EMBEDDED SYSTEMS WEEK
OCTOBER 10-15, 2021 | VIRTUAL CONFERENCE

nature COMMUNICATIONS

Tutorial on QuantumFlow: A Co-Design Framework of Neural Network and Quantum Circuit towards Quantum Advantage

Weiwen Jiang, Ph.D.

Electrical and Computer Engineering

George Mason University

wjiang8@gmu.edu

Agenda

- **Background and Motivation**
  - What is machine learning
  - Why using quantum computer
  - Our goals
- **General Framework and Case Study (Tutorial 1-2 on GitHub)**
  - Implementing neural network accelerators: from classical to quantum
  - A case study on MNIST dataset
- **Optimization towards Quantum Advantage (Tutorial 3-4 on GitHub)**
  - The existing challenges
  - The proposed co-design framework: QuantumFlow

Background and Motivation

What is Machine Learning?

Supervised Learning

Example: Classification

Training

Given: Labeled data as training dataset

(x_i, y_i): x_i training data, y_i: label

$x_i =$  $y_i = 3$

Output: A learned function f from X to Y

$f: x \mapsto y$

Inference/Execution

Given: Unseen data test dataset
A learned function f

Do: $f($  $) = 3$

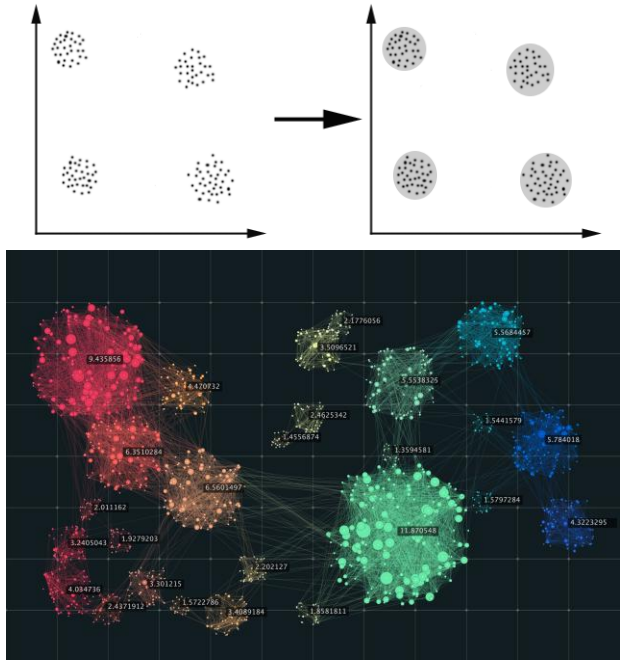
Unsupervised Learning

Example: Clustering

Given: Unlabeled data

(x_i)

Goal: discover the "natural groupings" present in the data

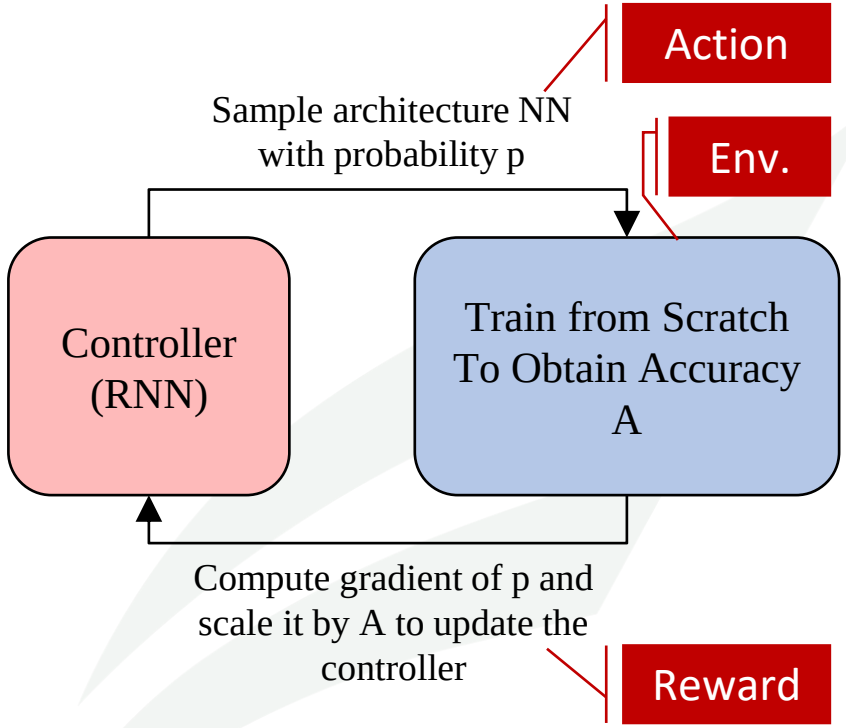


Reinforcement Learning

Example: Neural Architecture Search

Given: An environment that can give us reward based on our action

Goal: Maximize the expected rewards



What is Machine Learning? --- Our Focus

Supervised Learning

Example: Classification

Training

Given: Labeled data as training dataset

(x_i, y_i): x_i training data, y_i: label

$x_i =$  $y_i = 3$

Output: A learned function f from X to Y

$f: x \mapsto y$

Inference/Execution

Given: Unseen data test dataset
A learned function f

Do: $f($  $) = 3$

Unsupervised Learning

Example: Clustering

Given: Unlabeled data

(x_i)

Goal: discover the "natural groupings" present in the data

Reinforcement Learning

Example: Neural Architecture Search

Given: An environment that can give us reward based on our action

Goal: Maximize the expected rewards

What is Neural Network?

Supervised Learning

Example: Classification

Training

Given: Labeled data as training dataset

(x_i, y_i): x_i training data, y_i: label

$x_i =$  $y_i = 3$

Output: A learned function f from X to Y

$f: x \mapsto y$

Inference/Execution

Given: Unseen data test dataset

A learned function f

Do: $f($  $) = 3$

An unknown classification function: g

$$y = g(x);\ s.t.\ y_i = g(x_i)$$

Learn a function f with parameters θ, b to approximate g:

$$\hat{y} = f(x, \theta, b)$$

Training is to minimize the loss function by adjusting parameters θ, b

$$min:\ \mathcal{L}(f) = \sum_i (f(x_i, \theta, b) - y_i)$$

Perceptron model, where σ is a non-linear function:

$$\hat{y} = \sigma(\theta x + b)$$

Feedforward neural network:

$$l_1 = \sigma_1(\theta_1 x + b_1)$$
$$l_2 = \sigma_2(\theta_2 l_1 + b_2)$$
$$\dots\ \dots$$
$$l_n = \sigma_n(\theta_n l_{n-1} + b_n)$$
$$\hat{y} = classifier(l_n)$$

What is Neural Network?

Supervised Learning

Example: Classification

Training

Given: Labeled data as training dataset

(x_i, y_i): x_i training data, y_i: label

$x_i =$ $y_i = 3$

Output: A learned function f from X to Y

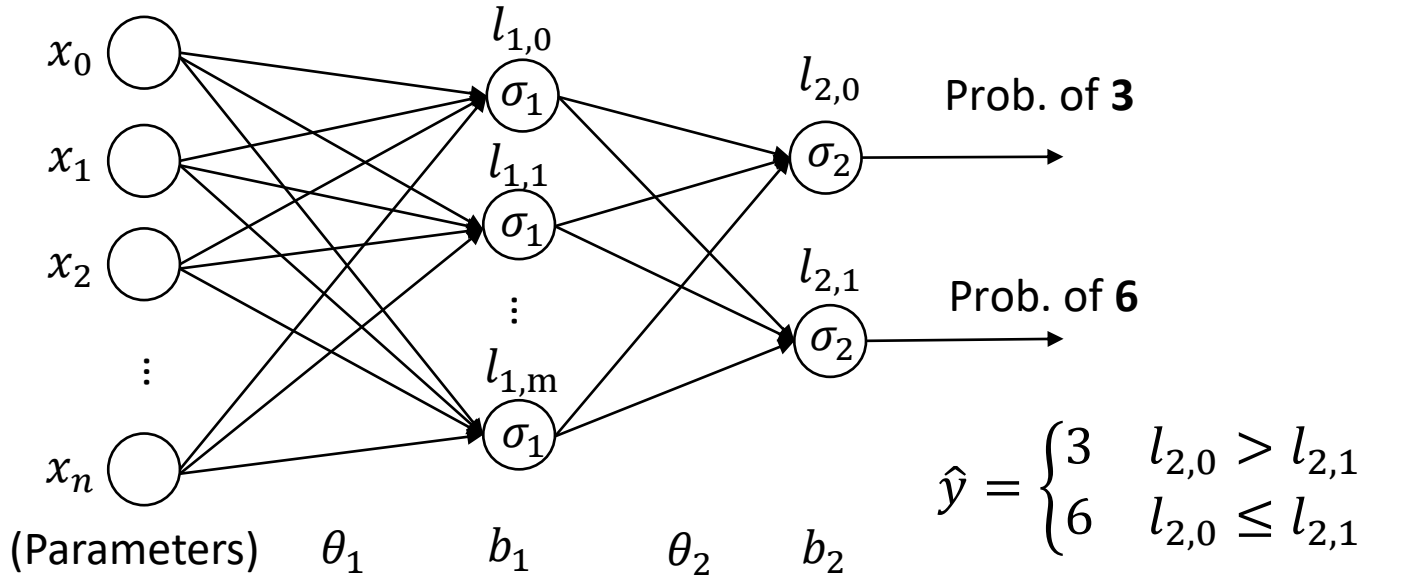
$f: x \mapsto y$

Inference/Execution

Given: Unseen data test dataset

A learned function f

Do: $f($  $) = 3$



Example of feedforward neural network for $n = 2$

Perceptron model, where $\boldsymbol{\sigma}$ is a non-linear function:

$$\hat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{\theta x} + \boldsymbol{b})$$

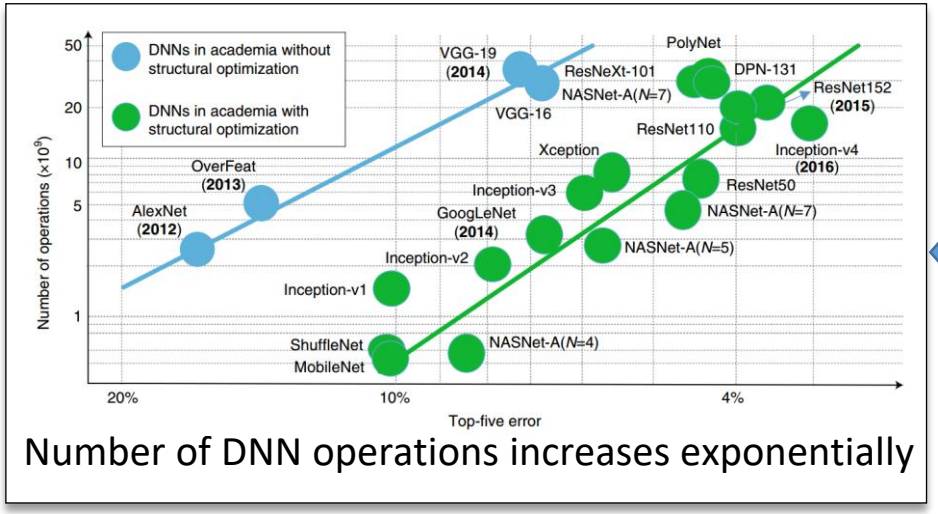
Feedforward neural network:

$$\boldsymbol{l_1} = \boldsymbol{\sigma_1}(\boldsymbol{\theta_1 x} + \boldsymbol{b_1})$$
$$\boldsymbol{l_2} = \boldsymbol{\sigma_2}(\boldsymbol{\theta_2 l_1} + \boldsymbol{b_2})$$
$$\dots\ \dots$$
$$\boldsymbol{l_n} = \boldsymbol{\sigma_n}(\boldsymbol{\theta_n l_{n-1}} + \boldsymbol{b_n})$$
$$\hat{\boldsymbol{y}} = \boldsymbol{classifier}(\boldsymbol{l_n})$$

Why Using Quantum Computer for Machine Learning?

