The basics of quantum machine learning: Classification of the Iris dataset by use of quantum ciruits

Jonny Aarstad Igeh
Department of Physics, University of Oslo, Norway

June 4, 2024

Abstract

The merging of quantum computing and machine learning was the stuff of science fiction just a few decades ago, but now it is a natural progression in the field of quantum technology. In this project we've studied a simple quantum circuit for data classification, utilizing qubit-based data encoding and a quantum circuit ansatz with variational parameters for making predictions. The parameters have been optimized using a classical optimizer, and the model has been tested on the Iris dataset. The results show that the model is able to classify the training data with an accuracy of around 80%, but fails to classify the test data, reaching an accuracy of 50%. This illustrates the potential in quantum machine learning, as well as the need for more research in the field. Our implementation found the importance of entanglement, and how entanglement is used to great effect in quantum circuit to capture complex patterns in data.

Contents

1 Introduction

With the advent of quantum computing, the field of quantum machine learning has seen a surge in interest within the field. Theoretically, quantum computers already outperform classical computers in certain tasks, and new algorithms are continually developed to exceed their classical counterparts. The field of machine learning is a massive field that has taken the world by storm the last decade, and the combination of quantum computing and machine learning is a natural progression. In this project, we will explore the basics of quantum machine learning, and how quantum circuits can be used to classify data.

One area where quantum machine learning could prove useful is for classification purposes, where the goal is to predict the class of a given input data point. This is due to the nature of measurements in a circuit translates to probabilities, which is a natural fit for classification tasks. In this report, we will explore the Iris dataset [2], one of the earliest and most well-known datasets for evaluating classification algorithms.

We will employ a simple quantum circuit for data classification, utilizing qubit-based data encoding and a quantum circuit ansatz with adaptable parameters for making predictions. The framework will be implemented in Python, leveraging the Qiskit library for quantum computing and the Scikit-learn library for classical machine learning. This report will cover the theoretical background of quantum machine learning in the theory section 2, followed by the implementation details of the quantum circuit and classical optimizer in method & impleentation section 3. The result section 4 will present the project outcomes, with an analysis of the results and methods used, before we conclude our findings and provide some final insights on quantum machine learning section 5.

2 Theory

2.1 Classification

We will build, and use, our quantum circuit for classification purposes. Classification is a supervised learning task, where we have a dataset with input samples and corresponding target values. The goal of the classifier is to learn the patterns in the dataset, and make predictions on new, unseen data. In our case, we will use a quantum circuit to classify the data, and we will use the cross-entropy loss function to train the model. The classifier circuit will be trained on a dataset, and will make predictions on new samples by encoding the samples into the quantum circuit, and then measuring the state of the qubits to make predictions.

2.1.1 Iris dataset

The Iris dataset is a well-known dataset in the field of machine learning, and is often used to test classification algorithms. The dataset consists of 150 samples of iris flowers, with four features for each sample: sepal length, sepal width, petal length, and petal width. The target values are the species of the iris flowers, and there are three possible species: setosa, versicolor, and virginica. The Iris dataset is a simple dataset, and is well-suited for testing classification algorithms.

2.2 Encoding data

In order to use a quantum circuit for classification, we need to encode the data into the circuit. This is done by giving each feature in the dataset a qubit, i.e n features requires n qubits, and the subsequent sample(s) (of a given feature) will be converted into an angle - and be encoded to the circuit by a R_z rotation gate. We will first split the dataset into training, and test data - where the training data is, as you'd expect, used for training the quantum circuit, and the test data is used to validate the performance of our circuit. Thereafter, we need to scale the feature values, and convert them into radians so they can be used to tune our rotation gates.
The encoding of the data is a crucial step in the classification process, and the choice of encoding will affect the performance of the classifier. Our choice of encoding, by using a single Hadamard gate (to achieve superposition), and a subsequent R_z gate with the sample is a very basic encoding - but will be a sufficient starting point for our simple classifier. Such encodings are often referred to as a *feature map*.

2.3 Predictions

When we want to perform measurements on our quantum system, we need to measure the state of the qubits in a basis of choice. For our purposes, we will measure the qubits in the σ_z (spin-z) basis. This basis has two possible 1 qubit states, $|0\rangle$ and $|1\rangle$, and, in a n qubit system, the possible states are the following

$$\Psi = \bigotimes_i x\,|j\rangle$$

where j can take the values 0 or 1, spin down or up respectively.
For our classification algorithm, we need to map the measurements of our quantum state to the target values, and this mapping is arbitrary. A natural choice for our simple classification problem, using only 2 of the 3 possible targets in the Iris dataset, we have the following mapping

- $|0\rangle \to 0$
- $|1\rangle \to 1$

2.4 Quantum Circuit

The core of our classifier algorithm is the quantum circuit. A quantum circuit is a collection of qubits, and quantum gates, that are used to perform quantum computations. It is used to manipulate an initial state of qubits into a final state, and the final state of the qubits is then measured to make predictions. By applying various rotations and entanglement gates, we can shape the initial state of the qubits to capture the patterns in our dataset, and make accurate predictions.
A schematic overview of the quantum circuit is shown in figure 1.

