A Simple Model for Quantum Neural Network

林君錡 白曜瑋
Department of Physics,
National Taiwan University
Taipei, Taiwan

***Abstract*—In this project, we implement a modified model for quantum neural network (QNN) proposed in Ref. [1]. We will describe the model in terms of a quantum circuit and explain how it works mathematically. When the QNN is compared with its classical counterpart, we will see that the main difference between them lies in the nature of their activation functions.**

I. Introduction

Neural networks (NN) are composed of many non-linear components that mimic the learning mechanism of a human brain. The training is done by adjusting the weights and biases applied to the input parameters. The computational advantage (speed-up) of QNN over classical NN has been discussed in Ref. [1] and hence it will not be emphasized here. In this project, we simply focus on the implementation of our QNN and its mathematical formulation. The Task for our QNN is to categorize irises among the given data set imported from *sklearn*.

Before we go on to the formulation of our QNN model, keeping the classical counterpart in mind will help to understand the whole story: *The classical NN contains only* 1 *hidden layer, which has only* 2 *hidden neurons* (Of course one may consider more hidden neurons, but we use two for simplicity). For example, the diagram of a 2-input model is shown in Fig.1. Note that there are no weights or biases between the hidden layer and the output layer.

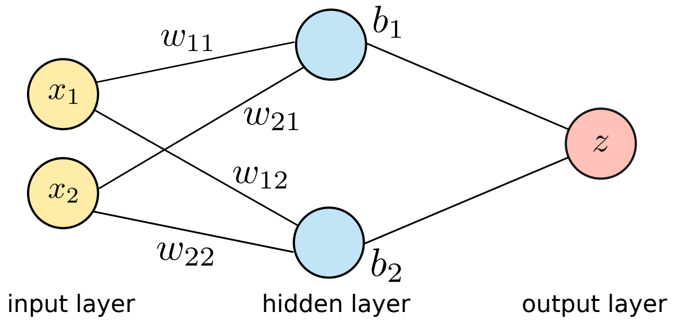

Fig. 1: Classical NN with 2 inputs and 2 hidden neurons

II. The Quantum Circuit for QNN

Fig.2 is a compact diagram of the QNN circuit. In the following sections, we will implement three models: one is a 2-input network *without* biases, another one is a 2-input network *with* biases, and the other is a 4-input network *with* biases.

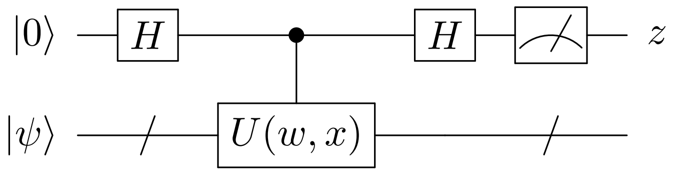

Fig. 2: The proposed single-output QNN

A. 2-input Unbiased QNN

Consider the operation

$$U_{x_1} \equiv \begin{bmatrix} e^{iw_{11}x_1} & 0 \\ 0 & e^{iw_{12}x_1} \end{bmatrix}. \tag{1}$$

To implement such a gate in *qiskit*, we have to decompose it into

$$U_{x_1} = X\, U_1(w_{11}x_1)\, X\, U_1(w_{12}x_1), \tag{2}$$

where $U_1(\theta)$ is one of the elementary operations in *qiskit* defined by

$$U_1(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}. \tag{3}$$

The gate $U(w,x)$ in the 2-input Unbiased QNN is therefore

$$U(w,x) = U_{x_1} \otimes U_{x_2} = \begin{bmatrix} e^{i\alpha} & 0 & 0 & 0 \\ 0 & e^{i\beta} & 0 & 0 \\ 0 & 0 & e^{i\gamma} & 0 \\ 0 & 0 & 0 & e^{i\delta} \end{bmatrix}, \tag{4}$$

where

$$\begin{cases} \alpha = w_{11}x_1 + w_{21}x_2 \\ \beta = w_{11}x_1 + w_{22}x_2 \\ \gamma = w_{12}x_1 + w_{21}x_2 \\ \delta = w_{12}x_1 + w_{22}x_2 \end{cases}. \tag{5}$$

A more detailed visualization of the circuit is shown in Fig.3.