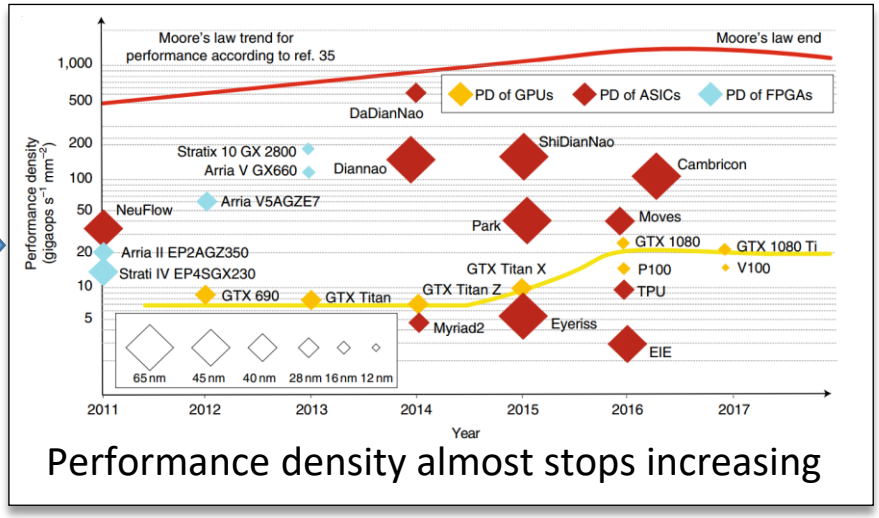
- Imbalanced “demand and supply” of NN on classical computing
- The growing power of quantum computing
- Linear algebra is central for both quantum computing and machine learning

NN on Classical Computer: Computation & Storage Demand > Supply



Number of DNN operations increases exponentially

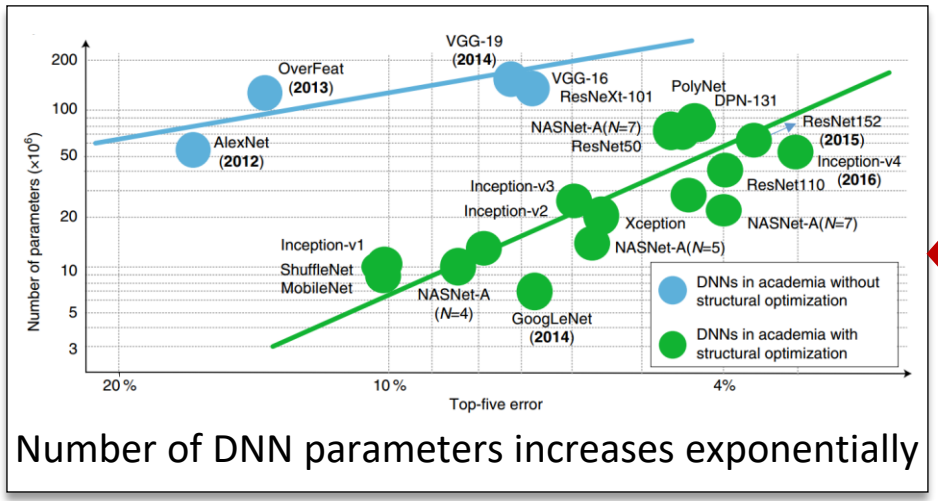
Computation Gap



Performance density almost stops increasing

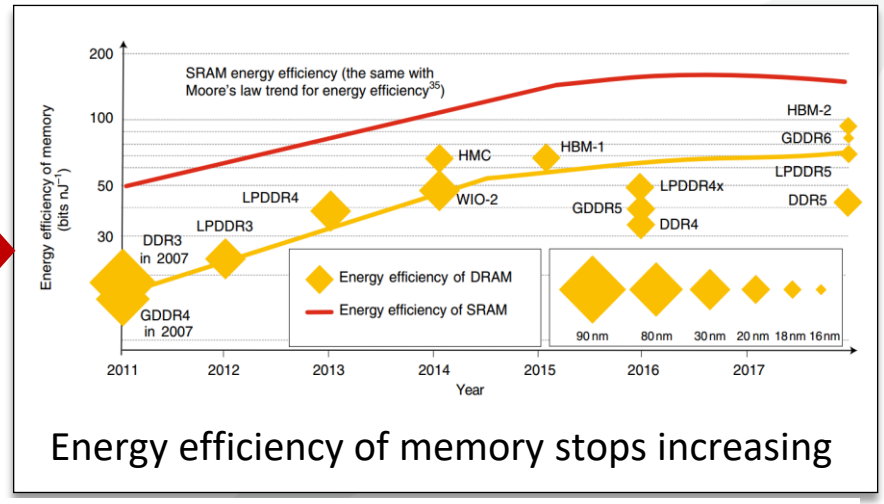
Neural Network Size

Traditional Hardware Capability



Number of DNN parameters increases exponentially

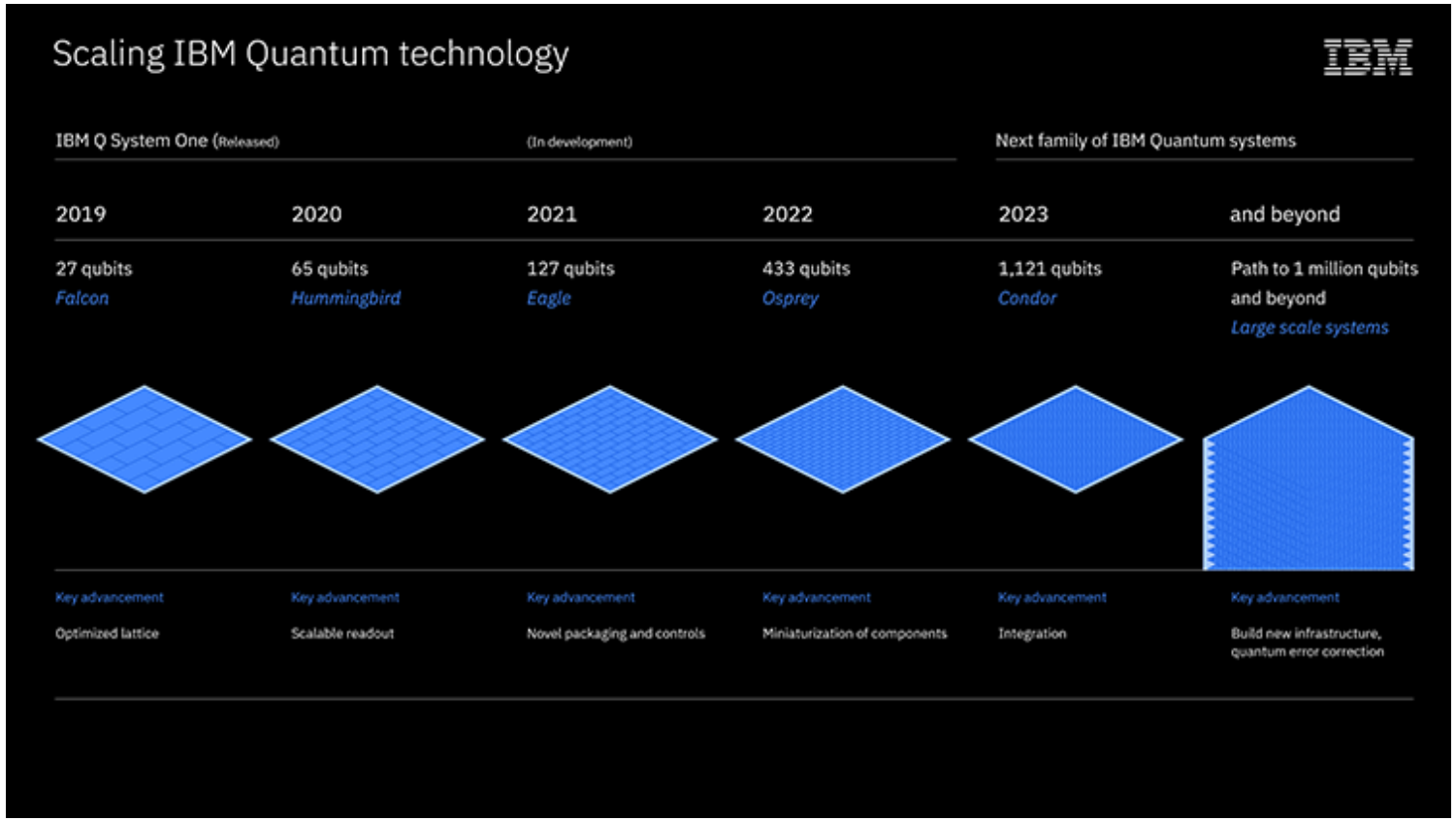
Storage Gap

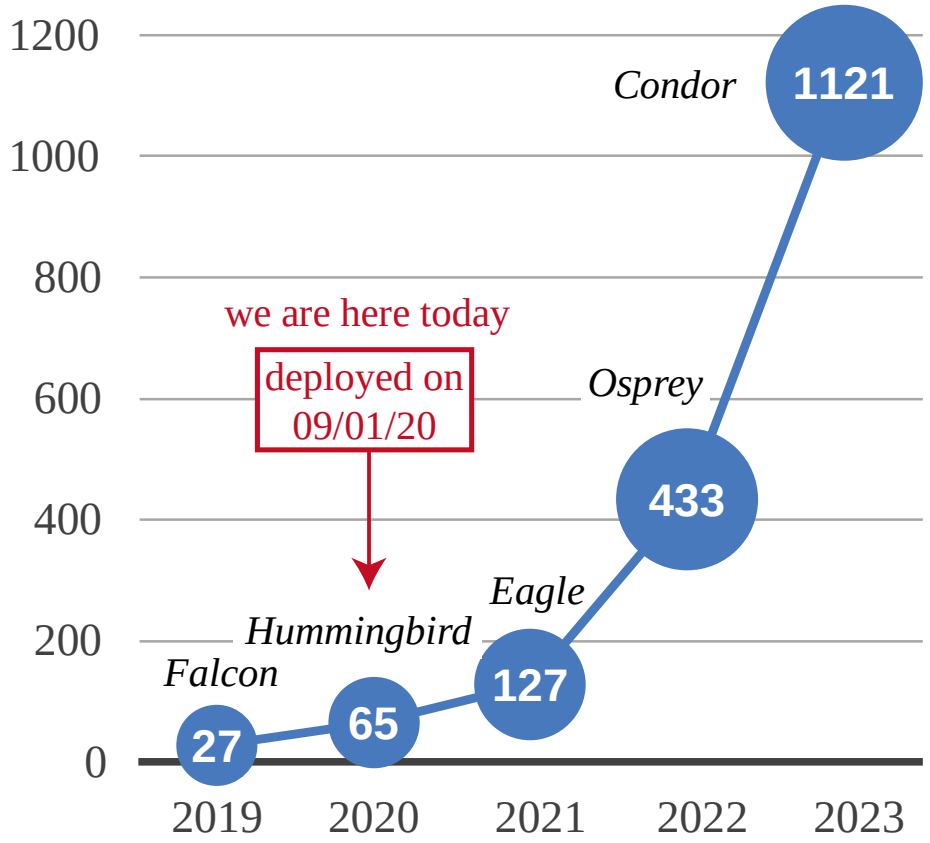


Energy efficiency of memory stops increasing

[ref] Xu, X., et al. 2018. Scaling for edge inference of deep neural networks. *Nature Electronics*, *1*(4), pp.216-222.

