}
$$

$$
\begin{aligned}
\cos(x) &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \\
\sin(x) &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots
\end{aligned}
\tag{2.5}
$$

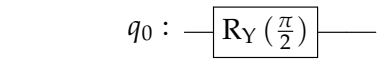


FIGURE 2.9: Quantum circuit with a single qubit and RY gate parameterized to $\frac{\pi}{2}$

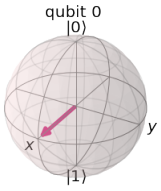


FIGURE 2.10: Bloch sphere with the equal superposition state $\cos\left(\frac{\pi}{4}\right)|0\rangle + \sin\left(\frac{\pi}{4}\right)|1\rangle$

Quantum circuits with more than one qubit can use the entanglement of multiple qubits for advanced unitary operations. When entangling two qubits together, one has to start with calculating the tensor product of their state vectors. For circuit 2.12, the calculations are outlined in equation 2.6.

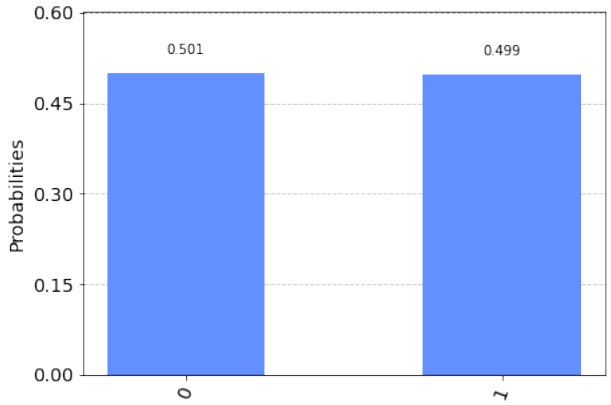


FIGURE 2.11: Histogram of the resulting probabilities for $|0\rangle$ and $|1\rangle$ on circuit 2.9

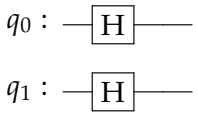


FIGURE 2.12: Quantum circuit with two qubits and two Hadamard gates

$$\begin{aligned} \mathrm{H}|0\rangle &= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ \vec{s} &= \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix} \otimes \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix} = \begin{pmatrix}\frac{1}{2}\\ \frac{1}{2}\\ \frac{1}{2}\\ \frac{1}{2}\end{pmatrix} \end{aligned} \tag{2.6}$$

Quantum gates can be of any size and use as many qubits as they're designed for. Gates such as CY[12], CX[13] and CZ[14] cover two qubits, of which one is a *control* qubit. Depending on its state, the rotation is then applied to the second qubit. Figures 2.13, 2.14 and 2.15 show the circuit design and the corresponding equations 2.7, 2.8 and 2.9 show the behaviour of the state vector when entanglement is applied.

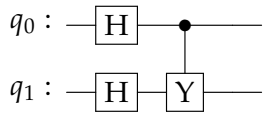


FIGURE 2.13: Quantum circuit with two qubits, two Hadamard gates and entangled with a CY gate

$$\begin{aligned} \vec{s}_{\mathrm{CY}} &= \mathrm{CY} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & 1 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \\ \vec{s}_{\mathrm{CY}} &= \begin{pmatrix} \frac{1}{2} \\ \frac{-i}{2} \\ \frac{1}{2} \\ \frac{i}{2} \end{pmatrix} \end{aligned} \tag{2.7}$$

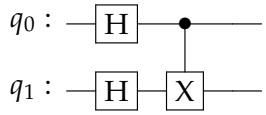


FIGURE 2.14: Quantum circuit with two qubits, two Hadamard gates and entangled with a CX gate

$$\begin{aligned} \vec{s}_{\mathrm{CX}} &= \mathrm{CX} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \\ \vec{s}_{\mathrm{CX}} &= \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \end{aligned} \tag{2.8}$$

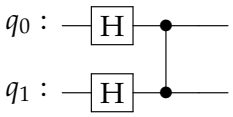


FIGURE 2.15: Quantum circuit with two qubits, two Hadamard gates and entangled with a CZ gate

$$\vec{s}_{\mathrm{CZ}} = \mathrm{CZ} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$
$$\vec{s}_{\mathrm{CZ}} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{-1}{2} \end{pmatrix} \tag{2.9}$$

2.2.1 Measurement

The statement in chapter 2.1 of $\alpha^2 + \beta^2 = 1$ is a gross simplification of the measurement process. When measuring a quantum circuit, it always returns one of all the possible values. This is why, when determining it, multiples *shots* are executed until the achieved distribution of values converges onto their probabilities. This also applies when calculating the probabilities. Whilst it is not necessary to do multiple calculations for each state, each state does need to be calculated separately. To measure the probability of 01 in a two qubit circuit, we have to use a measurement matrix and surround it with our state $|\psi\rangle$[2] and $\langle\psi|$, which is the Hermitian transposition[15] of $|\psi\rangle$. Equation 2.10 demonstrates the steps needed to measure the probability of state 01 and equation 2.11 for state 11. Further calculations in this chapter will use M to resemble all four, single calculations and show them as a single vector.

$$\mathrm{P}_{Example} = \langle\psi| M_2 |\psi\rangle = \begin{pmatrix} \frac{-i}{\sqrt{2}} & \frac{i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{i}{\sqrt{2}} \\ \frac{-i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix} = \frac{1}{2} \tag{2.10}$$

$$\mathrm{P}_{Example} = \langle\psi| M_4 |\psi\rangle = \begin{pmatrix} \frac{-i}{\sqrt{2}} & \frac{i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{i}{\sqrt{2}} \\ \frac{-i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix} = \frac{1}{2} \tag{2.11}$$

As entanglement connects both qubits, any gate that is applied to one of those is directly applied to the other one. The circuit 2.16 shows the RY gate, parameterized to $\frac{\pi}{4}$, after the entanglement. To calculate this quantum circuit, an increase in dimensionality of the target gate is done. Equation 2.12 demonstrates how the identity matrix is used on the RY gate.

$$\mathrm{RY}(\theta)^{2\times2} = I^{2\times2} \otimes \mathrm{RY}(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} & 0 & 0 \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} & 0 & 0 \\ 0 & 0 & \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ 0 & 0 & \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{2.12}$$

[2] $|\psi\rangle$ stands for any arbitrary qubit state

Equations 2.13, 2.14 and 2.15 show the calculations of the resulting probabilities for each type of entanglement.

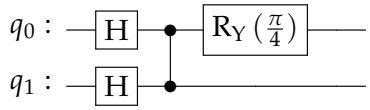


FIGURE 2.16: Quantum circuit with two qubits, two Hadamard gates, entangled with a CZ gate and a follow-up RY gate

$$\mathrm{RY}(\theta)^{2\times 2}\vec{s}_{\mathrm{CZ}} = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} & 0 & 0 \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} & 0 & 0 \\ 0 & 0 & \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ 0 & 0 & \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{-1}{2} \end{pmatrix}$$

$$|\psi_{CZ}\rangle = \begin{pmatrix} 0.2706 \\ 0.65328 \\ 0.65328 \\ -0.2706 \end{pmatrix} \tag{2.13}$$

$$\mathrm{P_{CZ}} = \langle\psi_{CZ}| M |\psi_{CZ}\rangle = \begin{pmatrix} 0.073223 \\ 0.42678 \\ 0.42678 \\ 0.073223 \end{pmatrix}$$

$$\mathrm{P_{CX}} = \langle\psi_{CX}| M |\psi_{CX}\rangle \rightarrow \begin{pmatrix} 0.073223 \\ 0.42678 \\ 0.073223 \\ 0.42678 \end{pmatrix} \tag{2.14}$$

$$\mathrm{P_{CY}} = \langle\psi_{CY}| M |\psi_{CY}\rangle \rightarrow \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} \tag{2.15}$$

To assess the differences between the application of the rotation gate RY before the entanglement, circuit 2.17 is calculated. The initial step of applying $\mathrm{RY}(\theta)$ to q_0, is demonstrated in equation 2.16.

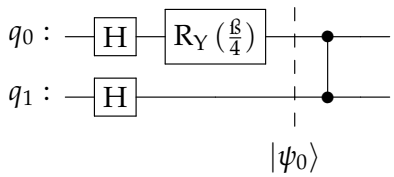


FIGURE 2.17: Quantum circuit with two qubits, two Hadamard gates, single RY gate and entangled with a CZ gate

$$
\begin{aligned}
|\psi_0\rangle = \mathrm{RY}(\frac{\pi}{4})\mathrm{H}\,|0\rangle = \begin{pmatrix} \cos\frac{\pi}{8} & -\sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & \cos\frac{\pi}{8} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \\
|\psi_0\rangle = \begin{pmatrix} 0.38268 \\ 0.92388 \end{pmatrix}
\end{aligned}
\tag{2.16}
$$

$$
\begin{aligned}
|\psi_{CZ}\rangle = \mathrm{CZ}\,|\psi_0\rangle \otimes \mathrm{H}\,|0\rangle = \begin{pmatrix} 0.2706 \\ 0.2706 \\ 0.65328 \\ 0.65328 \end{pmatrix} \\
\mathrm{P_{CZ}} = \langle\psi_{CZ}|\, M\, |\psi_{CZ}\rangle \rightarrow \begin{pmatrix} 0.073223 \\ 0.073223 \\ 0.42678 \\ 0.42678 \end{pmatrix}
\end{aligned}
\tag{2.17}
$$

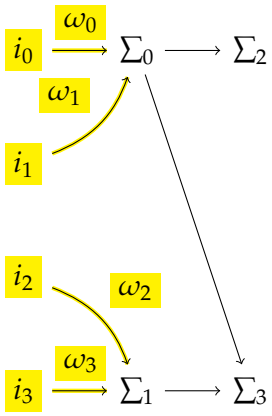
$$
\begin{aligned}
|\psi_{CX}\rangle = \mathrm{CX}\,|\psi_0\rangle \otimes \mathrm{H}\,|0\rangle = \begin{pmatrix} 0.2706 \\ 0.2706 \\ 0.65328 \\ 0.65328 \end{pmatrix} \\
\mathrm{P_{CX}} = \langle\psi_{CX}|\, M\, |\psi_{CX}\rangle \rightarrow \begin{pmatrix} 0.073223 \\ 0.073223 \\ 0.42678 \\ 0.42678 \end{pmatrix}
\end{aligned}
\tag{2.18}
$$

$$
\begin{aligned}
|\psi_{CY}\rangle = \mathrm{CY}\,|\psi_0\rangle \otimes \mathrm{H}\,|0\rangle = \begin{pmatrix} 0.2706 \\ 0 - 0.65328i \\ 0.65328 \\ 0 + 0.2706i \end{pmatrix} \\
\mathrm{P_{CY}} = \langle\psi_{CY}|\, M\, |\psi_{CY}\rangle \rightarrow \begin{pmatrix} 0.0732 \\ 0.4268 \\ 0.4268 \\ 0.0732 \end{pmatrix}
\end{aligned}
\tag{2.19}
$$

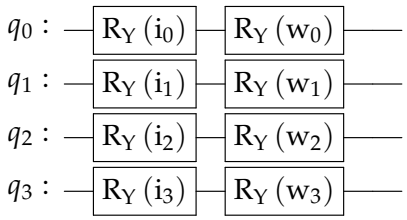
Whereas equation 2.15 shows that when applying RY after the entanglement, it returns the states to their previous probabilities, equation 2.19 losses none of its operations. This indicates that applying the weights before the entanglement can result in a wider range of results, which *could* lead to a more powerful classifier.

2.3 Visual Comparison Between A Neural Network And Quantum Circuit

To design the quantum equivalent to a neural net, a visual comparison is created and traced. Figure 2.18 contains an exemplary neural network, where each $\sum_i$ represents a single node. The highlighted paths are converted into their quantum counterparts. In an initial step, the input layer with features and weights is created. Equation 2.20 shows the mathematical operation that is executed on a single qubit.



(A) Example neural network, with the weighted input connections being highlighted



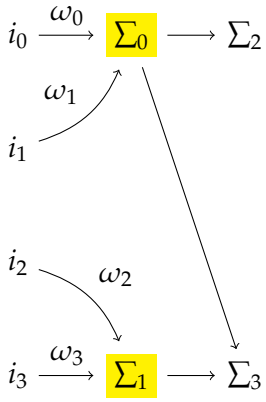
(B) Quantum circuit with four qubits and a pair of two RY gates per qubit, one for the input feature i_x and one for the weight w_x

FIGURE 2.18: Comparison of neural network and input and weight gates of the quantum circuit

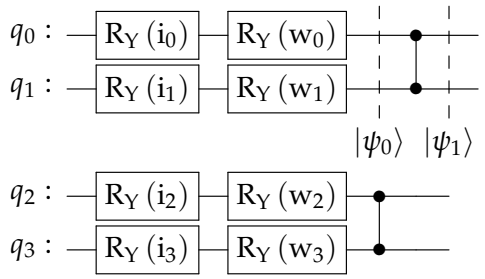
$$\begin{aligned} \mathrm{RY}(\omega_0)\mathrm{RY}(i_0)\,|0\rangle &= \begin{pmatrix} \cos\frac{\omega_0}{2} & -\sin\frac{\omega_0}{2} \\ \sin\frac{\omega_0}{2} & \cos\frac{\omega_0}{2} \end{pmatrix} \begin{pmatrix} \cos\frac{i_0}{2} & -\sin\frac{i_0}{2} \\ \sin\frac{i_0}{2} & \cos\frac{i_0}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \rightarrow q_0 &= \begin{pmatrix} \cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2} \\ \cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2} \end{pmatrix} \end{aligned} \tag{2.20}$$

It is important to note that whilst graphically they appear to do the same, the behaviour is quite different. Where the computational neuron performs classical multiplications, the quantum counterpart operates with rotations, as demonstrated in chapter 2.2. The *probability* of a given state is, depending on the rotation, reinforced or weakened.

The next step is to rebuild the two partially connected nodes $\sum_0$ and $\sum_1$, as highlighted in figure 2.19. Note that we first have to increase the dimension of the state vector by calculating the tensor product of q_0 and q_1, as seen in equation 2.21. The entanglement itself is calculated in equation 2.22 using the CZ gate.



(A) Example neural network, with the first two nodes highlighted



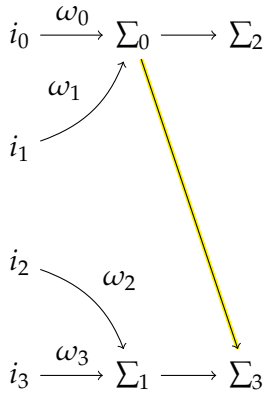
(B) Quantum circuit with four qubits, weighted input gates and entanglement between q_0, q_1 and q_2, q_3

FIGURE 2.19: Comparison of the summation nodes of the neural network and their quantum entanglement equivalent.

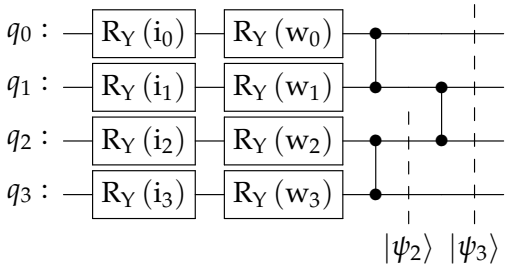
$$
\begin{aligned}
|\psi_0\rangle &= \mathrm{RY}(\omega_0)\mathrm{RY}(i_0) \otimes \mathrm{RY}(\omega_1)\mathrm{RY}(i_1) \\
|\psi_0\rangle &= \begin{pmatrix} \cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2} \\ \cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2} \end{pmatrix} \otimes \begin{pmatrix} \cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2} \\ \cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2} \end{pmatrix} \\
|\psi_0\rangle &= \begin{pmatrix} (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2}) \end{pmatrix}
\end{aligned}
\tag{2.21}
$$

$$
\begin{aligned}
|\psi_1\rangle &= CZ_{q_0,q_1}\,|\psi_0\rangle \\
|\psi_1\rangle &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2}) \end{pmatrix} \\
|\psi_1\rangle &= \begin{pmatrix} (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2}) \\ -(\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2}) \end{pmatrix}
\end{aligned}
\tag{2.22}
$$

At the end of the neural network, we have two more nodes, one of which is fully connected.



(A) Example neural network, with the connection between node $\sum_0$ and sum_3 highlighted



(B) Quantum circuit with four qubits, weighted input gates and entanglement between q_0, q_1 and q_2, q_3, and entanglement between q_{01} and q_{23}

FIGURE 2.20: Overview of the final quantum circuit in comparison to the neural network

The shown neural network is designed to illustrate the core problem that entanglement brings. If we are to recreate the network shown in figure 2.20, we get full entanglement of all present qubits. Equation 2.23 shows the resulting state $|\psi_3\rangle$. Figure 2.21 shows different neural network designs that lead to the same quantum circuit. Note that one could always measure a part of the circuit, and use that as an input for the next part that is represented in a different quantum circuit. This would allow us to recreate the given neural network fully, but increase complexity substantially, whilst also reducing the amount of transmitted information throughout the complete neural network.

$$|\psi_3\rangle = (I^4 \otimes CZ)(|\psi_1\rangle \otimes |\psi_2\rangle) = \begin{pmatrix} (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ (\cos\frac{i_0}{2}\cos\frac{\omega_0}{2} - \sin\frac{i_0}{2}\sin\frac{\omega_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ (\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\cos\frac{\omega_1}{2} - \sin\frac{i_1}{2}\sin\frac{\omega_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ -(\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ -(\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\cos\frac{\omega_2}{2} - \sin\frac{i_2}{2}\sin\frac{\omega_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \\ -(\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\cos\frac{\omega_3}{2} - \sin\frac{i_3}{2}\sin\frac{\omega_3}{2}) \\ -(\cos\frac{i_0}{2}\sin\frac{\omega_0}{2} + \cos\frac{\omega_0}{2}\sin\frac{i_0}{2})(\cos\frac{i_1}{2}\sin\frac{\omega_1}{2} + \cos\frac{\omega_1}{2}\sin\frac{i_1}{2})(\cos\frac{i_2}{2}\sin\frac{\omega_2}{2} + \cos\frac{\omega_2}{2}\sin\frac{i_2}{2})(\cos\frac{i_3}{2}\sin\frac{\omega_3}{2} + \cos\frac{\omega_3}{2}\sin\frac{i_3}{2}) \end{pmatrix} \tag{2.23}$$

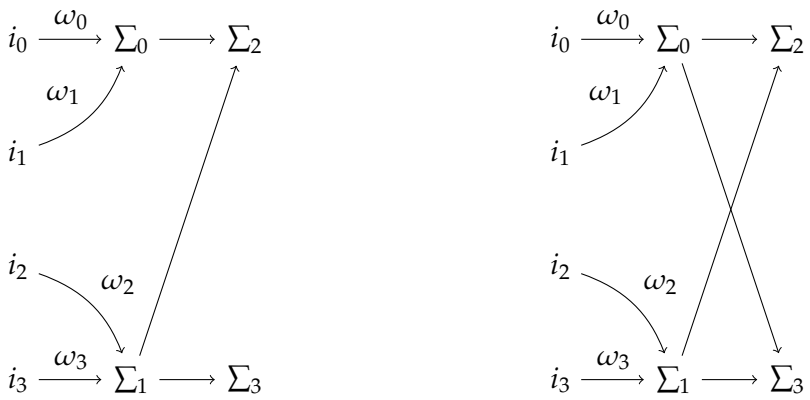


FIGURE 2.21: Example neural networks that all lead to the same quantum circuit illustrated in figure 2.19

Chapter 3

Quantum Embedding

3.1 Quantum Data Encoding

Quantum data encoding or quantum data embedding[1] describes the process of loading classical data in a quantum circuit by encoding it into the state of the qubits. In fact, the classical data is encoded in a Hilbert space using a quantum feature map. Lloyd et al.[16] defines quantum embedding as a quantum state $|x\rangle$ that represents a data input $x \in X$. It is facilitated by a quantum feature map $\phi : x \to |x\rangle$. For example, the quantum feature map can be executed by a quantum circuit $\Phi(x)$ whose gate parameters depend on x.

There are multiple ways of encoding classical data onto quantum states (see list 3.1) using a quantum feature map which directly affects the computational power of the quantum circuit, emphasizing that the encoding is a crucial part in designing quantum algorithms.[16–22]
An incomplete list of encoding patterns for quantum algorithms[21–23]:

- Basis Encoding (See section 3.2)
- Angle Encoding or Tensor Product Encoding (See section 3.3) [24]
- Amplitude Encoding & Repeated Amplitude Encoding
- Quantum Associative Memory (QUAM) Encoding
- Quantum Random Access Memory (QRAM) Encoding
  (No hardware implementation of QRAM exists until today [23])
- Coherent State Encoding
- General Near-term Encoding
- Divide-and-conquer Encoding [25]

3.1.1 State Preparation (Initialization)

The section in a quantum circuit where the classical data encoding is initialized is referred to as *state preparation*[22, 23] (see figure 3.1) and can consist of multiple different gates depending on the requirements of the chosen encoding and the quantum algorithm.

There is a trade-off between the following three major criteria[23]:

- *The amount of qubits needed for the encoding should be minimal* because current devices are of intermediate size and thus only contain a limited amount of qubits

[1] The terms *encoding* and *embedding* can be used interchangeably in this context.

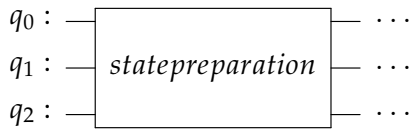


FIGURE 3.1: Abstract example quantum circuit with 3 qubits which encodes classical data and starts with a *state preparation* routine

- *The number of parallel operations needed to realize the encoding should be minimal to minimize the width of the quantum circuit* - ideally, the loading routine is of constant or logarithmic complexity
- *The data must be represented in a suitable manner* for further calculations, e.g., arithmetic operations.

3.2 Basis Embedding

The main idea of basis embedding is to encode a real number, approximated by a binary format, into the computational basis $\{|0\ldots00\rangle, |0\ldots01\rangle, \ldots, |1\ldots11\rangle\}$ of the quantum state. For example, the number "5" (decimal) is represented as 101 (binary format) which is then encoded by $|101\rangle$[21, 26]. The X gate (see section 2.1) is used to set a qubit with initial ground state $|0\rangle$ to $|1\rangle$ as shown in figure 3.2.

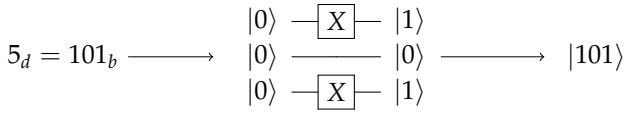


FIGURE 3.2: Basis encoding quantum circuit example with X gates to encode the decimal number 5 represented as 101_b into the quantum state

3.3 Angle Embedding

Whilst angle embedding is not optimal as it requires n qubits to represent n-dimensional data, it is efficient regarding operations and directly useful for processing data in quantum neural networks[23, 24]. Weigold et al. state that only single-qubit rotations are needed for the state preparation routine, which is highly efficient and can be done in parallel for each qubit.

The experiments are limited to angle encoding in the quantum circuits, where the value is directly encoded onto the rotation angle of the qubit state. Parametrizable gates[9] are used for the encoding. A detailed explanation of the used RY(θ) gate is available in chapter 2.2

Consider the example in figure 3.3 where the features x, y are angle encoded onto the qubits q_0, q_1 using the RY(θ) rotation gate, where θ is replaced with the corresponding feature.

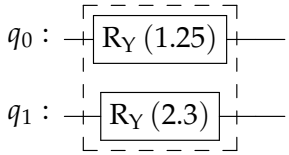


FIGURE 3.3: Angle encoding quantum circuit example with RY gates and grouping around the *state preparation*

The corresponding quantum state equation (3.1), as shown by Weigold et al.[23]:

$$|\psi\rangle = \begin{pmatrix} \cos\frac{1.25}{2} \\ \sin\frac{1.25}{2} \end{pmatrix} \otimes \begin{pmatrix} \cos\frac{2.3}{2} \\ \sin\frac{2.3}{2} \end{pmatrix} = \begin{pmatrix} \cos\frac{1.25}{2}\cos\frac{2.3}{2} \\ \cos\frac{1.25}{2}\sin\frac{2.3}{2} \\ \sin\frac{1.25}{2}\cos\frac{2.3}{2} \\ \sin\frac{1.25}{2}\sin\frac{2.3}{2} \end{pmatrix} \tag{3.1}$$

The example circuit from figure 3.3 can be written in Python with the Pennylane.ai library:

```
# pennylane.ai python library angle embedding example
import pennylane as qml
from pennylane import numpy as np

# features
x = 1.25
y = 2.3
data = np.array([[x, y]])

# quantum device (simulator) and circuit
dev = qml.device('default.qubit', wires=2)

@qml.qnode(dev)
def circuit(data, probs=False):
    for i in range(len(data)):
        qml.RY(data[i][0], wires=0) # input feature x
        qml.RY(data[i][1], wires=1) # input feature y
    if probs:
      return  qml.probs(wires=range(2))
    return  qml.state()

# print the probabilities
print("State vector:")
print(circuit(data))
print("Probabilities:")
print(circuit(data, True))
```

The code then returns the probability vectors:

```
State vector:
[0.33126825+0.j 0.74021789+0.j 0.23900489+0.j 0.53405569+0.j]
Probabilities:
[0.10973865 0.54792253 0.05712334 0.28521548]
```

The same example (figure 3.3) written in Python with the qiskit.org library:

```
# qiskit.org python library angle embedding example
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
import numpy as np

# features
x = 1.25
y = 2.3
data = np.array([[x, y]])

# create circuit
circuit = QuantumCircuit(2)
for i in range(len(data)):
  circuit.ry(data[i][0],0) # input feature x
  circuit.ry(data[i][1],1) # input feature y

# get the probabilities
state = Statevector.from_instruction(circuit.reverse_bits())
probs = state.probabilities()

# print the probabilities
print(state)
print("Probabilities:")
print(probs)
```

The code then returns the probability vectors:

```
Statevector([0.33126825+0.j, 0.74021789+0.j, 0.23900489+0.j,
             0.53405569+0.j],
            dims=(2, 2))
Probabilities:
[0.10973865 0.54792253 0.05712334 0.28521548]
```

Using the statement in chapter 2.1 $|\alpha|^2 + |\beta|^2 = 1$, the statevector gives the probabilities with:

$$\begin{aligned} |0.33126825|^2 + |0.74021789|^2 + |0.23900489|^2 + |0.53405569|^2 = \\ 0.10973 \cdots + 0.54792 \cdots + 0.05712 \cdots + 0.28521 \cdots = 1 \end{aligned} \tag{3.2}$$

The probability vector results are illustrated in the following table 3.1:

qubit state	probability
00	0.10973865
01	0.54792253
10	0.05712334
11	0.28521548

TABLE 3.1: Probability vector results

Table 3.1 can be interpreted that there is a probability of 0.10973865 for measuring "0" for qubit 0 (q_0) and "0" for qubit 1 (q_1), a probability of 0.54792253 for measuring

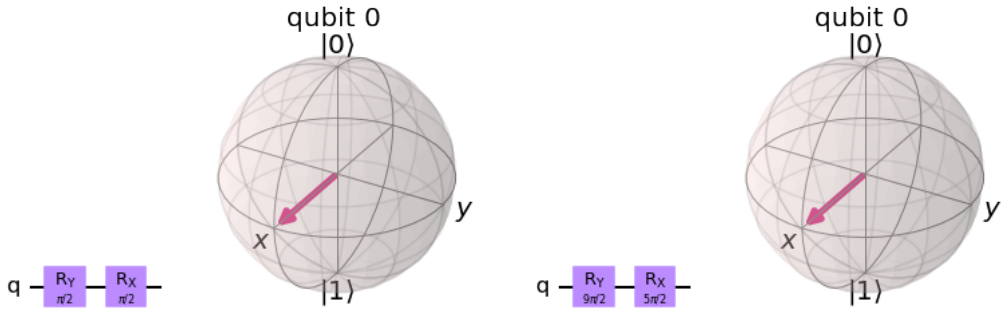
"0" for q_0 and "1" for q_1, a probability of 0.05712334 for measuring "1" for q_0 and "0" for q_1 and finally a probability of 0.28521548 for measuring "1" for q_0 and "1" for q_1. The most likely outcome is "01" as a final state with a probability greater than ˜0.54.