Now we go through the circuit step by step. First, we initialize the qubits in the state

$$H\,|0\rangle \otimes H^{\otimes 2}\,|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{1}{\sqrt{4}} \sum_{j=0}^{3} |j\rangle. \tag{6}$$

After the controlled-$U(w,x)$ gate, the state becomes

$$\frac{1}{\sqrt{8}}\,|0\rangle \sum_{j=0}^{3} |j\rangle + \frac{1}{\sqrt{8}}\,|1\rangle \sum_{j=0}^{3} U(w,x)\,|j\rangle. \tag{7}$$

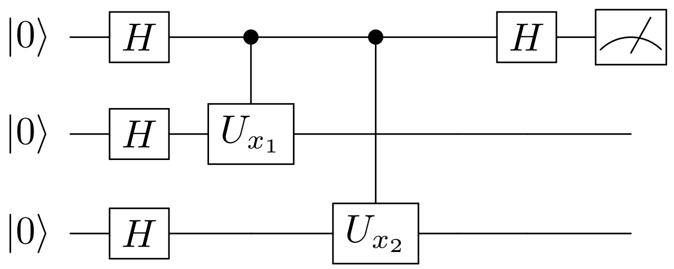

Fig. 3: The circuit for a 2-input unbiased QNN

Finally, after applying the Hadamard gate, we have

$$\begin{aligned}&\frac{|0\rangle+|1\rangle}{4}\sum_{j=0}^{3}|j\rangle+\frac{|0\rangle-|1\rangle}{4}\sum_{j=0}^{3}U(w,x)|j\rangle\\ &=\frac{|0\rangle}{4}\big[(1+e^{i\alpha})|0\rangle+(1+e^{i\beta})|1\rangle\\ &\quad+(1+e^{i\gamma})|2\rangle+(1+e^{i\delta})|3\rangle\big]+\\ &\quad\frac{|1\rangle}{4}\big[(1-e^{i\alpha})|0\rangle+(1-e^{i\beta})|1\rangle\\ &\quad+(1-e^{i\gamma})|2\rangle+(1-e^{i\delta})|3\rangle\big].\end{aligned}\tag{8}$$

If we measure the first qubit, then the probability that we obtain 1 will be

$$\begin{aligned}P^1=&\frac{1}{16}\big(\left|1-e^{i\alpha}\right|^2+\left|1-e^{i\beta}\right|^2\\ &+\left|1-e^{i\gamma}\right|^2+\left|1-e^{i\delta}\right|^2\big)\\ =&\frac{1}{8}\big(4-\cos\alpha-\cos\beta-\cos\gamma-\cos\delta\big).\end{aligned}\tag{9}$$

As for the output of the neural network, we simply let $z=P^1$ because our task is to classify 2 kinds of irises labeled 0 and 1 respectively. The QNN hence makes pedictions

$$\begin{cases}0, \text{ if } z\le 0.5\\ 1, \text{ if } z>0.5\end{cases}.$$

The workflow of what we have done so far is summarized in Fig. 4. From this, the main difference between our

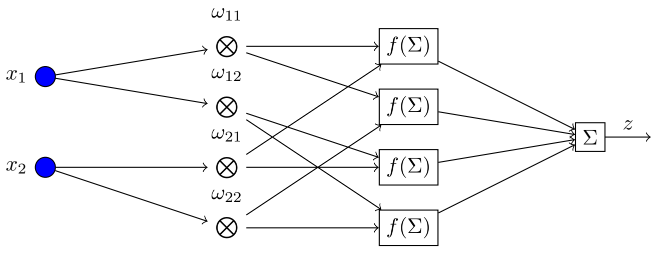

Fig. 4: The equivalent representation of QNN for two inputs and two weights for each input

QNN and its classical counterpart is clear: the activation functions are always periodic (i.e., always sine or cosine) in QNN.

Now the QNN has been constructed, so here comes the crucial part−−learning/training. We define the loss function by

$$L=\frac{1}{2s}\sum_{j=1}^{s}(z_j-d_j)^2,\tag{10}$$

the *mean squared error* (MSE), where s is the size of a batch (We train our QNN on batches.) and d is the target value (the desired output). The weights are updated according to

$$w_i\longrightarrow w_i-\eta\frac{\partial L}{\partial w_i},\tag{11}$$

where η is the learning rate. This whole process is called *stochastic gradient descent*. The derivative $\frac{\partial L}{\partial w_i}$ is easy to compute. For example,

$$\frac{\partial L}{\partial w_{11}}=\frac{1}{s}\sum_{j=1}^{s}(z_j-d_j)\frac{1}{8}\big(\sin\alpha_j+\sin\beta_j\big)x_1^j\tag{12}$$

How do we tackle those annoying sines appearing in $\frac{\partial L}{\partial w_i}$? Well, if we implement the same circuit with an additional phase gate S (as shown in Fig. 5), then we will have an output state

$$\begin{aligned}&\frac{|0\rangle}{4}\big[(1+ie^{i\alpha})|0\rangle+(1+ie^{i\beta})|1\rangle\\ &\quad+(1+ie^{i\gamma})|2\rangle+(1+ie^{i\delta})|3\rangle\big]+\\ &\frac{|1\rangle}{4}\big[(1-ie^{i\alpha})|0\rangle+(1-ie^{i\beta})|1\rangle\\ &\quad+(1-ie^{i\gamma})|2\rangle+(1-ie^{i\delta})|3\rangle\big].\end{aligned}\tag{13}$$