Consistently Increasing Qubits in Quantum Computers





The Power of Quantum Computers: Qubit

Classical Bit

$$X = 0 \ \textit{or} \ 1$$

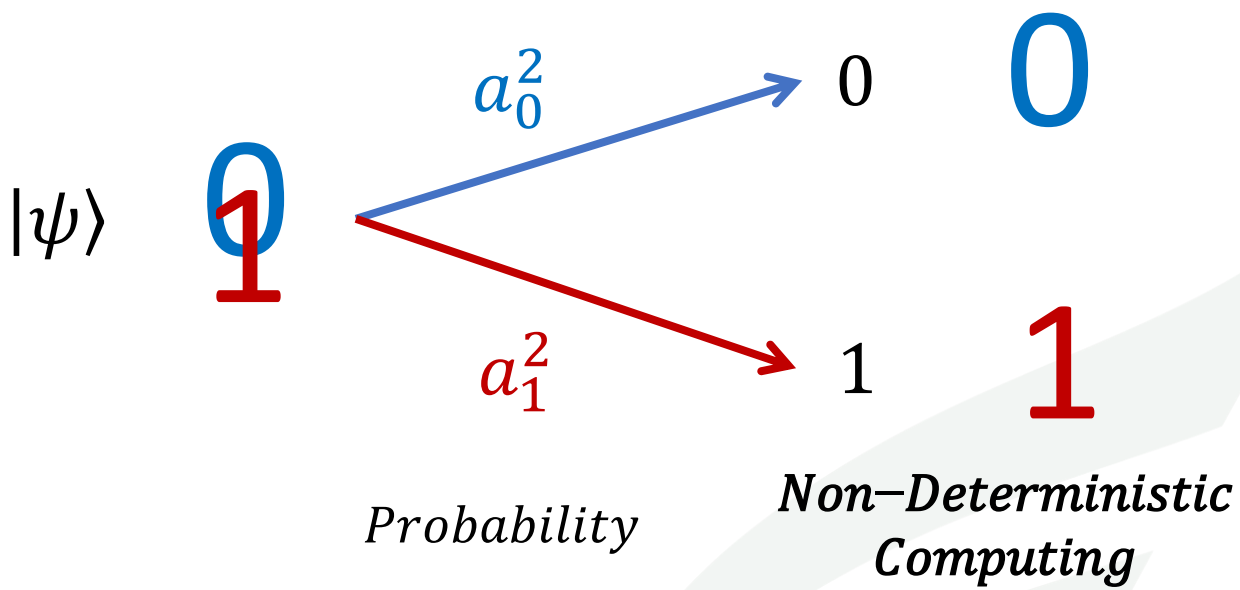
Quantum Bit (Qubit)

$$|\psi\rangle = |0\rangle \ \textit{and} \ |1\rangle$$

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle$$

$$\text{s.t. } a_0^2 + a_1^2 = 100\%$$

Reading out Information from Qubit (Measurement)



$$a_0^2 + a_1^2 = 100\%$$

$$40\% + 60\% = 100\%$$

The Power of Quantum Computers: Qubits

2 Classical Bits

00 **or** 01 **or** 10 **or** 11

n bits for 1 value

$\boldsymbol{x \in [0, 2^n - 1]}$

---

2 Qubits

$c_{00}|00\rangle$ **and** $c_{01}|01\rangle$ **and** $c_{10}|10\rangle$ **and** $c_{11}|11\rangle$

n bits for 2^n values

$\boldsymbol{a_{00}, a_{01}, a_{10}, a_{11}}$

Qubits: q_0, q_1

$$|q_0\rangle = a_0|0\rangle + a_1|1\rangle$$

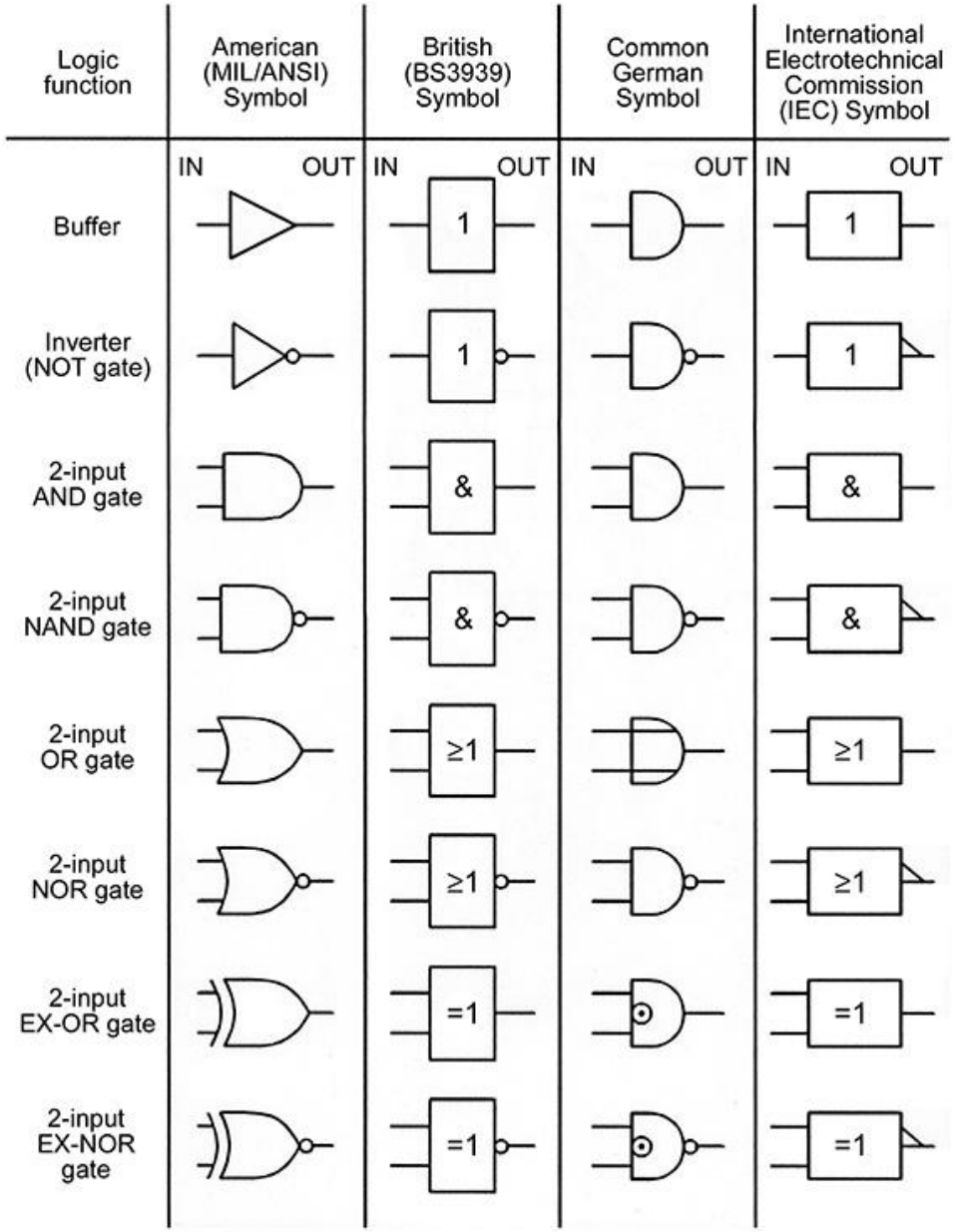
$$|q_1\rangle = b_0|0\rangle + b_1|1\rangle$$

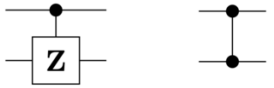
$$|\boldsymbol{q_0, q_1}\rangle = |\boldsymbol{q_0}\rangle \otimes |\boldsymbol{q_1}\rangle$$

$$= \boldsymbol{c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle}$$

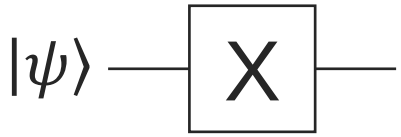
- $|00\rangle$: Both q_0 and q_1 are in state $|0\rangle$
- c_{00}^2: Probability of both q_0 and q_1 are in state $|0\rangle$
- $c_{00}^2 = a_0^2 \times b_0^2$
- $c_{00} = \sqrt{a_0^2 \times b_0^2} = a_0 \times b_0$

Logic Gates v.s. Quantum Logic Gates

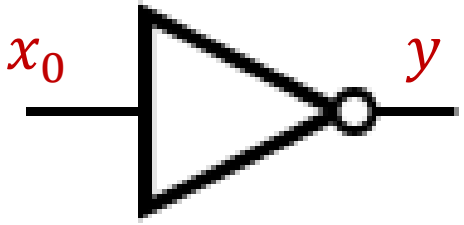


Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Logic Gates v.s. Quantum Logic Gates



Single-bit Gate

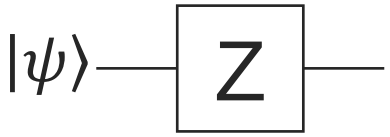


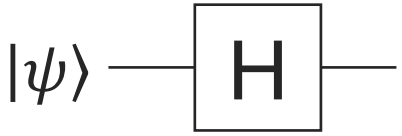
Not Gate

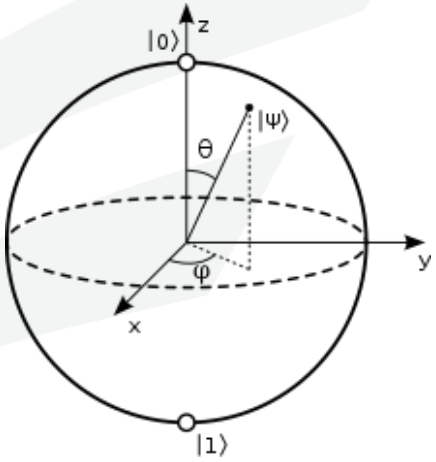
x_0	y
0	1
1	0

Single-Qubit Gates

- **Pauli operators: X, Y, Z Gates**
- **Hadamard gate: H Gate**
- General gate: U Gate

