A SIMPLE MODEL FOR QUANTUM NEURAL NETWORK

Daskin, Ammar.

2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2018.

Presenter: 物理四 白曜瑋

物理四 林君錡

OUTLINE

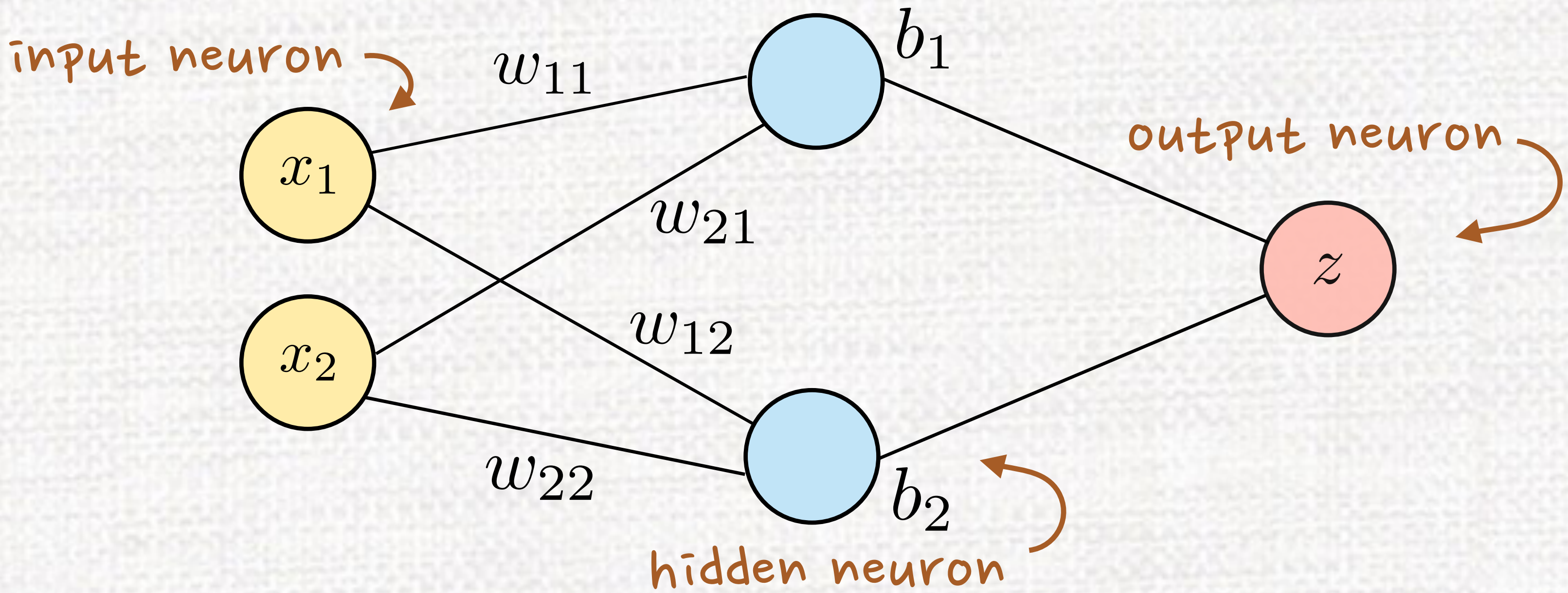
- Classical neural network (NN)
  - Stochastic gradient descent (SGD)
  - Backpropagation
- Quantum Neural Network
  - Two-input unbiased QNN
  - Two-input biased QNN
  - Four-input biased QNN

WHAT IS QUANTUM NEURAL NETWORK (QNN)?

- We will describe the model in terms of a quantum circuit and explain how it works mathematically.
- We implement a modified model for quantum neural network (QNN) proposed in Ref. [1], and run on Qiskit simulator.
- 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.
- In order to make the the concept of QNN clear, let's first introduce how a classical neural network (NN) works.

NEURAL NETWORKS (NN)

- 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.



- In order to train our network, we have to define a loss function. A typical choice is the mean squared error (MSE)

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

- It should be minimized during the training process.

STOCHASTIC GRADIENT DESCENT (SGD)

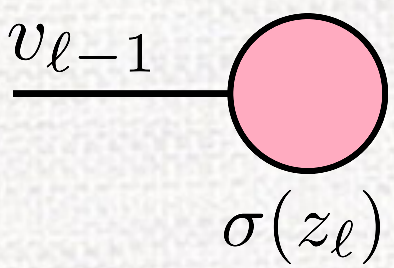
- A method named stochastic gradient descent (SGD) can help to speed up this process by limiting the calculation to a small randomly chosen subset of the input data, and the gradient can be calculated approximately:

$$\nabla L \approx \frac{1}{m} \sum_{x_k}^{m} \nabla L_{x_k} \quad \Rightarrow \quad w_i \to w_i' = w_i - \frac{\eta}{m} \sum_k \frac{\partial L_{x_k}}{\partial w_i}$$

- Such a small subset is usually called a mini-batch.
- Due to the special structure of neural network, the gradient can be, in fact, calculated in a very efficient way. This method is called backpropagation.

