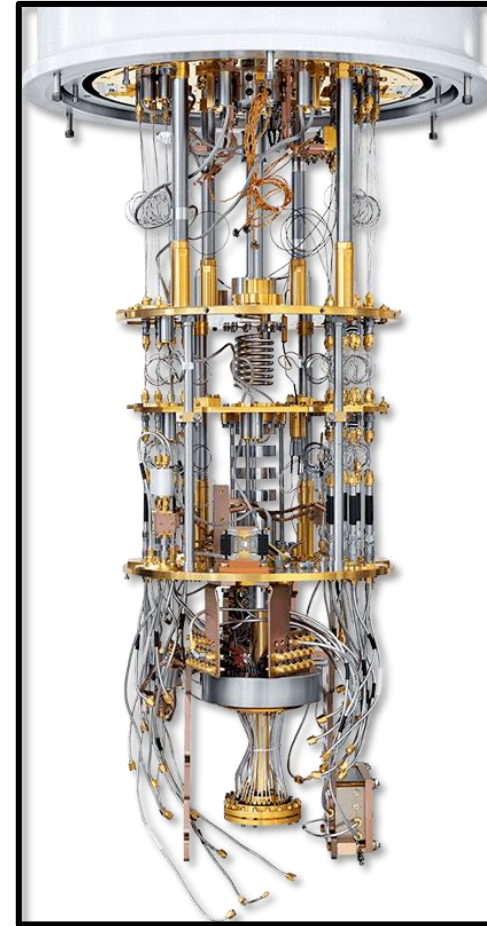


Recap

In the previous presentation, the basics of quantum computing were covered. First, we discussed what qubits are and how they form the foundation of quantum computing. Then, we studied different quantum gates.

We began with the Pauli X gate, which is equivalent to the NOT gate in traditional computers. Next, we examined the controlled-NOT (CNOT) gate, which serves as a source of non-linearity in quantum computers. The last gate we explored was the Hadamard gate, which transforms a qubit into a superposition state.

Finally, we built a simple quantum circuit consisting of two qubits, one Hadamard gate, and one CNOT gate.

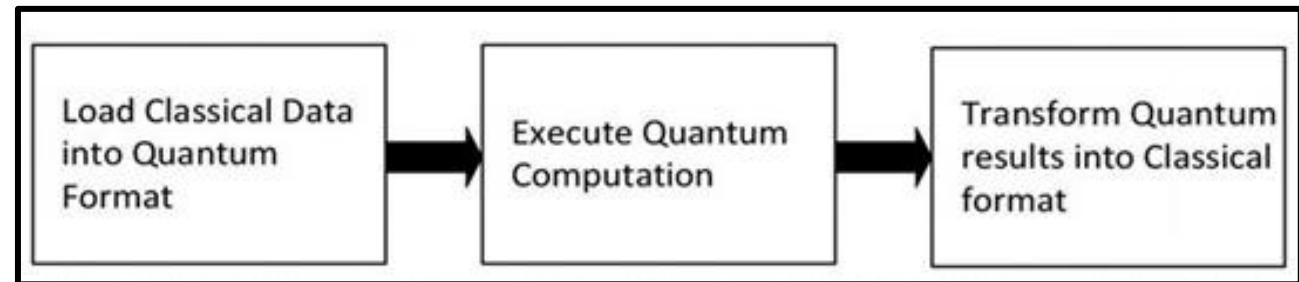


Steps to build a Quantum neural network

In this presentation, we will build a quantum neural network. The first step in constructing a neural network is to convert traditional data into quantum data. There are many different methods utilized to accomplish this goal; in this presentation, we will focus on the rotation method.

The next step is to define the network using quantum gates. It is important to understand what our parameters and hyperparameters are in this process.