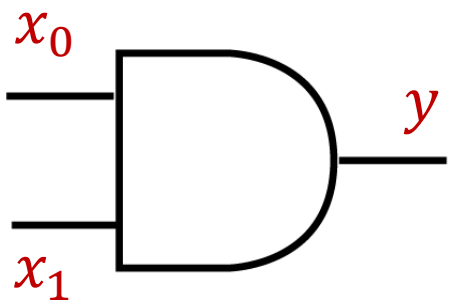






Logic Gates v.s. Quantum Logic Gates

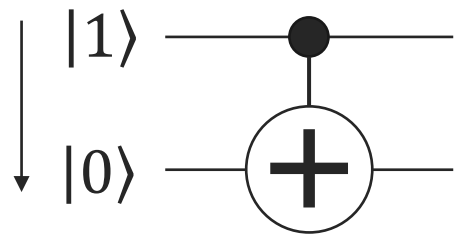
Two-bits Gate

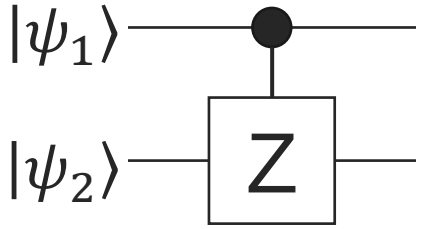


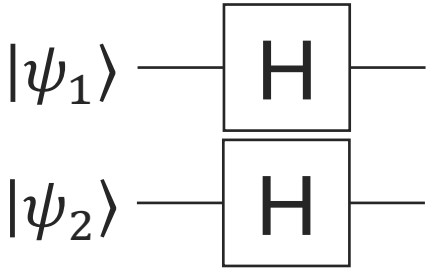
AND Gate

x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

- Multi-Qubit Gates
  - **Controlled-Pauli gates**
  - **Toffoli gate or CCNOT**
  - ……







Linear Algebra is also Central for Quantum Computing

$$A_{N,N} \times B_{N,1} = \frac{1}{2} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{bmatrix} = \begin{bmatrix} d_{00} \\ d_{01} \\ d_{10} \\ d_{11} \end{bmatrix}$$

$$|q_0, q_1\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$$

$$\rightarrow \begin{bmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{bmatrix} \quad \text{(vector representation)}$$

$$H \otimes H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = A_{N,N}$$

$$H \otimes H|q_0, q_1\rangle$$

$$= d_{00}|00\rangle + d_{01}|01\rangle + d_{10}|10\rangle + d_{11}|11\rangle$$

Matrix multiplication on classical computer using 16bit number

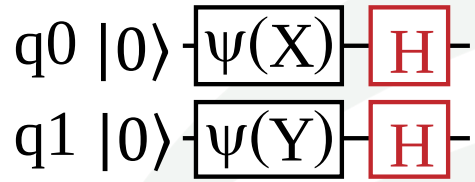
$$A_{N,N} \times B_{N,1} = C_{N,1}$$

Data: $(M \times M + 2 \times M) \times 16bit, \quad M = 2^2$

Operation: Multiplication: $\boldsymbol{M \times M}$

Accumulation: $\boldsymbol{M \times (M-1)}$

Special matrix multiplication on quantum computer

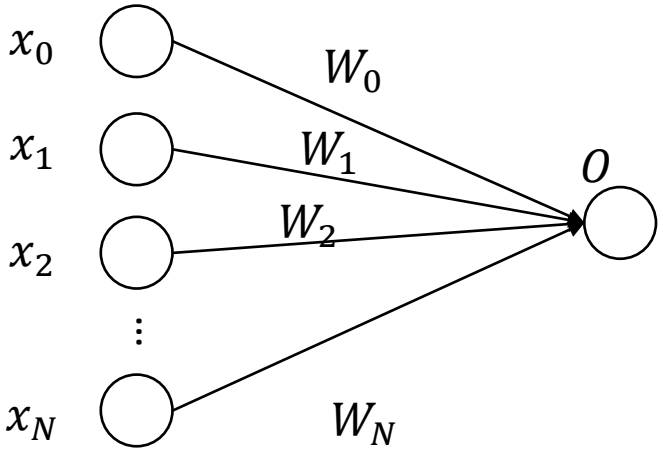


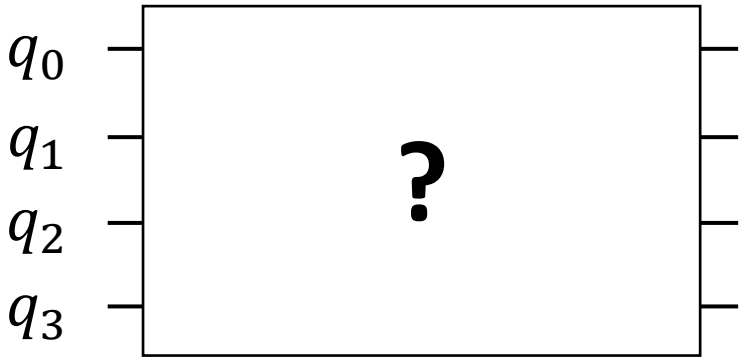
Data: **K** Qbits, **K = logM = 2**

Operation: **K** Hadamard (H) Gates

What's the Goals?

Goal 1: Correctly Implement!





Goal 2: Efficiently Implement!

$$O = \delta\left(\sum_{i\in[0,N)} x_i \times W_i\right)$$

where δ is a quadratic function

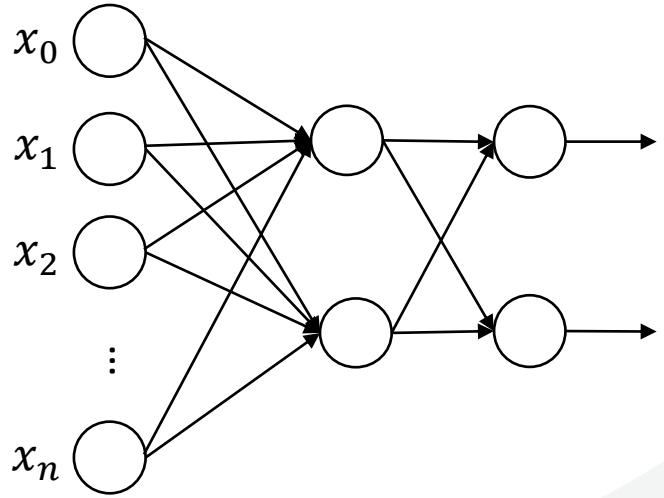
Classical Computing:

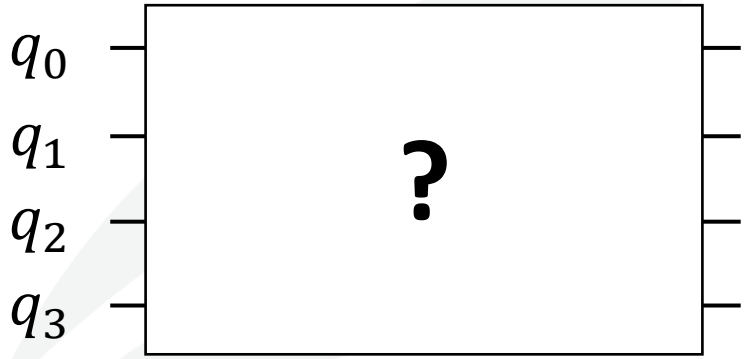
Complexity of $\boldsymbol{O(N)}$

Quantum Computing:

Can we reduce complexity to $\boldsymbol{O(ploylogN)}$, say $\boldsymbol{O(log^2 n)}$?

Goal 3: Scale-Up!