BACK PROPAGATION

- Let's explain how it works by starting from the ending (output) layer of the network with only one neuron. For a given data point *x* and target *t*, consider the following small variation:



$$v_\ell = \sigma(z_\ell) = \sigma(w_\ell v_{\ell-1} + b_\ell)$$

$$\delta_\ell = \frac{\partial L}{\partial z_\ell} \quad \text{where} \quad L = \frac{1}{2}(v_\ell - t)^2$$

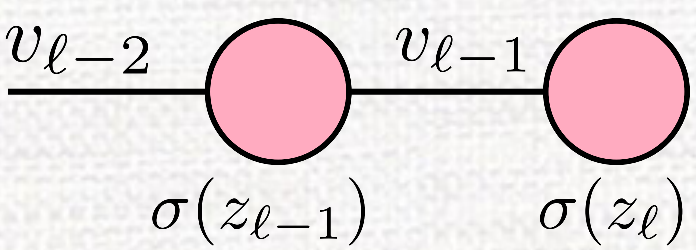
$$\Rightarrow \delta_\ell = \frac{\partial L}{\partial v_\ell}\frac{\partial v_\ell}{\partial z_\ell} = [\sigma(z_\ell) - t] \cdot \sigma'(z_\ell)$$

$$\begin{cases} \dfrac{\partial L}{\partial b_\ell} = \delta_\ell \\ \dfrac{\partial L}{\partial w_\ell} = \delta_\ell v_{\ell-1} \end{cases}$$

The differentials at the ending layer can be calculated easily!

BACK PROPAGATION (II)

- Then propagate back by another layer of single neuron:



$$v_\ell = \sigma(w_\ell v_{\ell-1} + b_\ell) = \sigma(w_\ell \sigma(z_{\ell-1}) + b_\ell)$$

$$\delta_{\ell-1} = \frac{\partial L}{\partial z_{\ell-1}} = \frac{\partial L}{\partial z_\ell}\frac{\partial z_\ell}{\partial z_{\ell-1}} = \delta_\ell w_\ell \sigma'(z_{\ell-1})$$

$$\text{since} \quad z_\ell = w_\ell \sigma(z_{\ell-1}) + b_\ell$$

$$\begin{cases} \dfrac{\partial L}{\partial b_{\ell-1}} = \delta_{\ell-1} \\ \dfrac{\partial L}{\partial w_{\ell-1}} = \delta_{\ell-1} v_{\ell-2} \end{cases}$$

Basically the calculations of the differentials are the same as the ending layer!

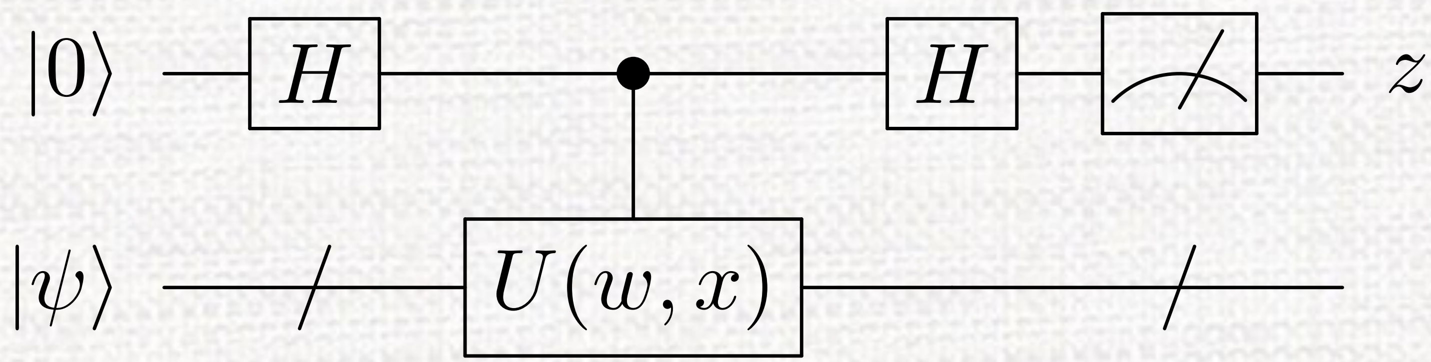
Based on this the gradient can be calculated, back propagated from the ending layer. And the feedforward calculation is performed only once!

IRIS FLOWER DATASET

- 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.