3.4 Data Preprocessing

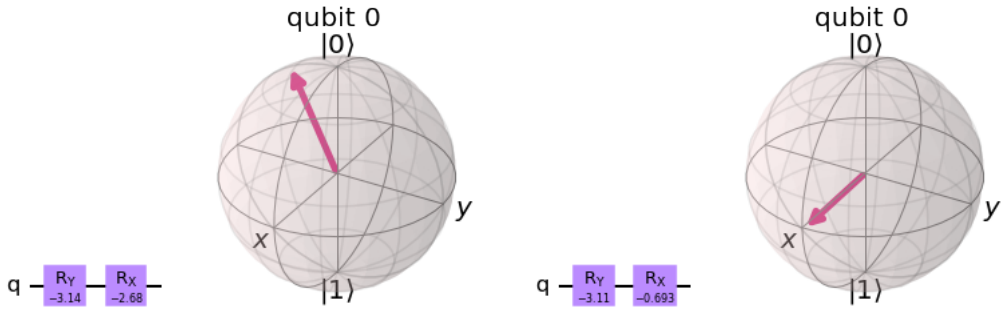
As in classical machine learning, creating usable classifiers requires the preprocessing of the available data. This can be achieved in a plethora of different ways, for example, mean reduction, feature scaling, normalization and padding. Depending on the input data and the chosen quantum data encoding, preprocessing is a crucial or even mandatory step[27–29].

One such an example is amplitude encoding, where the input feature vector is encoded into the amplitudes of the quantum state. Weigold et al. [26] states that the squared moduli of the amplitudes of a quantum state must sum up to 1, and therefore the input feature vector needs to be normalized to length 1. To associate each amplitude with a component of the input feature vector, the dimension of the feature vector must be equal to a power of two because the vector space of a n qubit register has dimension 2^n. If this is not the case, the input feature vector can be padded with additional zeros to increase the dimension of it. To summarize the preprocessing steps needed, the feature vectors may need to be padded and finally normalized to length 1 before given to the amplitude encoding routine.

The data used in our experiments is scaled to different ranges using MinMaxScaler[30] and StandardScaler[31] from sklearn. The StandardScaler centres and scales the data to unit variance and the MinMaxScaler is used to additionally scale the data between a given range $[x, y]$, where $x < y$, for the angle embedding. Since the angles for the rotation gates RY, RX and RZ are 2π-periodic, it makes sense to scale the data accordingly to ranges $[-1, 1]$, $[0, 2\pi]$[21] or $[-\pi, \pi]$. Scaled data can lead to a different state, as seen in figure 3.4.



(A) Original unscaled data: Circuits and Bloch spheres of features which result in an identical state



(B) Scaled data: Circuits and Bloch spheres of scaled features which result in a different state than the original data. In this example the same data - with additional data points that are not shown here - as in 3.4a from above has been scaled using the MinMaxScaler with range $[-\pi, \pi]$.

FIGURE 3.4: Original versus scaled data example circuits and states

Chapter 4

Computational Neuron

4.1 Classical Variant

The classic computational neuron, often referred to as a perceptron, is a multistep process that amounts to a single result that can be used in further neurons, or directly read out as an output. In their simplest form, the neuron has n input features that are multiplied with n weights. In addition to those features, there is a weighted bias, where the feature itself has the fixed value 1. These values are then summarized and evaluated through an activation function $f(x)$. Originally proposed by Rosenblatt F.[32], the structure of a neuron with $n = 4$ input features and one bias feature is shown in figure 4.1[1]. A more formal definition is presented in equation 4.1, where $j = 0$ is the bias.

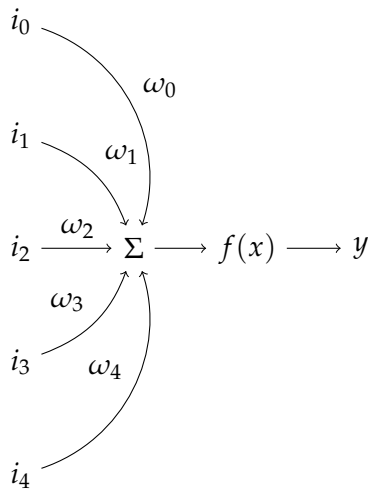


FIGURE 4.1: Visual representation of one perceptron with 4 features and 1 bias

$$y = f\left(\sum_{j=0}^{n} i_j \omega_j\right) \tag{4.1}$$

[1]There are many iterations of this design that range from a different handling of the bias to multiple output designs like LSTMs[33]

The activation function $f(x)$, also called transfer function, can be selected from a plethora of valid choices[34], depending on the problem at hand. Dissecting the neuron leads to the two key components needed for the creation of an equivalent quantum neuron: A function for addition U_+ and one for multiplication $U_\times$ (The activation function can then be constructed from these). Whilst a single qubit can be in a coherent superposition of two different states[35], its measurements always amount to either 0 or 1, with their respective probabilities. As a result, the implementation uses binary data embedding. The implementation also neglects the usage of *entanglement* and *superposition*, as the arithmetic functions do not make usage of probabilistic calculations.

4.2 Binary Addition

Binary addition allows the summation of all components $i_j\omega_j$ into a single variable x, which is then given to the activation function. It is also a prerequisite to creating the binary multiplication circuit. The rules that define binary addition are relatively simple and summarized in table 4.1 for the expression $x_A + x_B = y$

x_A	x_B	y
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 4.1: Rule set for the binary addition for two 1 bit wide words

With table 4.1 a matrix U_+ can be created, that follows the given rules that apply to the creation of quantum gates[36]. Simply speaking, the matrix has to fulfill $U_+U_+^\dagger = I$. This is also often referred to as *bijective* when working with functions. It dictates that any single resulting state can be followed back to a single input state. To fulfill these prerequisites, we have to use 3 qubits to assert no loss of information. The first two qubits are input bits x_A and x_B, respectively, whilst the third one is output bit y. In this case, the vector $\vec{S}$ stands for any of the $2^3 = 8$ possible states the 3 qubits can be in, and vector $\vec{O}$ to any possible output state, as shown in equation 4.2.

$$U_+\vec{S} = \vec{O} \rightarrow U_+ = OS^T \tag{4.2}$$

First, the state matrix S is created that represents all possible input states of our quantum circuit.

$$\vec{S_0} = |0\rangle \otimes |0\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix} \tag{4.3}$$

$$\rightarrow S = I^{8\times 8}$$

Using the same method and the rules from table 4.1, the output state matrix O is constructed in equation 4.5.

$$\begin{aligned}
&1.\ \text{Row: } x_A + x_B = y \rightarrow |0\rangle + |0\rangle = |0\rangle \\
&2.\ \text{Row: } x_A + x_B = y \rightarrow |0\rangle + |1\rangle = |1\rangle \\
&3.\ \text{Row: } x_A + x_B = y \rightarrow |1\rangle + |0\rangle = |1\rangle \\
&4.\ \text{Row: } x_A + x_B = y \rightarrow |1\rangle + |1\rangle = |0\rangle
\end{aligned} \tag{4.4}$$

$$\vec{O}_0 = |0\rangle \otimes |0\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

$$\rightarrow O = \begin{pmatrix}1&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&0&0&0&1\end{pmatrix} \tag{4.5}$$

Note that the matrix O in equation 4.5 can be flipped, depending on what qubit is selected as output. The two matrices O and S are now inserted into $U_+ = OS^T$

and solved for U_+, as per equation 4.6

$$
\begin{aligned}
U_+ = OS^T &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\
\rightarrow U_+ &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
\tag{4.6}
$$

The calculated matrix can easily be turned into a quantum gate using `qiskit`.

```
from qiskit import QuantumCircuit
import qiskit.quantum_info as qi

n_qubits = 3

#Create the Gate
addition = qi.Operator([[1, 0, 0, 0, 0, 0, 0, 0],
                        [0, 0, 0, 0, 0, 1, 0, 0],
                        [0, 0, 0, 0, 0, 0, 1, 0],
                        [0, 0, 0, 1, 0, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0, 0, 0],
                        [0, 1, 0, 0, 0, 0, 0, 0],
                        [0, 0, 1, 0, 0, 0, 0, 0],
                        [0, 0, 0, 0, 0, 0, 0, 1]])
```

Binary addition with bit words longer than 1 use a carry bit, which is then used as the input for the next addition. As the carry over is not always needed, a separate quantum gate is created. The rules for binary addition with carry over are listed in table 4.2

$carry_{in}$	x_A	x_B	y	$carry_{out}$
0	0	0	0	0
1	0	0	1	0
0	0	1	1	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	1
0	1	1	0	1
1	1	1	1	1

TABLE 4.2: Rule set for the binary addition for two 1 bit wide words with carry over

It is important to note here that the $carry_{in}$ bit from table 4.2 is being reused as the output bit y. Using the *Toffoli Gate*, also called `CCNOT`[37], we can use two controlling qubits to invert the state of a third one, as demonstrated with figure 4.2.

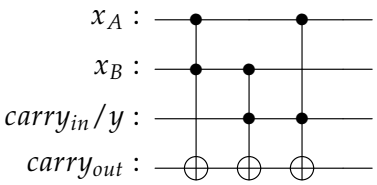


FIGURE 4.2: Quantum circuit to calculate the carry over bit

Combining these two solutions, we construct a quantum circuit to add any n bit wide words together. Figure 4.3 demonstrates a circuit for the addition of two 4 bit wide words.

In figure 4.3 the i in x_{Ai} and x_{Bi} stands for the ith bit in the word, starting with the LSB[2]. Both the *carry over* and *addition* gates are the previously defined ones. The

[2]Least significant bit

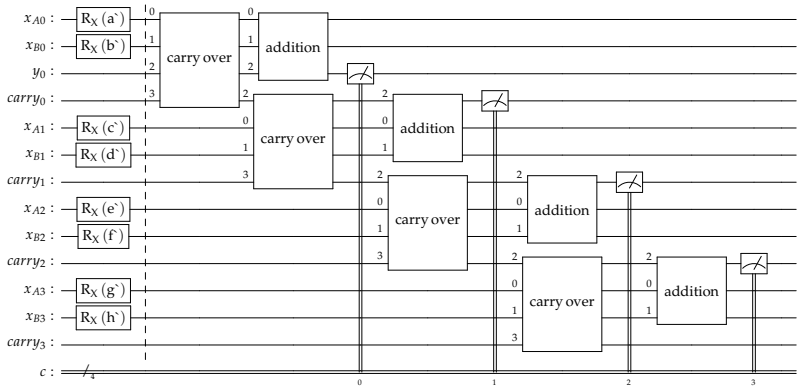


FIGURE 4.3: A 4 bit wide full adder realized as a quantum circuit

numbering on their input side resembles the mapping of the incoming qubit to the *internal qubit*. The figure also illustrates how the $carry_i$ qubit is reused as the output qubit of the next addition step.

4.3 Binary multiplication

Using the same steps as in chapter 4.2, table 4.3 illustrates the binary multiplication rules.

x_A	x_B	y
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 4.3: Binary multiplication rules

The defined rules resemble only a subset of what makes binary multiplication work. The results of the multiplication get shifted one by one and then summarized. Figure 4.4 illustrates the procedure with a basic example.

x_A			1	0	1	1	0	1
x_B						1	0	1
			1	0	1	1	0	1
		0	0	0	0	0	0	X
$+$	1	0	1	1	0	1	X	X
y	1	1	1	0	0	0	0	1

FIGURE 4.4: Rule set for the binary multiplication of two 1 bit wide words

Quantum rules with our multiplication rules have a problem, as illustrated in table 4.4

$\|\psi_{in}\rangle$	$\|\psi_{out}\rangle$
000	000
001	000
010	010
011	010
100	100
101	100
110	111
111	111

TABLE 4.4: 3 qubit state multiplications

Table 4.4 shows multiple states in $|\psi_{out}\rangle$ that are duplicates, which, as discussed in chapter 4.2, is not allowed. To solve this the assumption has to be made, that any implementation will always enforce $y_{in} = |0\rangle$. Table 4.5 is corrected for the new behaviour, which fulfils all requirements.

$\|\psi_{in}\rangle$	$\|\psi_{out}\rangle$
000	000
001	001
010	010
011	011
100	100
101	101
110	111
111	110

TABLE 4.5: Illustration of input states $|x_A x_B y_{in}\rangle$ and output states $ketx_A x_B y_{out}$, where non-bijective behaviour is shown

There is no perfect solution for this problem, as binary multiplication does not use $y_{in} = 1$ in its rule set and therefore, any usage where that state does exit is either on *purpose* or due to an implementation error. Equation 4.3 demonstrates that the state matrix S equals the identity matrix I of the same dimensionality (as we have 3 qubits, we get $2^3 = 8$). The output matrix O is constructed using the states defined in table 4.5 and used to calculate $U_\times$, as demonstrated in equation 4.7.

$$\begin{aligned} U_\times = OS^T = & \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&1&0&0&0&0 \end{pmatrix} \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix} \\ \rightarrow U_\times = & \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&1&0&0&0&0 \end{pmatrix} \end{aligned} \tag{4.7}$$

The calculated matrix $U_\times$ is then turned into an usable quantum gate in `qiskit`.

```
from qiskit import QuantumCircuit
import qiskit.quantum_info as qi

n_qubits = 3

#Create the Gate
multiplication = qi.Operator([[1, 0, 0, 0, 0, 0, 0, 0],
                              [0, 1, 0, 0, 0, 0, 0, 0],
                              [0, 0, 1, 0, 0, 0, 0, 0],
                              [0, 0, 0, 0, 0, 0, 0, 1],
                              [0, 0, 0, 0, 1, 0, 0, 0],
                              [0, 0, 0, 0, 0, 1, 0, 0],
                              [0, 0, 0, 0, 0, 0, 1, 0],
                              [0, 0, 0, 1, 0, 0, 0, 0]])
```

With $U_\times$ and U_+ calculated as well as the *carry over* gate defined, a full binary multiplier can be constructed, demonstrated in figure 4.5.

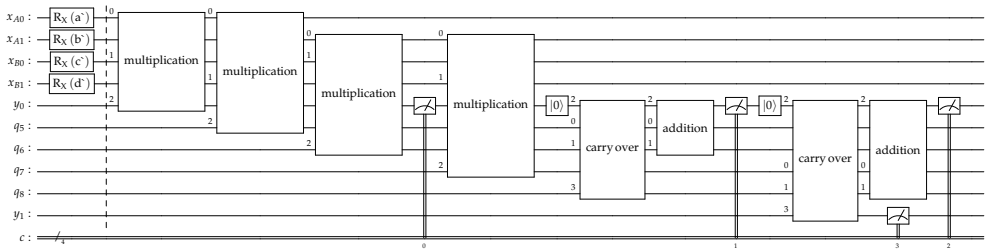


FIGURE 4.5: A 2 bit wide binary multiplier realized as a quantum circuit

Figure 4.5 shows a circuit that can multiply two 2 bit wide words. The presented form is optimized, as in, the qubit that has been measured is reset and reused for the next calculation step. In comparison to the non-optimized solution, this reduces the amount of required qubits, in this example, from 12 to 10. Equations 4.8 and 4.9 demonstrate how the required amount of qubits can be calculated. $n_{bits} = 1$ is a special case, as the multiplication's result is always 1 bit wide. With $n_{bits} > 1$ the result's length l is in the range $n_{bits} \leq l \leq 2n_{bits}$

$$\begin{aligned} n_{bits} &= 1 \\ n_{measure} &= 1 \\ n_{qubits} &= 2n_{bits} + n_{measure} = 3 \end{aligned} \tag{4.8}$$

$n_{bits} > 1$ adds a special rules that can be observed in figure 4.5. The intermediate results of every multiplication, except the first, must be preserved. Additional qubits are then needed to act as *carry* for the binary addition. For this case, n_{carry} equals 1, as we can reuse the last measurement qubit. This was omitted in equation 4.8 as there was no need for binary addition. Note that $n_{measure}$ is always 2 as we can reuse the first measurement, but need a second one for the last measurement.

$$\begin{aligned} n_{bits} &= 2 \\ n_{U_\times} &= 2n_{bits} - 1 \\ n_{measure} &= 2 \\ n_{carry} &= 1 \\ n_{qubits} &= 2n_{bits} + n_{U_\times} + n_{carry} + n_{measure} = 10 \end{aligned} \tag{4.9}$$

Any larger sizes could not be created nor evaluated. The required amount of qubits needed to create and simulate a circuit for a 3 bit wide multiplier disallowed any experiments, as the `qiskit` simulator could not handle it. It can be expected that the amount of qubits increases is, *at least*, linearly. The amount of binary additions required in the classical solution for n_{bits} are listed in 4.6.

n_{bits}	number of additions
1	0
2	1
3	4
4	9

TABLE 4.6: Number of binary addition steps for n_{bits} in a classical solution

4.4 Conclusion

The solutions presented in this chapter show that the replication of a computational neuron as a quantum circuit is possible. The amount of required qubits for the implementation of the basic arithmetic operations seem to be a deterrent factor when it comes to further pursuing this solution. If one would to implement this solution, it would either be partial, as in, after each arithmetic step a measurement is made, and inserted into the next quantum circuit, or one would have to wait for the availability of hardware with enough qubits[38]. On the other hand, the missing usage of quantum-specific features will not allow for any quantum speed up in comparison to the classical variant, as we are merely emulating it. Current quantum solutions for classical problems like *Multiple Query Optimization* as demonstrated by Fankhauser et al.[3], can be solved without the need to replicate classical methods with the usage of superpositions and entanglement, which are completely absent from the solutions presented in chapters 4.2 and 4.3, and also *outperform* the classical counterpart.

This does not mean that the proposed circuits serve no purpose. These could, in the future, be used to allow basic arithmetic on quantum-only devices. The reversed state of the n-bit multiplier shown in figure 4.5 could be adapted, using superpositions and entanglement, to calculate two factors used to attain the given result.

Chapter 5

Quantum Classifiers And Decision Boundaries

To analyse the expressiveness, accuracy and the impact of small gate changes for quantum classifier circuits in comparison to a classical Multi-layer Perceptron classifier (sklearn neural network `MLPClassifier`), contour plots are used to visualize the decision boundaries. A quantum classifier is defined as follows by Lloyd et al.[16]: *Let $|x\rangle$ be an embedding of a data input $x \in X$. A quantum classifier is a classifier where $f(x)$ is an expectation $\langle x| M |x\rangle$ of a quantum measurement of the observable M.* The classifier which receives the measured output of the quantum circuit (see 5.1) is defined as follows[16]:

$f : X \to \mathbb{R}$. The quantum classifier assigns a binary label to x according to the threshold rule:

$$y = \begin{cases} -1 \; if \; f(x) < \tau, \\ 1 \; if \; f(x) \geq \tau \end{cases} \tag{5.1}$$

with $\tau \in \mathbb{R}$ and if no other information is provided, τ is assumed to be zero.

```
# Compute predictions on train and validation set
predictions_train = [np.sign(quantum_classifier(var, f)) for f in x_train]
predictions_val = [np.sign(quantum_classifier(var, f)) for f in x_test]
```

FIGURE 5.1: Classifier uses `np.sign` (see `numpy.sign()`) function to assign label after classification

Lloyd et al.[16] further states that quantum binary classification is defined as: *For a given embedding Φ, quantum binary classification is the problem of assigning $|x\rangle$ to one of two "data-ensembles" of quantum feature states, $\varrho = \frac{1}{M_a} \sum_a |a\rangle \langle a|$ or $\sigma = \frac{1}{M_b} \sum_b |b\rangle \langle b|$, where ϱ and σ are mixed states that describe the process of selecting an embedded data point $|a\rangle$, $|b\rangle$ with uniform probability from a training set.*

The following experiments in subsections 5.0.1, 5.0.2 and 5.0.3 show three different artificial problems consisting of linearly separable (*make_classfication*), circle shaped (*make_circles*) and moon shaped data (*make_moons*), ordered from easier to more difficult. All data sets have only two classes (binary) and are limited to 100 data points. The data given to the quantum classifiers has been scaled to $[-1, 1]$ except for the "Two qubit circuits with 3 Layers" in subsection 5.0.3 where the data has been scaled to $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Data was further divided into 75% being used for training and 25% for testing. The Pennylane.ai framework has been used with the `default.qubit` simulator. Optimization is done in a classical way with the `NesterovMomentumOptimizer` optimizer. For each iteration step, the optimizer updates the weights and gives the new values again to the quantum model circuit. The schematic setup for the quantum circuit and classical parts is identical as seen in figure 6.3.

```
MLPClassifier(random_state=1, max_iter=max_iterations,
        solver="sgd", nesterovs_momentum=True)
```

(A) Python code: `MLPClassifier` configuration for single and 2 qubit classifier circuits (see 5.0.1 and 5.0.2)

```
MLPClassifier(random_state=1,
        hidden_layer_sizes=(100,100,100),
        max_iter=max_iterations, solver="sgd",
        nesterovs_momentum=True),
```

(B) Python code: `MLPClassifier` configuration for 2 qubit classifier circuits with 3 layers (see 5.0.3)

FIGURE 5.2: `MLPClassifier` configurations

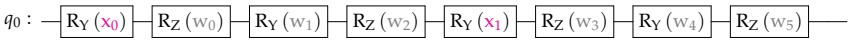
All quantum circuits in subsections 5.0.1, 5.0.2 and 5.0.3 are finally measured in the Z axis (PauliZ) for qubit q_0.

```
@qml.qnode(dev)
def circuit(weights, data):
    ...
    return qml.expval(qml.PauliZ(0))
```

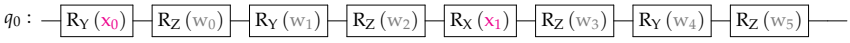
FIGURE 5.3: Python code: Circuit measurement of Z axis for qubit q_0

5.0.1 Single Qubit Circuits

In this subsection, two single qubit quantum classifiers with angle embedding for the features x_0, x_1 and a total count of six weights ($w_0, \ldots, w_5$) have been trained. For comparison, there is also a Multi-layer Perceptron classifier in the plots with the same iteration count as the quantum classifiers. Figure 5.5a has a iteration count of 50, figure 5.6a shows 200 iterations and the last figure 5.7a shows the classifiers after 400 iterations. The single qubit quantum classifier circuits can be seen in figures 5.4a and 5.4b.



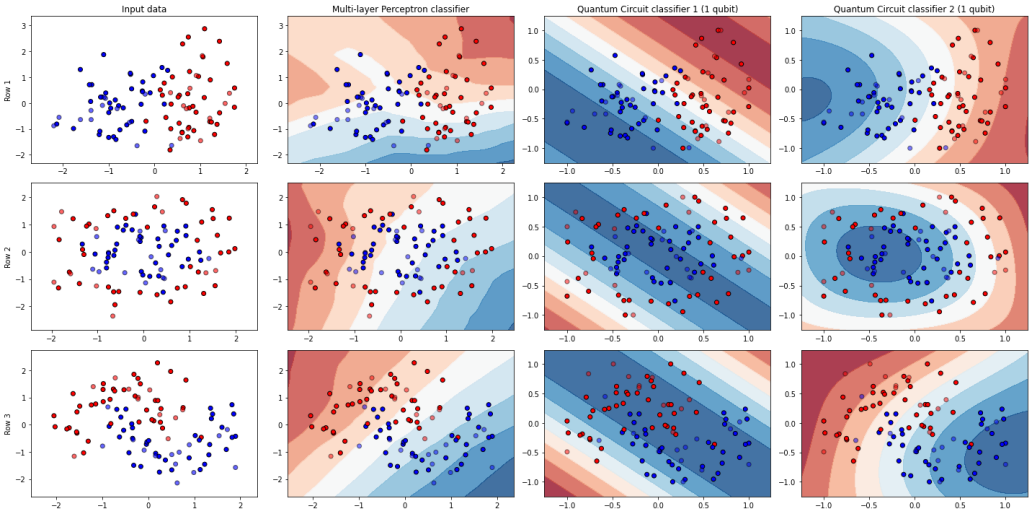
(A) **Quantum Circuit classifier 1** with only RY and RZ gates using 6 weights ($w_0, \ldots, w_5$) and two input features x_0, x_1



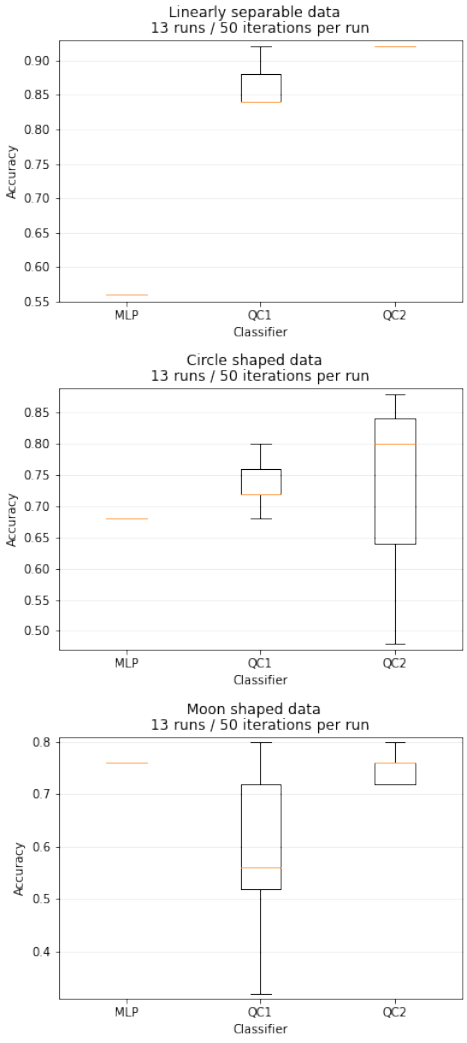
(B) **Quantum Circuit classifier 2** with RY, RZ gates and a RX gate using also 6 weights ($w_0, \ldots, w_5$) and two input features x_0, x_1

FIGURE 5.4: Single qubit quantum circuits referring to **Quantum Circuit classifier 1** and **Quantum Circuit classifier 2** in figures 5.5a, 5.6a and 5.7a.