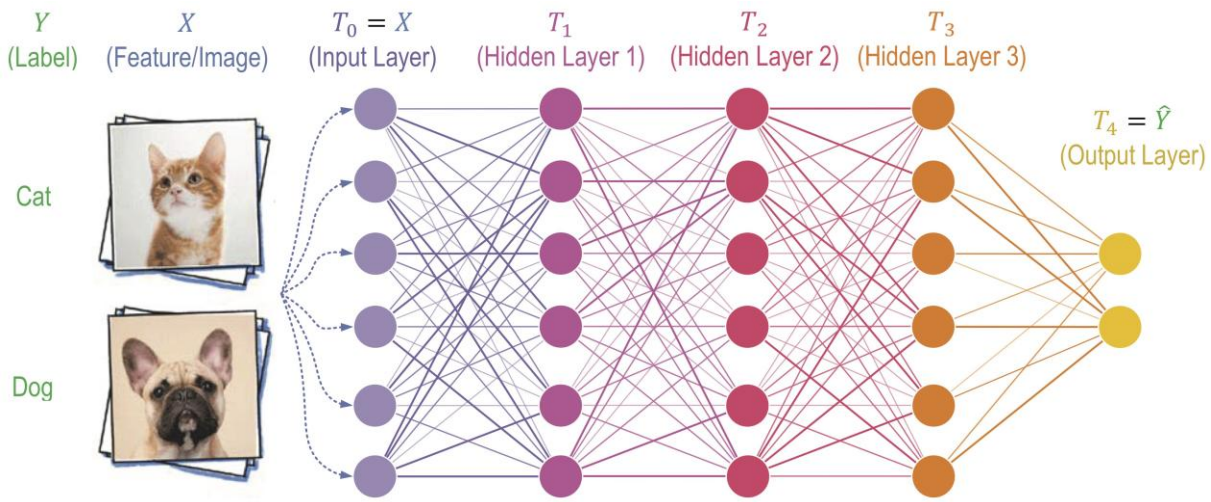


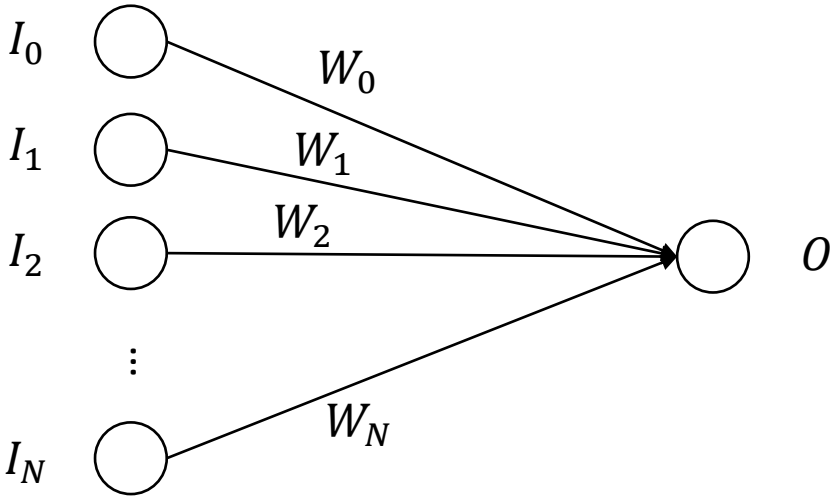


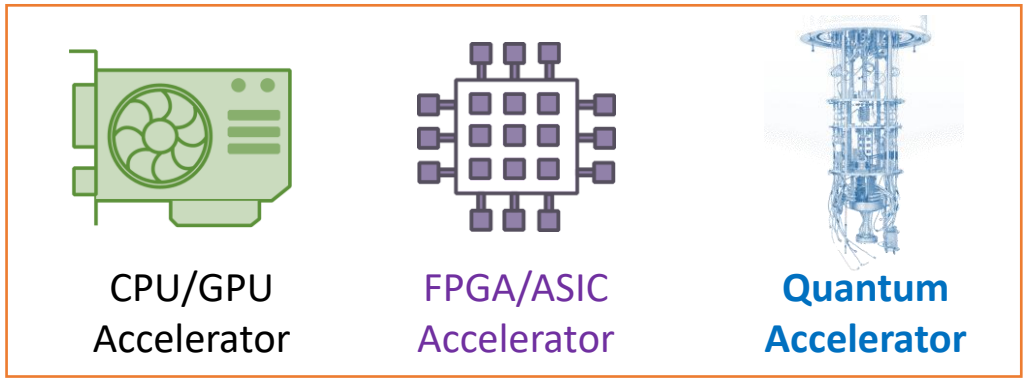
Agenda

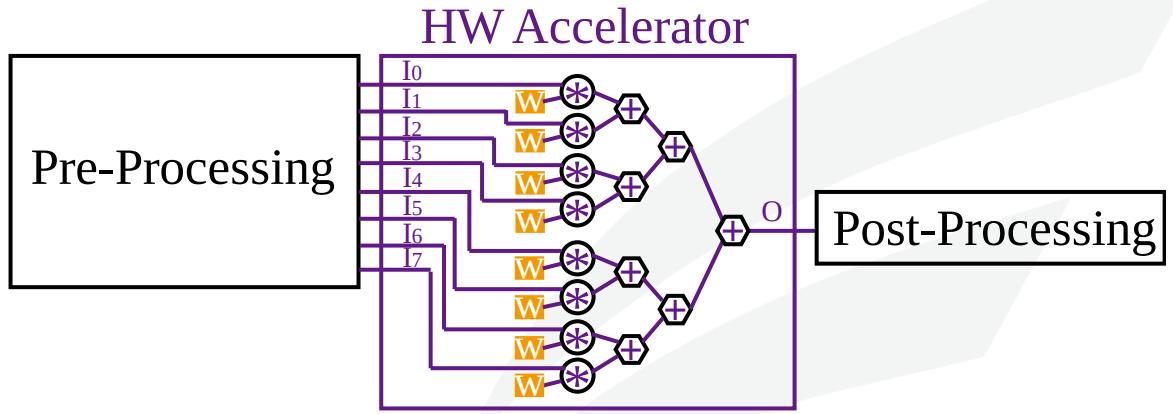
- **Background and Motivation**
  - What is machine learning
  - Why using quantum computer
  - Our goals
- **General Framework and Case Study (Tutorial 1-2 on GitHub)**
  - Implementing neural network accelerators: from classical to quantum
  - A case study on MNIST dataset
- **Optimization towards Quantum Advantage (Tutorial 3-4 on GitHub)**
  - The existing challenges
  - The proposed co-design framework: QuantumFlow

Neural Network Accelerator Design on Classical Hardware

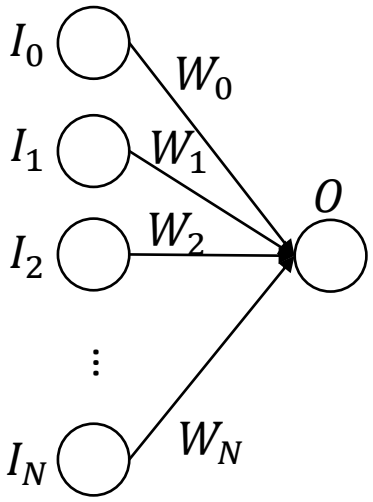


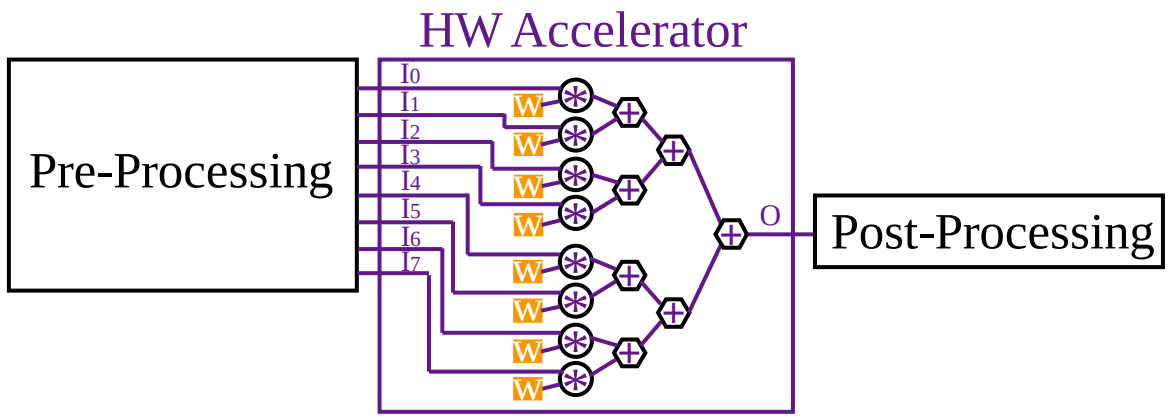


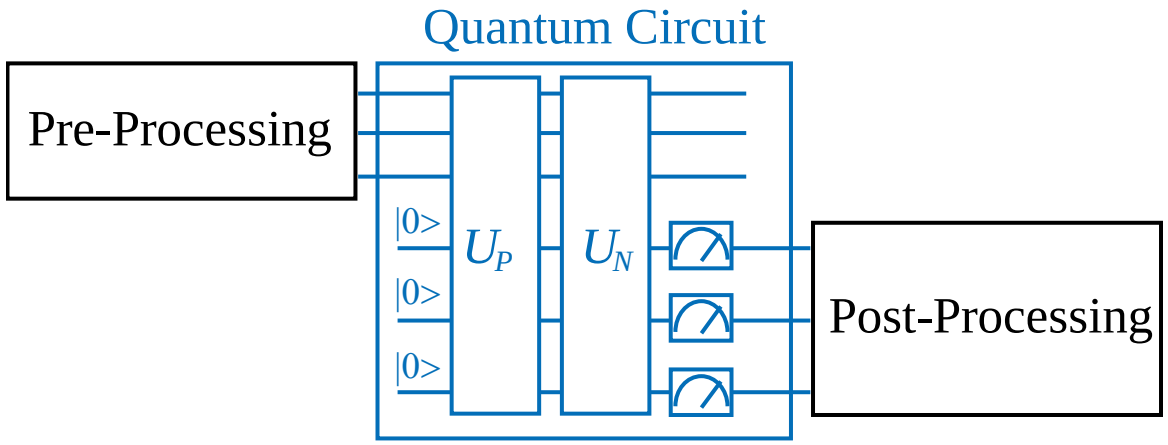




Neural Network Accelerator Design from Classical to Quantum Computing







(1) Data Pre-Processing (*PreP*)

(2) HW Acceleration

(3) Data Post-Processing (*PostP*)

(1) Data Pre-Processing (*PreP*)

(2) HW/Quantum Acceleration

(2.1) U_p Quantum-State-Preparation

(2.2) U_N Quantum Neural Computation

(2.3) *M* Measurement

(3) Data Post-Processing (PostP)

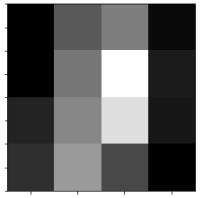
PreP + U_P + U_N + M + PostP

PreP + U_P + U_N + *M* + *PostP* : Data Pre-Processing

- **Given:** (1) 28×28 image, (2) the number of qubits to encode data (say Q=4 qubits in the example)
- **Do:** (1) downsampling from 28×28 to $2^Q = 16 = 4 \times 4$; (2) converting data to be the state vector in a unitary matrix
- **Output:** A unitary matrix, $M_{16\times16}$

Step 1: Downsampling

From 28×28 to 4×4



$$\begin{bmatrix} 0.0039 & 0.2118 & 0.2941 & 0.0275 \\ 0.0039 & 0.2784 & 0.5961 & 0.0667 \\ 0.0863 & 0.3176 & 0.5216 & 0.0588 \\ 0.1137 & 0.3608 & 0.1725 & 0.0039 \end{bmatrix}$$

$$\begin{bmatrix} 0.0039 & 0.2118 & 0.2941 & 0.0275 \\ 0.0039 & 0.2784 & 0.5961 & 0.0667 \\ 0.0863 & 0.3176 & 0.5216 & 0.0588 \\ 0.1137 & 0.3608 & 0.1725 & 0.0039 \end{bmatrix}$$

Step 2: Formulate Unitary Matrix

Applying SVD method
(See Listing 1 in ASP-DAC SS Paper)

Unitary matrix: $M_{16\times16}$

[SS] W. Jiang, et al. When Machine Learning Meets Quantum Computers: A Case Study, ASP-DAC'21

PreP + U_P + U_N + *M* + *PostP* --- Data Encoding / Quantum State Preparation

- **Given:** The unitary matrix provided by *PreP*, $M_{16\times16}$
- **Do:** Quantum-State-Preparation, encoding data to qubits
- **Verification:** Check the amplitude of states are consistent with the data in the unitary matrix, $M_{16\times16}$

Let's use a 2-qubit system as an example to encode a matrix $M_{4\times4}$

$$\begin{bmatrix} 0.3 & 0.5 \\ 0.7 & 0.9 \end{bmatrix} \xrightarrow{PreP} \text{data\_matrix} \begin{bmatrix} \mathbf{0.2343} & X & X & X \\ \mathbf{0.3904} & X & X & X \\ \mathbf{0.5466} & X & X & X \\ \mathbf{0.7028} & X & X & X \end{bmatrix} \xrightarrow{U_P}$$

State Transition:

data_matrix $|00\rangle$

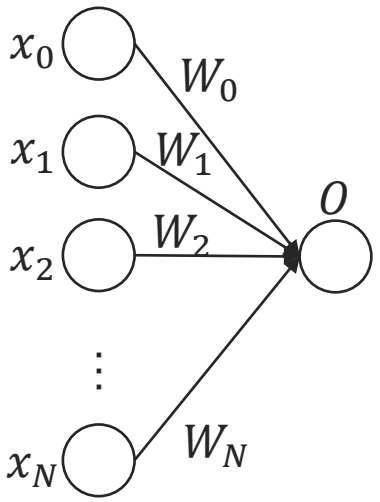
IBM Qiskit Implementation:

```
inp = QuantumRegister(4, ”in_qubit”)
circ = QuantumCircuit(inp)
iniG = UnitaryGate(data_matrix, label=”input”)
circ.append(iniG, inp[0:4])
```

Hands-On

Tutorial 1: *PreP + U_P*

PreP + U_P + U_N + M + PostP --- Neural Computation



- **Given:** (1) A circuit with encoded input data $\boldsymbol{x}$; (2) the trained binary weights $\boldsymbol{w}$ for one neural computation, which will be associated to each data.
- **Do:** Place quantum gates on the qubits, such that it performs $\frac{(\boldsymbol{x}*\boldsymbol{w})^2}{\|\boldsymbol{x}\|}$.
- **Verification:** Whether the output data of quantum circuit and the output computed using torch on classical computer are the same.

Target: $O = \left[\frac{\sum_i (x_i \times w_i)}{\sqrt{\|x\|}}\right]^2$

- **Assumption 1:** Parameters/weights (W_0 --- W_N) are binary weight, either +1 or -1
- **Assumption 2:** The weight $W_0 = +1$, otherwise we can use $-\boldsymbol{w}$ (quadratic func.)