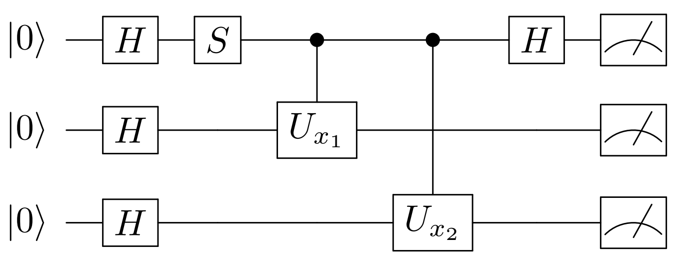

Fig. 5: The "learning" circuit

Since $\left|1-ie^{i\theta}\right|^2=2(1+\sin\theta)$, we have

$$\begin{cases}P_0^1=\frac{1}{8}(1+\sin\alpha)\\ P_1^1=\frac{1}{8}(1+\sin\beta)\\ P_2^1=\frac{1}{8}(1+\sin\gamma)\\ P_3^1=\frac{1}{8}(1+\sin\delta)\end{cases}.$$

The derivatives are hence

$$\begin{cases}\frac{\partial L}{\partial w_{11}}=\frac{1}{s}\sum(z_j-d_j)\left(P_0^1+P_1^1-\frac{1}{4}\right)x_1^j\\ \frac{\partial L}{\partial w_{12}}=\frac{1}{s}\sum(z_j-d_j)\left(P_2^1+P_3^1-\frac{1}{4}\right)x_1^j\\ \frac{\partial L}{\partial w_{21}}=\frac{1}{s}\sum(z_j-d_j)\left(P_0^1+P_2^1-\frac{1}{4}\right)x_2^j\\ \frac{\partial L}{\partial w_{22}}=\frac{1}{s}\sum(z_j-d_j)\left(P_1^1+P_3^1-\frac{1}{4}\right)x_2^j\end{cases}.$$

B. 2-input biased QNN

Biases may be easily included in our QNN by simply adding the gate

$$U_b=\begin{bmatrix}e^{ib_1}&0\\0&e^{ib_2}\end{bmatrix}.\tag{14}$$

That is,

$$U(w,b,x)=U_{x_1}\otimes U_{x_2}\otimes U_b.$$

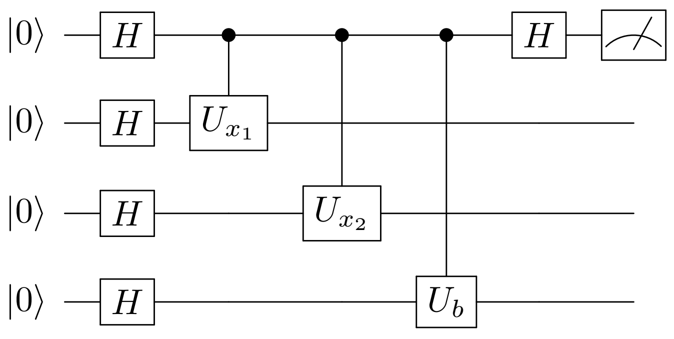

Fig. 6: The circuit for a 2-input biased QNN

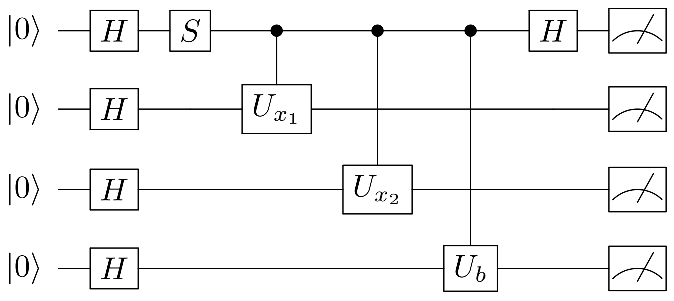

Fig. 7: The "learning" circuit

A more detailed visualization of the circuit is shown in Fig. 6. Through a similar circuit (shown in Fig. 7) as in the unbiased case, we can obtain