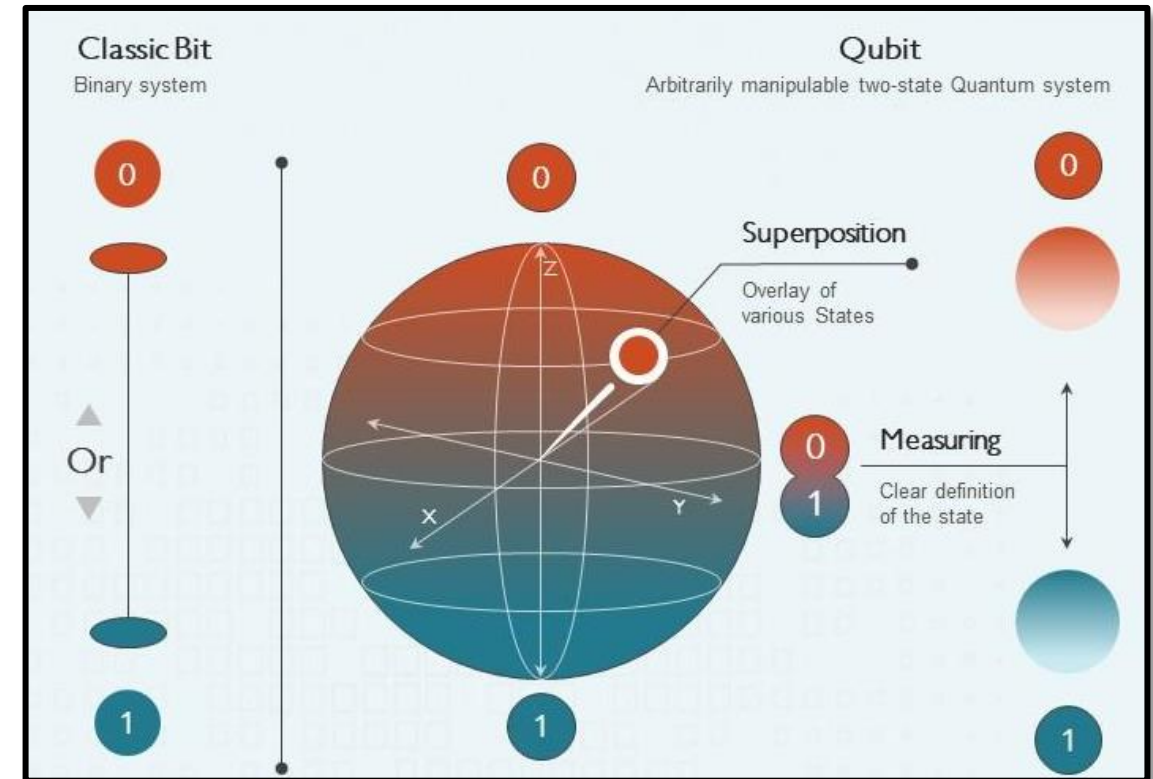
The final step is to train the model. In this step, we will perform forward propagation and backpropagation to optimize our parameters.



Convert Traditional Data into Quantum Data

In the previous presentation, we covered the importance of quantum decoherence. This phenomenon limits the number of qubits available to us in a quantum computer, so we must utilize these qubits wisely. Therefore, choosing the best method to convert our traditional data into qubits is crucial in quantum machine learning.

There are many different methods to convert traditional data into qubits. The method we will use in this presentation is the angle encoding. While there are more advanced and complex conversion techniques that might be more efficient, angle encoding method is chosen for its simplicity.