	Attribute Information
1	sepal length
2	sepal width
3	petal length
4	petal width

QUANTUM NEURAL NETWORK



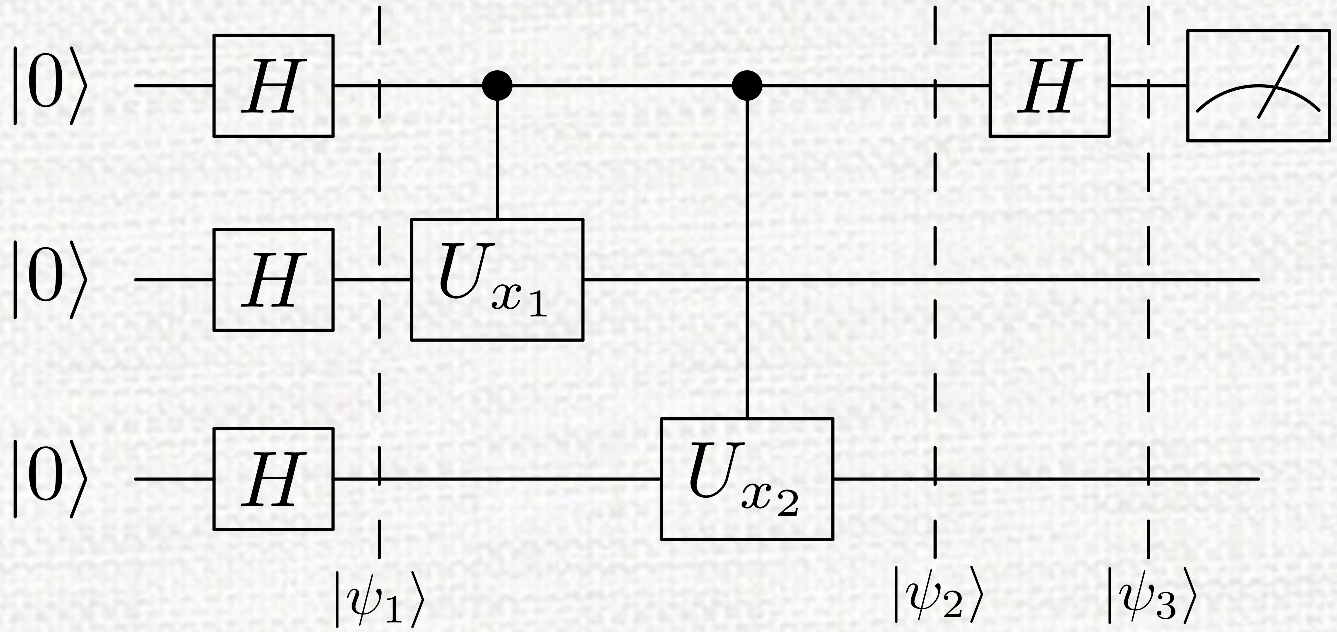
$$U_{x_1} \equiv \begin{bmatrix} e^{iw_{11}x_1} & 0 \\ 0 & e^{iw_{12}x_1} \end{bmatrix}$$

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

$$\text{where} \quad \begin{cases} \alpha = w_{11}x_1 + w_{21}x_2 \\ \beta = w_{11}x_1 + w_{22}x_2 \\ \gamma = w_{12}x_1 + w_{21}x_2 \\ \delta = w_{12}x_1 + w_{22}x_2 \end{cases}$$

- We will implement three models, and select different features to perform five experiments:
  - 2-input network without biases
  - 2-input network with biases
  - 4-input network with biases

[MODEL 1] TWO-INPUT UNBIASED QNN



$$|\psi_1\rangle = H|0\rangle \otimes H^{\otimes 2}|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{1}{\sqrt{4}} \sum_{j=0}^{3} |j\rangle$$

$$|\psi_2\rangle = \frac{1}{\sqrt{8}}|0\rangle \sum_{j=0}^{3}|j\rangle + \frac{1}{\sqrt{8}}|1\rangle \sum_{j=0}^{3} U(w,x)|j\rangle$$

$$|\psi_3\rangle = \frac{|0\rangle + |1\rangle}{4} \sum_{j=0}^{3}|j\rangle + \frac{|0\rangle - |1\rangle}{4} \sum_{j=0}^{3} U(w,x)|j\rangle$$

$$= \frac{|0\rangle}{4}\left[(1+e^{i\alpha})|0\rangle + (1+e^{i\beta})|1\rangle + (1+e^{i\gamma})|2\rangle + (1+e^{i\delta})|3\rangle\right] + \frac{|1\rangle}{4}\left[(1-e^{i\alpha})|0\rangle + (1-e^{i\beta})|1\rangle + (1-e^{i\gamma})|2\rangle + (1-e^{i\delta})|3\rangle\right]$$

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

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

Activation functions are periodic in QNN!