Step 1: $m_i = x_i \times w_i$

Step 2: $n = \left[\frac{\sum_i (m_i)}{\sqrt{\|x\|}}\right]$

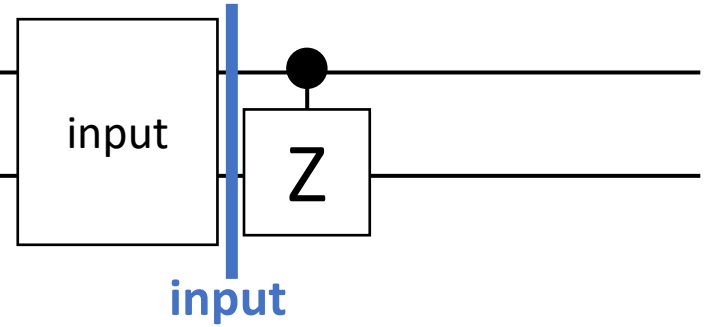
Step 3: $O = n^2$

PreP + U_P + U_N + M + PostP --- Neural Computation: Step 1

Step 1: $m_i = x_i \times w_i$

EX: 4 input data on 2 qubits

$$\boldsymbol{w} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$



Input

	$\|00\rangle$
	$\|01\rangle$
	$\|10\rangle$
	$\|11\rangle$

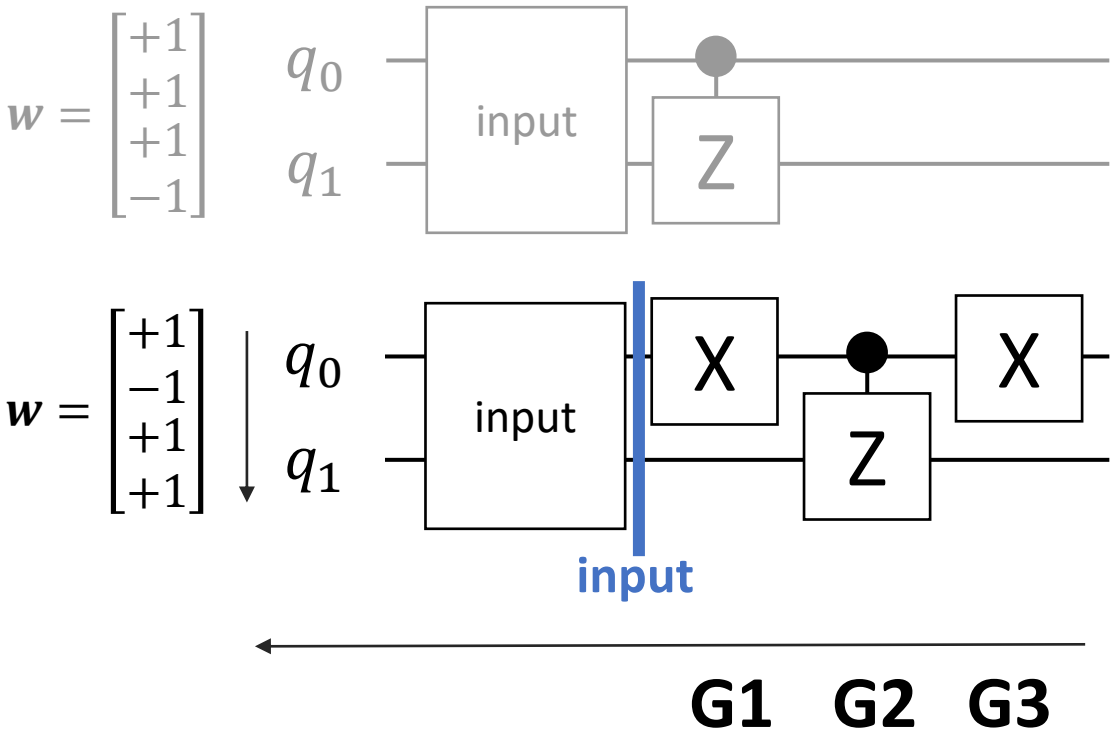
CZ

Output

PreP + U_P + U_N + M + PostP --- Neural Computation: Step 1

Step 1: $m_i = x_i \times w_i$

EX: 4 input data on 2 qubits



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times$$

Input	
a_0	$\|00\rangle$
a_1	$\|01\rangle$
a_2	$\|10\rangle$
a_3	$\|11\rangle$

$\longrightarrow$

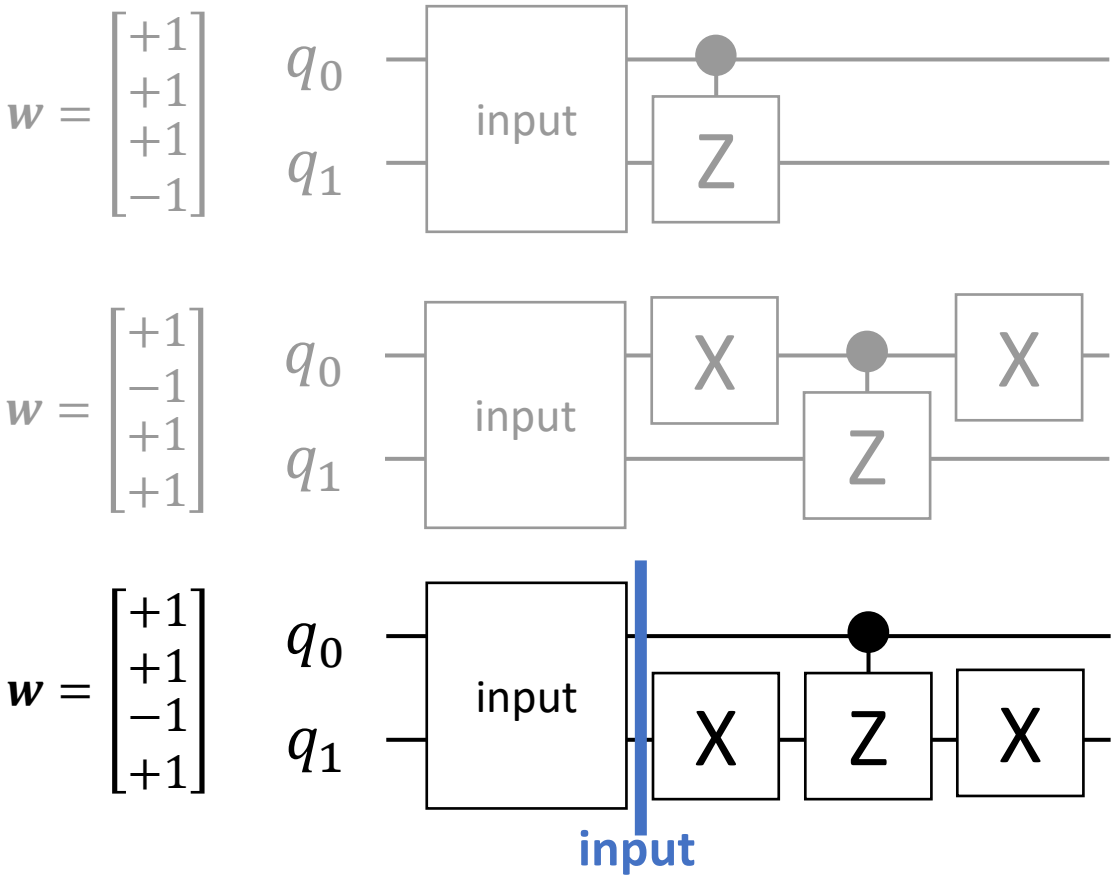
Output	
a_0	$\|00\rangle$
$-a_1$	$\|01\rangle$
a_2	$\|10\rangle$
a_3	$\|11\rangle$

$$\mathbf{G3} \times (\mathbf{G2} \times (\mathbf{G1} \times |q_0 q_1\rangle)) =$$
$$(G3 \times G2 \times G1) \times |q_0 q_1\rangle =$$
$$(X \otimes I) \times CZ \times (X \otimes I)$$

PreP + U_P + U_N + M + PostP --- Neural Computation: Step 1

Step 1: $m_i = x_i \times w_i$

EX: 4 input data on 2 qubits



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times$$

Input	
$\boldsymbol{a_0}$	$\|00\rangle$
$\boldsymbol{a_1}$	$\|01\rangle$
$\boldsymbol{a_2}$	$\|10\rangle$
$\boldsymbol{a_3}$	$\|11\rangle$

$\longrightarrow$

Output	
$\boldsymbol{a_0}$	$\|00\rangle$
$\boldsymbol{a_1}$	$\|01\rangle$
$-\boldsymbol{a_2}$	$\|10\rangle$
$\boldsymbol{a_3}$	$\|11\rangle$

$$(\boldsymbol{I} \otimes \boldsymbol{X}) \times \boldsymbol{CZ} \times (\boldsymbol{I} \otimes \boldsymbol{X})$$

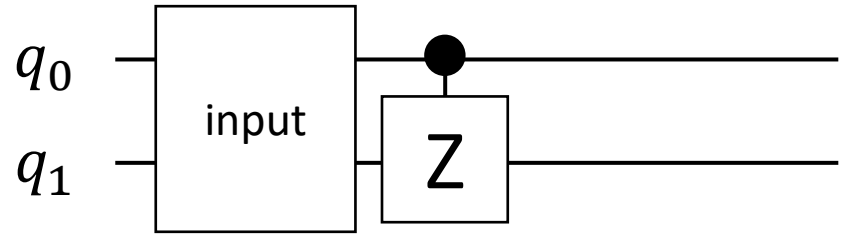
PreP + U_P + U_N + M + PostP --- Neural Computation: Step 1

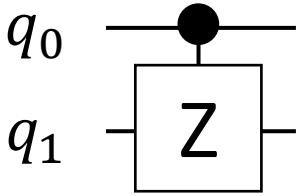
Step 1: $m_i = x_i \times w_i$

$$\boldsymbol{w} = \begin{bmatrix}+1\\+1\\+1\\-1\end{bmatrix} \text{or} \begin{bmatrix}+1\\+1\\-1\\-1\end{bmatrix} \text{or} \begin{bmatrix}+1\\-1\\-1\\-1\end{bmatrix} \text{or} \begin{bmatrix}+1\\+1\\-1\\+1\end{bmatrix} \text{or} \begin{bmatrix}+1\\-1\\-1\\+1\end{bmatrix} \text{or} \begin{bmatrix}+1\\-1\\+1\\+1\end{bmatrix}$$

EX: 4 input data on 2 qubits

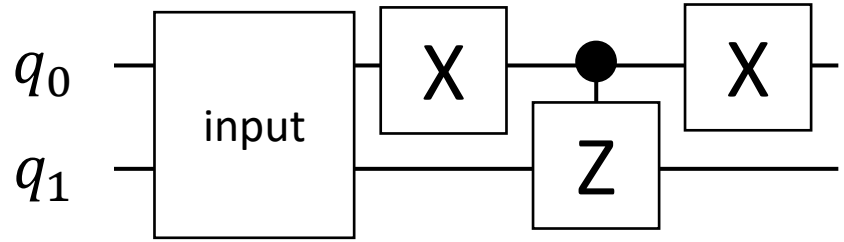
$\boldsymbol{w} = \begin{bmatrix}+1\\+1\\+1\\-1\end{bmatrix}$