$$\begin{cases} \frac{\partial L}{\partial w_{11}} = \frac{1}{s}\sum\left(z_j - d_j\right)\left(P_0^1 + P_1^1 + P_2^1 + P_3^1 - \frac{1}{4}\right)x_1^j \\ \frac{\partial L}{\partial w_{12}} = \frac{1}{s}\sum\left(z_j - d_j\right)\left(P_4^1 + P_5^1 + P_6^1 + P_7^1 - \frac{1}{4}\right)x_1^j \\ \frac{\partial L}{\partial w_{21}} = \frac{1}{s}\sum\left(z_j - d_j\right)\left(P_0^1 + P_1^1 + P_4^1 + P_5^1 - \frac{1}{4}\right)x_2^j \\ \frac{\partial L}{\partial w_{22}} = \frac{1}{s}\sum\left(z_j - d_j\right)\left(P_2^1 + P_3^1 + P_6^1 + P_7^1 - \frac{1}{4}\right)x_2^j \\ \frac{\partial L}{\partial b_1} = \frac{1}{s}\sum\left(z_j - d_j\right)\left(P_0^1 + P_2^1 + P_4^1 + P_6^1 - \frac{1}{4}\right) \\ \frac{\partial L}{\partial b_2} = \frac{1}{s}\sum\left(z_j - d_j\right)\left(P_1^1 + P_3^1 + P_5^1 + P_7^1 - \frac{1}{4}\right) \end{cases}.$$

C. 4-input Biased QNN

For a 4-input biased QNN, the operation $U(w,b,x)$ is

$$U(w,b,x) = U_{x_1} \otimes U_{x_2} \otimes U_{x_3} \otimes U_{x_4} \otimes U_b. \tag{15}$$

The derivatives of L with respect to $\{w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32}, w_{41}, w_{42}, b_1, b_2\}$ contain the following factors

$$\begin{cases} P_0^1 + \cdots + P_{15}^1 - \frac{1}{4} \\ P_{16}^1 + \cdots + P_{31}^1 - \frac{1}{4} \\ P_0^1 + \cdots + P_7^1 + P_{16}^1 + \cdots + P_{23}^1 - \frac{1}{4} \\ P_8^1 + \cdots + P_{15}^1 + P_{24}^1 + \cdots + P_{31}^1 - \frac{1}{4} \\ P_0^1 + \cdots + P_3^1 + P_8^1 + \cdots + P_{11}^1 + \\ \quad P_{16}^1 + \cdots + P_{19}^1 + P_{24}^1 + \cdots + P_{27}^1 - \frac{1}{4} \\ P_4^1 + \cdots + P_7^1 + P_{12}^1 + \cdots + P_{15}^1 + \\ \quad P_{20}^1 + \cdots + P_{23}^1 + P_{28}^1 + \cdots + P_{31}^1 - \frac{1}{4} \\ P_0^1 + P_1^1 + P_4^1 + P_5^1 + \cdots + P_{28}^1 + P_{29}^1 - \frac{1}{4} \\ P_2^1 + P_3^1 + P_6^1 + P_7^1 + \cdots + P_{30}^1 + P_{31}^1 - \frac{1}{4} \\ P_0^1 + P_2^1 + P_4^1 + P_6^1 + \cdots + P_{28}^1 + P_{30}^1 - \frac{1}{4} \\ P_1^1 + P_3^1 + P_5^1 + P_7^1 + \cdots + P_{29}^1 + P_{31}^1 - \frac{1}{4} \end{cases}$$

respectively.

III. Experimental results

The iris data set consists of 3 classes, each of which contains 50 instances, but only two classes (labeled 0 and 1) were used. We mapped the inputs into the range $[0,1]$, and randomly split out 80% of the samples as the training data; the remaining 20% are used for testing the models. The results of our models are listed in TABLE I. The full program is available at https://github.com/qqqquincy/QC (Notebook version available at https://bit.ly/2IKV4fX).

TABLE I: Accuracy of the 5 implemented models

Model	Selected features	Training Accuracy	Testing Accuracy
Unbiased	Petal width & length	100%	100%
	Sepal width & length	83.75%	95%
Biased	Petal width & length	100%	100%
	Sepal width & length	100%	95%
	Petal & Sepal width & length	88.75%	100%

IV. Conclusion

From the results of the above models, we can see that all of them are able to accurately distinguish the two kinds of irises. This QNN model is easy to implement, but it might not work efficiently. During the learning process, the computation is done both on a quantum computer and a classical one. Switching between the two devices might slow down the learning. Therefore, it is worth studying whether a neural network can be run completely on a quantum computer.

Also, the number of neurons is limited by the number of available qubits. The more features of input data and hidden neurons we want to incorporate into our QNN, the more qubits we will have to use. It is hence clear that this simple model cannot deal with high-dimensional input data (e.g., the famous 28×28 handwriting pictures from MNIST) or a large amount of hidden neurons efficiently.

References

[1] A. Daskin, "A simple quantum neural net with a periodic activation function," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2018, pp. 2887–2891.