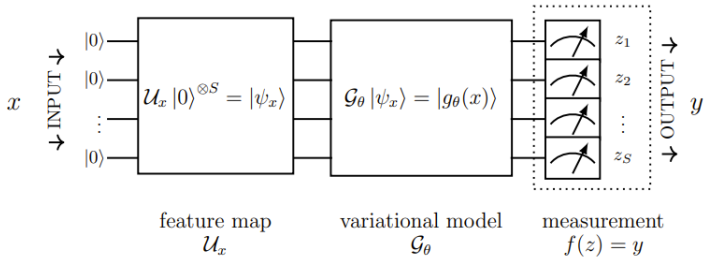

Figure 1: A schematic overview of the quantum circuit used for classification. The data is encoded into the circuit, and the state of the qubits is measured to make predictions. Source: [1].

2.5 Quantum Gates

Quantum gates are the building blocks of quantum circuits. They are the quantum analogues of classical logic gates, and are used to manipulate the state of a quantum system. In this section, we will discuss some of the most important quantum gates, and how they can be used to perform quantum computations.

2.5.1 The Hadamard Gate

The Hadamard gate is a single-qubit gate, and is defined as follows:

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

The Hadamard gate is used to create superposition, and is also used to perform a change of basis.

2.5.2 Rotation Gates

The rotation gates are single-qubit gates, and are defined as follows:

$$R_x(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix},$$
$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix},$$
$$R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}.$$

The rotation gates are used to rotate the state of a qubit around the x, y, and z-axis, respectively. For our classifier algorithm, the rotation gates will be used for two purposes:

- To encode the samples into the quantum circuit
- To create the variational ansatz, where the angles in the rotation gates are the parameters we want to optimize.

2.5.3 The CNOT Gate

The CNOT gate is a two-qubit gate, and is defined as follows:

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

The CNOT gate is used to create entanglement, and is also used to perform conditional operations. The CNOT gate is also known as the controlled-X gate, and the need to create entanglement arises when we are working in a system where the ground state may be an entangled state. Then we need to be able to create entanglement in our trial wavefunction, otherwise we will not be able to properly converge towards the ground state.

2.6 Computational basis

The computational basis is the basis that we can use to represent the state of a quantum system. In a quantum computer, the computational basis is the set of all possible states of a qubit. The qubits in our system can be in a superposition of states, and we represent this superposition using a linear combination of the computational basis states. For a single qubit system, the computational basis is as follows:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

This can easily be extended to a multi-qubit system by taking the tensor product of the single-qubit basis states. E.g for a 2 qubit system,

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

2.7 Entanglement

A very important topic for quantum computing is the concept of entanglement. This is a property of quantum systems that do not occur in classical systems, and is widely used when we perform quantum computing. Many quantum computing algorithms would not be possible without entanglement.
The concept of entanglement (in a quantum computer) is the fact that two (or more) qubits become correlated in such a way that the state of one qubit is directly dependent on the state of the other qubit(s). This is a property we can use, since we then need only perform a measurement on one qubit to immediately know the state of the other qubit(s)!
Entanglement is a very important property, and is used in many quantum algorithms. In our classifier, we will use the concept of entanglement to capture complex patterns in the dataset, and to make more accurate predictions with our model circuit.

2.8 Parameter optimization

In order to optimize the parameters of our quantum circuit, we need to define a cost function that we want to minimize. In our case, we will use the cross-entropy loss function.
The loss function will evaluate the difference between the predicted probabilities of the model and the true probabilities of the data, effectively giving a measure of how good the current parameters are. We can also use that fact that we'd like to minimize the loss function, and calculate the loss-functions gradient w.r.t. to the variational parameters - and use this minimization scheme to update the parameters.

There are many methods for such optimization, and in this report, we will use some of the traditional methods - which we will introduce later in this section.

2.8.1 Cross-entropy

The cross-entropy loss function is a measure of how well the predicted probabilities of the model match the true probabilities of the data. It is defined as follows:

$$L(\theta) = -\sum_{i=1}^{N} y_i \ln f(x_i; \theta), \tag{1}$$

where y_i is a given target sample and $f(x_i; \theta)$ is the predicted probability of the model given an input sample x_i and the parameters θ. The cross-entropy loss function is a common choice for classification problems, and is used to train the model paramers by minimizing this loss function, as explained in the

previous section. For our binary classifier problem, we need to use the binary cross-entropy function, which is defined as follows:

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \ln f(x_i;\theta) + (1-y_i)\ln(1-f(x_i;\theta)), \tag{2}$$

where N is the number of samples in the dataset.