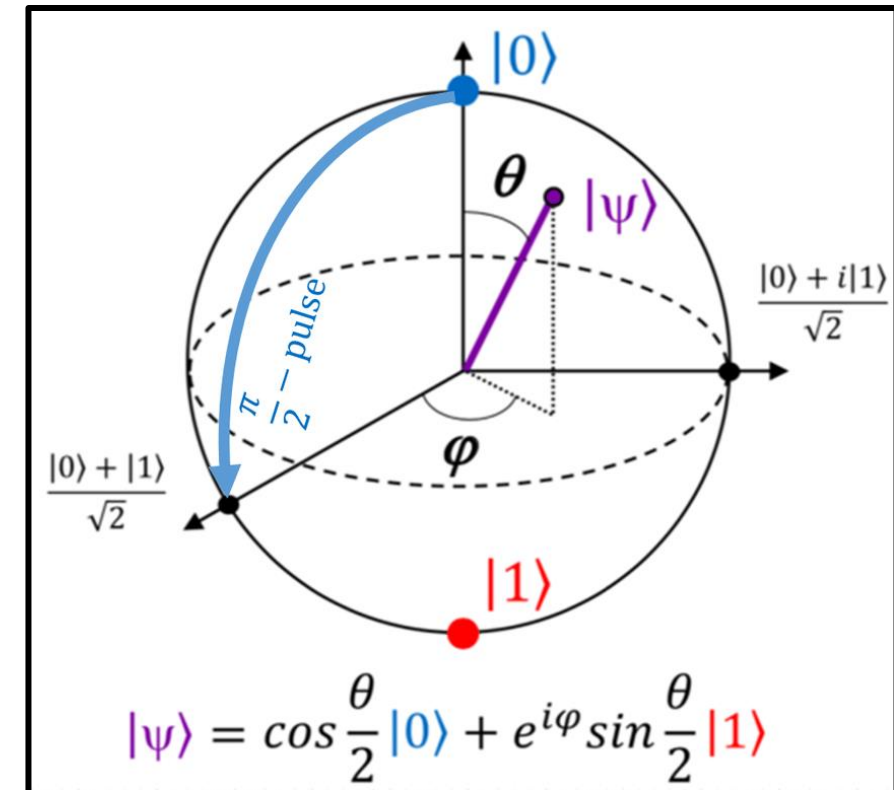


Revision of Bloch Sphere

Reviewing the Bloch sphere representation of the qubit is very helpful for understanding the rotation method.

We can visualize a qubit as a sphere, where its state can be located anywhere on this sphere. The upper pole of the sphere corresponds to the state $|0\rangle$, while the lower pole corresponds to the state $|1\rangle$. The points in between these two poles represent superposition, where the qubit can be in both states $|0\rangle$ and $|1\rangle$ simultaneously. We can only determine the value of the qubit after measurement.

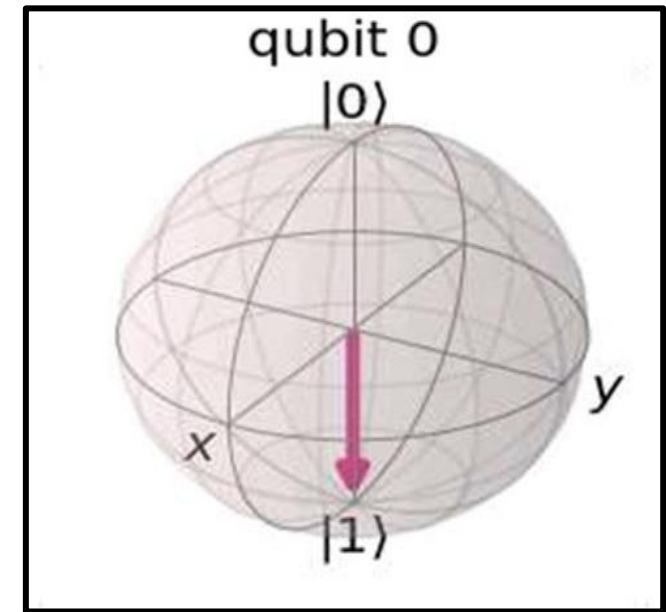
Rotating the qubit around different axes will result in different positions on the Bloch sphere, which corresponds to varying probabilities of measuring the qubit as $|0\rangle$ or $|1\rangle$.