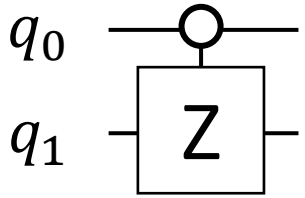




Flip the sign of $|11\rangle$

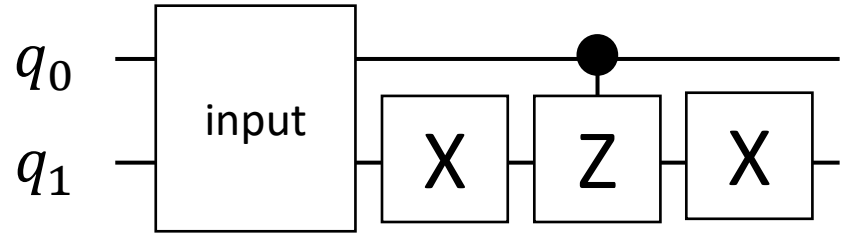
$\boldsymbol{w} = \begin{bmatrix}+1\\-1\\+1\\+1\end{bmatrix}$

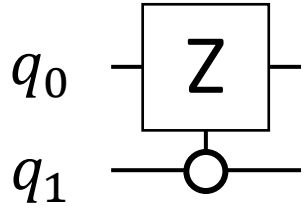




Flip the sign of $|01\rangle$

$\boldsymbol{w} = \begin{bmatrix}+1\\+1\\-1\\+1\end{bmatrix}$





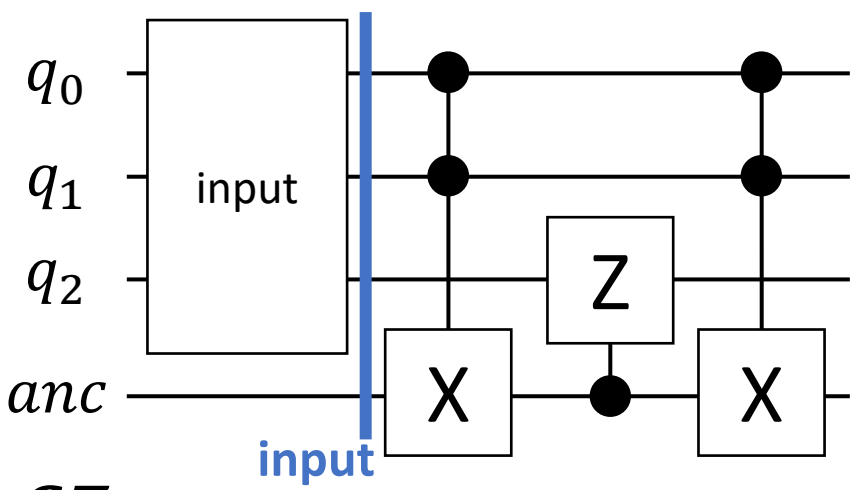
Flip the sign of $|10\rangle$

PreP + U_P + U_N + M + PostP --- Neu utati L

Step 1: $m_i = x_i \times w_i$

EX: 8 input data on 3 qubits

$$w = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$



$\boldsymbol{I \otimes I \otimes CZ}$:

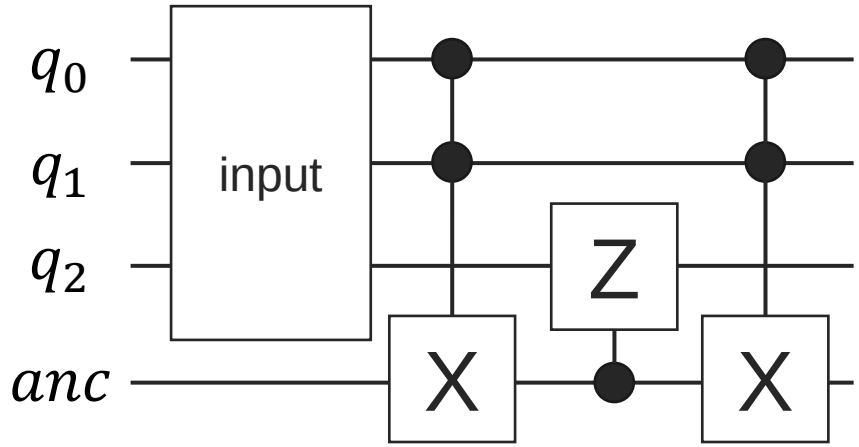
Input	CCIX	CCIX
$\|0000\rangle$	$\|0000\rangle$	$\|0000\rangle$
$\|0001\rangle$	$\|0001\rangle$	$\|0001\rangle$
$\|0010\rangle$	$\|0010\rangle$	$\|0010\rangle$
$\|0011\rangle$	$\|0011\rangle$	$\|0011\rangle$
$\|0100\rangle$	$\|0100\rangle$	$\|0100\rangle$
$\|0101\rangle$	$\|0101\rangle$	$\|0101\rangle$
$\|0110\rangle$	$\|0110\rangle$	$\|0110\rangle$
$\|0111\rangle$	$\|0111\rangle$	$\|0111\rangle$
$\|1000\rangle$	$\|1000\rangle$	$\|1000\rangle$
$\|1001\rangle$	$\|1001\rangle$	$\|1001\rangle$
$\|1010\rangle$	$\|1010\rangle$	$\|1010\rangle$
$\|1011\rangle$	$\|1011\rangle$	$\|1011\rangle$
$\|\mathbf{11}00\rangle$	$\|1100\rangle$	$\|1100\rangle$
$\|\mathbf{11}01\rangle$	$\|1101\rangle$	$\|1101\rangle$
$\|\mathbf{11}10\rangle$	$\|1110\rangle$	$\|1110\rangle$
$\|\mathbf{11}11\rangle$	$\|1111\rangle$	$\|1111\rangle$

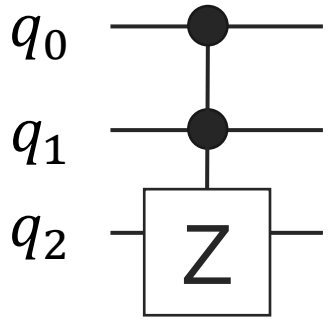
PreP + U_P + U_N + M + PostP --- Neural Computation: Step 1