2.8.2 Parameter-Shift Rule

To find the gradient of the cross-entropy, we need to evaluate the following

$$\frac{\partial}{\partial\theta_k}L = -\frac{1}{N}\sum_i (\frac{y_i}{f_i} - \frac{1-y_i}{1-f_i})\frac{\partial}{\partial\theta_k}f_i \tag{3}$$

where $f_i = f(x_i;\theta)$ is the predicted probability of the model. Finding the gradient of the model output is non-trivial, and we will use the parameter-shift rule for evaluating the gradient of the model parameters, and this is found by the following:

$$\frac{\partial f(x_i;\theta_1,\theta_2,...,\theta_k)}{\partial\theta_j} = \frac{f(x_i;\theta_1,\theta_2,...,\theta_j+\pi/2,...,\theta_k) - f(x_i;\theta_1,\theta_2,...,\theta_j-\pi/2,...,\theta_k)}{2} \tag{4}$$

2.8.3 Gradient Descent methods & Scipy.minimize

When trying to learn patterns in the dataset, we will continously calculate the gradients of our loss-function, and "move" in the negative direction of the gradient in the parameter space. This is the essence of optimization, and at the center of gradient descent methods. However, when optimizing quantum circuits, we need to be careful. The presence of "barren plateaus", regions in parameter space where the gradients are exponentially small, pose a big challenge for most gradient methods. However, for our simple dataset, these barren plateus should not pose significant issues - but it is important to keep in mind when working with gradient descent methods in quantum machine learning.
The gradient descent algorithm is as follows:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta\nabla_\theta L(\theta_{\text{old}}), \tag{5}$$

where η is the learning rate, and $\nabla_\theta L(\theta_{\text{old}})$ is the gradient of the loss function w.r.t. the parameters at the current iteration.
I.e we calculate the gradient of the parameter(s) we are using to minimize the expectation value of the Hamiltonian, and then update the parameters by taking a step in the direction of the negative gradient.

We will also use more sophisticated optimization methods, available to us in the SciPy library in Python. Namely, we will be using the COBYLA, which is the method of choice for many applications of quantum machine learning. The COBYLA method is a gradient-free optimization method, and is well-suited for optimization problems where the gradient is not known, or is difficult to calculate. The COBYLA method is a good choice for our problem, and we will use it to optimize the parameters of our quantum circuit.

3 Methodology & implementation

In this section, we will give brief introductions on the implementation of the various algorithms and methods presented in the theory section. These introductions will give a general overview of the steps taken in our project, and how the different parts of the project are connected. We will also present some insights behind some of our choices in the implementation.

3.1 Data encoding

For our simple, 4 feature IRIS dataset, we will encode the data on qubits in the following way:

- For each feature, we assign a qubit
- For each feature value, we assign a rotation angle θ, scaled with the sample value, to the corresponding qubit

3.1.1 Pre-processing of the dataset

As explained in the theory section (2.2), we need to scale the feature values (the samples in the dataset) to the interval $[0, 4\pi]$. To do so, we will use the built in MinMaxScaler in the Scikit-learn library [4], which scales the values to the interval $[0, 1]$. We will then multiply the scaled values with 4π to get the desired interval.
We will also split the dataset into training and testing sets, using the built-in method train_test_split in Scikit-learn. This method will split the dataset into two parts, one for training the model and one for testing the model. We will use 80% of the dataset for training and 20% for testing, and this split is necessary to prevent overfitting to our dataset, and to assess the generalization capabilities of our model.

3.1.2 Encoding the data on qubits

This is easily done using the Qiskit framework, see [3], using the gates presented in section 2.5. For each scaled feature value, first apply a Hadamard gate before we apply a rotation gate (with the corresponding angle θ) to the corresponding qubit.

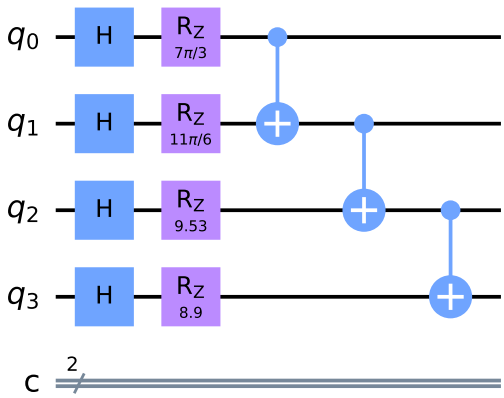

Figure 2: A quantum circuit encoding a single sample of the IRIS dataset. The rotation angles are scaled with the feature values.