Pauli X gate as a Rotation gate

To better understand what rotation means in the context of the Bloch sphere, we can examine the NOT gate from a different perspective. We can interpret the NOT gate as a rotation around the x-axis by π radians. These two operations are exactly equivalent.

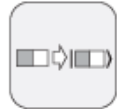

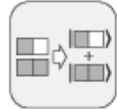
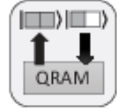

For example, if we apply the NOT gate to $|0\rangle$, we obtain $|1\rangle$. Similarly, if we rotate $|0\rangle$ around the x-axis by π radians, we will also arrive at $|1\rangle$.



Angle Encoding

Angle encoding is a method used in quantum computing to represent classical data in quantum states, particularly within quantum neural networks. This technique encodes classical information by utilizing rotation gates to manipulate qubits, allowing for efficient data representation and processing.

In angle encoding, classical data points are transformed into quantum states through a series of rotations around the Bloch sphere. Each classical data value is associated with a specific angle, which is used to define the state of a qubit.

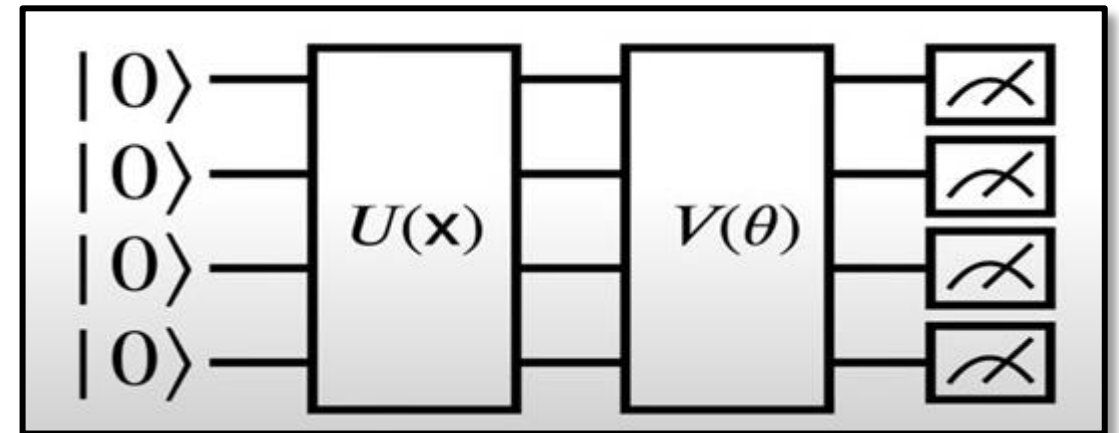
Encoding Pattern	Encoding	Req. Qubits
 BASIS ENCODING [1]	$x_i \approx \sum_{i=-k}^m b_i 2^i \mapsto b_m \dots b_{-k}\rangle$	$l = k+m$ per data-point
 ANGLE ENCODING	$x_i \mapsto \cos(x_i) 0\rangle + \sin(x_i) 1\rangle$	1 per data-point
 QUAM ENCODING [1]	$X \mapsto \sum_{i=0}^{n-1} \frac{1}{\sqrt{n}} x_i\rangle$	l
 QRAM ENCODING	$X \mapsto \sum_{i=0}^{n-1} \frac{1}{\sqrt{n}} i\rangle x_i\rangle$	$\lceil \log n \rceil + l$
 AMPLITUDE ENCODING [1]	$X \mapsto \sum_{i=0}^{n-1} x_i i\rangle$	$\lceil \log n \rceil$

Source: www.semanticscholar.org

Quantum Neural Network

A quantum neural network (QNN) consists of various components. The first component is responsible for performing angle encoding and is referred to as the encoding circuit. The QNN has several parameters, one of the most widely used being the rotation gate. The angles by which the qubits are rotated are the parameters we aim to optimize in QNNs.