- Let $z = P^1$ for the output, QNN hence makes predictions

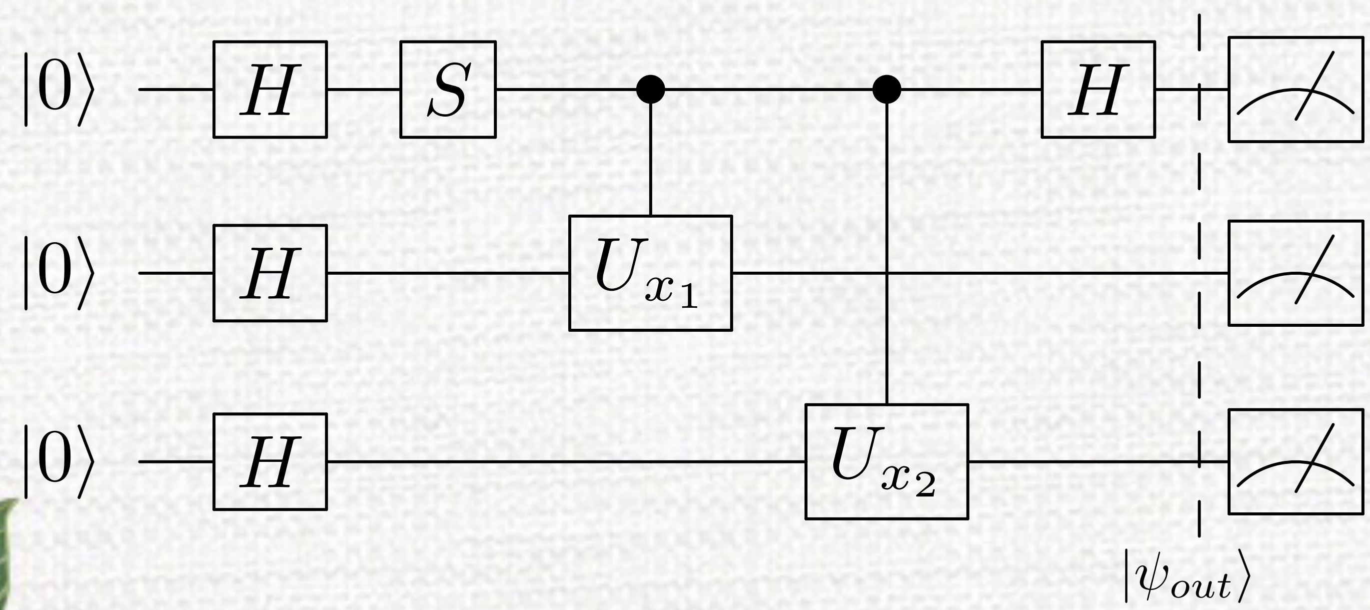
$$\begin{cases} 0, \text{ if } z \le 0.5 \\ 1, \text{ if } z > 0.5 \end{cases}$$

[MODEL 1] THE "LEARNING" CIRCUIT

- The weights are updated according to stochastic gradient descent.

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

- We can tackle the annoying sines appearing in gradient by implementing the "learning" circuit.



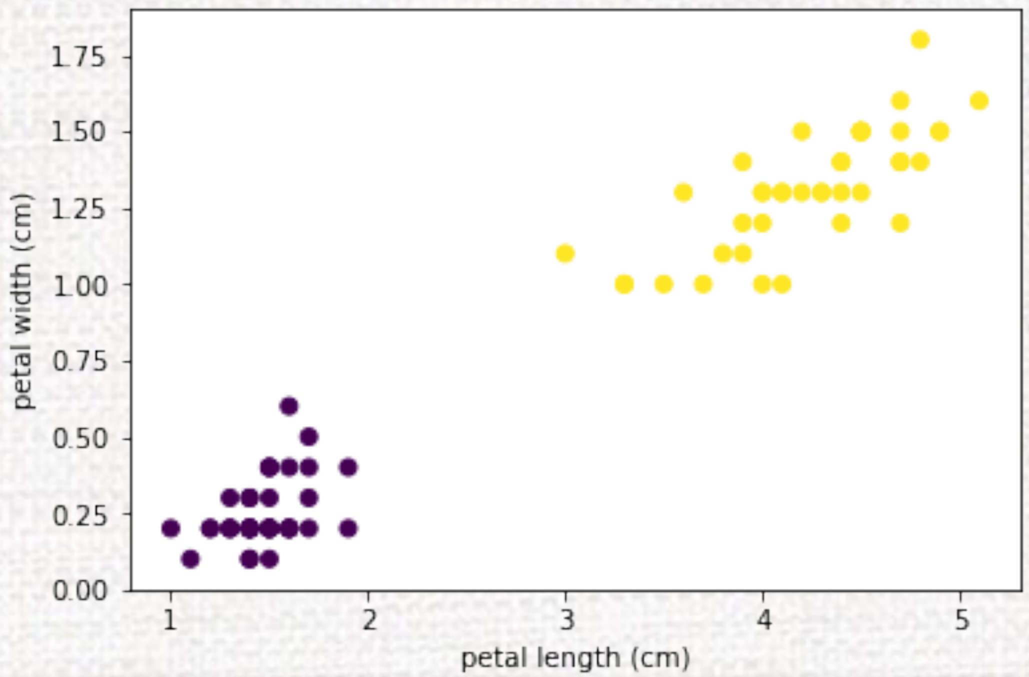
- The derivatives are hence

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

$$\begin{aligned} |\psi_{out}\rangle = \frac{|0\rangle}{4}\Big[&(1+ie^{i\alpha})|0\rangle + (1+ie^{i\beta})|1\rangle + (1+ie^{i\gamma})|2\rangle + (1+ie^{i\delta})|3\rangle\Big] + \\ \frac{|1\rangle}{4}\Big[&(1-ie^{i\alpha})|0\rangle + (1-ie^{i\beta})|1\rangle + (1-ie^{i\gamma})|2\rangle + (1-ie^{i\delta})|3\rangle\Big] \end{aligned}$$