FIGURE 5.5: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1 and Quantum Circuit classifier 2.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data

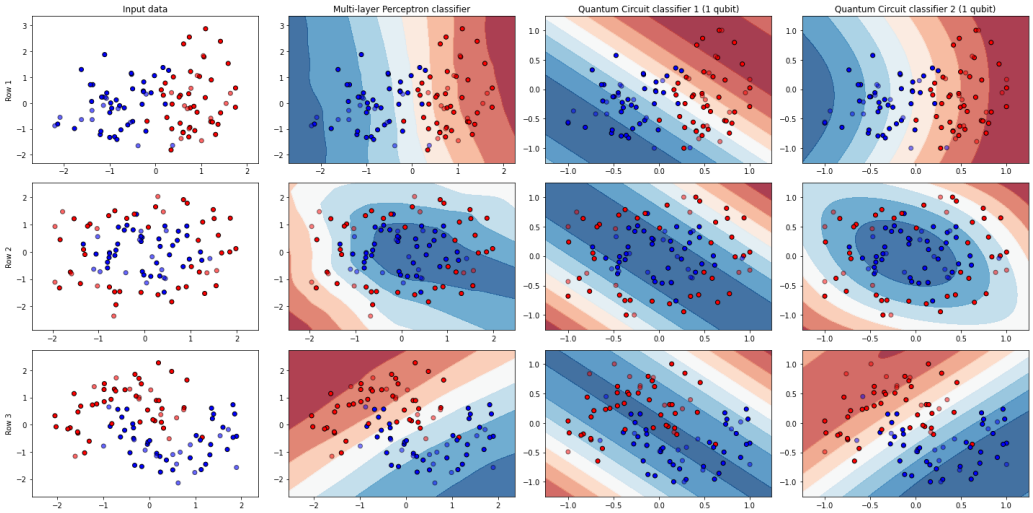


(A) Decision boundary plot: **50 iterations** for all classifiers using **single qubit quantum circuits**.

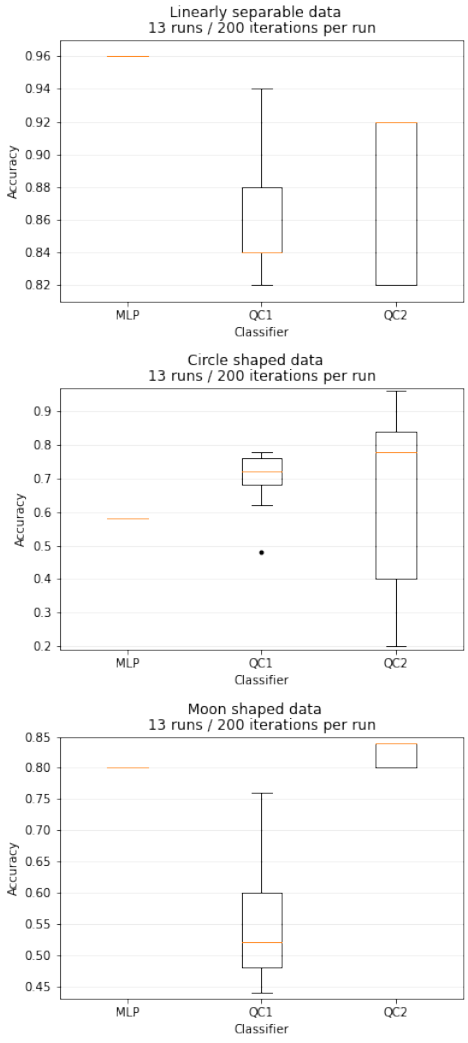


(B) Accuracy box plots: **50 iterations** for all classifiers using **single qubit quantum circuits**.
MLP, **QC1** and **QC2** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1** and **Quantum Circuit classifier 2** in figure 5.5a accordingly.

FIGURE 5.6: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1 and Quantum Circuit classifier 2.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data

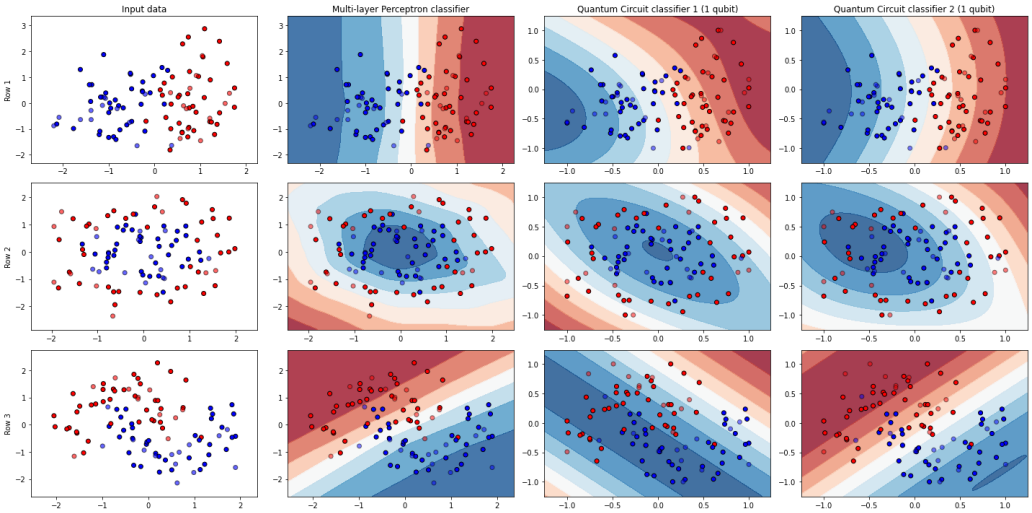


(A) Decision boundary plot: **200 iterations** for all classifiers using **single qubit quantum circuits**.

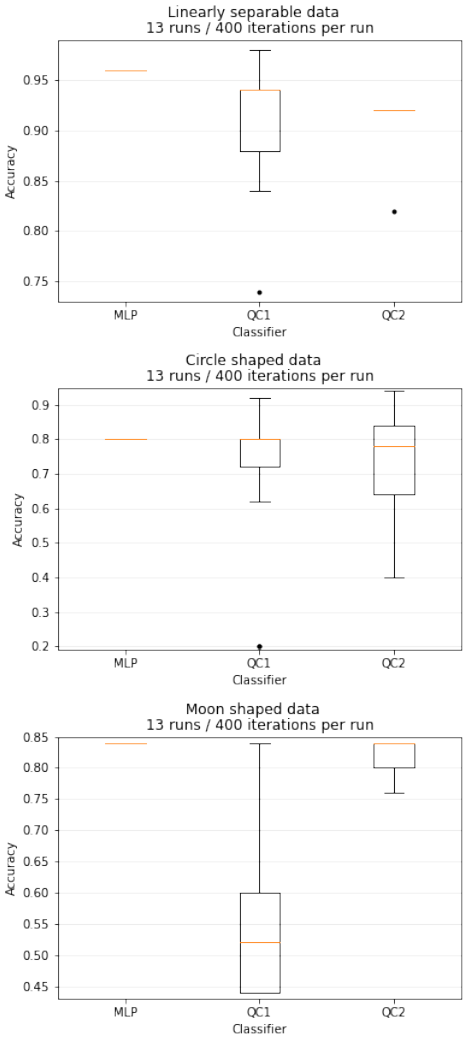


(B) Accuracy box plots: **200 iterations** for all classifiers using **single qubit quantum circuits**.
MLP, **QC1** and **QC2** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1** and **Quantum Circuit classifier 2** in figure 5.6a accordingly.

FIGURE 5.7: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1 and Quantum Circuit classifier 2.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data



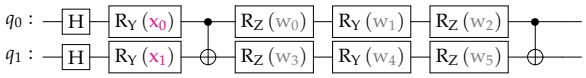
(A) Decision boundary plot: **400 iterations** for all classifiers using **single qubit quantum circuits**.



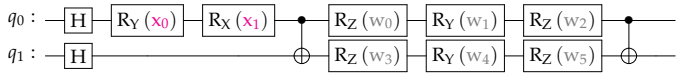
(B) Accuracy box plots: **400 iterations** for all classifiers using **single qubit quantum circuits**.
MLP, **QC1** and **QC2** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1** and **Quantum Circuit classifier 2** in figure 5.7a accordingly.

5.0.2 Two qubit circuits

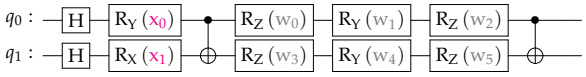
In this subsection, three two qubit quantum classifiers with angle embedding and a total count of six weights are trained. For comparison, there is also a Multi-layer Perceptron classifier (sklearn neural network MLPClassifier) in the plots with the same iteration count as the quantum classifiers. Figure 5.9a has an iteration count of 50, figure 5.10a shows 200 iterations and the last figure 5.11a shows the classifiers after 400 iterations. The two qubit quantum classifier circuits can be seen in figures 5.8a, 5.8b and 5.8c.



(A) **Quantum Circuit classifier 1** with starting Hadamard (H) gates and angle embedding using a RY gates entangled with a CX gate followed by multiple RY and RZ rotation gates containing six weights in total, entangled with a final CX gate



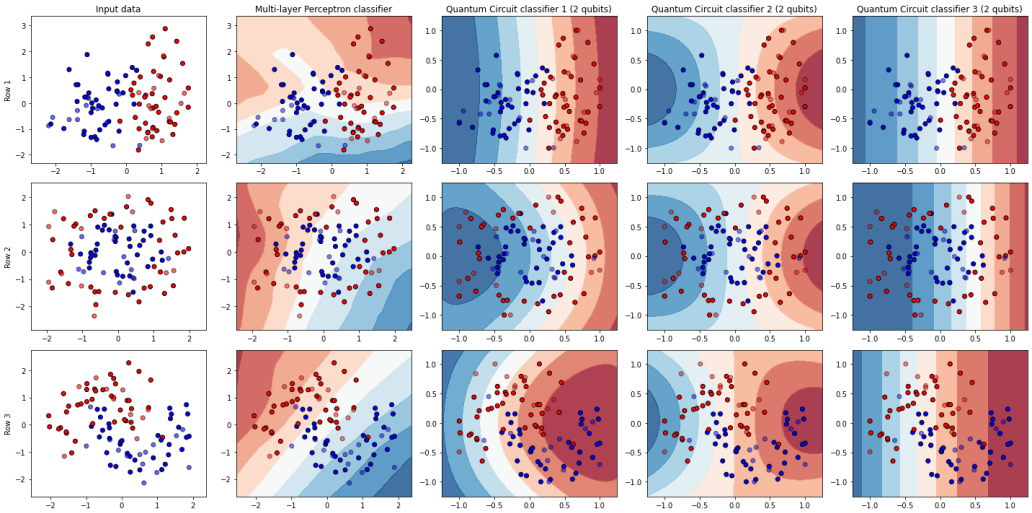
(B) **Quantum Circuit classifier 2** differs from **Quantum Circuit classifier 1** 5.8a in terms of angle embedding using a RY and a RX gate to encode both features x_0, x_1 on the first qubit q_0 instead of using both qubits



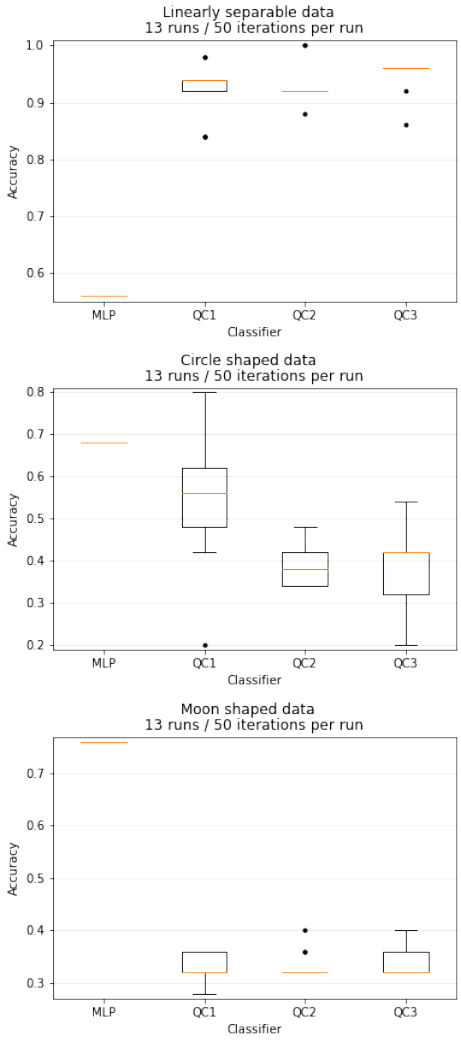
(C) **Quantum Circuit classifier 3** with starting Hadamard (H) gates and angle embedding using a RY and a RX gate

FIGURE 5.8: Two qubit quantum circuits referring to the **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figures 5.9a, 5.10a and 5.11a.

FIGURE 5.9: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1, Quantum Circuit classifier 2 and Quantum Circuit classifier 3.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data

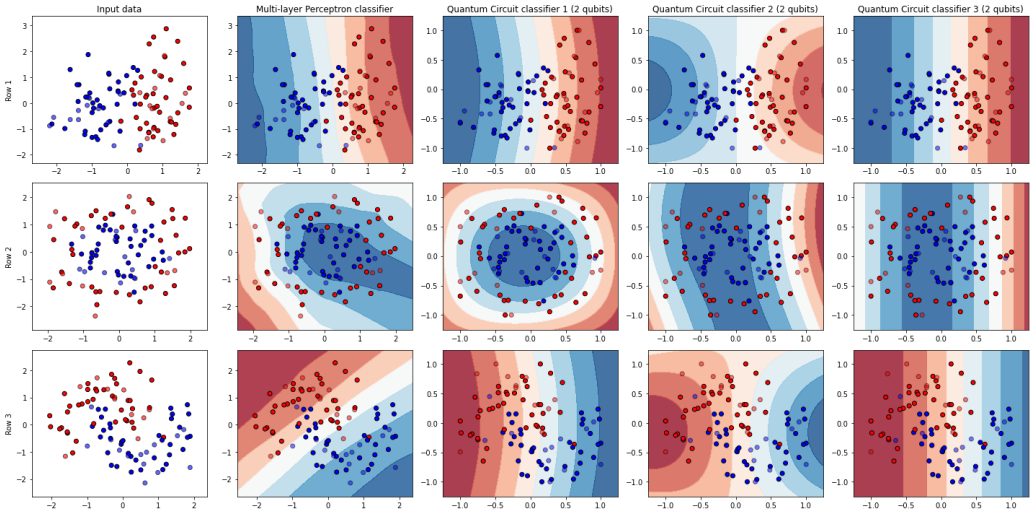


(A) **50 iterations** for all classifiers using **two qubit quantum circuits**.

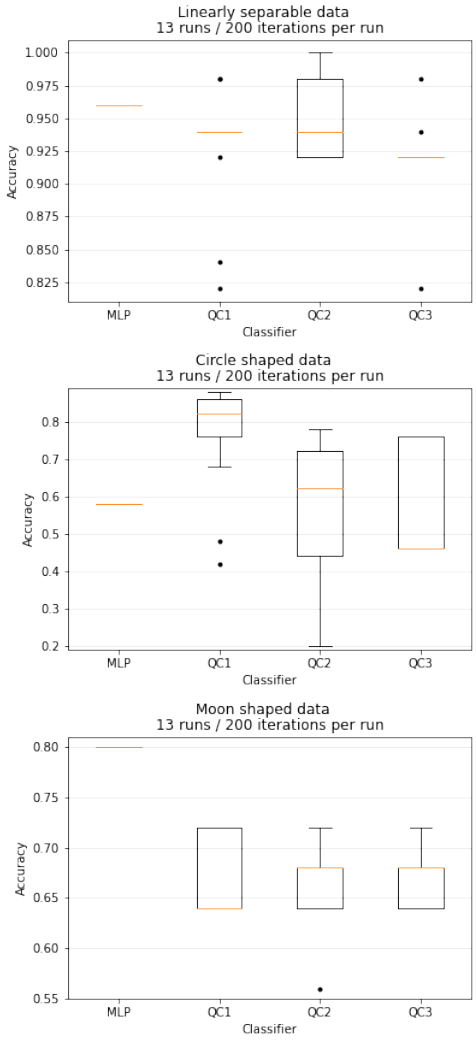


(B) Accuracy box plots: **50 iterations** for all classifiers using **two qubit quantum circuits**.
MLP, **QC1**, **QC2** and **QC3** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figure 5.9a accordingly.

FIGURE 5.10: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1, Quantum Circuit classifier 2 and Quantum Circuit classifier 3.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data

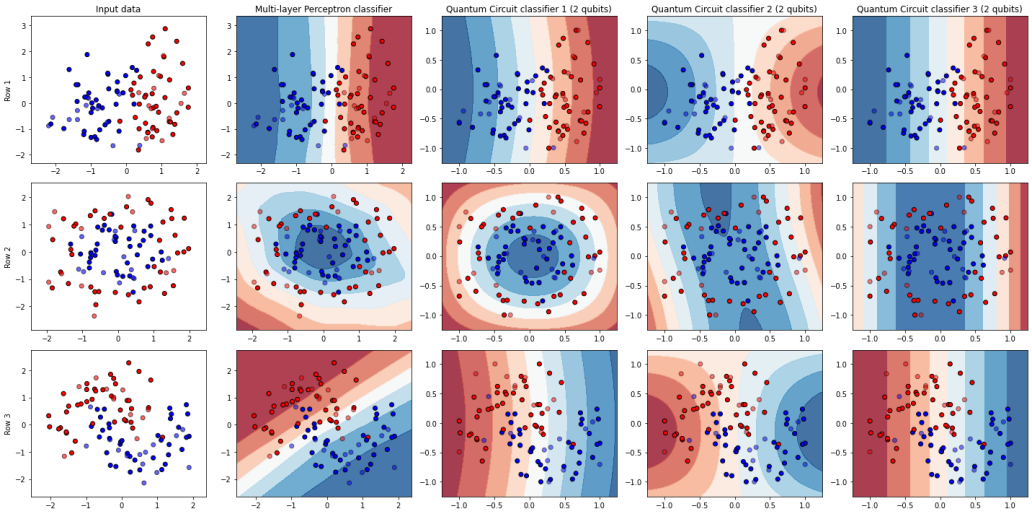


(A) **200 iterations** for all classifiers using **two qubit quantum circuits**.

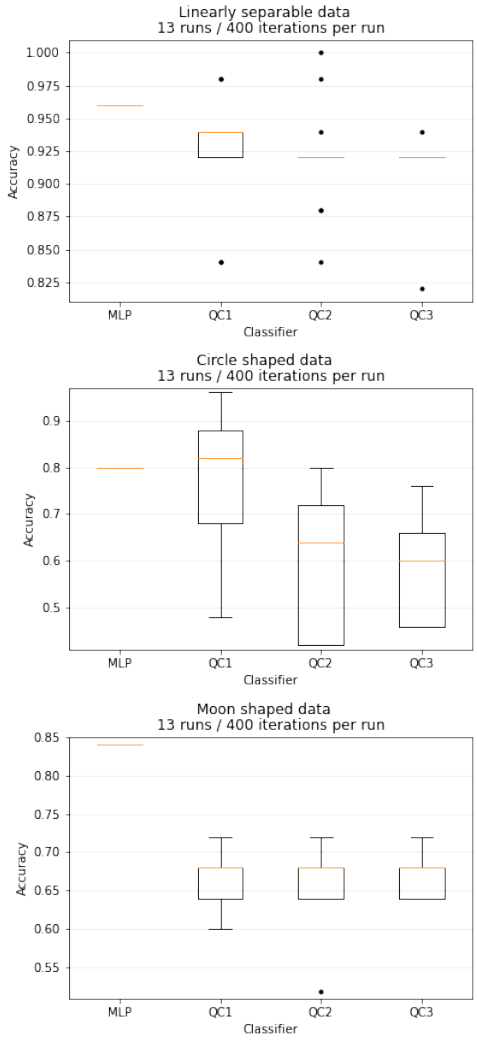


(B) Accuracy box plots: **200 iterations** for all classifiers using **two qubit quantum circuits**. **MLP**, **QC1**, **QC2** and **QC3** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figure 5.10a accordingly.

FIGURE 5.11: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1, Quantum Circuit classifier 2 and Quantum Circuit classifier 3.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data



(A) **400 iterations** for all classifiers using **two qubit quantum circuits**.



(B) Accuracy box plots: **400 iterations** for all classifiers using **two qubit quantum circuits**. **MLP**, **QC1**, **QC2** and **QC3** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figure 5.11a accordingly.

5.0.3 Two qubit circuits with 3 layers

In this subsection, three two qubit quantum classifiers with three layers (the same quantum circuits as in subsection 5.0.2 repeated three times, excluding the Hadamard (H) gates) using angle embedding and a total count of $3 \times 6 = 18$ weights have been trained. Again for comparison, there is also a Multi-layer Perceptron classifier (sklearn neural network MLPClassifier) in the plots with the same iteration count as the quantum classifiers, but this time with 3 hidden layers as seen in figure 5.2b. Figure 5.13a has an iteration count of 50, figure 5.14a shows 200 iterations and the last figure 5.15a shows the classifiers after 400 iterations. The two qubit quantum classifiers with three layer circuits can be seen in figures 5.12a, 5.12b and 5.12c.

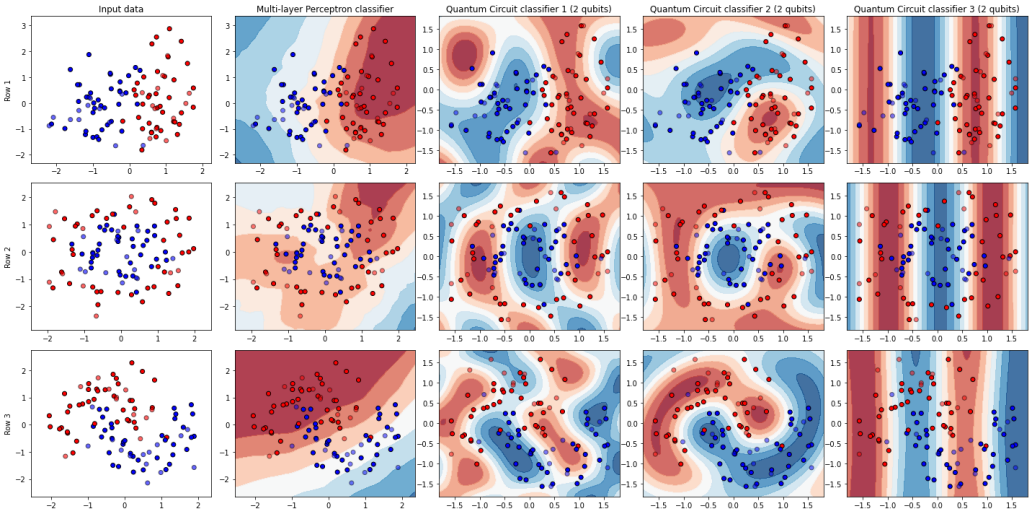
(A) **Quantum Circuit classifier 1** same as the two qubit classifier in figure 5.8a with 2 additional "layers"

(B) **Quantum Circuit classifier 2** same as the two qubit classifier in figure 5.8b with 2 additional "layers"

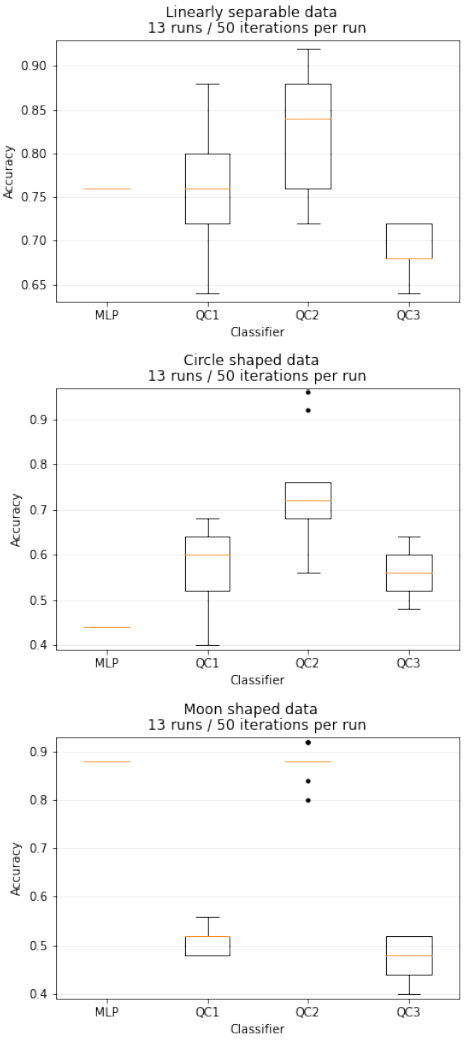
(C) **Quantum Circuit classifier 3** same as the two qubit classifier in figure 5.8c with 2 additional "layers"

FIGURE 5.12: Two qubit quantum circuits with 3 layers referring to the **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figures 5.13a, 5.14a and 5.15a. For better visual understanding three barriers have been added, marking the beginning of each layer. The barriers have no impact on the circuits themselves and are only used for visualization purposes.

FIGURE 5.13: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1, Quantum Circuit classifier 2 and Quantum Circuit classifier 3.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data

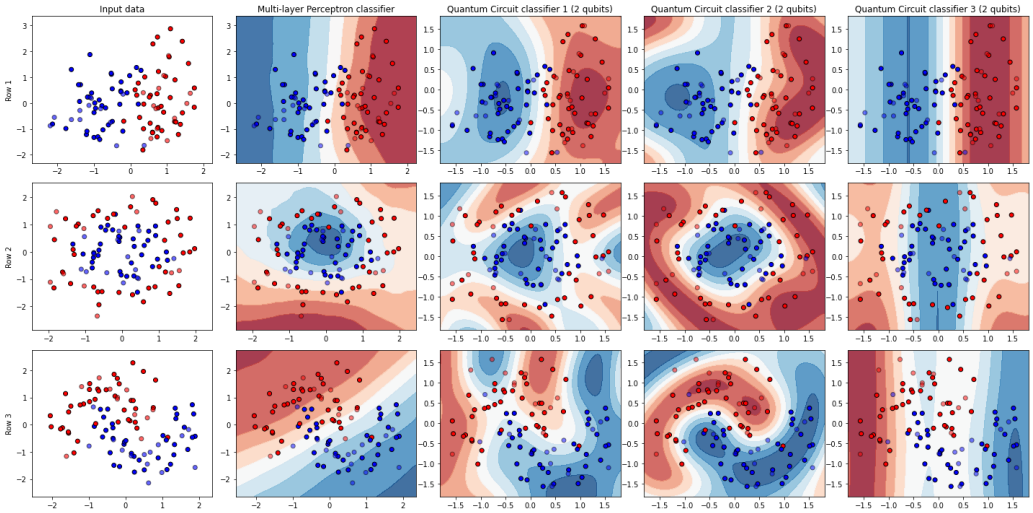


(A) **50 iterations** for all classifiers using **two qubit quantum circuits with 3 layers**.