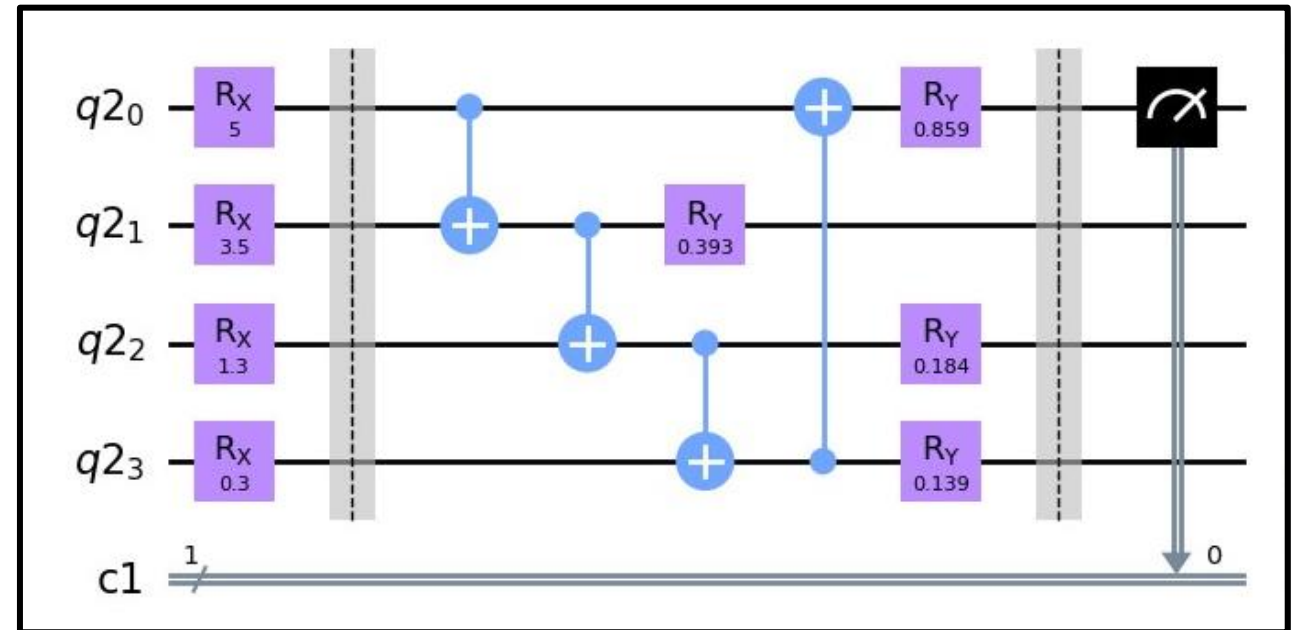
Similar to classical neural networks, we require sources of non-linearity in QNNs. We use CNOT gates and other similar gates to introduce this non-linearity. The circuit that includes these gates is called the variational circuit. Finally, we need to measure some of the qubits to generate our output.



QNN

In this slide, we will examine a sample QNN. In this network, we have four qubits, which means we have four features in our dataset. We also have one classical bit that we will use to store the measurement result.