A circuit of a single encoded sample is shown in figure 2, where the Hadamard gate is used to create superposition in each qubit.

3.2 Quantum Circuit

As stated previously, we will be using the Qiskit framework to implement our quantum circuits. This framework allows for— easy implementation of quantum computing algorithms, and provides a wide range of tools for quantum computing. As an example, the following code snippet will reproduce the data encoding circuit shown in figure 2:

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
qreg = QuantumRegister(4)
creg = ClassicalRegister(2)
circuit = QuantumCircuit(qreg, creg)
circuit.h(qreg)
circuit.ry(theta1, qreg[0])
circuit.ry(theta2, qreg[1])
circuit.ry(theta3, qreg[2])
circuit.ry(theta4, qreg[3])
circuit.barrier()
```

where the angles θ should be calculated from the sample values as explained in the previous section.

3.3 Variational ansatz

The variational ansatz is implemented in a similar way as the data encoding, using the Qiskit framework. The ansatz is a simple circuit, consisting of CNOT gates to create entanglement between the qubits, and rotation gates to introduce the variational parameters. The following code snippet shows how this can be done:

```
some_params = ...
circuit = QuantumCircuit(qreg, creg)
circuit.h(qreg)
for i in range(4):
    circuit.ry(some_params[i], qreg[i])
for c in range(4):
    for t in range(c, 4):
        if c != t:
            circuit.cx(qreg[c], qreg[t])
.circuit.ry(some_params[-1], 3)
```

where the variational parameters are stored in a list, and the circuit is constructed similarly to the data encoding circuit. Here we've put CNOT gates between each pair of qubits, but the entanglement can be constructed in any way that is deemed necessary for the problem at hand. We've also added a "bias" term at the end of the circuit, on the qubit that we are going to measure - just to add more complexity to our model. For our actual quantum circuit, we will initialize the parameters randomly in the domain $[-2\pi, 2\pi]$, before starting the optimization.
Running the above snippet yields the following circuit shown in figure 3.

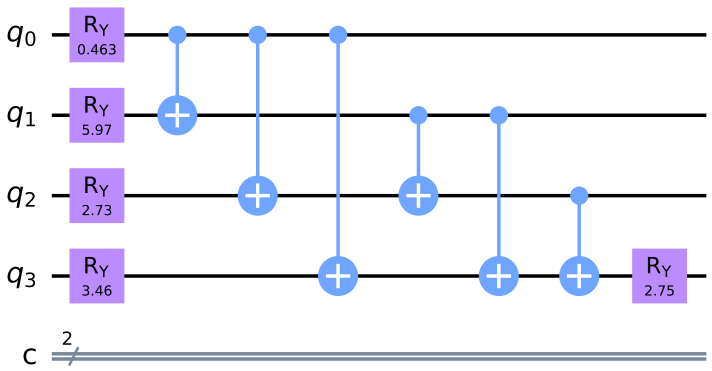

Figure 3: A simple variational ansatz circuit, with rotation gates and CNOT gates.

3.4 Making predictions

To use our quantum circuit for classification tasks, we will need to map the measurements done on the qubits to an output that mimics the target matrix of our dataset. The measurements themselves are easily performed in Qiskits framework - using a quantum simulator (since we currently lack access to a quantum computer) - and the results are stored in the classical registers. Using the mapping presented in section (2.3), we identify that we need only to measure two of the four qubits needed to encode the Iris dataset. This is done by applying a measurement gate to the two qubits we want to measure, and then store the result in a matrix to be used to evaluate the loss function. A simple example of how this can be done is shown in the following code snippet:

```
frorm qiskit import transpile
from qiskit_aer import AerSimulator
circuit.measure(qreg[-1], creg[0])
backend = AerSimulator(method="statevector")
circuit = transpile(circuit, backend)
result = backend.run(circuit, shots=10000).result()
counts = result.get_counts()
tmp = 0.0
for key in counts.keys():
    if key[0] == '1':
        tmp += 1 * counts[key]
prediction = tmp / 10000
```

where the measurement is here done on the last qubit. The result is then stored in the variable prediction, which can be used to evaluate the loss function. It is important that we use the AerSimulator with the statevector argument to get the correct probabilities, as the other noisy simulators will struggle with convergence for such few shots.

3.5 Optimization

3.5.1 Loss function

Implementing the binary cross-entropy loss function (2) is pretty straightforward in Python, and can be done using the following code snippet:

```
def cross_entropy_loss(y_true, y_pred):
    loss = 0
    for i in range(len(y_true)):
        loss += - (y_true[i]*np.log(y_pred[i]) + (1 - y_true[i])*np.log(1 - y_pred[i]))
    return loss
```

where y_{true} is the target vector and y_{pred} is the predicted vector. The loss function is then used to evaluate the performance of the model, and to update the variational parameters in the optimization process, as explained in section (3.5).

3.5.2 Gradient Descent

The simplest of optimization algorithms that we will use to update the variational parameters of our quantum circuit is the gradient descent algorithm. For our quantum circuit, this gradient descent optimization algorithm is as follows:

Algorithm 1 Gradient Descent

1: Initialize the variational parameters θ
2: **while** not converged **do**
3: Calculate the gradient of the loss function with respect to the variational parameters
4: Update the variational parameters using the gradient
5: Evaluate the loss function and check for convergence
6: **end while**

3.5.3 Scipy-minimize by gradient-free methods

Due to the barren plateu problem that often arises when optmizing quantum circuit, we will also implement a non-gradient based optimization algorithm, namely the COBYLA algorithm from the Scipy library. COBYLA (Constrained Optimization by Linear Approximations) is a derivative-free optimization algorithm that is well suited for optimization problems where the gradient is not known[5].

4 Results and analysis

4.1 Iris dataset

As stated in section 2.3, we will investigate only two of the three classes in the Iris dataset - which allows us to only make measurements on one of the qubits to determine the class of the sample. Using a train-test-split of 80/20, with two thirds of the dataset gives us 80 samples, and subsequently 80 different quantum ciruits that we will evaluate, and take measurements on, to make predictions.

4.2 Training the model circuit

Using the quantum circuit described in section 3, with 13 learnable parameters (3 sequences, and 1 bias term) we encode our dataset and train the model using a standard gradient descent method (see section 3.5), we are ready to evaluate the performance of our model.
Running for four different learning rates, and 35 epochs (with 10000 shots per epoch), resetting the parameters to the same starting point before each run, produces the following graphs:

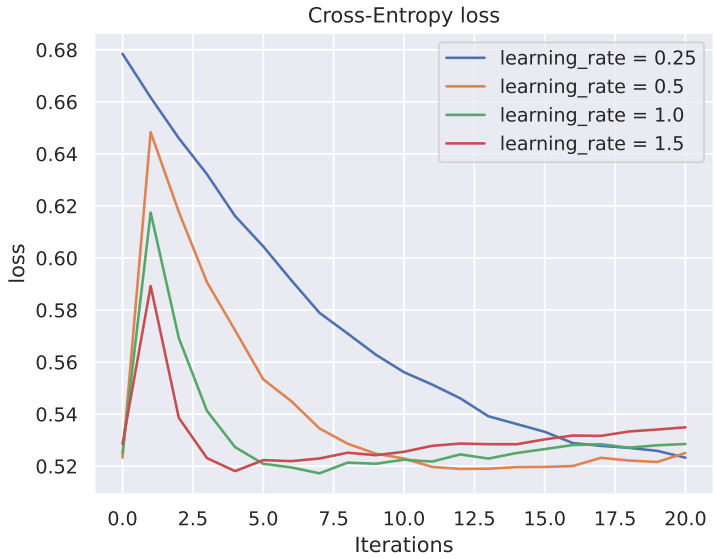

Figure 4: Cross entropy as a function of training iterations for different learning rates

From the graph in figure (4) we can see that all 4 choices of learning rates converge to the same minima, $\approx$ CE = 0.52, but at different rates. The higher learning rates converge quicker, which can be intepreted as them taking "larger steps" in parameter space, and thus finding a minima much quicker, which is consistent with the results in the graph.
It seems also that the algorithm diverges slightly for the higher learning rates, which could be caused by the fact that the optimization "jumps" in and out of the local minima that the algorithm converges to when the learning rate is larger. This is a common problem with gradient descent, and is usually solved by decreasing the learning rate as the algorithm converges (using more sophisticated, momentum-based gradient descent methods, like ADAM). One interesting observation is the fact that for $l = 0.25$, it seems that the algorithm starts at a much higher initial loss - and this is most likely due to the randomness of the predictions, as we only ran 10 000 shots per epoch, we cannot guarantee that we predict the true state of the system.