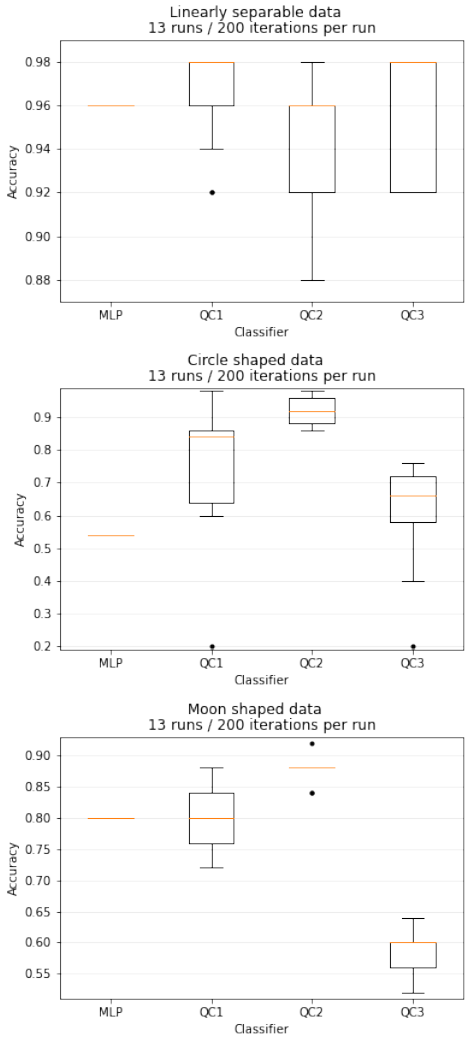


(B) Accuracy box plots: **50 iterations** for all classifiers using **two qubit quantum circuits with 3 layers**. **MLP**, **QC1**, **QC2** and **QC3** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figure 5.13a accordingly.

FIGURE 5.14: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1, Quantum Circuit classifier 2 and Quantum Circuit classifier 3.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data

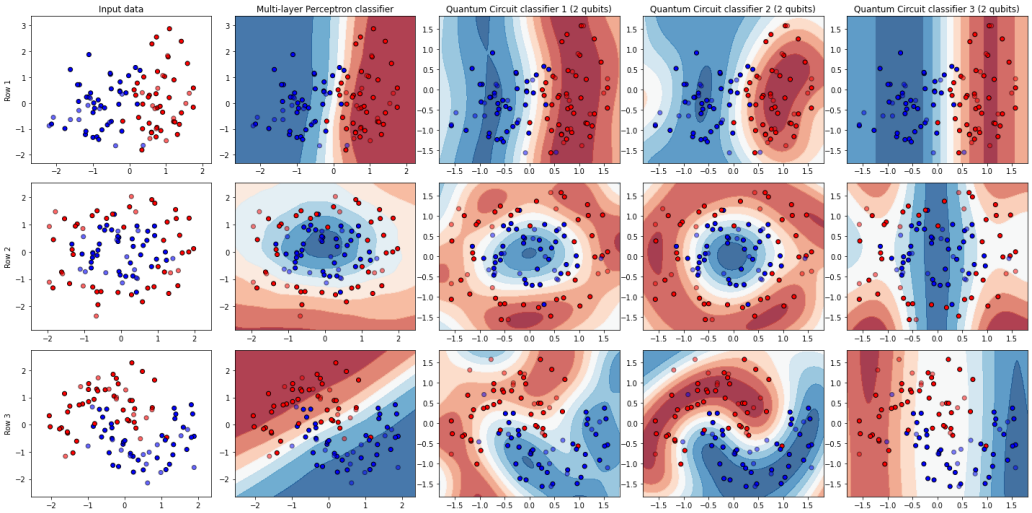


(A) **200 iterations** for all classifiers using **two qubit quantum circuits with 3 layers**.

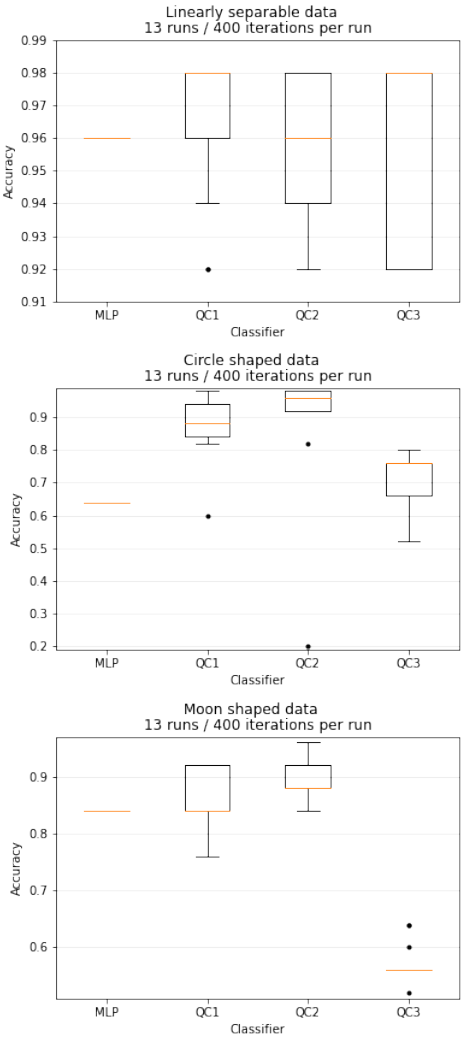


(B) Accuracy box plots: **200 iterations** for all classifiers using **two qubit quantum circuits with 3 layers**.
MLP, **QC1**, **QC2** and **QC3** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figure 5.14a accordingly.

FIGURE 5.15: Decision boundary and accuracy comparison between Multi-layer Perceptron classifier, Quantum Circuit classifier 1, Quantum Circuit classifier 2 and Quantum Circuit classifier 3.
Row 1: Linearly separable data / *Row 2*: Circle shaped data / *Row 3*: Moon shaped data



(A) **400 iterations** for all classifiers using **two qubit quantum circuits with 3 layers**.



(B) Accuracy box plots: **400 iterations** for all classifiers using **two qubit quantum circuits with 3 layers**.
MLP, **QC1**, **QC2** and **QC3** are referring to **Multi-layer Perceptron classifier**, **Quantum Circuit classifier 1**, **Quantum Circuit classifier 2** and **Quantum Circuit classifier 3** in figure 5.15a accordingly.

5.1 Discussion

5.1.1 Single qubit classifiers

Examining the single qubit classifier plots and comparing the quantum classifiers decision boundaries, it becomes obvious that the replacing of the RY gate in *Quantum Circuit classifier 1* (5.4a) with a RX gate in *Quantum Circuit classifier 2* (5.4b) for the feature x_1 enhanced the expressiveness of the *Quantum Circuit classifier 2* in a mostly beneficial way, resulting in a higher score over nearly all iteration counts.

Another interesting observation is that the *Quantum Circuit classifier 1* only consisting of RY and RZ gates can adapt the weights to more and more resemble the data, mostly increasing the score with higher iterations, but achieving this much slower than *Quantum Circuit classifier 2*.

Comparing the linearly separable data (*Row 1* of figure 5.5a) between the *Multi-layer Perceptron classifier* and *Quantum Circuit classifier 2* with only 50 iterations, the *Quantum Circuit classifier 2* outperforms the *Multi-layer Perceptron classifier*, but the *Multi-layer Perceptron classifier* can keep up again and over perform with more iterations as seen in figures 5.6a and 5.7a, at least for the linearly separable data.

By comparing the contour plots of figure 5.7a, one can see that all the quantum circuit classifiers behave almost like the *Multi-layer Perceptron classifier*. The accuracies in figure 5.7b are also very close, except for *Quantum Circuit classifier 1* with the moon shaped dataset in *Row 3*.

One of the most important observations, however, is that the quantum circuit classifiers cannot produce smaller angles in all the contour plots, indicating that either the gates used are incorrect or their number is insufficient. Although smaller angles are needed to achieve a higher score when data points start to overlap, this can also have a negative impact on generalization like overfitting.

5.1.2 Two qubit classifiers

Interestingly, the accuracy for the two qubit classifiers (figures 5.9b, 5.10b, 5.11b) is not better than the single qubit classifiers (figures 5.5b, 5.6b, 5.7b) even though two more quantum gates were used e.g. the CX and the Hadamard (H) see circuits 5.8a, 5.8b, 5.8c. For the circular (*Row 2*) and moon shaped (*Row 2*) datasets the accuracy is even worse compared to the single qubit classifiers, which is also evident in the contour plots because these do not show a significantly higher expression strength. Only the linear data set has a similarly high and slightly more stable accuracy.

As with the single qubit classifiers, comparing the linearly separable data (*Row 1* of figure 5.13a) between the *Multi-layer Perceptron classifier* and all of the two qubit classifiers with only 50 iterations, all two qubit classifiers outperform the *Multi-layer Perceptron classifier*, but the *Multi-layer Perceptron classifier* can keep up again and slightly over perform with more iterations as seen in figures 5.10a and 5.15a.

Quantum Circuit classifier 3 can only perform a linear classification and is very limited in its expressiveness, as the contour plots (figures 5.9a, 5.10a, 5.11a) indicate. The accuracy for the non-linear separable data sets (figures 5.9b, 5.10b, 5.11b) is correspondingly poor.

The *Multi-layer Perceptron classifier* leaves all two qubit classifiers behind regarding accuracy for the moon shaped data sets for all iteration counts (figures 5.13b, 5.14b, 5.15b).

5.1.3 Two qubit classifiers with 3 layers

When inspecting the contour plots in figures 5.13a, 5.14a and 5.15a, it is obvious that the contours can now take on much smaller angles or fields and the expressiveness has increased a lot even with a few iterations.

The accuracy advantage over the *Multi-layer Perceptron classifier* at 50 iterations - observed with the single qubit 5.1.1 and two qubit classifiers 5.1.2 - has significantly decreased for the linearly separable dataset *Row 1*, which makes sense since now a lot more weights have to be trained and the features are additionally encoded twice resulting in more complex computation of the quantum circuit (see figures 5.13b, 5.14b, 5.15b).

Quantum Circuit classifier 3 behaves no longer only "linearly" and can improve with higher iteration numbers its expression strength, even if only slowly. *Quantum Circuit classifier 1* and *Quantum Circuit classifier 2* show promising contours and accuracies with three layers starting at 100 iterations (see figures 5.13a, 5.14a, 5.15a). All three circuits tend to achieve higher accuracy than the *Multi-layer Perceptron classifier* in the circular and moon-shaped data sets starting at 100 iterations, except for *Quantum Circuit classifier 3* on the moon-shaped data set (see figures 5.13b, 5.14b, 5.15b).

5.2 Conclusion

The experiments with different quantum classifier circuits using contour and box plots in comparison with a classical *Multi-layer Perceptron classifier* showed that small gates changes - for example a rotation around another axis - can result in a very different decision boundary improving/decreasing the accuracy or can lead to slower/faster adaption of the decision boundary when trained.

Not only can single qubit classifiers perform well on certain data sets, they can also be more accurate than two qubit classifiers, depending on the data set complexity and the gates used. A big advantage are the quantum classifier circuits with multiple layers that have more weights and gates and thus a larger expression strength with smaller angles and fields that can adapt much better to more difficult datasets and this at an iteration count where the accuracy of the *Multi-layer Perceptron classifier* is not nearly sufficient.

We cannot say how well the quantum classifier circuits used in the above experiments (5.0.1, 5.0.2 and 5.0.3) would perform in a real quantum computer given the accuracy, since we could only use quantum computer simulators due to time constraints. The accuracy of the different runs with the optimal simulator still varies enormously, in contrast to the *Multi-layer Perceptron classifier*. This leads to the fact that runs have to be performed several times because an above-average bad/good run can occur more often.

It certainly needs more experiments and mathematical analysis of the quantum classifier circuits to be more precise. However, we advise using the contour plots as an additional tool to visually estimate the expression strength and adaptation to the data set for the used quantum circuit in the context of the training.

Chapter 6

Quantum Classification Circuits

6.1 Quantum circuits

Using the definitions from chapter 3 and 4, several quantum circuits are constructed for classification. `qiskit` offers a flexible approach for the gradient-based optimization of weights in a given circuit, as well as fine control of all components in the circuit and around the optimization. Based on the paper by Schuld et al. [39], we can optimize any given quantum circuit with a `NeuralNetworkClassifier`[40]. The code to turn a single circuit into a classifier is shown in figure 6.1. Note that the optimization itself is done in a classical fashion - only the classification is done with a quantum circuit. All calculations and classifications were done with the 'aer_simulator' from `qiskit`. The `NeuralNetworkClassifier` with `CircuitQNN` uses a measurement of all given qubits and the parity function 6.2 to determine the label, where n is the amount of different labels in the dataset.

```
circuit_qnn = CircuitQNN(circuit=circuit,
                         input_params=inputs,
                         weight_params=weights,
                         interpret=parity,
                         output_shape=output_shape,
                         quantum_instance=quantum_instance)

nn_classifier = NeuralNetworkClassifier(neural_network=circuit_qnn,
                                        optimizer=COBYLA())
```

FIGURE 6.1: Python code to create a trainable neural network classifier from a quantum circuit

```
def parity(x):
    return '{:b}'.format(x).count('1') % n
```

FIGURE 6.2: Parity function to assign label after classification

Figure 6.3 is an overview of the training process. The features of a given dataset are preprocessed and then parameterized onto the state preparation part of the circuit. The weights are set from the optimizer, which also adapts them during training.

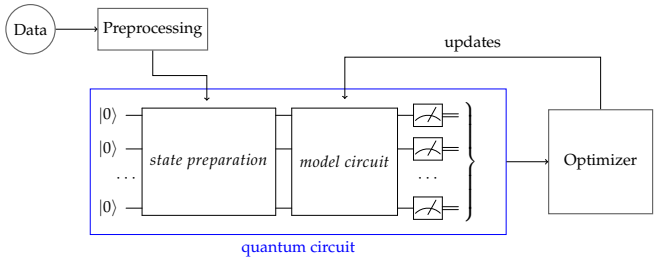


FIGURE 6.3: Schematic view of classical and quantum circuit

Figures 6.4 until 6.11 show the designs of the circuits used for the evaluation.

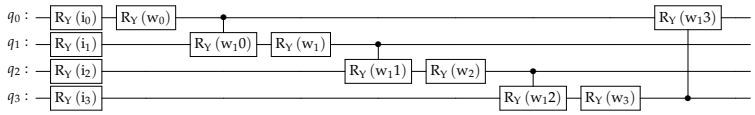


FIGURE 6.4: Circuit 1 with 4 qubits, RY gates before each entanglement, and entanglement with parameterized CRY gates

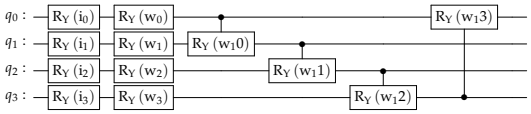


FIGURE 6.5: Circuit 2 with 4 qubits, RY for input and weights before entanglement, and entanglement with parameterized CRY gates

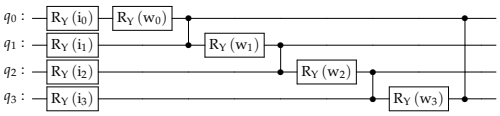


FIGURE 6.6: Circuit 3 with 4 qubits, RY gates before each entanglement, and entanglement with parameterized CZ gates

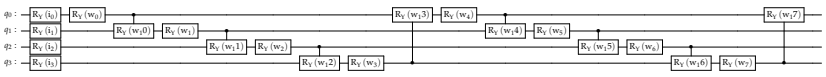


FIGURE 6.7: Circuit 4, which equals figure 6.4, with repeated entanglement process.

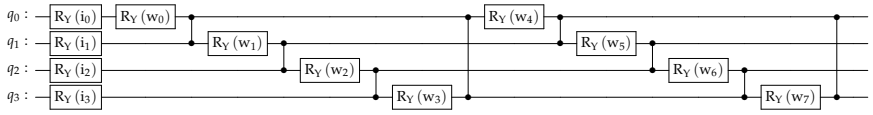


FIGURE 6.8: Circuit 5, which equals figure 6.5, with repeated entanglement process.

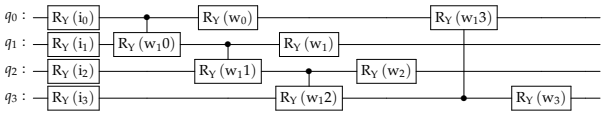


FIGURE 6.9: Circuit 6 with 4 qubits, RY gates after each entanglement, and entanglement with parameterized CRY gates

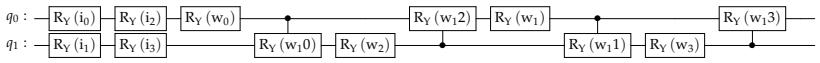


FIGURE 6.10: Circuit 7 with 2 qubits, which represents a minified version of circuit 6.4

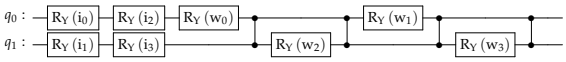


FIGURE 6.11: Circuit 8 with 2 qubits, which represents a minified version of circuit 6.6

6.2 Datasets

The circuits shown in chapter 6.1 are used as shown or adapted for the given number of features and classes of the testing dataset. The datasets are used with three different forms of normalization. The first one uses no preprocessing, that is, the features are parameterized as is. The second one normalizes all features to the range $[-1, 1]$, and the last one to $[-\pi, \pi]$. As seen in chapters 3 and 2.2, π corresponds to a 180 rotation, so should offer some additional expressibility as well as the normalization to $[-1, 1]$, albeit by a smaller factor. The data collected on each circuit was done trough 10 training iteration, where each was started with an empty set of weights. Each measurement in the simulator of `qiskit` was done with 1024 shots, and the data split was 25% testing and 75% training. Total training was done twice - once with random shuffle of the data between each training iteration, and once without. The seeds were all set for any randomized function, *except* the measurement simulator from `qiskit`. This because any measurement in real life cannot be predicted through a random seed. This is supposed to allow better evaluation and understanding of the training behaviour of a quantum classifier.

6.2.1 Iris

The IRIS [41] dataset is one of the most well-known datasets with plenty of available solutions. This allows it to be a good example for general comparison to classical classification methods. All the features were used in the circuit after passing through the normalization step. It contains 3 different classes, each with 4 discrete features, amounting to 150 entries.

6.2.2 Heart Failure Prediction

This dataset [42] was selected due to several reasons. It is biased towards the *survivor* classification with roughly 68% of the entries being of class 0, and 32% for class 1, as well as having a total of 299 entries. There exist several high accuracy offerings when it comes to classifying the complete dataset that can be used for comparison. The one picked for comparison[43] achieves a maximal accuracy of 93.33%. To make a direct comparison possible, the same 3 features out of 12 are used[1].

6.2.3 Artificial Problem

To examine the expressibility of the given circuits, the artificial problem from Havlicek et al. [4] offers a problem space that is inherently hard to solve through classical methods. Havlicek et al. also purpose a circuit that can solve the given problem with perfect accuracy, which offers itself as a good comparison to the capabilities the created circuits offer. It consists of 2 classes with 2 features, amounting to a total of 250 entries. The problem space is dynamically generated and can be adapted to one's desired setup, as shown in figure 6.12. A visual representation of the problem space is shown in figure 6.13.

[1] In this dataset, `time` refers to the time during study of the patients and data collection, so a direct correlation between death and time exists, as a given time can typically be attributed to death or no death. For comparison's sake, this error in feature selection was neglected, as we are only interested in achievable accuracy

```
adhoc_dimension = 2
train_features, train_labels, test_features, test_labels, adhoc_total = ad_hoc_data(
    training_size=188,
    test_size=67,
    n=adhoc_dimension,
    gap=0.3,
    plot_data=False, one_hot=False, include_sample_total=True)
```

FIGURE 6.12: Python code used to create an artificial problem space for classification

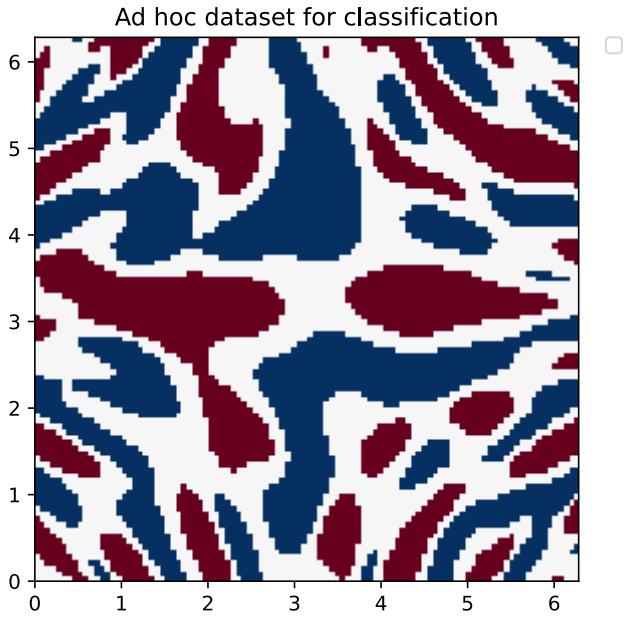


FIGURE 6.13: Visualization of the classified problem space, where x and y axis are the features, and red/blue the corresponding class

6.2.4 Classification On Quantum Hardware

Circuit 6.4 is used in the trained state and evaluated 5 times on a real quantum computer. Due to access to real hardware being behind a queue system, the workload was split across multiple real devices, which are `ibmq_belem` and `ibmq_lima`. `qiskit` is used for this as it offers easy access to the hardware as well as selection of the least used system. Note that the trained weights defined in equation 6.1 are static in the circuit and therefore don't need to be parameterized.

$$\begin{aligned} w = \ & \{0.5551258, -0.07536535, 1.5141565, 1.08361588, \\ & -0.2558132, -0.57076258, 1.80514739, 1.46449782\} \end{aligned} \tag{6.1}$$

```
shots = 1024
IBMQ.load_account()
amount_of_runs = 5
runs = []
for i in range(amount_of_runs):
    results = []
    for features in x_test:
        _, backend = use_least_busy_real_device()
        qc = circuit.copy()
        qc = qc.bind_parameters(features)
        job_exp = execute(qc, backend, shots=shots)
        job_monitor(job_exp)
        results.append(job_exp.result().get_counts(qc))
    runs.append(results)
```

FIGURE 6.14: Python code used to run trained classification circuit on a real quantum computer

Chapter 7

Results

The results in this chapter are in condensed form. Please refer to appendix A for more detailed figures. The following figures display the achieved accuracy on the y axis, and the number of the circuit on the x axis, and are limited to only the achieved testing accuracy. For the randomized segments, 10 training runs had its own data shuffling, which are equal for every circuit. Additionally, the IRIS dataset is also classified on a basic MLP classifier.

7.1 Non Randomized IRIS

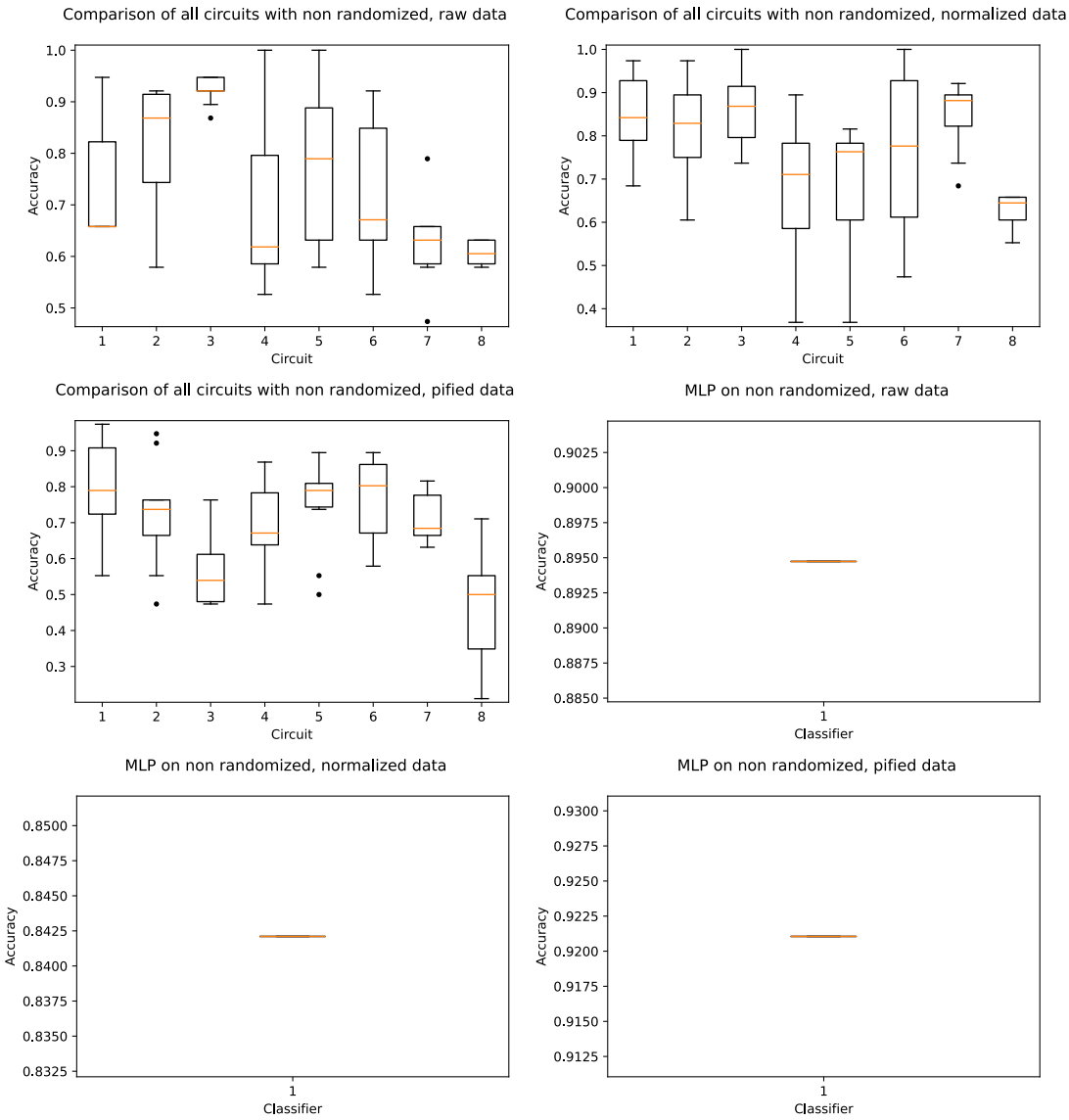


FIGURE 7.1: Comparison of test results from all trained circuits on the non-randomized IRIS dataset