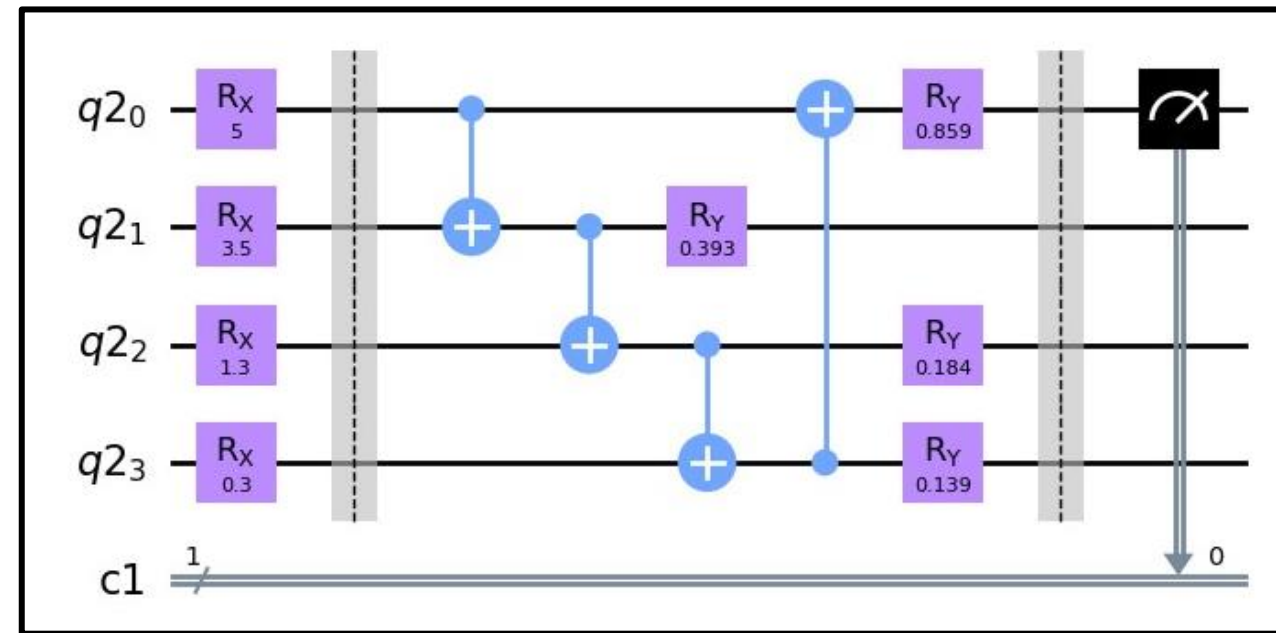
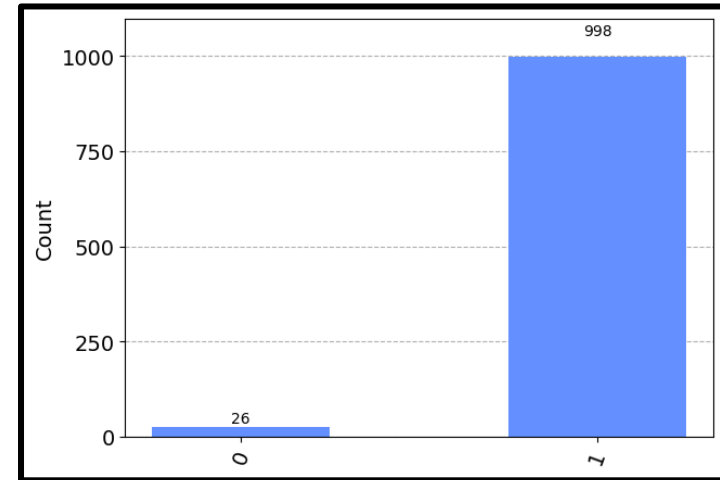
First, we have the encoding circuit, where we perform angle encoding. Next, we have our variational circuit. In this circuit, we will include a rotation gate around the y-axis for each qubit. The angles of these rotations serve as the parameters of the network. For every two consecutive qubits, we add a CNOT gate, which introduces non-linearity to our network. Finally, we measure the first qubit and store the result in the classical bit. This will be the output of our network.



Calculate Final Result

Imagine we use this network to solve a binary classification problem. We expect the network to output a probability that indicates how likely a given data point belongs to the first class.