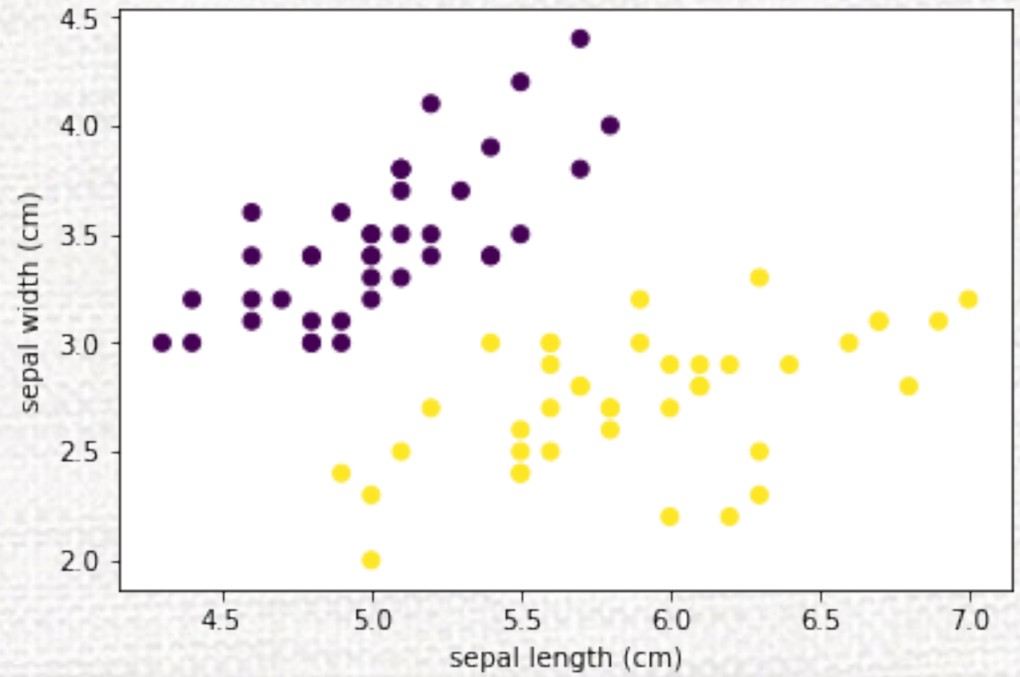
[MODEL 1] IMPLEMENTATION ON QISKIT SIMULATOR

- EXP #1
  - Input features: Petal width & length




```
Epoch 0: Loss = 0.0206013620
Epoch 1: Loss = 0.0115800500
Epoch 2: Loss = 0.0147680640
Epoch 3: Loss = 0.0139822662
Epoch 4: Loss = 0.0157027543
Training Acc = 100.00%
Testing Acc = 100.00%
```

- EXP #2
  - Input features: Sepal width & length




```
Epoch 0: Loss = 0.1956299543
Epoch 1: Loss = 0.1199962795
Epoch 2: Loss = 0.0942521095
Epoch 3: Loss = 0.0850783587
Epoch 4: Loss = 0.0947985053
Training Acc = 83.75%
Testing Acc = 95.00%
```

[MODEL 2] TWO-INPUT BIASED QNN

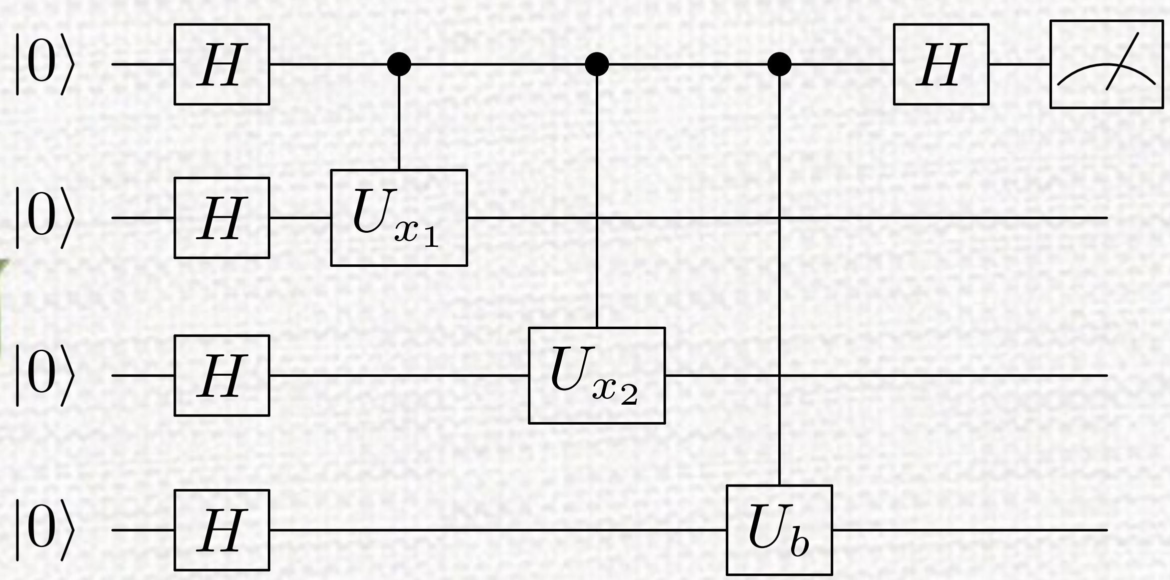
- Biases may be easily included in our QNN by simply adding the gate