A more rigorous study would employ a much larger number of shots, and also run the optimization multiple times to attempt to minimize the probability that the initial state is mis-read.

A different, and maybe even more informative measure for model performance, is the check the prediction accuracy for the four different circuit setups. In figure (5) we again identify the fault that arises from the randomness of the measurements, as the initial accuracy scores are somewhat different, even for the same initial parameters. What we do see, however, is that the larger learning rates again converge

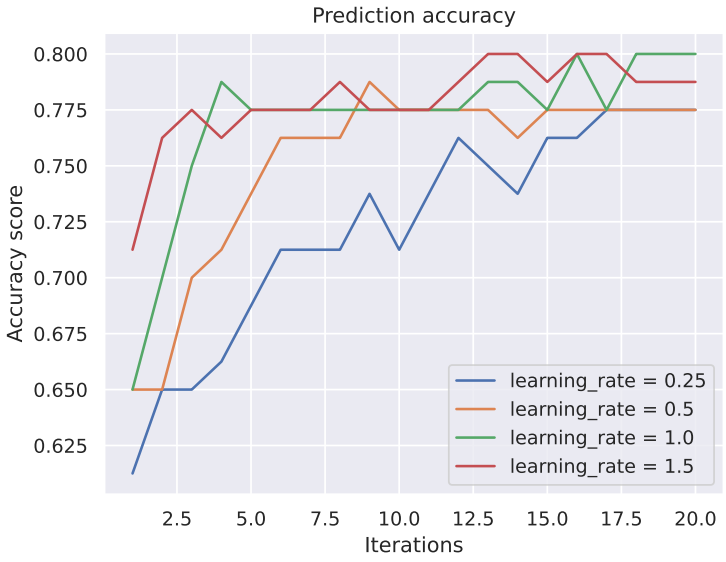

Figure 5: Prediction accuracy as a function of training iterations for different learning rates

faster to a high accuracy score of $\approx 80\%$, while the smallest learning rate $= 0.5$ take more iterations to converge, and to a lower accuracy. Some of the reason for the varying levels of accuracy is due to us having to "round" the predictions when measuring the accuracy scores. Since the predicted measurements will be averages over 10 000 shots, they will not be integer numbers - maybe 8000 shots are evaluated to 0, still 2000 are prerdicted 1. Naturally, there is the possibility to be "lucky" that your floating point predictions minimize the loss-function, while the rounded predictions yield a "low" accuracy score. More shots per epoch would yield a more accurate prediction, but also increase the computational cost of the model - which is why we've chosen to use 10 000 shots per epoch.

4.3 Data encoding

Using the encoding described in section 2.3, we allow the model to properly entangle differently for each sample - yielding a more diverse starting point for the ansatz to classify. We could've added even more rotation gates, by also looking at the difference between sample points - to make the qubit-states more dependent on all the feature values in a sample - but as we can see from figure (5), the entanglement introduced from the CNOT gates, both in the encoding and the ansatz, is sufficient for the model to accurately classify the samples. More complex encoding could potentially yield more rapid convergence in our model circuit, or even improve on the accuracy scores shown in figure (5).

To see the importance of a good encoding, we will attempt to run the model circuit with two different encodings:

- A simple encoding, with only one rotation gate per qubit
- A more complex encoding, with a rotation gate per qubit, and CNOT gates to entangle them

Both encoding still make use of a Hadamard gate to create superpositons - and the circuit is ran with a learning rate $= 1.0$, and an ansatz that applies one rotation gate per qubit, and also 3 CNOT gates between the pairs. Running for 10 epochs to visualize the necessity for entanglement yields the results shown in figure (6)

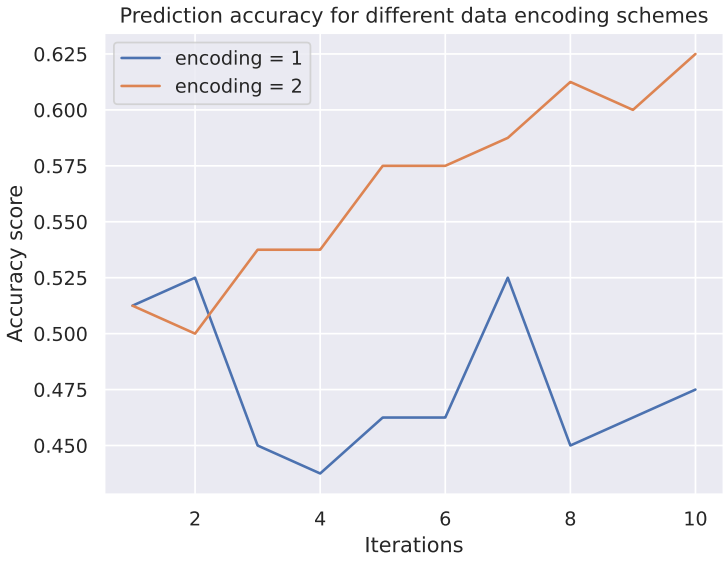

Figure 6: Prediction accuracy as a function of epochs (iterations) for different encodings

where we see that the non-entangled encoding performs much worse than the entangled encoding, which is to be expected. It diverges to a low accuracy score of ≈ 0.5, which is the same as qualified guessing. Checking the loss function, we get the following: This graph gives us a good indication that the

