1 Theory

1.1 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.

1.1.1 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.

1.2 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*.

1.3 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

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

1.4 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 **??**.

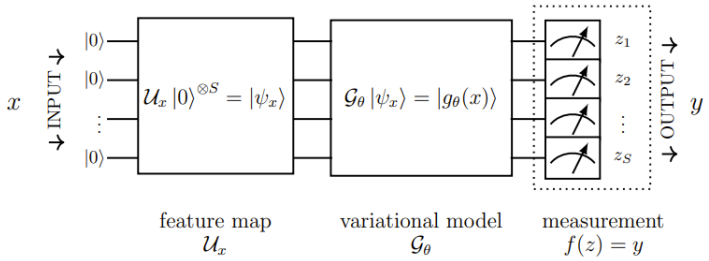

Figure 1: A schematic overview of the quantum circuit used for classification. The data is encoded into the circuit, and the state of the qubits is measured to make predictions. Source: [**Abbas'2021**].

1.5 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.

1.5.1 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.

1.5.2 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.

1.5.3 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.

1.6 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,

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

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

1.7 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.

1.8 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.

1.8.1 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.

1.8.2 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:

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

1.8.3 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.