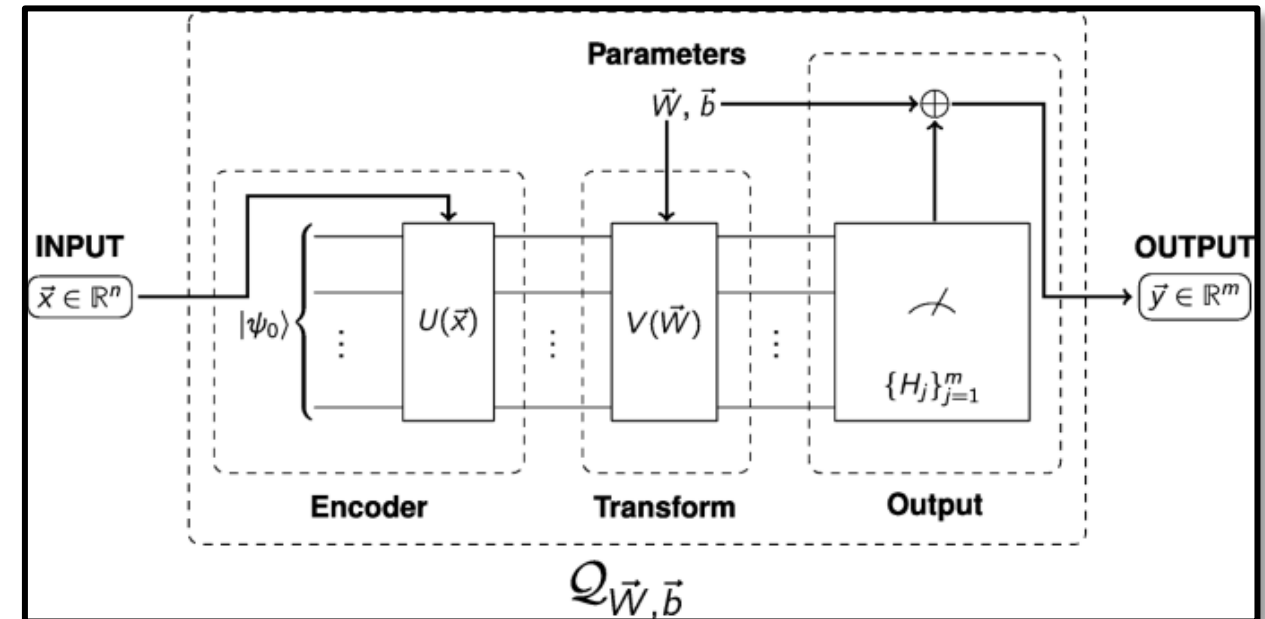
To calculate this probability using the QNN, we need to repeat the measurement process multiple times. This involves measuring the output several times to assess the likelihood of obtaining 0 as the output versus obtaining 1. Typically, we repeat this process more than 100,000 times. Afterward, we analyze the results using a histogram to determine the probability of each class.



Updating Parameters

The final step is adjusting the parameters after each iteration. Similar to classical neural networks, after making predictions, we need to calculate the loss function. Next, we compute the gradient of the loss function with respect to the parameters. We then take a small step in the direction of the gradient to update the parameters. This process is repeated until convergence is achieved, meaning that the parameters stabilize and the loss function reaches a minimum value.

It is important to note that the optimization techniques used in quantum neural networks may differ from those in classical networks due to the unique properties of quantum states. Adapting these techniques can enhance the performance of QNNs in various applications.



Summary

In this presentation, we covered the various components of a quantum neural network (QNN). We began with the angle encoding method, which is used to convert traditional data into qubits. Next, we focused on building the variational part of the network. This section is where we implement different gates to manipulate the data, enabling our network to perform its tasks effectively.

We utilize gates such as CNOT to introduce non-linearity into the network, while rotation gates serve as our parameters. Finally, we measure some qubits to generate our output. We then calculate the loss and update the parameters until convergence is achieved.

In the next presentation, we will demonstrate this entire process using Python and the Qiskit library, providing a hands-on approach to implementing a quantum neural network.