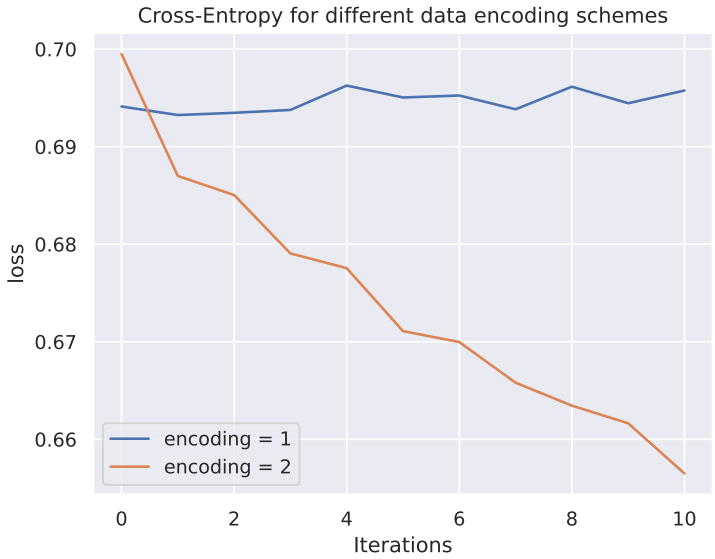

Figure 7: Cross-entropy loss as a function of epochs (iterations) for different encodings

entangled encoding performs at a higher level - and that we need the entanglement to properly classify the samples. Without the entanglement introduced in the encoding, the model can not properly mimic the patterns in the dataset, and it does not converge whatsoever.

4.4 Variational ansatz

Our choice of ansatz, with multiple CNOT gates and rotation gates, allowed our model to capture the underlying structure of the dataset, accurately predicting the training set. Evaluating our model circuit with the optimized parameters found when running the scheme that produced the plots in figure (4), yielded the following scores

- Test accuracy: 0.52
- Test loss: 0.79

We could see that our model potentially did overfit the data, as the test accuracy is not on par with the training accuracy - indicating a low level of generalization in our model circuit. The accuracy is no better than guessing, so our model clearly failed at predicting on this unseen test-data. This is something we could expect, given how simple the circuit is, and more complex datasets would require even more complex ansatzes to achieve similar accuracy scores. This is reason for concern if we'd like to apply such a quantum machine learning circuit on real life classification tasks - but as a proof of concept, we can be satisfied with convergence, and a decent accuracy score on the training data. This shows the potential of quantum machine learning, but also highlights the necessity for continued research in the field - to develop more complex models that can generalize better to unseen data.
Given a learning rate of $l = 1.0$, which from figure (5) and (4) seems to be an optimal choice of hyperparameter, we can test different variants of the variational ansatz to see how the model complexity affets the accuracy scores. This has been done for 3 different variants:

- A simple ansatz, with only one set of rotation gates, and no CNOT gates
- A more complex ansatz, with one set of rotation gates, and some CNOT gates
- A full ansatz, with multiple sequences of rotation gates, and CNOT gates between each pair of qubits, and a bias term.

The results are shown in figure (8)

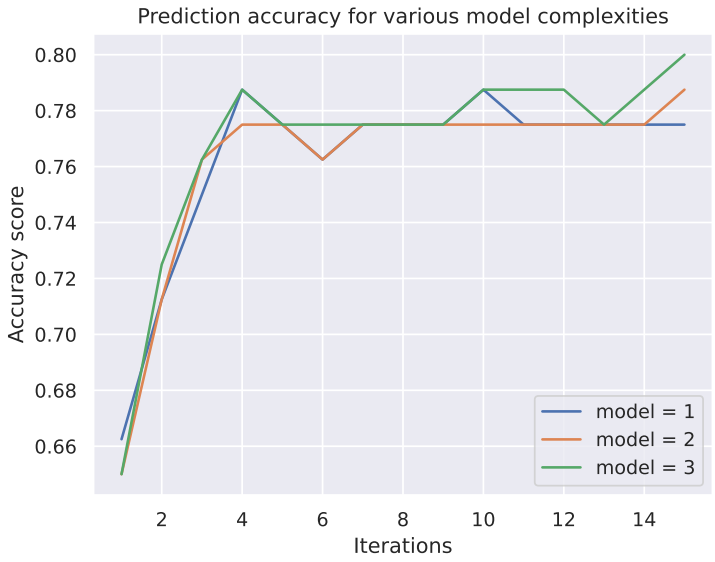

Figure 8: Prediction accuracy as a function of ansatz complexity

Interestingly, all 3 levels of complexity perform at the same level, accuracy wise. This indicates that we may not need a very complex ansatz to capture the patterns in the Iris dataset, and could make do with a very simple model ansatz. Another point this could show, is that our encoding is very efficient, and our good encoding provides a great initial starting point for our optimizatoin algorithm, allowing the model to accurately predict with few learnable parameters. We also check the loss function, to see if the model complexity affects the convergence rate of the model, presented in figure (9)

Figure (9) further backs our initial discovery that the model complexity does not necessarily need to be very high, and a very simple ansatz can capture the structure in the data. We still do not achieve a higher accuracy than 80%, with the loss function plateuing at $CE \approx 0.5$, which is an indicator that the model lands in a local minima. Another possibility is that the model suffers from "barren plateus", regions in the parameter space where the gradients are vanishingly small (or zero), without the model having reached a minima. The barren plateu problem is a common problem in quantum computing, and