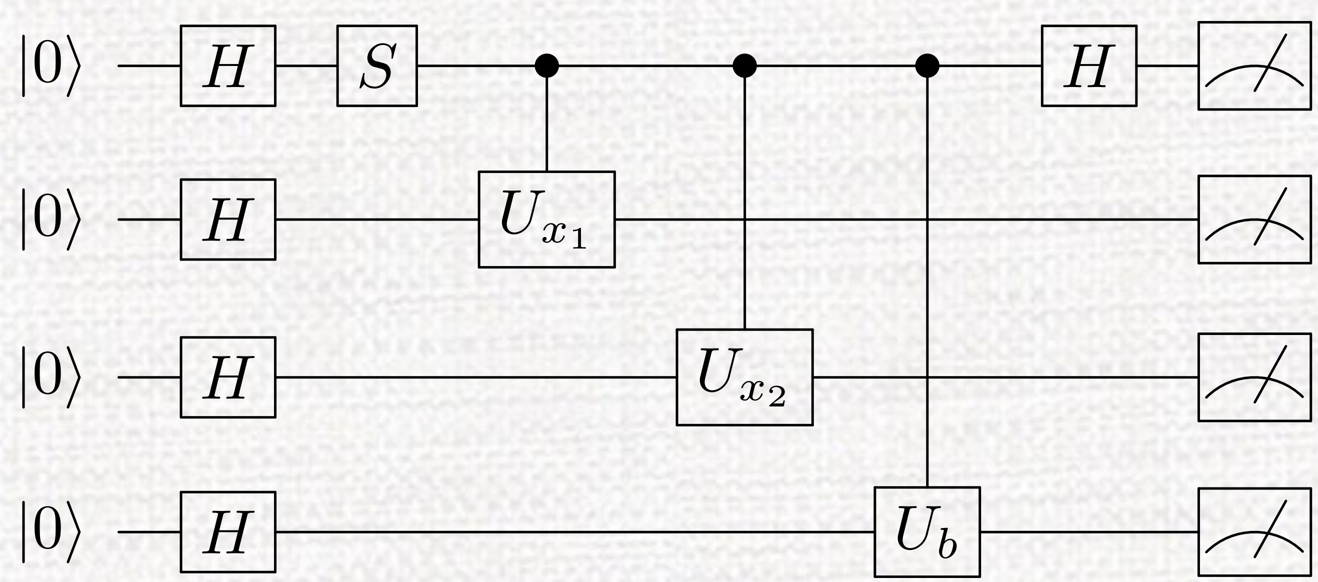
$$U_b = \begin{bmatrix} e^{ib_1} & 0 \\ 0 & e^{ib_2} \end{bmatrix}$$