7.2 Randomized IRIS

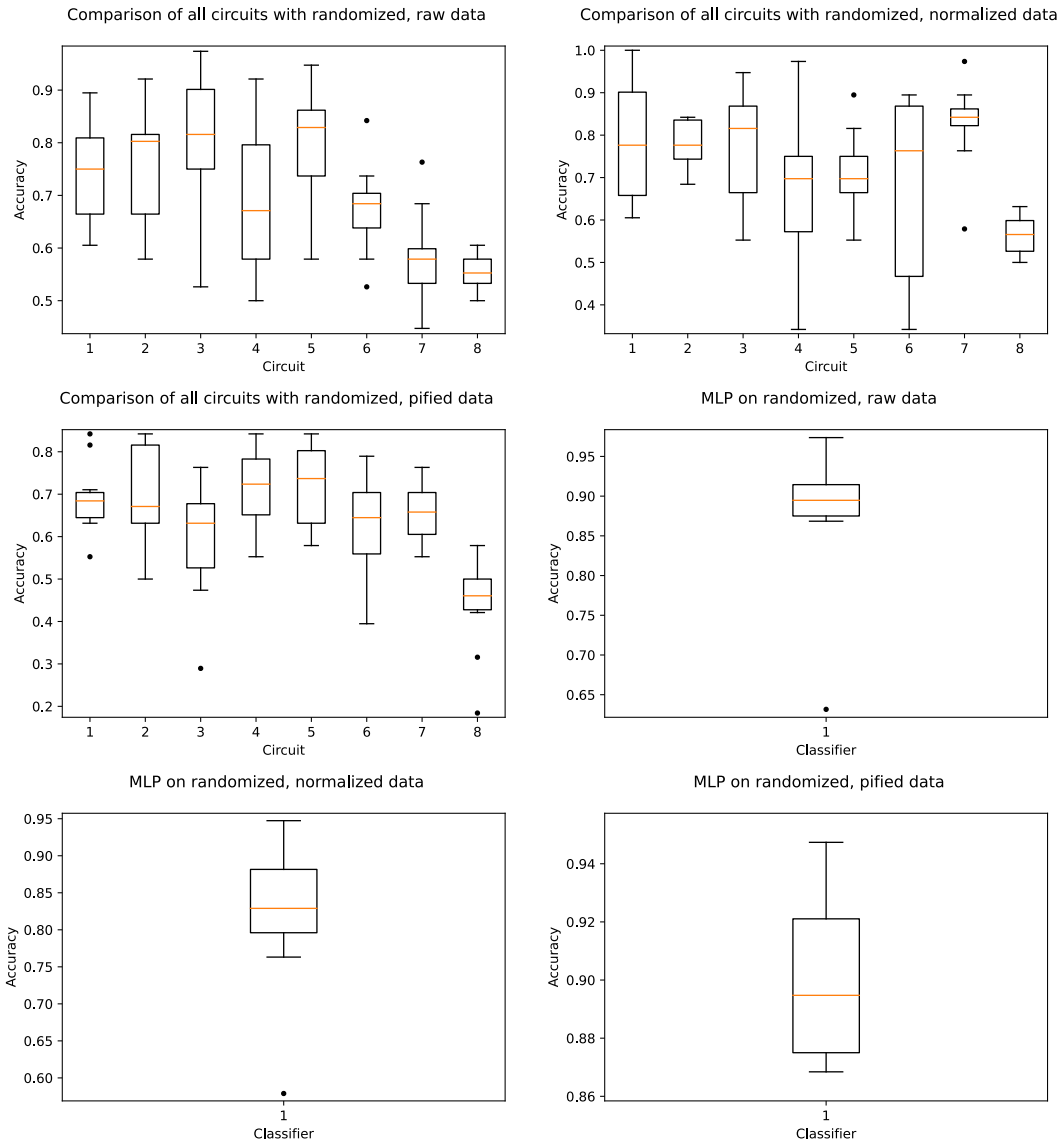


FIGURE 7.2: Comparison of test results from all trained circuits on the randomized IRIS dataset

7.2.1 Classification Of IRIS On Quantum Hardware

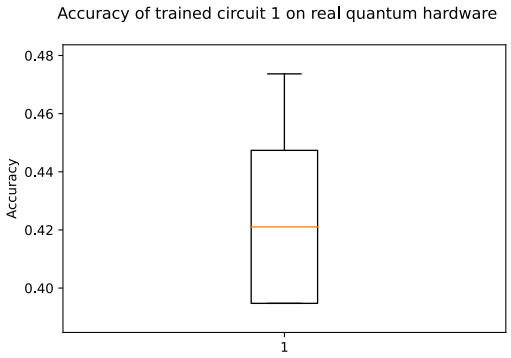


FIGURE 7.3: Achieved accuracy of circuit 1 on real quantum hardware, on the randomized IRIS dataset

7.3 Non Randomized Heart Failure Prediction

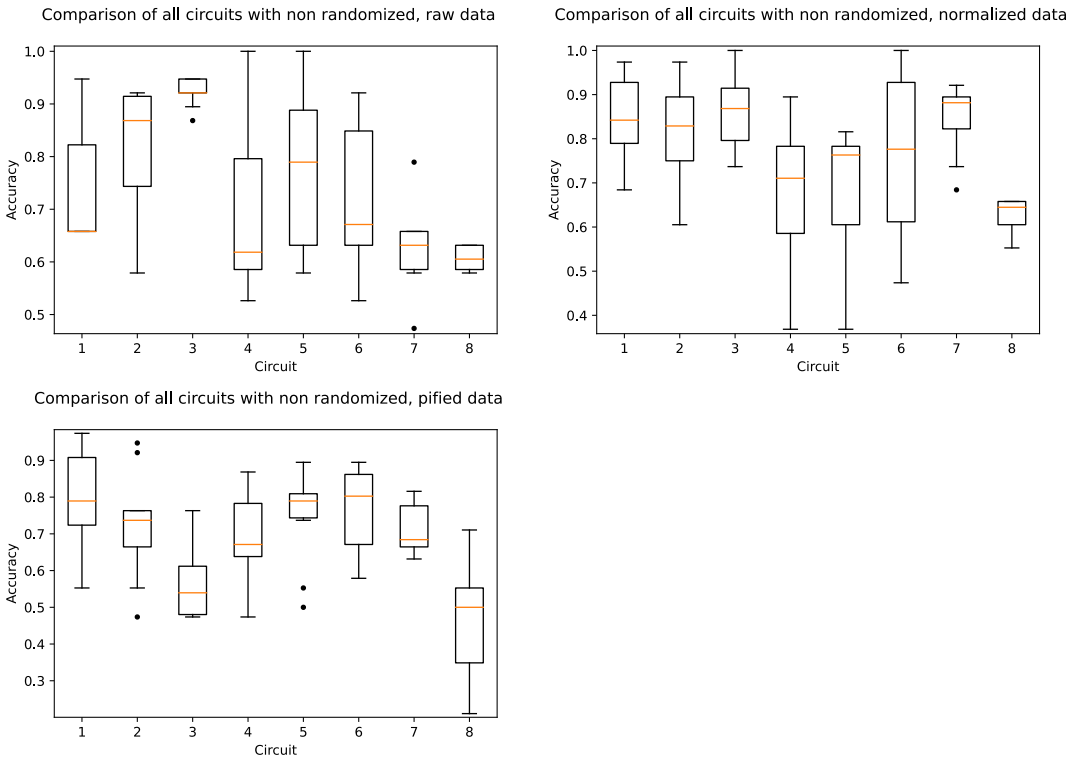


FIGURE 7.4: Comparison of test results from all trained circuits on the non-randomized Heart Failure dataset

7.4 Randomized Heart Failure Prediction

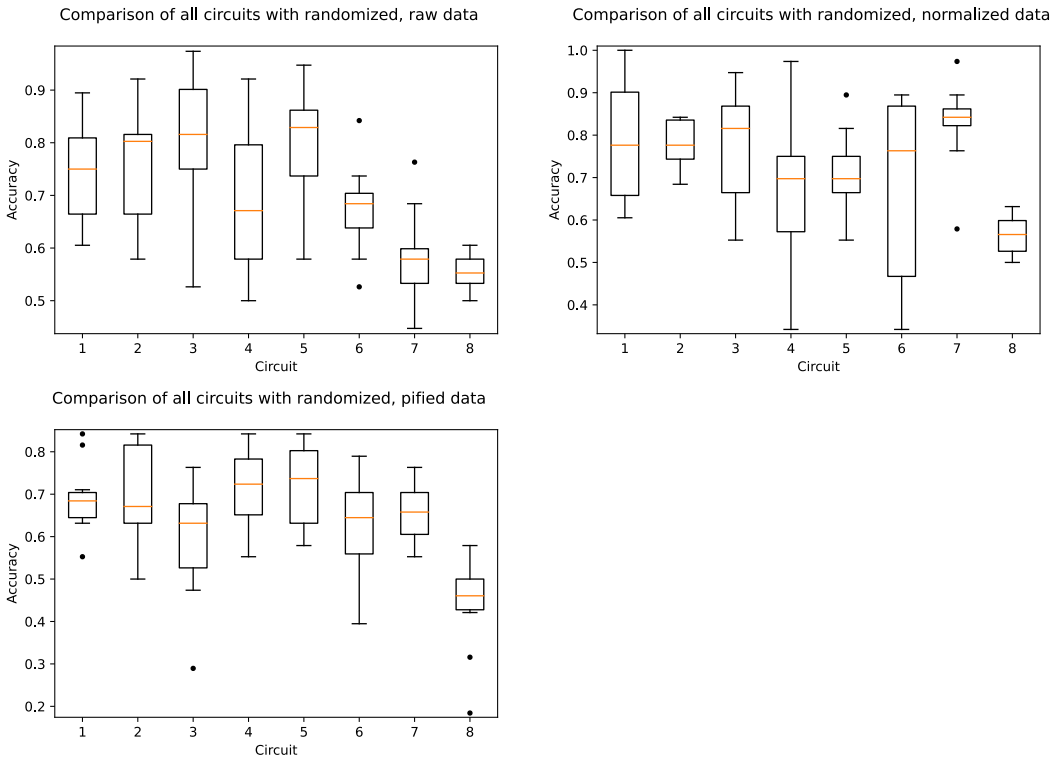


FIGURE 7.5: Comparison of test results from all trained circuits on the randomized Hearth Failure dataset

7.5 Non-randomized Artificial Problem

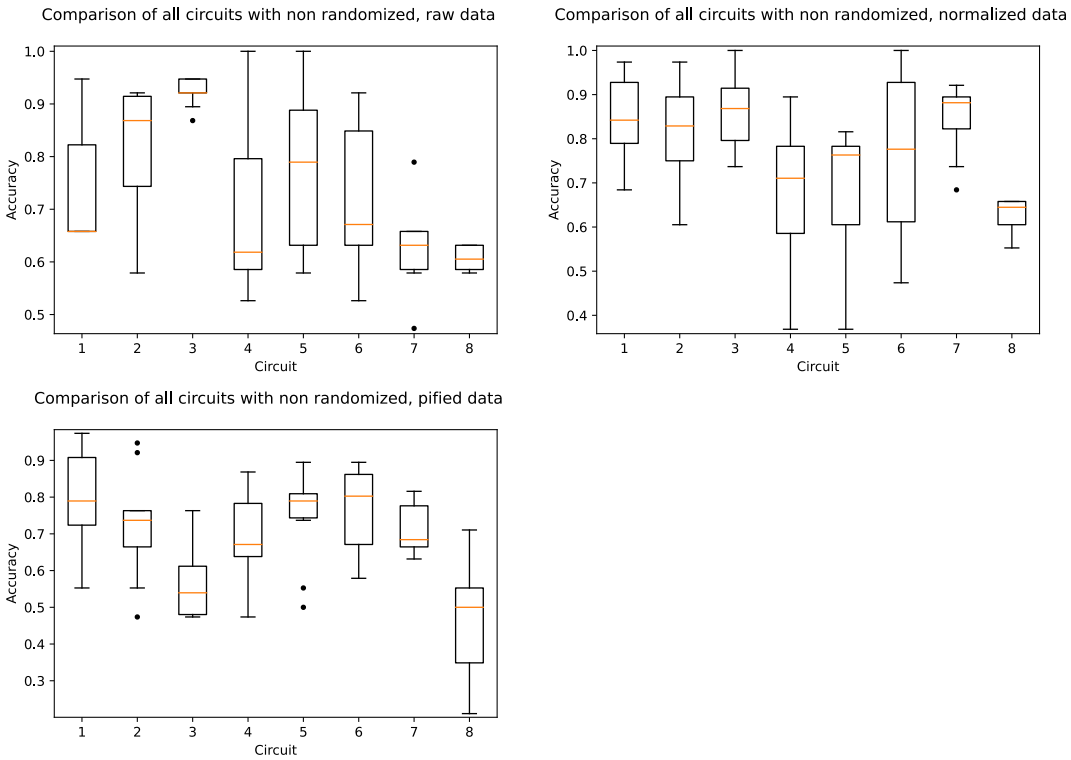


FIGURE 7.6: Comparison of test results from all trained circuits on the non-randomized Artificial Problem dataset

7.6 Randomized Artificial Problem

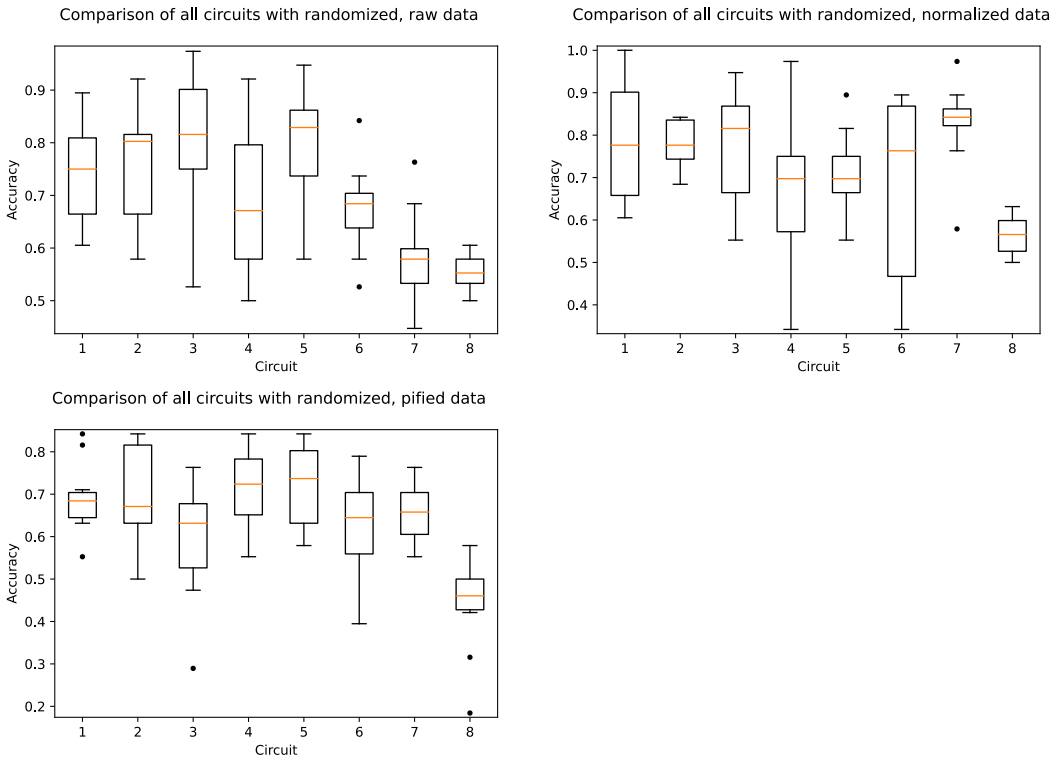


FIGURE 7.7: Comparison of test results from all trained circuits on the randomized Artificial Problem dataset

7.7 Discussion

The results when training is done with static training/test data shuffle show that due to the nature of probabilistic measurements, one cannot *naturally* expect to achieve the same accuracy every time training is done. As stated in chapter 6.2, the seed for the random measurements the simulator does on a quantum circuit is not fixated, as this cannot be done on real hardware.

The collected data shows that even a usable circuit that *can* achieve perfect testing accuracy, as seen in figure 7.7, will often end up below that. When compared to the circuit used by Havlicek et al.[4], which was adapted to be trained with the same methods, it achieved near perfect *or* perfect accuracy from the beginning of the training (on the artificial problem). This leads to the assumption, that some circuits might be easier to adapt to the problem space, and therefore, easier to learn, but cannot guarantee that a given circuit will always return the same results. This is also observable in the Heart Failure results when compared to the referred solution. Whilst cases exist, where it beats the comparison, most achieved scores are below the 0.933 of said solution.

Due to the limited availability of real hardware to perform long training sessions on, the training is done on the simulator with reduced amount of measurement shots (which are summed up to show the probabilities) of 1024. It can be argued that this is not enough to reach converging probabilities. When comparing to Havlicek et al. solution, it is clear that a circuit, which is more refined to the problem at hand, *will* achieve better resulting accuracy, faster and with less variance, than general purpose circuits. It is not yet clear if such a circuit can be dynamically built for every problem space, or if they have to be carefully designed.

All figures in chapter 7 display an overall increase in achieved accuracy from the performed normalization, which scaled down the feature space to the range $[-1, 1]$. At the same time, it shows that there is a net loss of achieved accuracy when using a normalization with the range $[-\pi, \pi]$ on IRIS. An explanation is that any rotations that go beyond the point of $|0\rangle$ and $|1\rangle$ start to *decrease* the probability of the given state again, even though initially, it was increasing it, as elaborated on in the chapter 3. When looking at the range of achieved accuracy for IRIS, the net gain for a normalization on $[-1, 1]$ is in the range of $[0.020, 0.028]$ and for the normalization on range $[-\pi, \pi]$ it was a net loss of $[-0.059, -0.054]$, when compared to the raw data scores. The same comparison on the heart failure data leads to a net gain of $[0.087, 0.229]$ for $[-1, 1]$ and a net gain of $[0.070, 0.131]$ for $[-\pi, \pi]$.

Comparing the raw features of the IRIS dataset to the ones of the Heart Failure dataset, the *initial* range of feature values is larger for the Heart Failure dataset. Therefore, normalizing IRIS to $[-\pi, \pi]$ increases the total range of single features, which makes the feature space more *sparse*, whilst in the Heart Failure dataset it decreases the range of single features, making it more dense. This explains why the classification of IRIS has a net loss and classifying Heart Failure a net gain when using a normalization of $[-\pi, \pi]$.

Collected data suggests that given an enough complex quantum circuit, it is possible to achieve excellent classification results for any given dataset. This can be seen in figures 7.1 up to 7.7, where circuit 4 managed to achieve good results across all datasets. This is supported by the findings of Sim et al.[44], which show that combinations of gates can make use of a wider range of possible states, therefore making them more capable of trespassing the problem space. Observing the results of the classification on real quantum hardware in figure 7.3, the measured accuracy is 0.55 *less* than the one achieved in the simulator. There are several potential reasons for

this. Initially, some variance is introduced from the setup through the limited number of shots. In addition, execution on real quantum hardware is always prone to noise which can falsify states, and therefore, lead to wrong measurements. This is an active problem that is still being tackled[45, 46], so future improvements in reliability of measurements are sure to increase classification scores.

A comparison of time complexity between our circuits and classical neural networks is difficult to create. Not only the lack of reliable references for this, but also flexibility of not only neural networks, but other machine learning algorithms too. These are commonly adapted to their classification problems, which makes a generic assumption of $O(x)$ unreliable. In addition, hardware with features that speed up neural networks[47] add to the unreliability of such a comparison.

7.8 Conclusion

The results show that it is indeed possible to create a quantum circuit that has the ability to classify a wide range of datasets. They also show that even though they offer high adaptability to multiple problem spaces, the probabilistic measurements and that versatility can backfire whilst training circuits with the goal of classifying datasets. The implication is that there is a requirement for multiple evaluations under the same conditions before a circuit's capabilities can be assessed. Another takeaway from the conducted research is that limiting the expressiveness of the circuit with rotation gates that have fixed values attributed to them *might* lead to less variance, as the optimizer cannot meddle with those values.

Chapter 8

Directory

List of Figures

List of Tables

Bibliography

[1] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/S0097539795293172. arXiv: quant-ph/9508027. URL: http://arxiv.org/abs/quant-ph/9508027 (visited on 12/20/2021).

[2] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A Quantum Approximate Optimization Algorithm". In: *arXiv:1411.4028 [quant-ph]* (Nov. 14, 2014). arXiv: 1411.4028. URL: http://arxiv.org/abs/1411.4028 (visited on 12/20/2021).

[3] Tobias Fankhauser et al. "Multiple Query Optimization using a Hybrid Approach of Classical and Quantum Computing". In: *arXiv:2107.10508 [quant-ph]* (July 22, 2021). arXiv: 2107.10508. URL: http://arxiv.org/abs/2107.10508 (visited on 11/26/2021).

[4] 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. arXiv: 1804.11326. URL: http://arxiv.org/abs/1804.11326 (visited on 12/18/2021).

[5] OpenAI. *AI and Compute*. OpenAI. May 16, 2018. URL: https://openai.com/blog/ai-and-compute/ (visited on 12/20/2021).

[6] F. Bloch. "Nuclear Induction". In: *Phys. Rev.* 70 (7-8 Oct. 1946), pp. 460–474. DOI: 10.1103/PhysRev.70.460. URL: https://link.aps.org/doi/10.1103/PhysRev.70.460.

[7] Qiskit Development Team. *XGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.XGate.html#qiskit.circuit.library.XGate (visited on 11/23/2021).

[8] Qiskit Development Team. *HGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.HGate.html (visited on 11/28/2021).

[9] Qiskit Development Team. *RYGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.RYGate.html (visited on 11/28/2021).

[10] Ahlfors Lars. *Complex analysis*. 3rd ed. International Series in Pure and Applied Mathematics. New York: McGraw-Hill Book Co., 1978. ISBN: 0-07-000657-1.

[11] Richard P. (Richard Phillips) Feynman 1918-1988. "The Feynman Lectures on Physics Vol. III Ch. 12: The Hyperfine Splitting in Hydrogen". 1965. URL: https://www.feynmanlectures.caltech.edu/III_12.html.

[12] Qiskit Development Team. *CYGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.CYGate.html (visited on 12/05/2021).

[13] Qiskit Development Team. *CXGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.CXGate.html (visited on 12/05/2021).

[14] Qiskit Development Team. *CZGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.CZGate.html (visited on 12/05/2021).

[15] Pease III Marshall C. *Methods of Matrix Algebra*. Academic Press, Jan. 1, 1964. 405 pp. ISBN: 978-0-08-095522-3. URL: https://www.elsevier.com/books/methods-of-matrix-algebra/pease/978-0-12-548850-1.

[16] Seth Lloyd et al. "Quantum embeddings for machine learning". In: (2020). arXiv: 2001.03622 [quant-ph].

[17] Maria Schuld and Nathan Killoran. "Quantum Machine Learning in Feature Hilbert Spaces". In: *Physical Review Letters* 122.4 (Feb. 2019). ISSN: 1079-7114. DOI: 10.1103/physrevlett.122.040504. URL: http://dx.doi.org/10.1103/PhysRevLett.122.040504.

[18] Vojtěch Havlíček et al. "Supervised learning with quantum-enhanced feature spaces". In: *Nature* 567.7747 (Mar. 2019), pp. 209–212. ISSN: 1476-4687. DOI: 10.1038/s41586-019-0980-2. URL: http://dx.doi.org/10.1038/s41586-019-0980-2.

[19] *Quantum embedding — PennyLane*. URL: https://pennylane.ai/qml/glossary/quantum_embedding.html (visited on 12/03/2021).

[20] Xanadu. *Quantum Feature Map — PennyLane*. URL: https://pennylane.ai/qml/glossary/quantum_feature_map.html (visited on 12/18/2021).

[21] Maria Schuld. *Supervised quantum machine learning models are kernel methods*. 2021. arXiv: 2101.11020 [quant-ph].

[22] Frank Leymann. *Towards a Pattern Language for Quantum Algorithms*. 2019. arXiv: 1906.03082 [quant-ph].

[23] Manuela Weigold et al. "Expanding Data Encoding Patterns For Quantum Algorithms". In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2021, pp. 95–101. DOI: 10.1109/ICSA-C52384.2021.00025.

[24] Frank Leymann and Johanna Barzen. "The Bitter Truth about Gate-Based Quantum Algorithms in the NISQ Era". In: *Quantum Science and Technology* 5.4 (Oct. 2020), p. 044007. ISSN: 2058-9565. DOI: 10.1088/2058-9565/abae7d.

[25] Israel F. Araujo et al. "A Divide-and-Conquer Algorithm for Quantum State Preparation". In: *Scientific Reports* 11.1 (Mar. 2021), p. 6329. ISSN: 2045-2322. DOI: 10.1038/s41598-021-85474-1.

[26] Manuela Weigold et al. "Encoding patterns for quantum algorithms". In: *IET Quantum Communication* 2.4 (2021), pp. 141–152. DOI: https://doi.org/10.1049/qtc2.12032. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/qtc2.12032. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/qtc2.12032.

[27] Daniel Sierra-Sosa et al. "Diabetes Type 2: Poincar Data Preprocessing for Quantum Machine Learning". In: *Computers, Materials and Continua* 67 (Feb. 2021), pp. 1849–1861. DOI: 10.32604/cmc.2021.013196.

[28] *Variational classifier — PennyLane*. URL: https://pennylane.ai/qml/demos/tutorial_variational_classifier.html (visited on 12/12/2021).

[29] Prakhar Shrivastava, Kapil Kumar Soni, and Akhtar Rasool. "Classical Equivalent Quantum Unsupervised Learning Algorithms". In: *Procedia Computer Science* 167 (2020), pp. 1849–1860. ISSN: 1877-0509. DOI: 10.1016/j.procs.2020.03.204.

[30] scikit-learn developers. *sklearn.preprocessing.MinMaxScaler*. scikit-learn. URL: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (visited on 12/20/2021).

[31] scikit-learn developers. *sklearn.preprocessing.StandardScaler*. scikit-learn. URL: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (visited on 12/20/2021).

[32] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958). Place: US Publisher: American Psychological Association, pp. 386–408. ISSN: 1939-1471. DOI: 10.1037/h0042519.

[33] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[34] Tomasz Szandała. "Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks". In: *arXiv:2010.09458 [cs]* 903 (2021). DOI: 10.1007/978-981-15-5495-7. arXiv: 2010.09458. URL: http://arxiv.org/abs/2010.09458 (visited on 11/23/2021).

[35] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. In collab. with Library Genesis. Cambridge ; New York : Cambridge University Press, 2010. 710 pp. ISBN: 978-1-107-00217-3. URL: http://archive.org/details/quantumcomputati00niel_993 (visited on 11/23/2021).

[36] Wolfgang Scherer. "Quantum Gates and Circuits for Elementary Calculations". In: *Mathematics of Quantum Computing: An Introduction*. Ed. by Wolfgang Scherer. Cham: Springer International Publishing, 2019, pp. 161–246. ISBN: 978-3-030-12358-1. DOI: 10.1007/978-3-030-12358-1_5. URL: https://doi.org/10.1007/978-3-030-12358-1_5 (visited on 12/20/2021).

[37] Qiskit Development Team. *CCXGate — Qiskit 0.32.1 documentation*. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.CCXGate.html (visited on 11/23/2021).

[38] IBM. *IBM Quantum*. IBM Quantum. URL: https://quantum-computing.ibm.com/services (visited on 12/26/2021).

[39] Maria Schuld et al. "Evaluating analytic gradients on quantum hardware". In: *Physical Review A* 99.3 (Mar. 21, 2019), p. 032331. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.99.032331. arXiv: 1811.11184. URL: http://arxiv.org/abs/1811.11184 (visited on 12/11/2021).