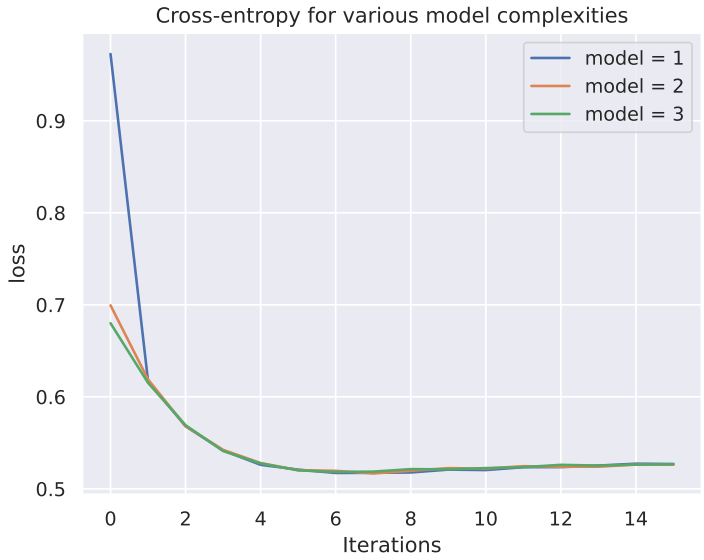

Figure 9: Cross entropy as a function of ansatz complexity

a common solution is to use non-gradient based methods. As a sanity check, we ran one model, with learning rate = 1.0, with the more complex model, using the COBYLA optimizer, as explained in section **??**. Doing so yields the following

- Accuracy score: 0.78
- Loss function: 0.49

which is strikingly similar to the results achieved using our gradient descent method. Thus we can assume that the barren plateu problem is not the main issue with our model, but rather that the model is stuck in a local minima.

5 Conclusion

This project explored the most basic of quantum machine learning algorithms by attempting to classify a subset of the Iris dataset. We have shown how we could encode such a dataset on qubits, scaling the feature values accordingly, and how a variational ansatz could be introduced to the quantum circuit to make predictions. These predictions, with the variational parameters as the only adjustable parameters, were then optimized using a classical optimizer. We were successful in the implementation of the quantum circuit, reaching a convergence of the cost function, and the classification accuracy was found to be around 80% for the training set. The circuit was also verified on a test set, where the model failed, and reached an accuracy of 50%. This shows room for improvement in the model, and the need for more complex circuits, and stronger optimization techniques. It could also be a sign of overfitting, however, as we have seen, reducing the complexity of the ansatz did not yield better results - meaning we can conclude with the necessity for more complex circuits, but with maybe fewer parameters. There are many ways to improve the complexity, by adding different rotation gates. Where we've used only R_y gates, we could've introduced both R_x and R_z gates (like we did in the encoding).

We did find however, that the quantum circuit was able to classify the known-data with a reasonable accuracy, and we saw the importance on choice of both encoding and ansatz. The lack on entanglement in the encoding of the data caused the model to fail even on the training set, highlighting the importance for entangling in such quantum circuit models when learning complex patterns in datasets.

Further work on this project would include more complex quantum circuits, both in encoding the data, and the variational ansatz. There is also a big improvement to made in the optimization schemes, looking at momentum-gradient based optimizers - such as ADAM - and adding more qubits, which would allow for more complex datasets to be classified.

References

[1] Amira Abbas et al. "The power of quantum neural networks". In: *Nature Computational Science* 1.6 (June 2021), pp. 403–409. ISSN: 2662-8457. DOI: 10.1038/s43588-021-00084-1. URL: http://dx.doi.org/10.1038/s43588-021-00084-1.

[2] R. A. Fisher. *Iris*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56C76. 1988.

[3] Ali Javadi-Abhari et al. *Quantum computing with Qiskit*. 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph].

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[5] Michael JD Powell. *A direct search optimization method that models the objective and constraint functions by linear interpolation*. Springer, 1994.