- That is,

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

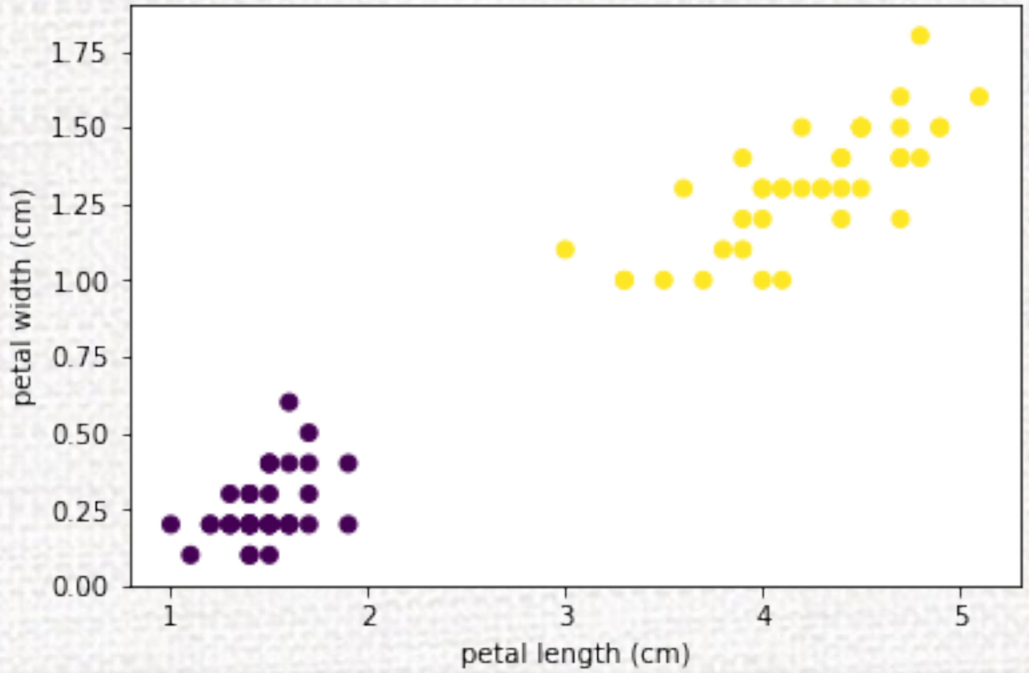
- The QNN and "learning" circuits are





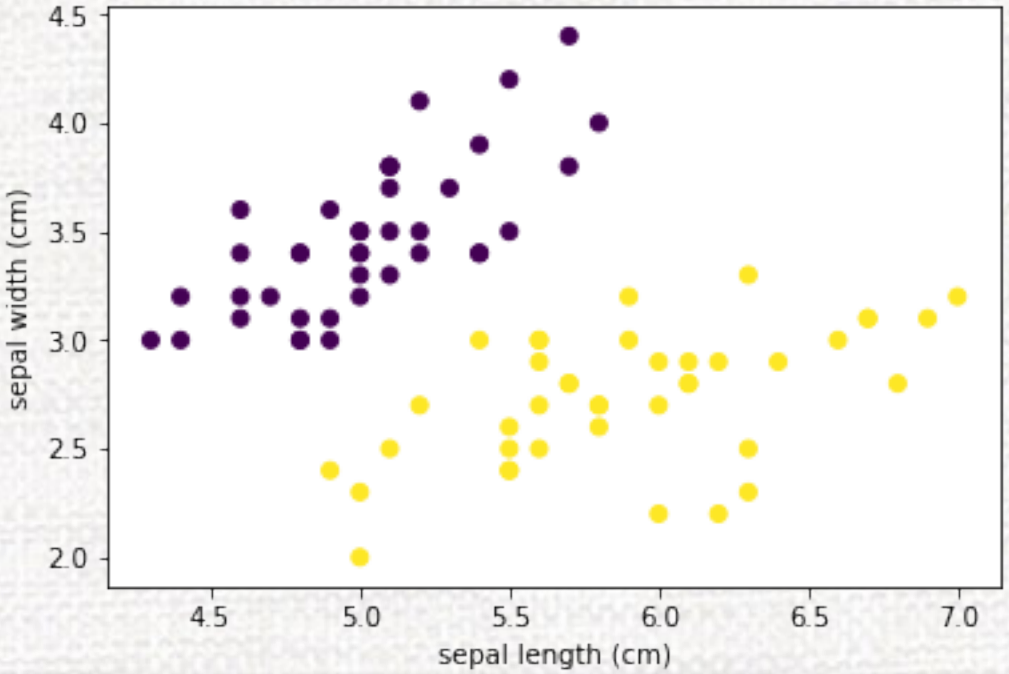
[MODEL 2] IMPLEMENTATION ON QISKIT SIMULATOR

- EXP #3
  - Input features: Petal width & length




```
Epoch 0: Loss = 0.1664802730
Epoch 1: Loss = 0.0644763708
Epoch 2: Loss = 0.0565638542
Epoch 3: Loss = 0.0464445651
Epoch 4: Loss = 0.0467461050
Training Acc = 100.00%
Testing Acc = 100.00%
```

- EXP #4
  - Input features: Sepal width & length




```
Epoch 0: Loss = 0.1533004761
Epoch 1: Loss = 0.0977748871
Epoch 2: Loss = 0.0883823156
Epoch 3: Loss = 0.0855589151
Epoch 4: Loss = 0.0902338982
Training Acc = 100.00%
Testing Acc = 95.00%
```