[40] Qiskit Development Team. *Neural Network Classifier & Regressor — Qiskit Machine Learning 0.2.1 documentation*. URL: https://qiskit.org/documentation/machine-learning/tutorials/02_neural_network_classifier_and_regressor.html (visited on 12/11/2021).

[41] R. A. Fisher. "The Use of Multiple Measurements in Taxonomic Problems". In: *Annals of Eugenics* 7.2 (1936). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-1809.1936.tb02137.x, pp. 179–188. ISSN: 2050-1439. DOI: 10.1111/j.1469-1809.1936.tb02137.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x (visited on 12/14/2021).

[42] Tanvir Ahmad et al. "Survival analysis of heart failure patients: A case study". In: *PLOS ONE* 12.7 (July 20, 2017). Publisher: Public Library of Science, e0181001. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0181001. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0181001 (visited on 12/18/2021).

[43] Nayan Sakhiya. *Heart Fail:Analysis and Quick-prediction*. URL: https://kaggle.com/nayansakhiya/heart-fail-analysis-and-quick-prediction (visited on 12/18/2021).

[44] Sukin Sim, Peter D. Johnson, and Alan Aspuru-Guzik. "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms". In: *Advanced Quantum Technologies* 2.12 (Dec. 2019), p. 1900070. ISSN: 2511-9044, 2511-9044. DOI: 10.1002/qute.201900070. arXiv: 1905.10876. URL: http://arxiv.org/abs/1905.10876 (visited on 12/18/2021).

[45] Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. "Modelling and Simulating the Noisy Behaviour of Near-term Quantum Computers". In: *Physical Review A* 104.6 (Dec. 17, 2021), p. 062432. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.104.062432. arXiv: 2101.02109. URL: http://arxiv.org/abs/2101.02109 (visited on 12/23/2021).

[46] Ali Shaib et al. "Efficient Noise Mitigation Technique for Quantum Computing". In: *arXiv:2109.05136 [quant-ph]* (Sept. 10, 2021). arXiv: 2109.05136. URL: http://arxiv.org/abs/2109.05136 (visited on 12/23/2021).

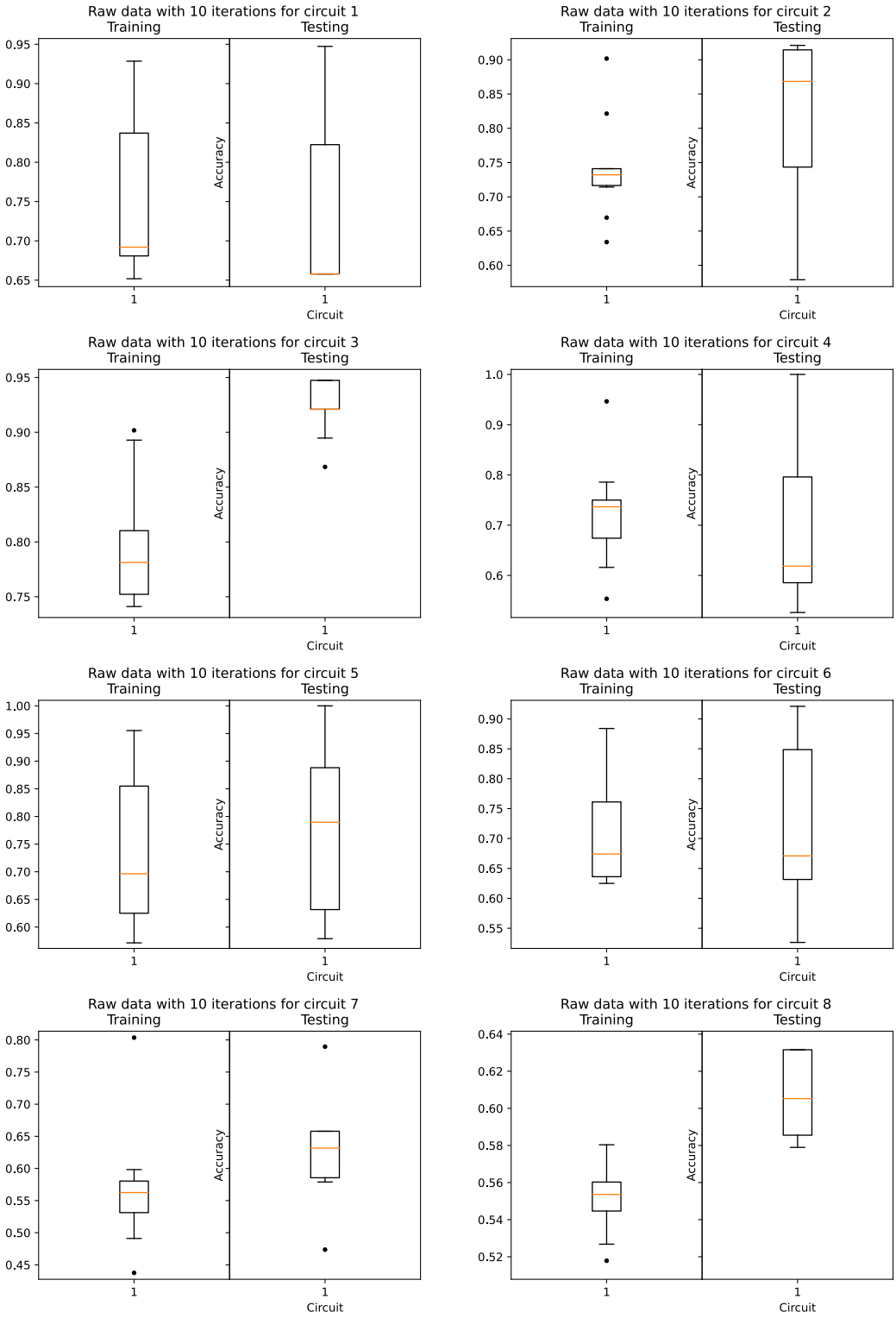
[47] Lukas Baischer, Matthias Wess, and Nima TaheriNejad. "Learning on Hardware: A Tutorial on Neural Network Accelerators and Co-Processors". In: *arXiv:2104.09252 [cs]* (Apr. 19, 2021). arXiv: 2104.09252. URL: http://arxiv.org/abs/2104.09252 (visited on 12/23/2021).

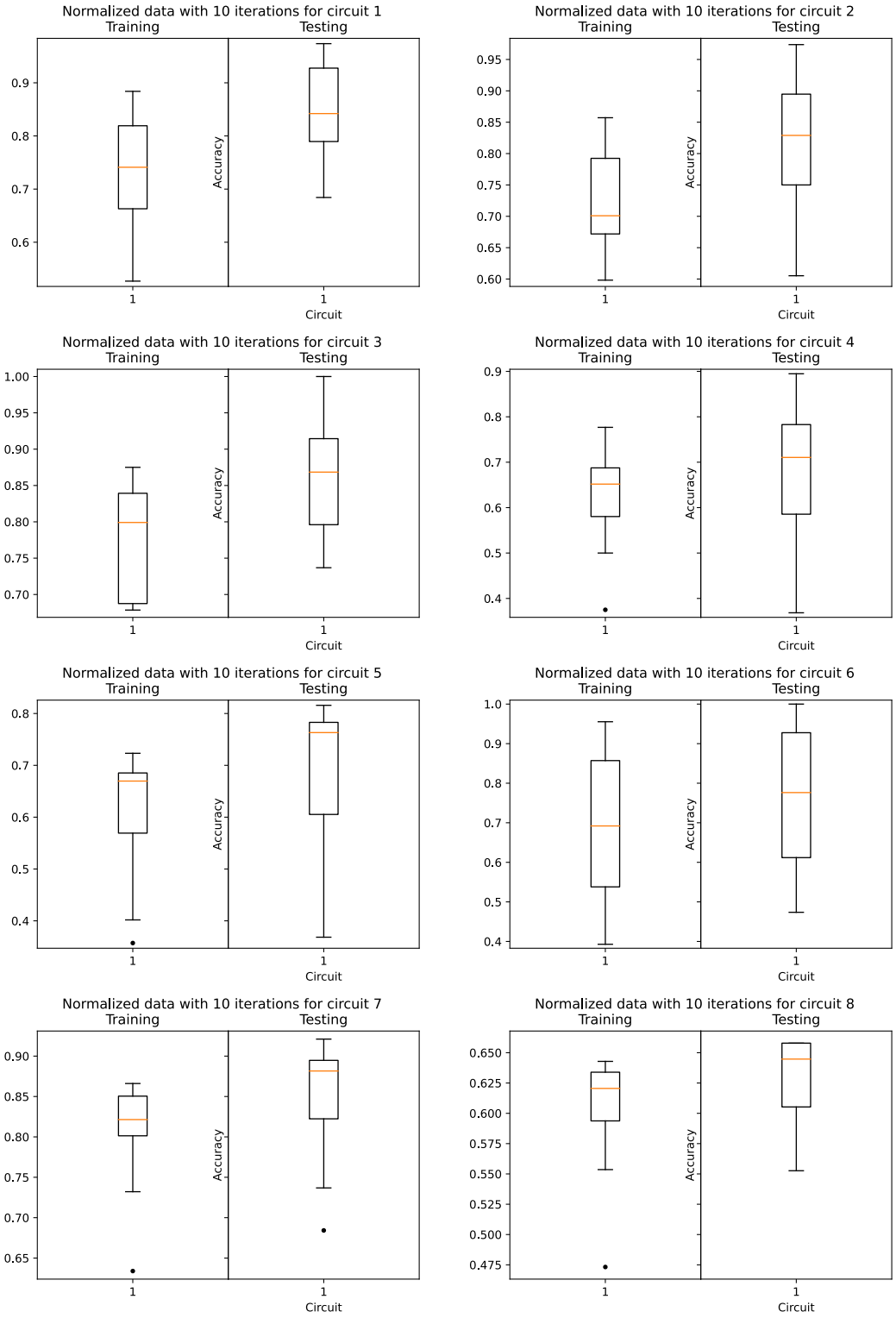
Appendix A

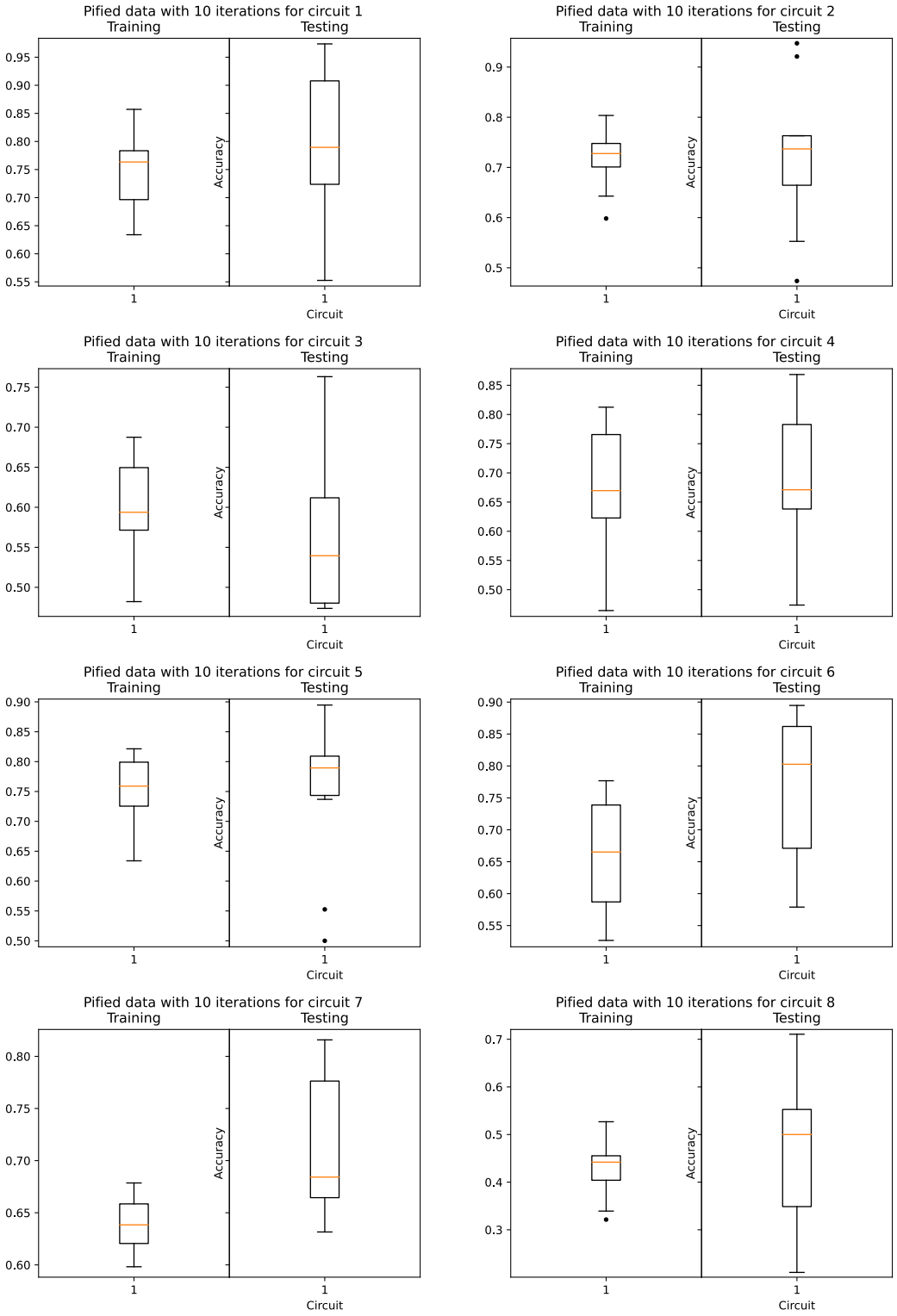
Detailed view of data collected for trained circuits

A.1 IRIS

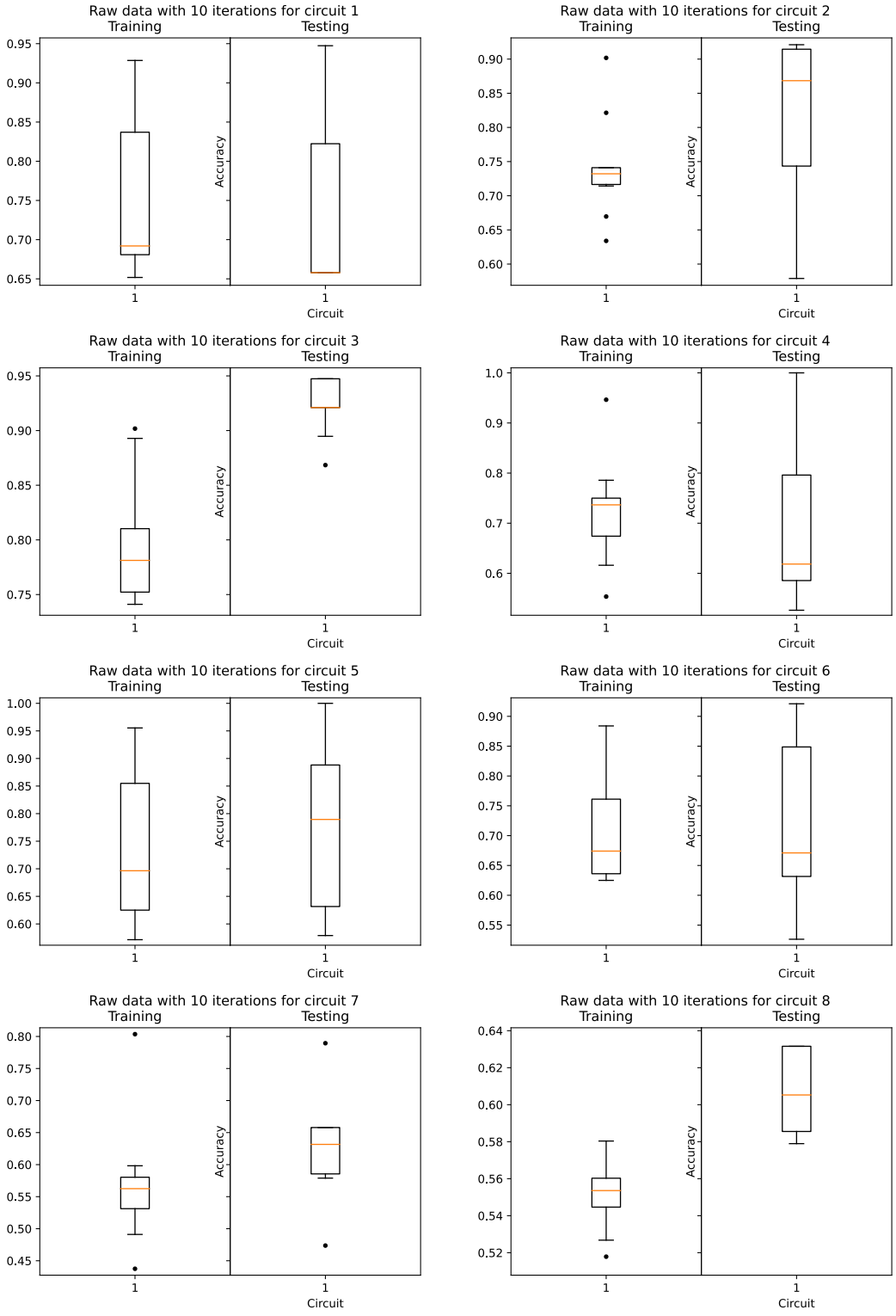
A.1.1 Non-randomized test/train data

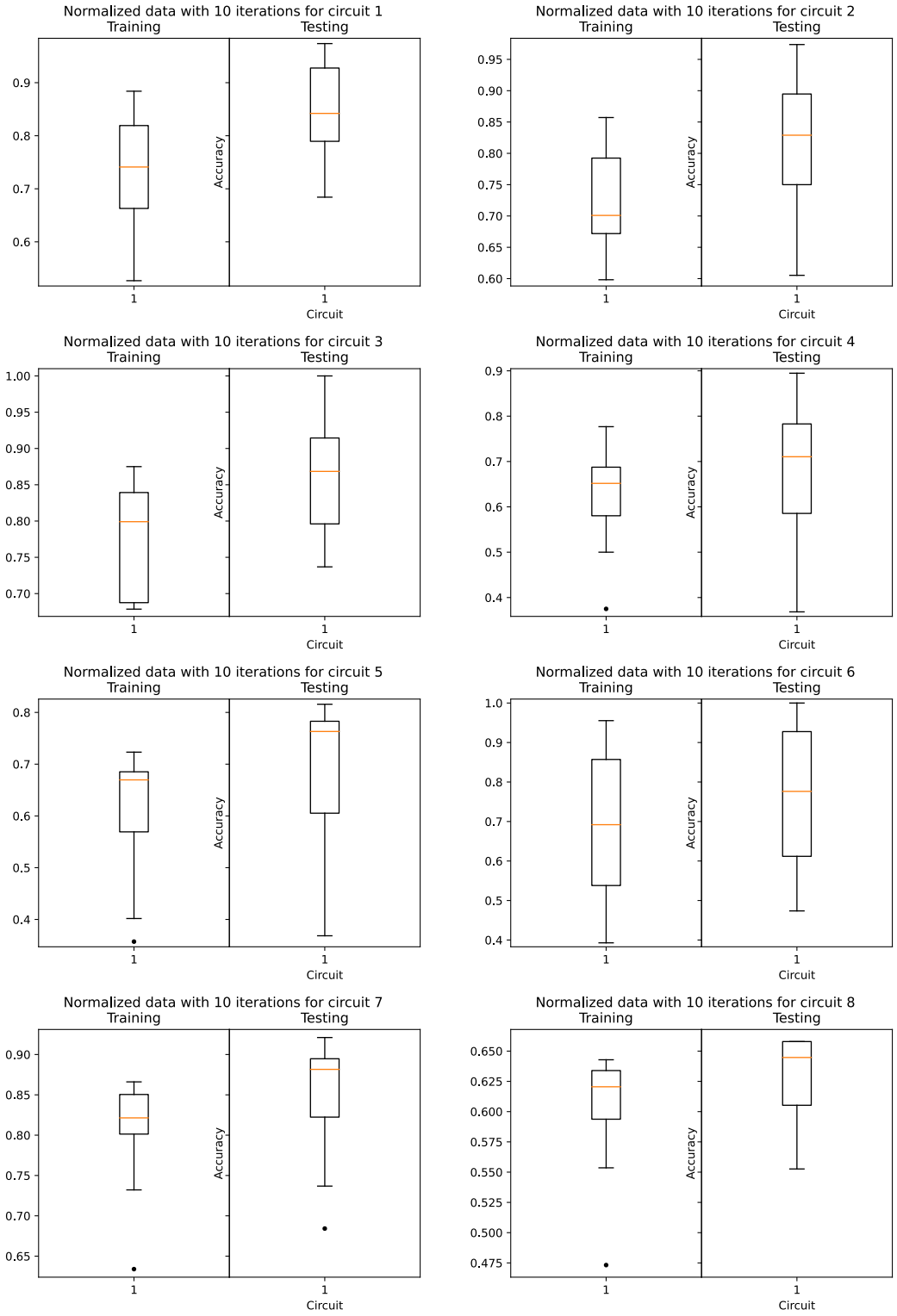


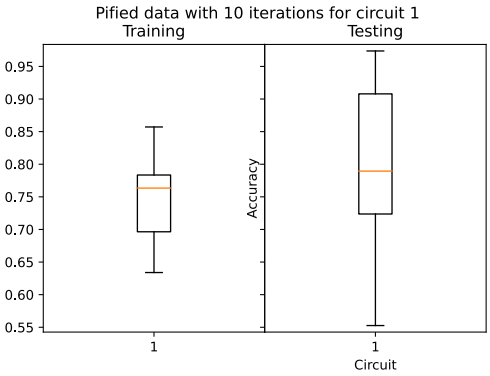

Normalized data with 10 iterations for circuit 1
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 2
Normalized data with 10 iterations for circuit 3
Normalized data with 10 iterations for circuit 4
Normalized data with 10 iterations for circuit 5
Normalized data with 10 iterations for circuit 6
Normalized data with 10 iterations for circuit 7
Normalized data with 10 iterations for circuit 8

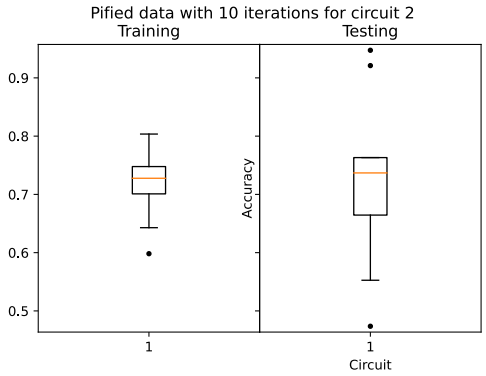

Pified data with 10 iterations for circuit 1
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 2
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 3
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 4
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 5
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 6
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 7
Training
Testing
Accuracy
Circuit
Pified data with 10 iterations for circuit 8
Training
Testing
Accuracy
Circuit

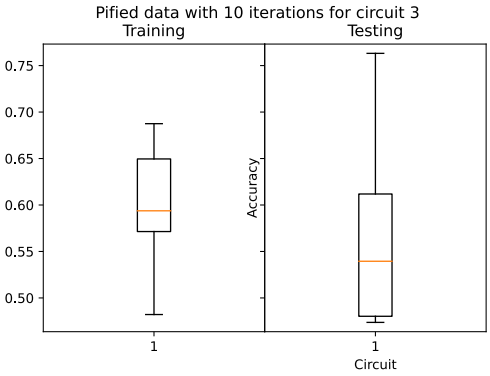
A.1.2 Randomized test/train data

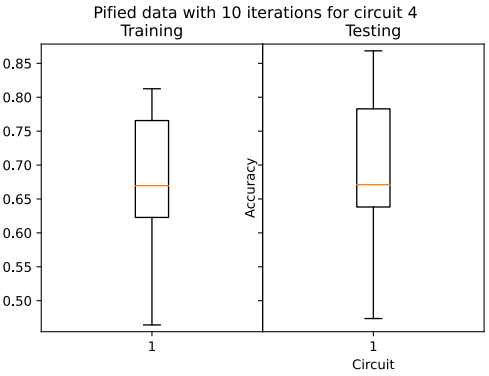


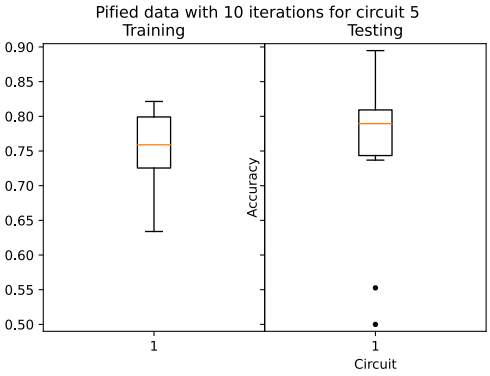

Normalized data with 10 iterations for circuit 1
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 2
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 3
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 4
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 5
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 6
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 7
Training
Testing
Accuracy
Circuit
Normalized data with 10 iterations for circuit 8
Training
Testing
Accuracy
Circuit

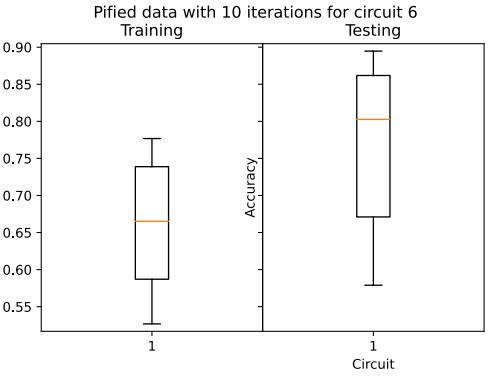

Pified data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

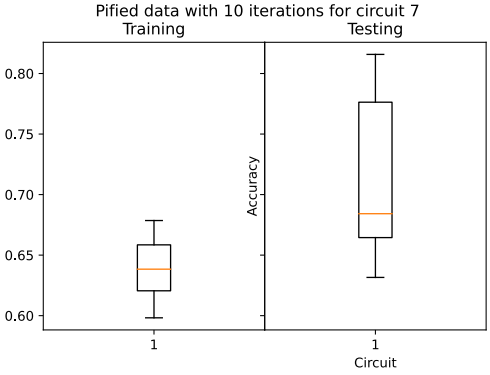

Pified data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
1
1
Circuit

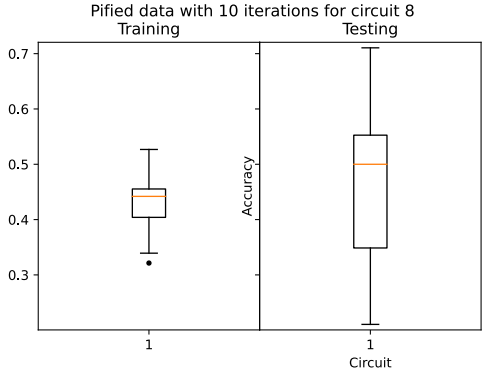

Pified data with 10 iterations for circuit 3
Training
Testing
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit


Pified data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit


Pified data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

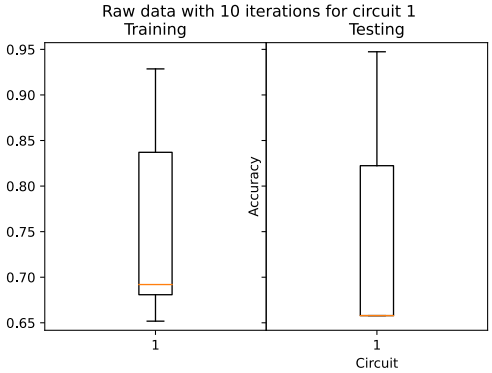

Pified data with 10 iterations for circuit 6
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

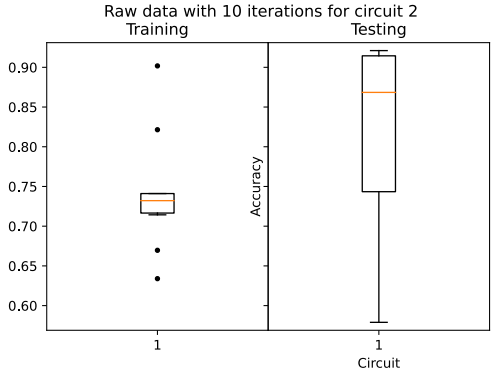

Pified data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

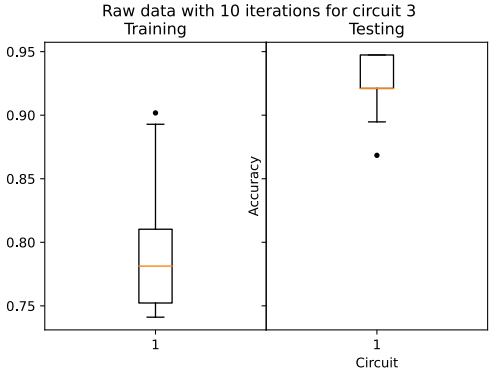

Pified data with 10 iterations for circuit 8
Training
Testing
Accuracy
0.7
0.6
0.5
0.4
0.3
1
1
Circuit

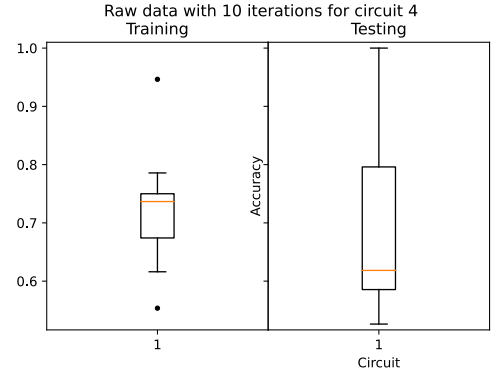
A.2 Heart Failure

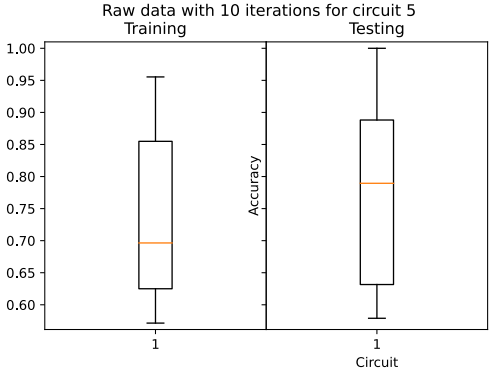
A.2.1 Non-randomized test/train data

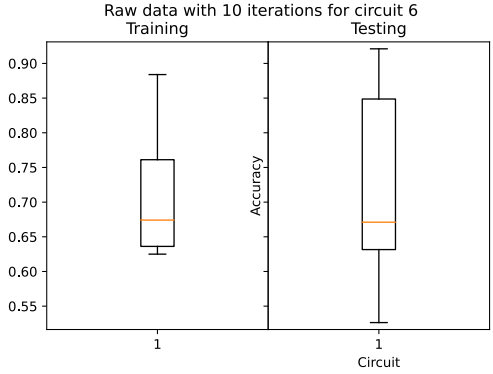


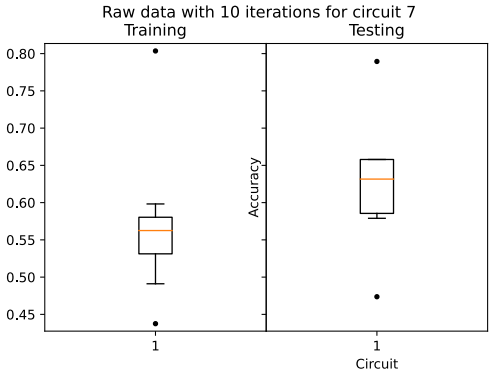


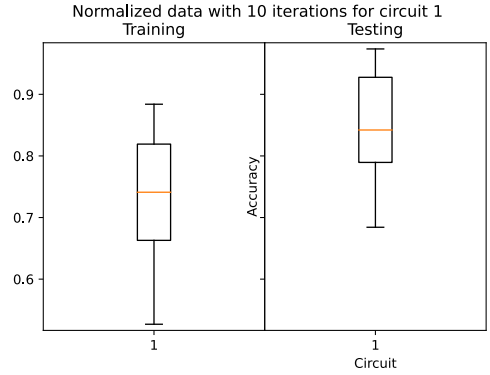


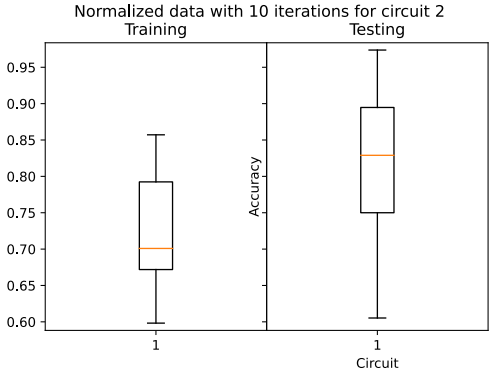


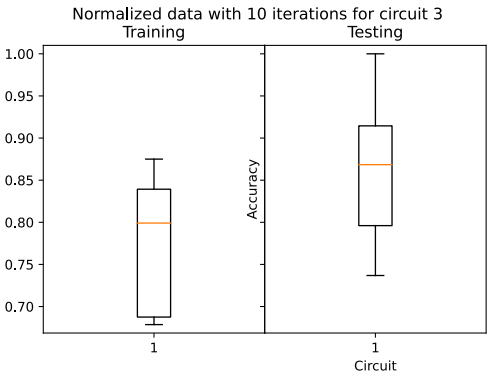


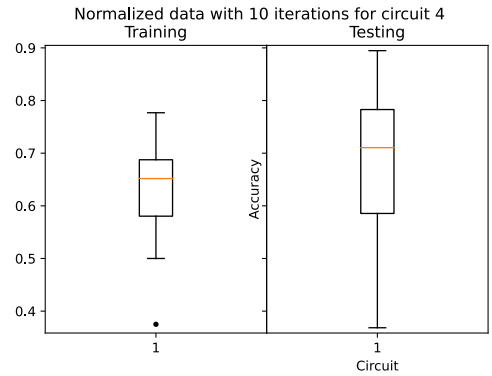


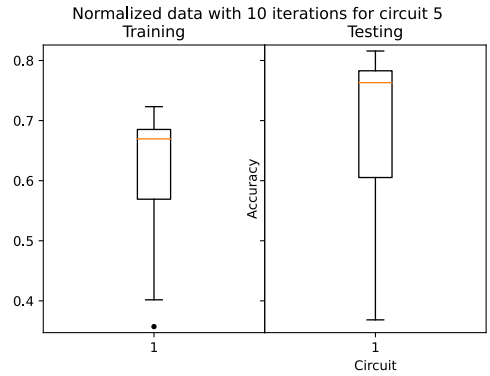


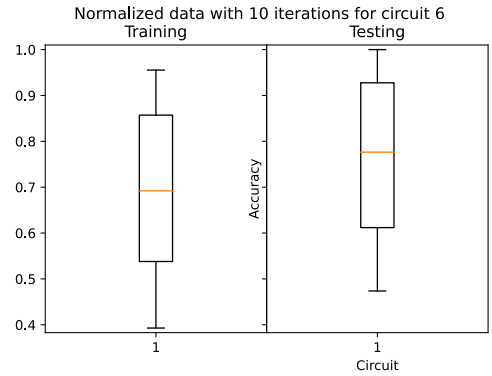

Normalized data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
1
1
Circuit

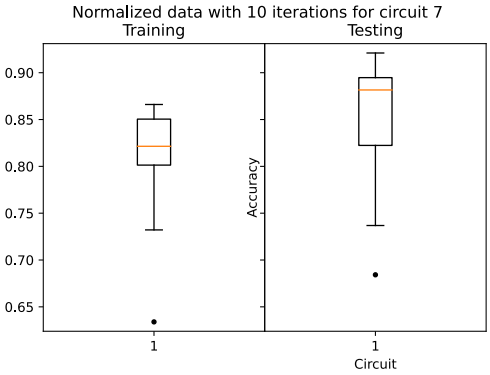

Normalized data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

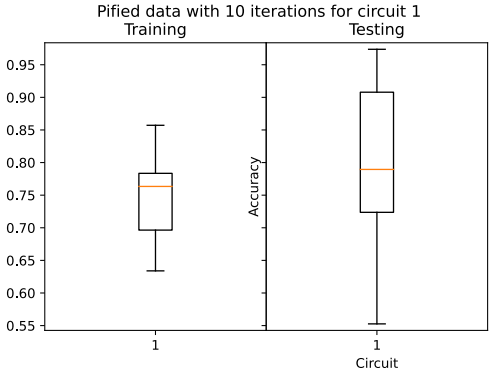

Normalized data with 10 iterations for circuit 3
Training
Testing
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
1
1
Circuit

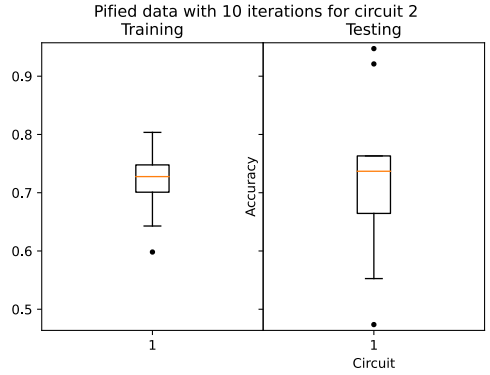

Normalized data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

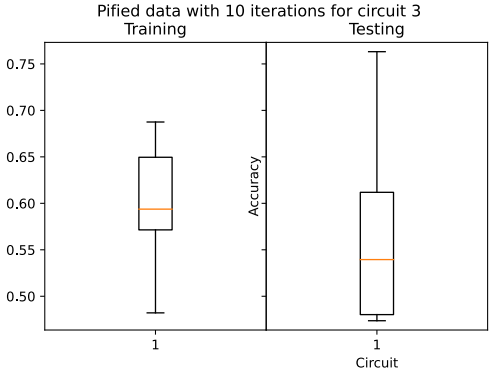

Normalized data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

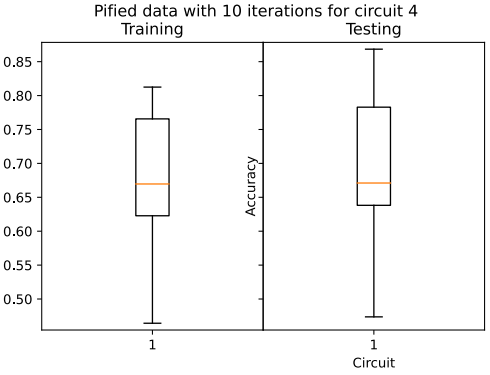

Normalized data with 10 iterations for circuit 6
Training
Testing
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

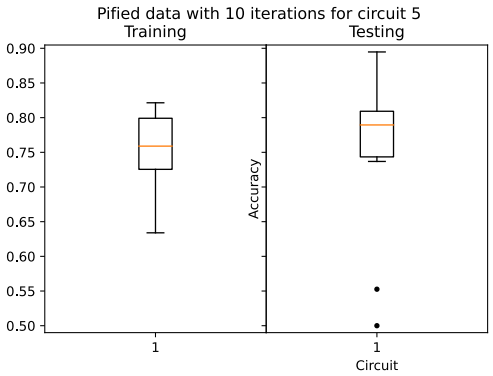

Normalized data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
1
1
Circuit

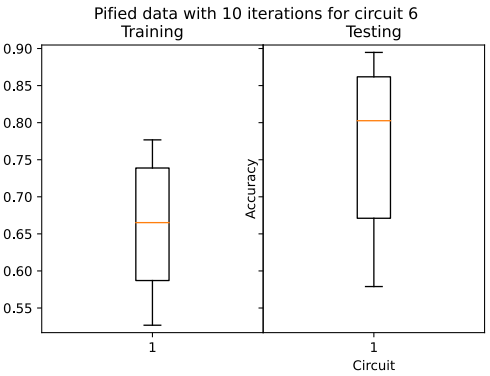

Pified data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

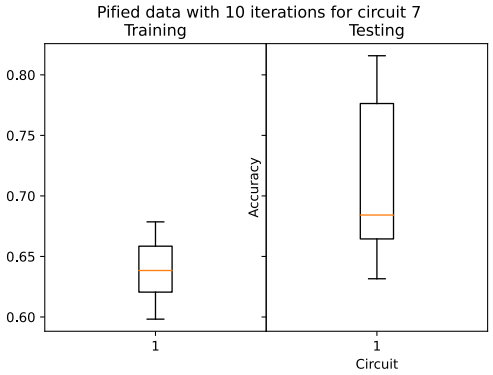

Pified data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
1
1
Circuit


Pified data with 10 iterations for circuit 3
Training
Testing
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

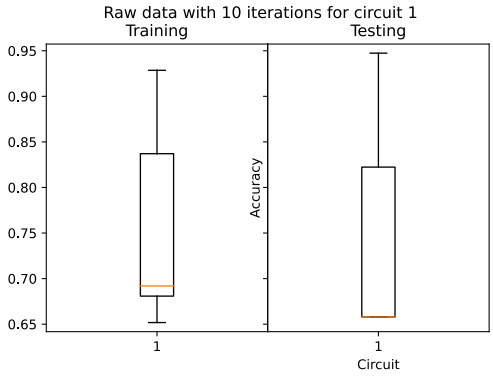

Pified data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

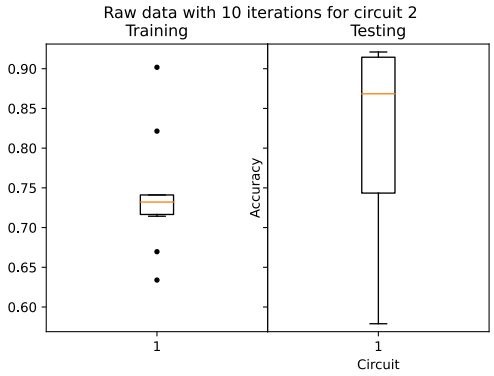

Pified data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

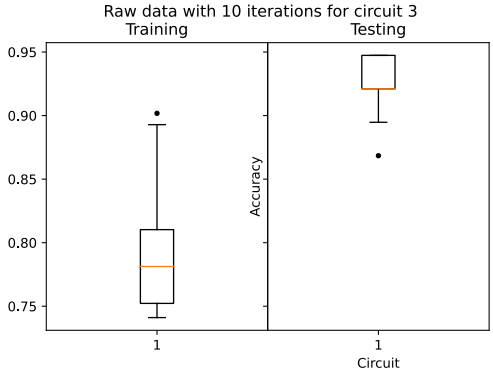

Pified data with 10 iterations for circuit 6
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

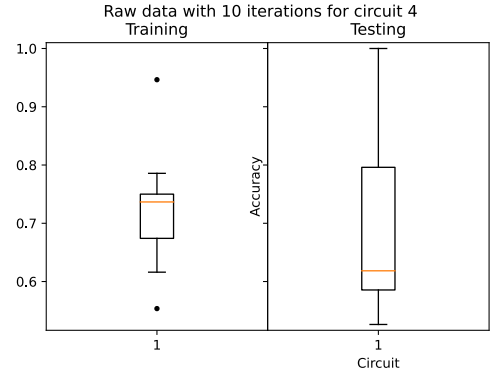

Pified data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

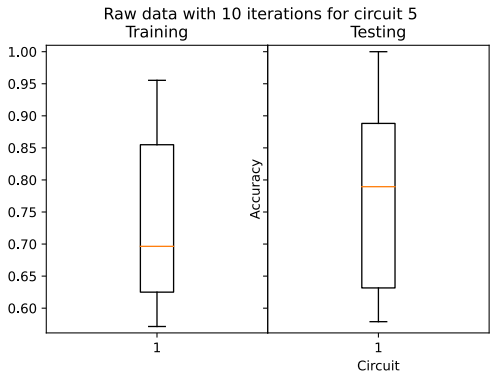
A.2.2 Randomized test/train data

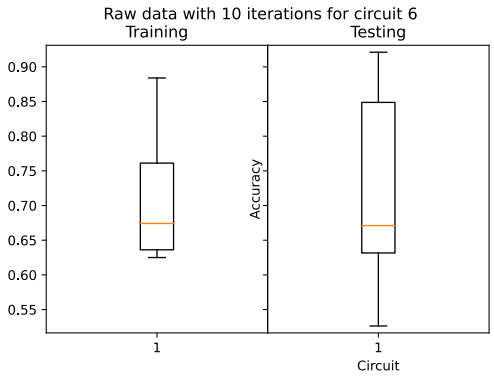


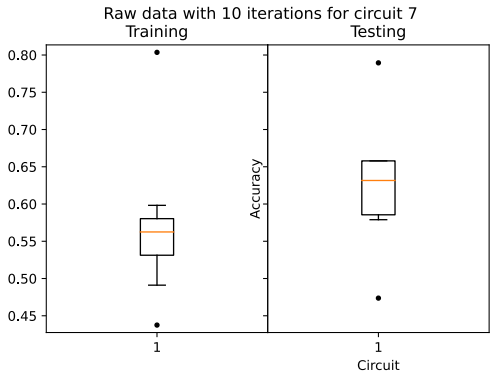


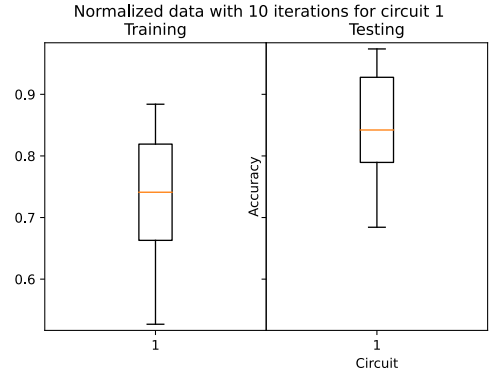


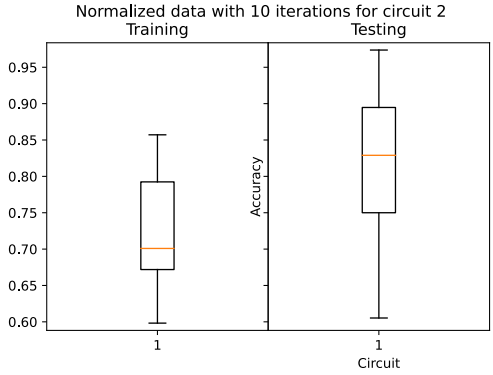


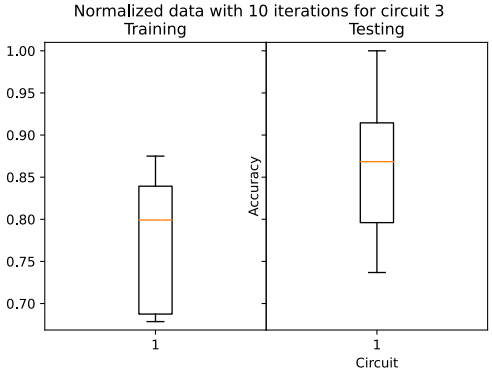


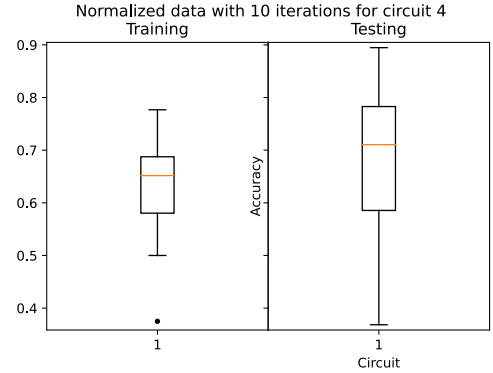


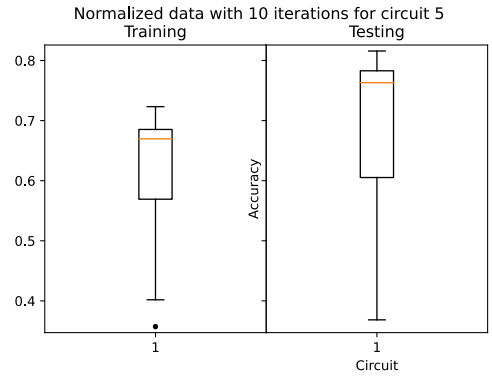


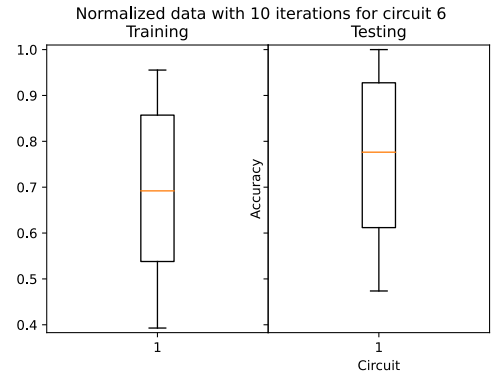

Normalized data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
1
1
Circuit

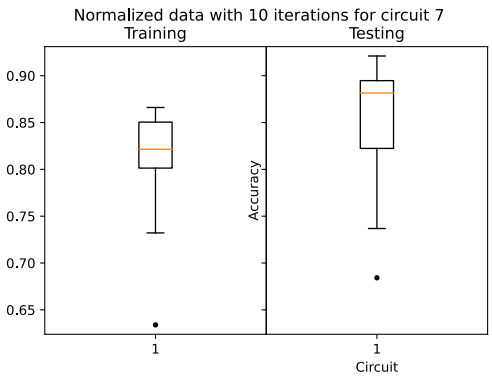

Normalized data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

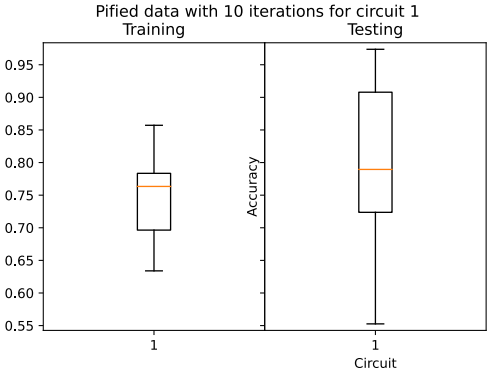

Normalized data with 10 iterations for circuit 3
Training
Testing
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
1
1
Circuit

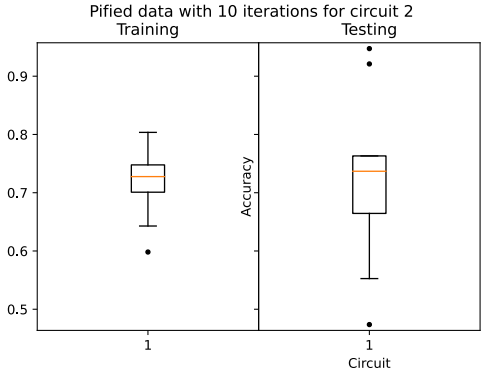

Normalized data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

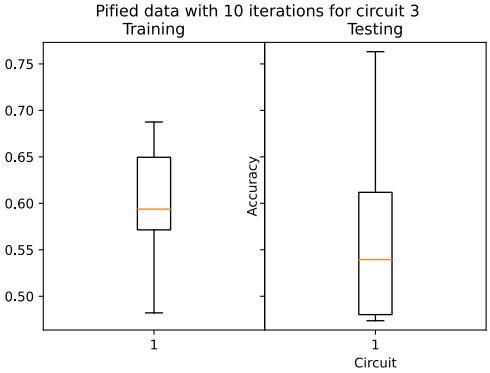

Normalized data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

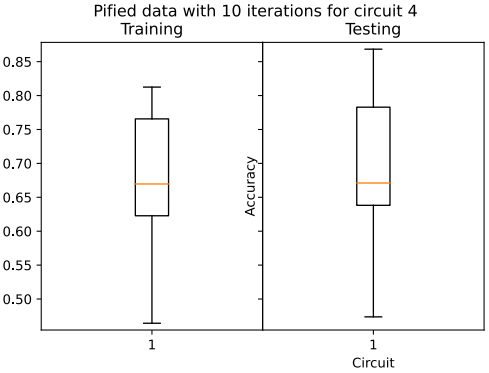

Normalized data with 10 iterations for circuit 6
Training
Testing
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

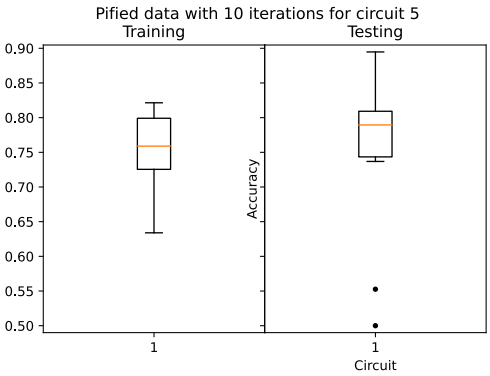

Normalized data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
1
1
Circuit

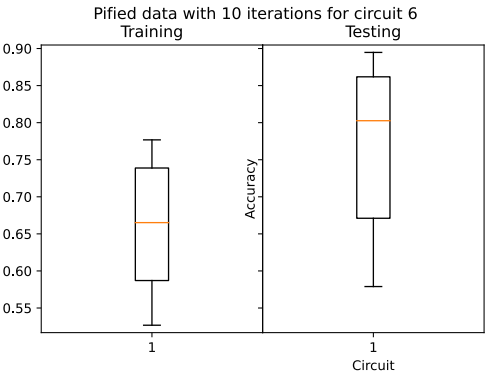

Pified data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

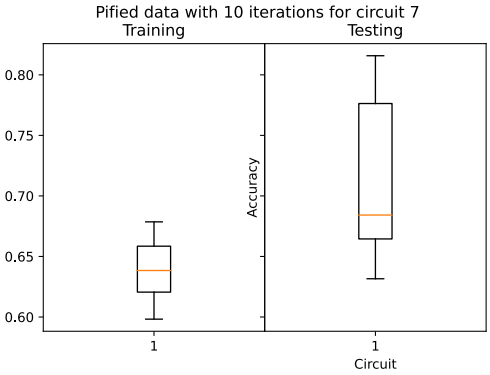

Pified data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
1
1
Circuit


Pified data with 10 iterations for circuit 3
Training
Testing
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit


Pified data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

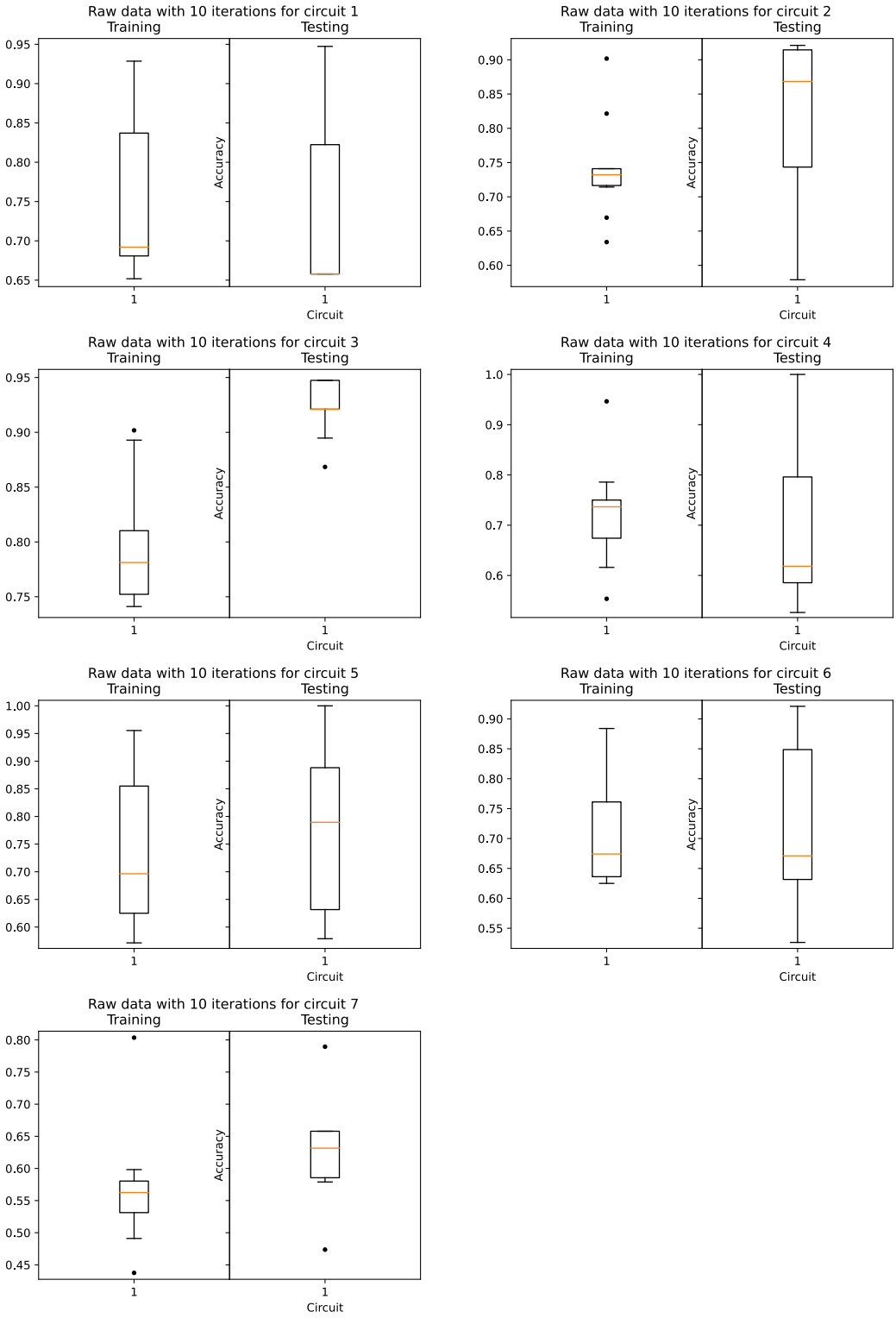

Pified data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

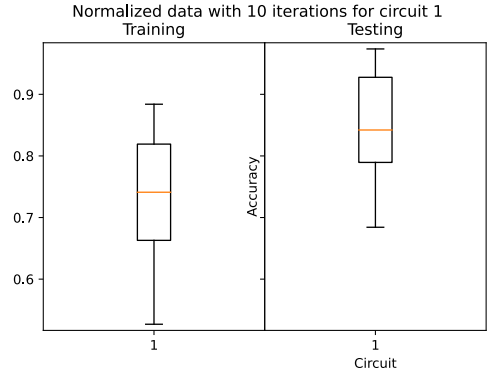

Pified data with 10 iterations for circuit 6
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

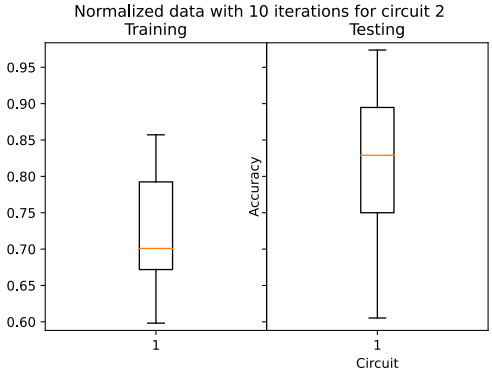

Pified data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

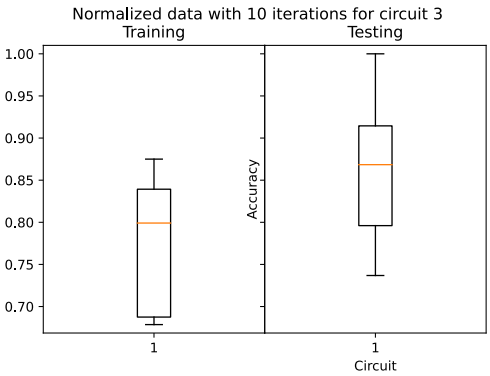
A.3 Artificial Problem

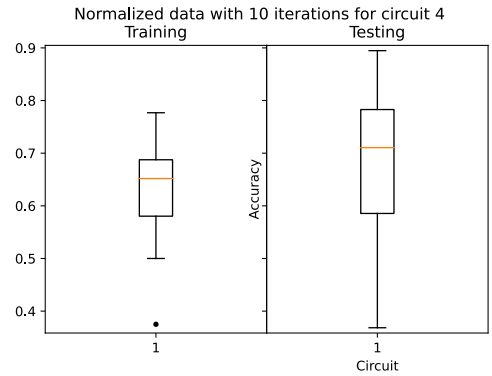
A.3.1 Non-randomized test/train data

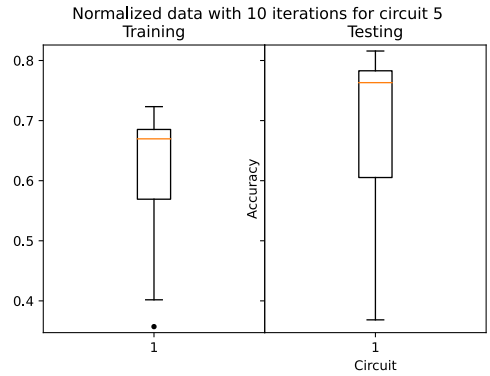


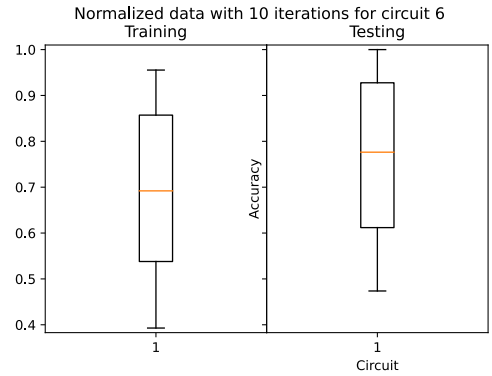

Normalized data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
1
1
Circuit

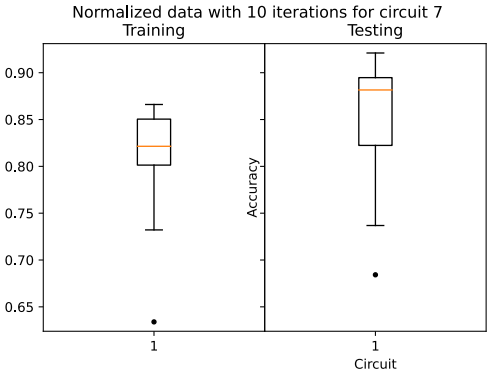

Normalized data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

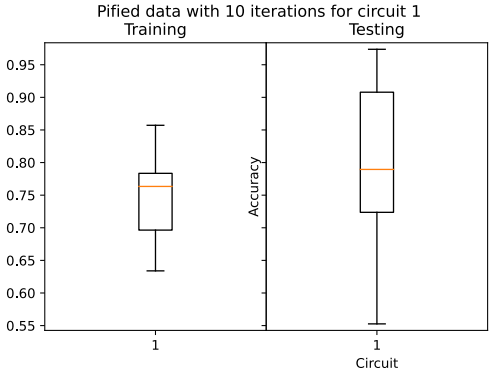

Normalized data with 10 iterations for circuit 3
Training
Testing
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
1
1
Circuit

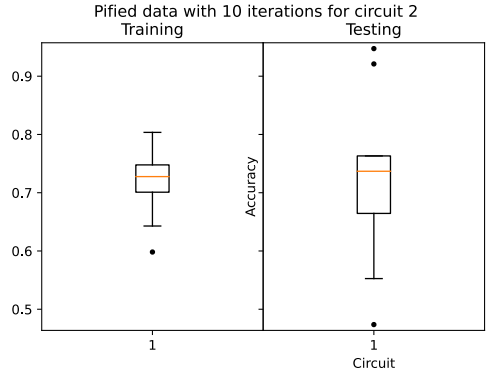

Normalized data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

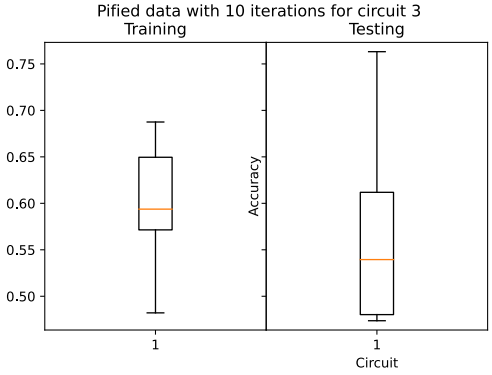

Normalized data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

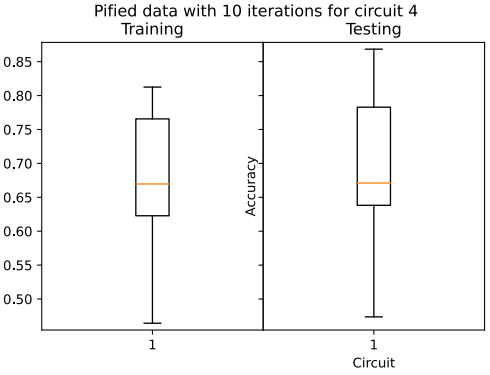

Normalized data with 10 iterations for circuit 6
Training
Testing
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

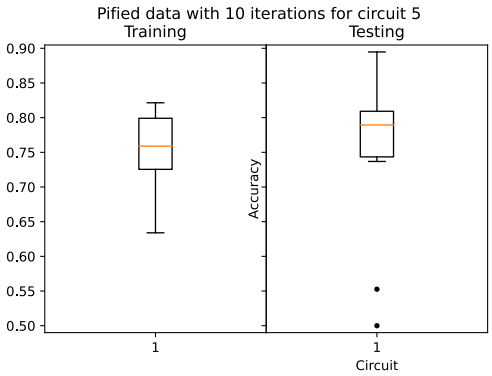

Normalized data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
1
1
Circuit

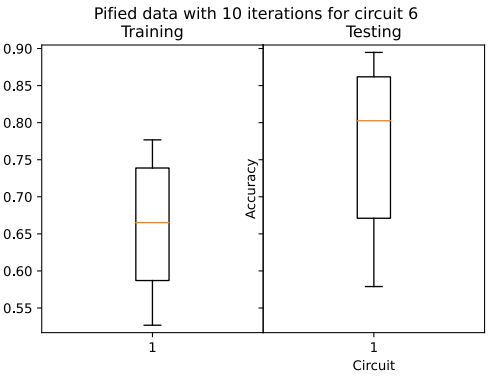

Pified data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

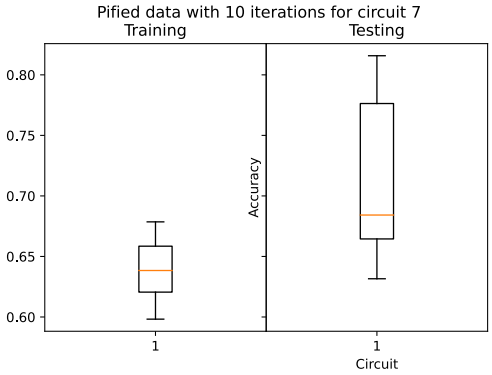

Pified data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
1
1
Circuit


Pified data with 10 iterations for circuit 3
Training
Testing
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

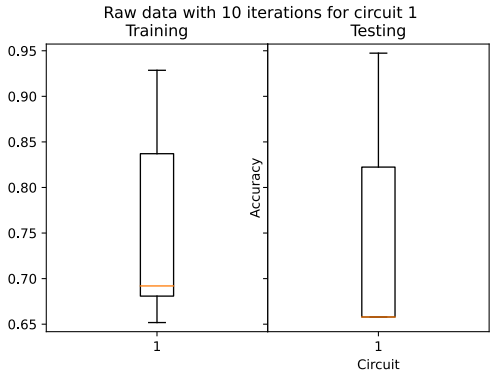

Pified data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

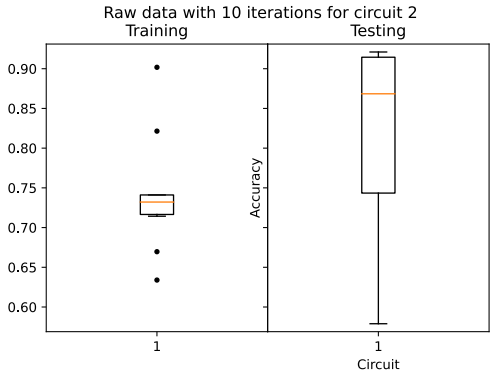

Pified data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit

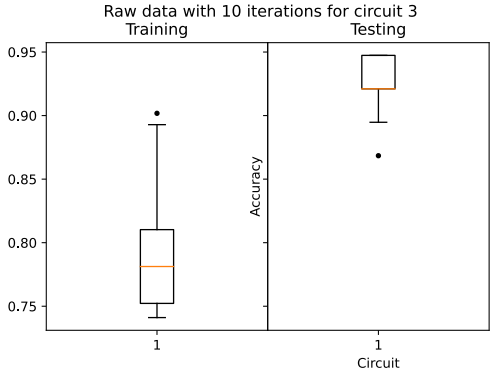

Pified data with 10 iterations for circuit 6
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

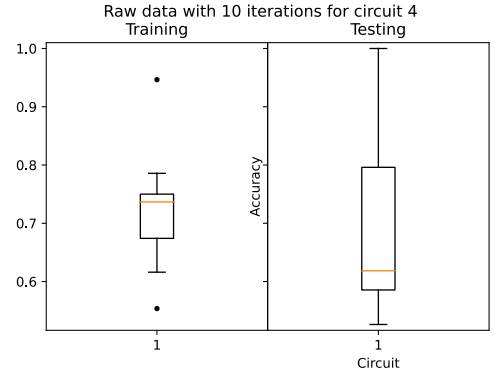

Pified data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

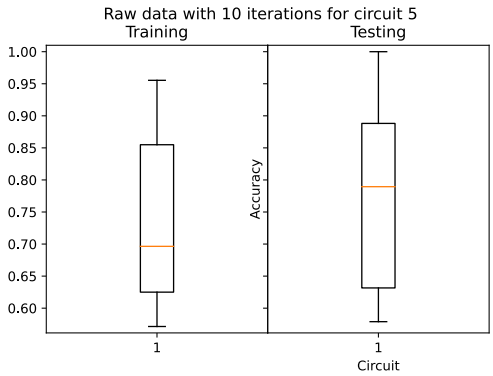
A.3.2 Randomized test/train data

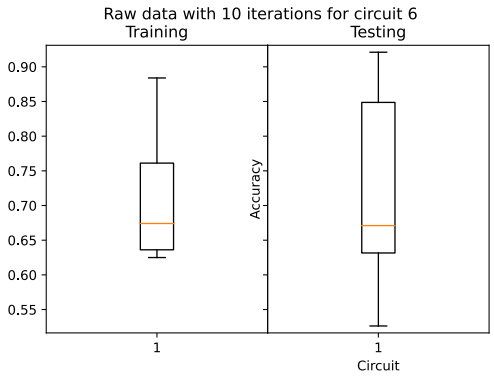


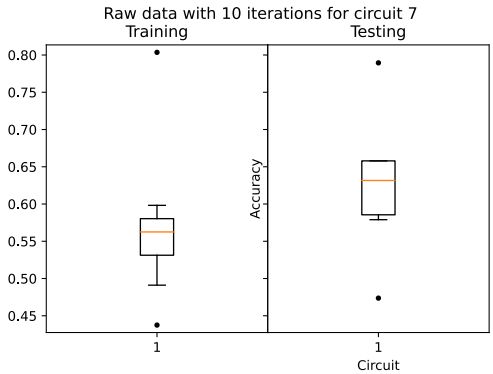


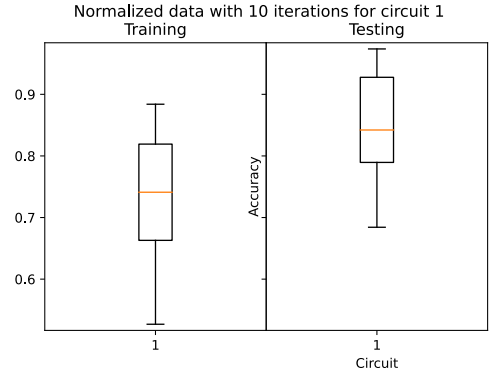


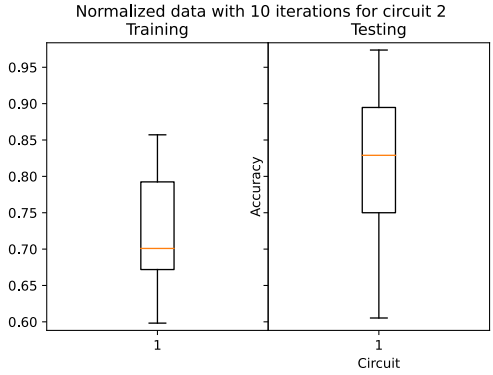


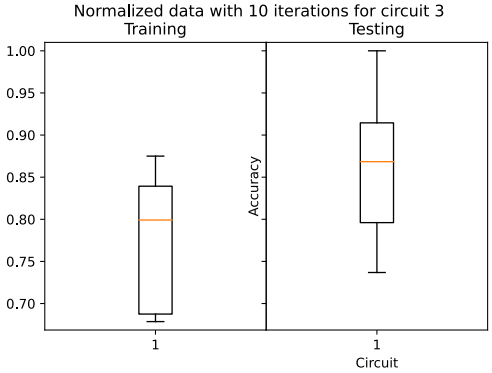


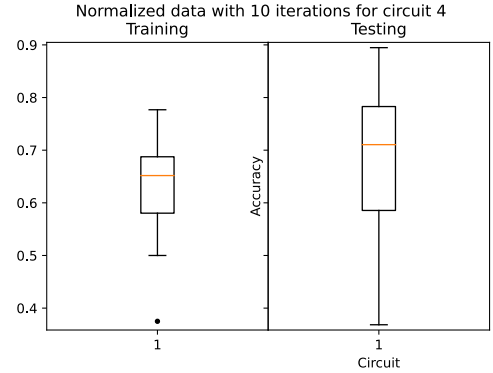


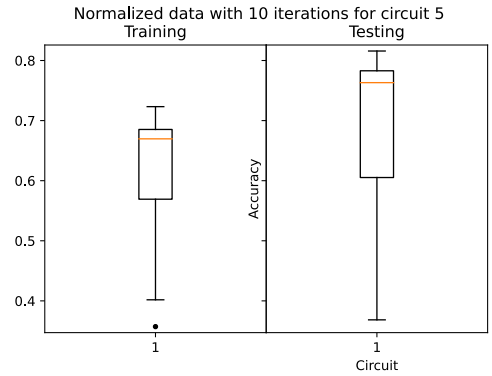


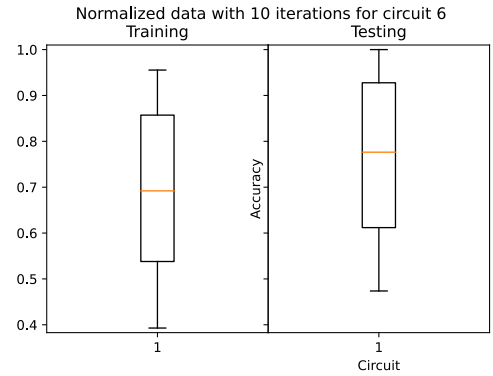

Normalized data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
1
1
Circuit

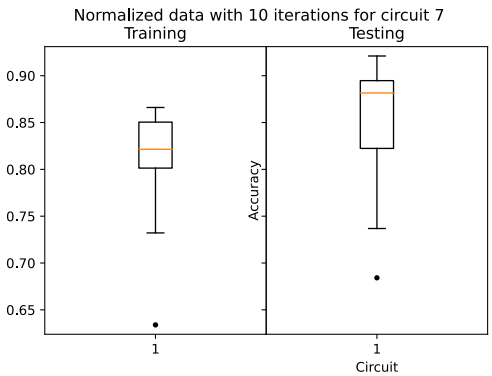

Normalized data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1
1
Circuit

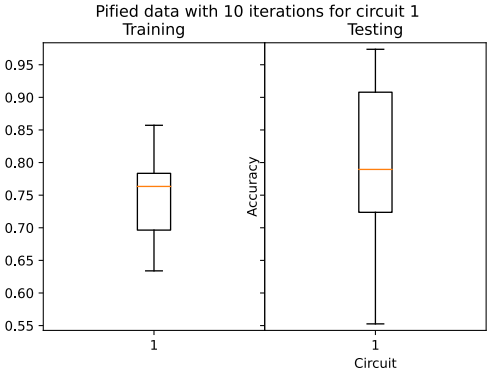

Normalized data with 10 iterations for circuit 3
Training
Testing
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
1
1
Circuit

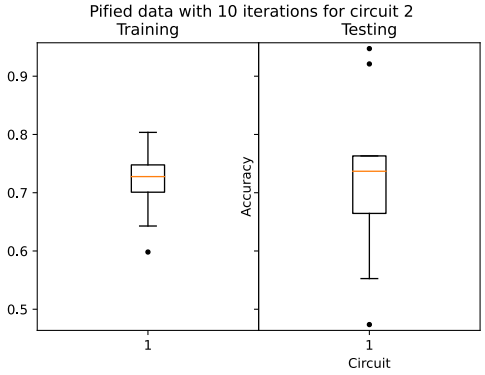

Normalized data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

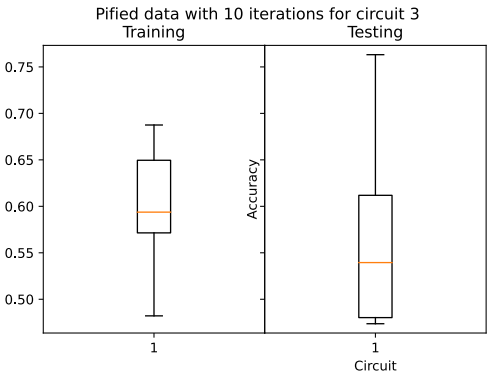

Normalized data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

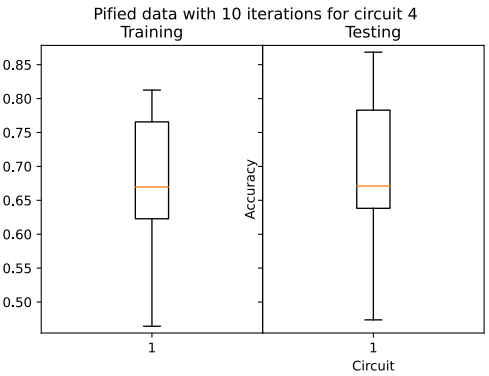

Normalized data with 10 iterations for circuit 6
Training
Testing
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
1
Circuit

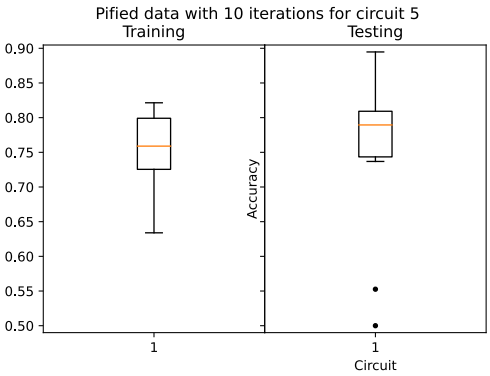

Normalized data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
1
1
Circuit

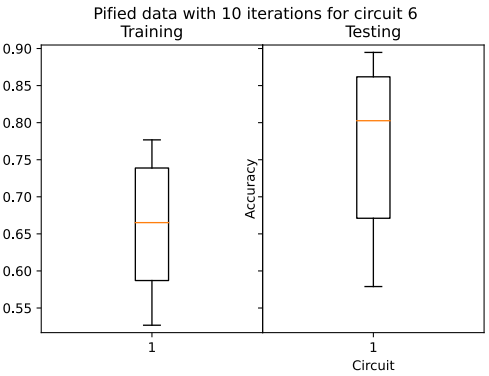

Pified data with 10 iterations for circuit 1
Training
Testing
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit

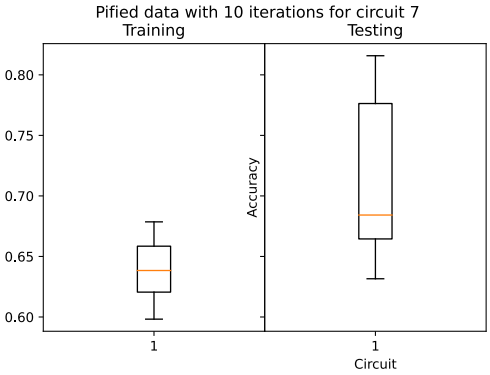

Pified data with 10 iterations for circuit 2
Training
Testing
Accuracy
0.9
0.8
0.7
0.6
0.5
1
1
Circuit


Pified data with 10 iterations for circuit 3
Training
Testing
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit


Pified data with 10 iterations for circuit 4
Training
Testing
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit


Pified data with 10 iterations for circuit 5
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
1
1
Circuit


Pified data with 10 iterations for circuit 6
Training
Testing
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1
1
Circuit


Pified data with 10 iterations for circuit 7
Training
Testing
Accuracy
0.80
0.75
0.70
0.65
0.60
1
1
Circuit