[MODEL 3] FOUR-INPUT BIASED QNN

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

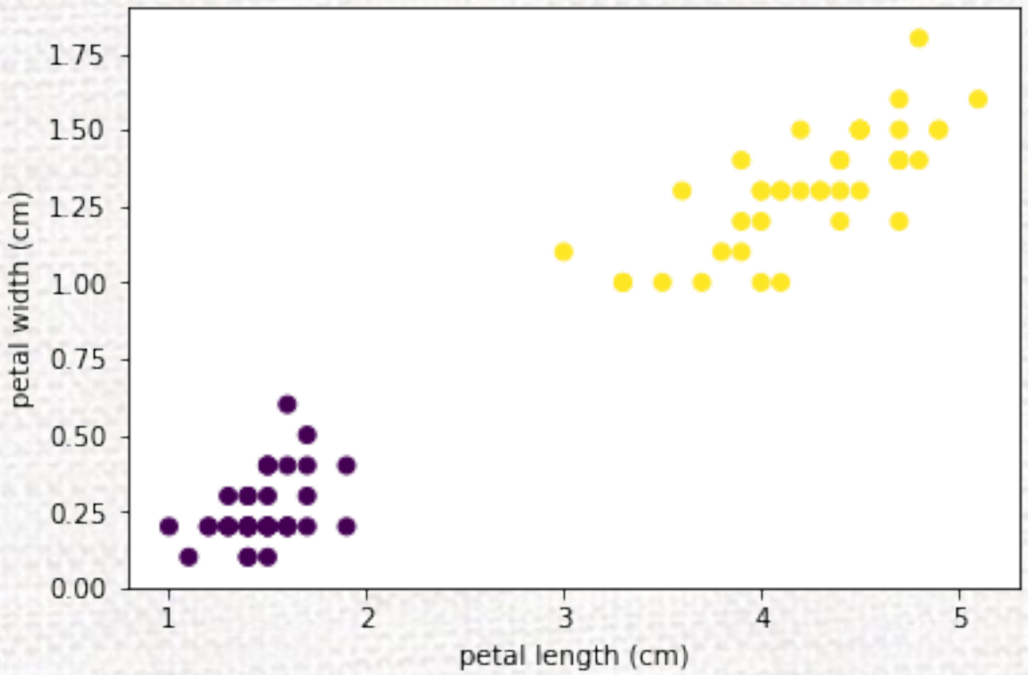
$$U(w,b,x) = U_{x_1} \otimes U_{x_2} \otimes U_{x_3} \otimes U_{x_4} \otimes U_b.$$

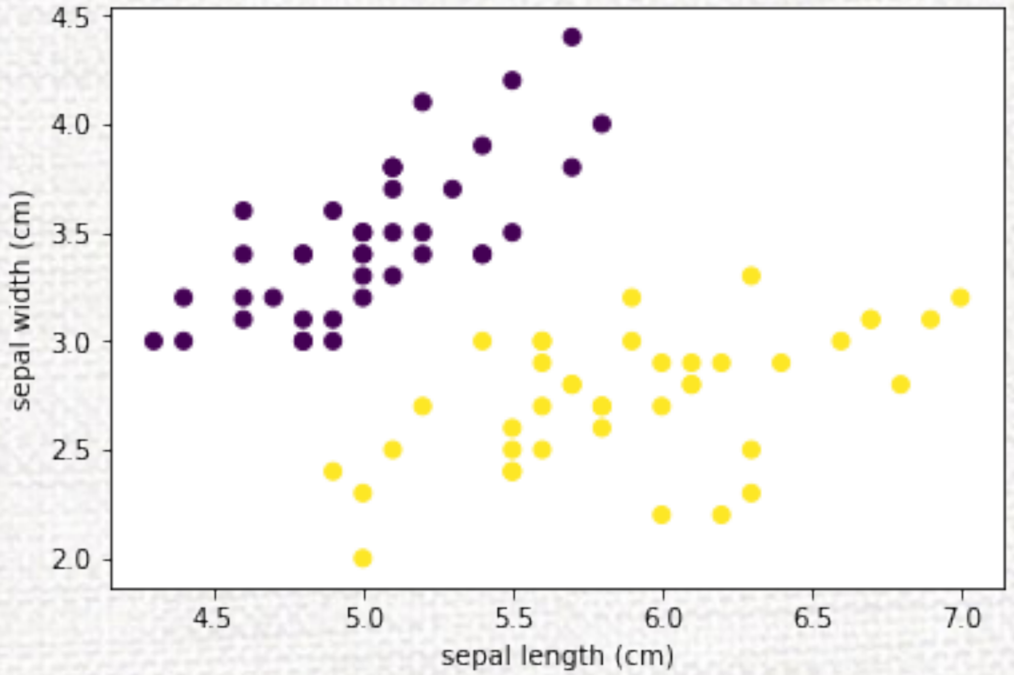
- The derivatives of L with respect to $\{w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32}, w_{41}, w_{42}, b_1, b_2\}$ contain the following factors

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

[MODEL 3] IMPLEMENTATION ON QISKIT SIMULATOR

- EXP #5
  - Input features: Petal width & length, Sepal width & length






```
Epoch 0: Loss = 0.0965277195
Epoch 1: Loss = 0.1126693964
Epoch 2: Loss = 0.0997015715
Epoch 3: Loss = 0.0907203436
Epoch 4: Loss = 0.0824208975
Training Acc = 88.75%
Testing Acc = 95.00%
```

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.
- 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.
- 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.

REFERENCE

- 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.