Step 1: $m_i = x_i \times w_i$

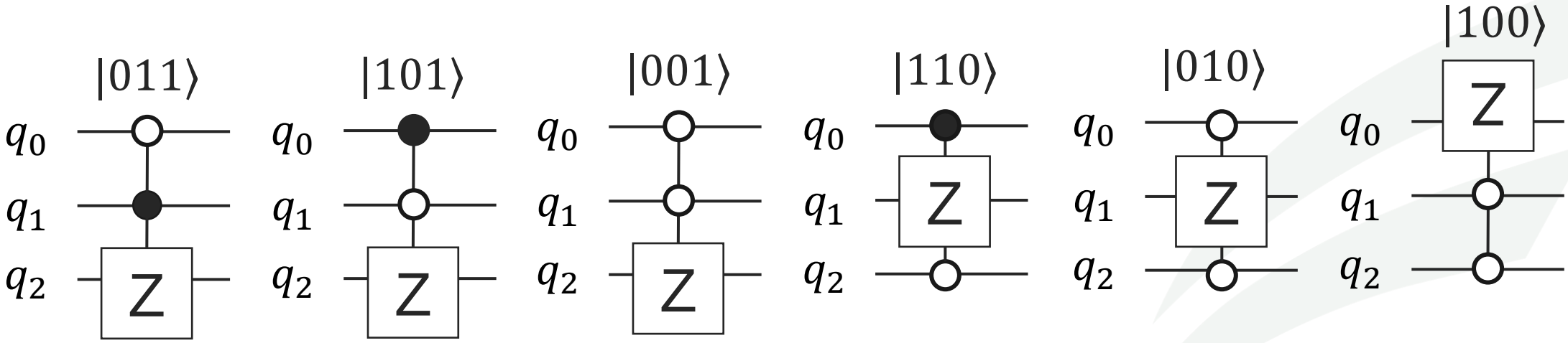
EX: 8 input data on 3 qubits

$$w = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$





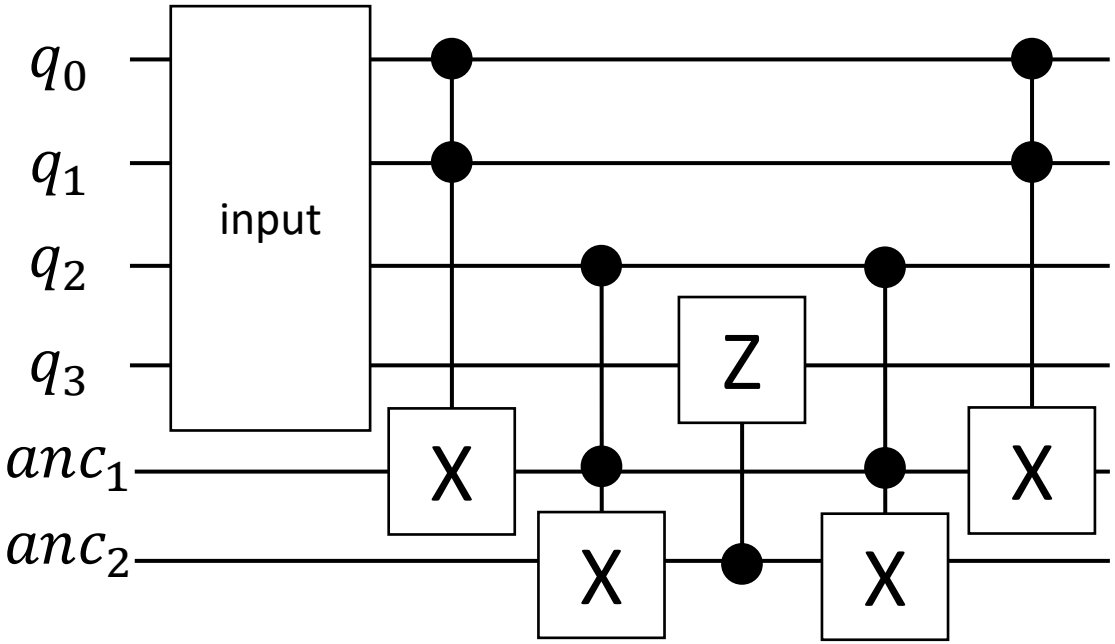
Flip the sign of $|111\rangle$

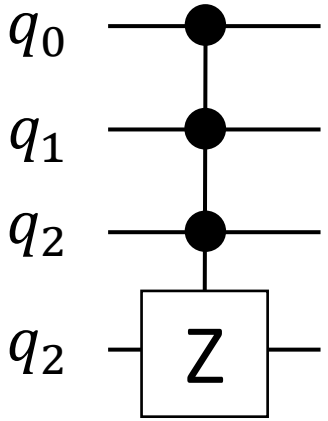


PreP + U_P + U_N + M + PostP --- Neural Computation: Step 1

Step 1: $m_i = x_i \times w_i$

EX: 16 input data on 4 qubits



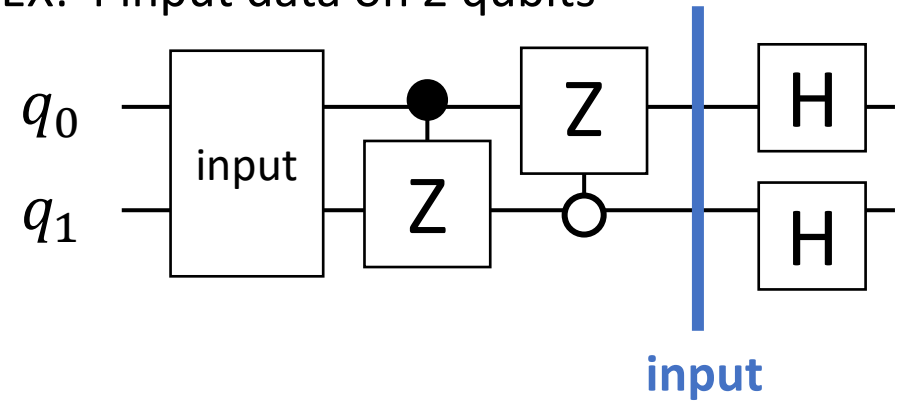


Flip the sign of $|1111\rangle$

PreP + U_P + U_N + M + PostP --- Neural Computation: Step 2

Step 2: $n = \left\lceil \frac{\sum_i (m_i)}{\sqrt{\|x\|}} \right\rceil$

EX: 4 input data on 2 qubits



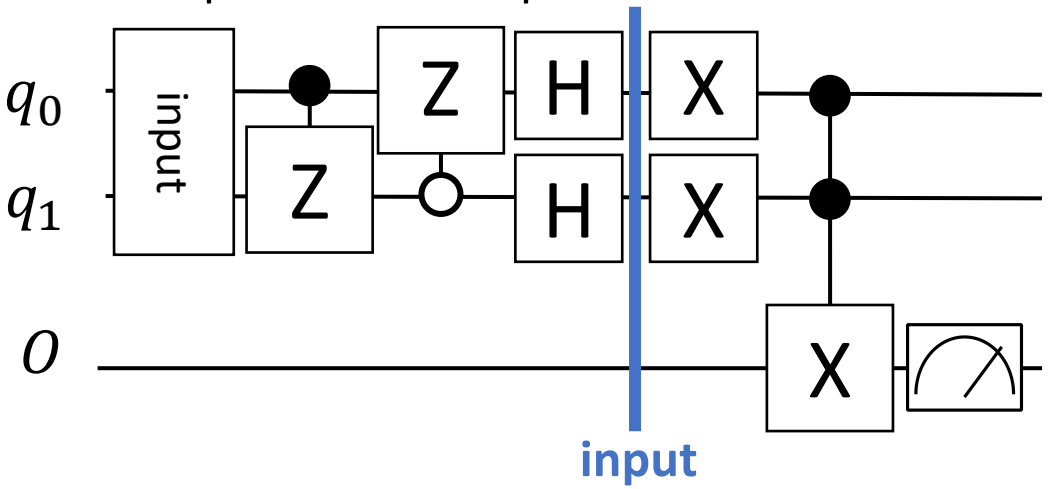
$\boldsymbol{H^{\otimes 2}}$

Input				Output	
a_0	m_0	$\|00\rangle$	=	$\sum_i (m_i)/\sqrt{\|x\|}$	$\|00\rangle$
a_1	m_1	$\|01\rangle$		Do not care 1	$\|01\rangle$
$-a_2$	m_2	$\|10\rangle$		Do not care 2	$\|10\rangle$
$-a_3$	m_3	$\|11\rangle$		Do not care 3	$\|11\rangle$

PreP + U_P + U_N + *M* + *PostP* -- Neural Computation (Step 3) & Measurement

Step 3: $O = n^2$

EX: 4 input data on 2 qubits



Input

$\sum_i (m_i)/\sqrt{\|\|x\|\|}$	$\|000\rangle$
0	$\|001\rangle$
Do not care 1	$\|010\rangle$
0	$\|011\rangle$
Do not care 2	$\|100\rangle$
0	$\|101\rangle$
Do not care 3	$\|110\rangle$
0	$\|111\rangle$

$X^{\otimes 2}$

	$\|000\rangle$
	$\|001\rangle$
	$\|010\rangle$
	$\|011\rangle$
	$\|100\rangle$
	$\|101\rangle$
$\sum_i (m_i)/\sqrt{\|\|x\|\|}$	$\|110\rangle$
	$\|111\rangle$

CCX

	$\|000\rangle$
	$\|001\rangle$
	$\|010\rangle$
	$\|011\rangle$
	$\|100\rangle$
	$\|101\rangle$
	$\|110\rangle$
$\sum_i (m_i)/\sqrt{\|\|x\|\|}$	$\|111\rangle$

Output

$$P\{O = |1\rangle\} = P\{|100\rangle\} + P\{|101\rangle\} + P\{|110\rangle\} + P\{|111\rangle\} = \left[\frac{\sum_i (m_i)}{\sqrt{\|\|x\|\|}}\right]^2$$

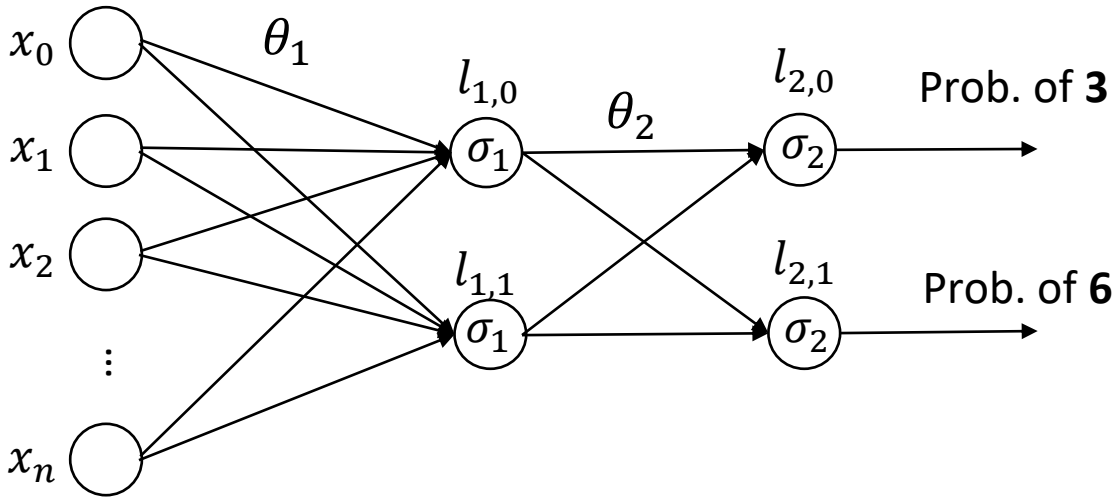
Hands-On

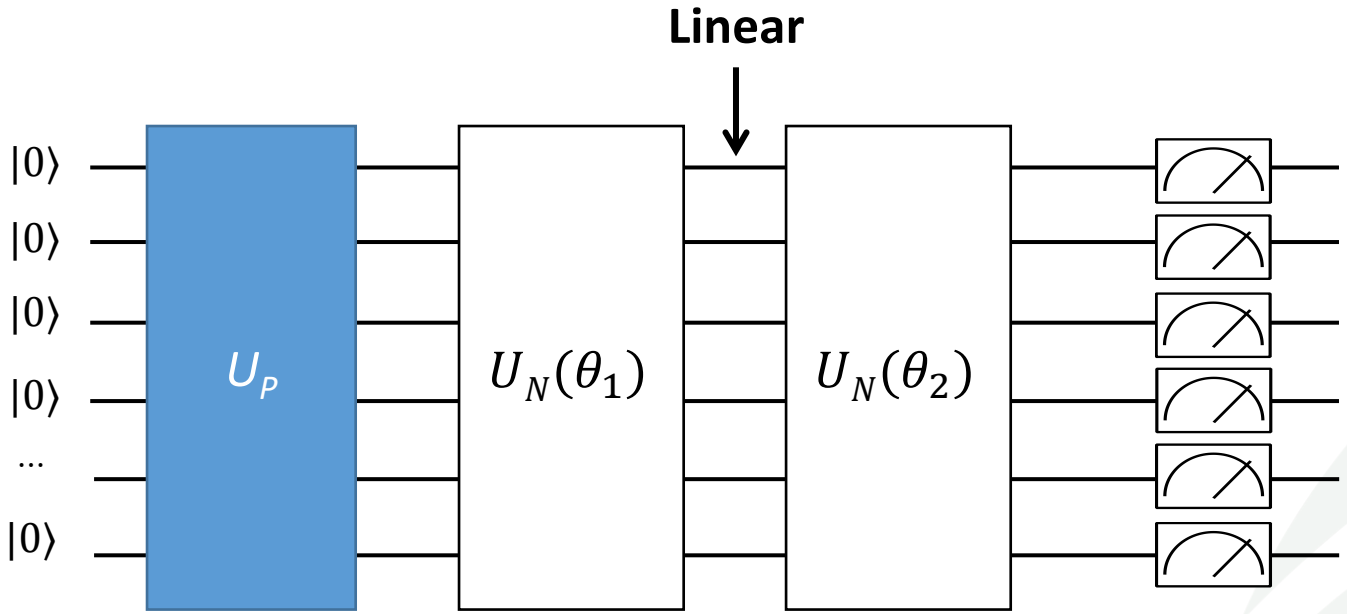
Tutorial 2: *PreP* + U_P + U_N

Agenda

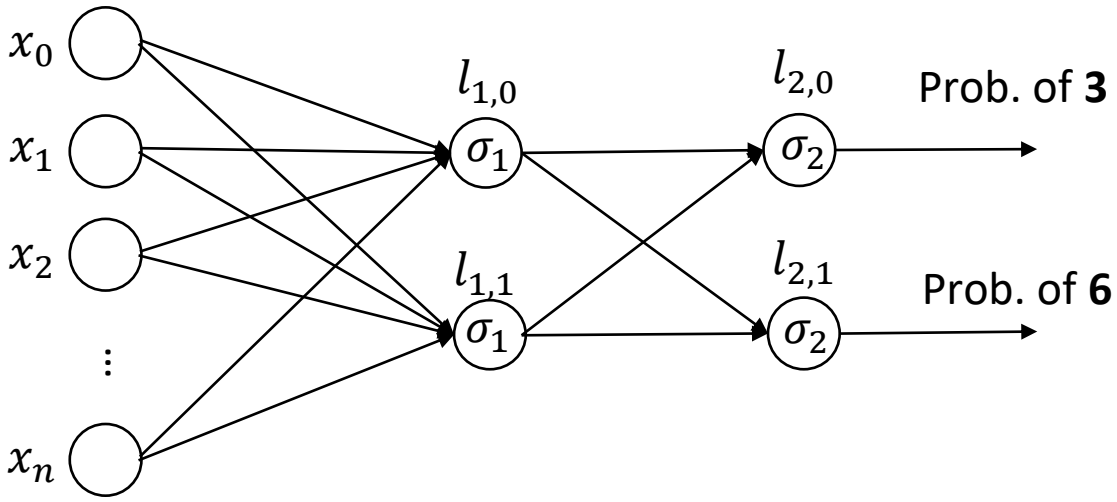
- **Background and Motivation**
  - What is machine learning
  - Why using quantum computer
  - Our goals
- **General Framework and Case Study (Tutorial 1-2 on GitHub)**
  - Implementing neural network accelerators: from classical to quantum
  - A case study on MNIST dataset
- **Optimization towards Quantum Advantage (Tutorial 3-4 on GitHub)**
  - The existing challenges
  - The proposed co-design framework: QuantumFlow

Challenge 1: Non-linearity is Needed, But Difficult in Quantum Circuit





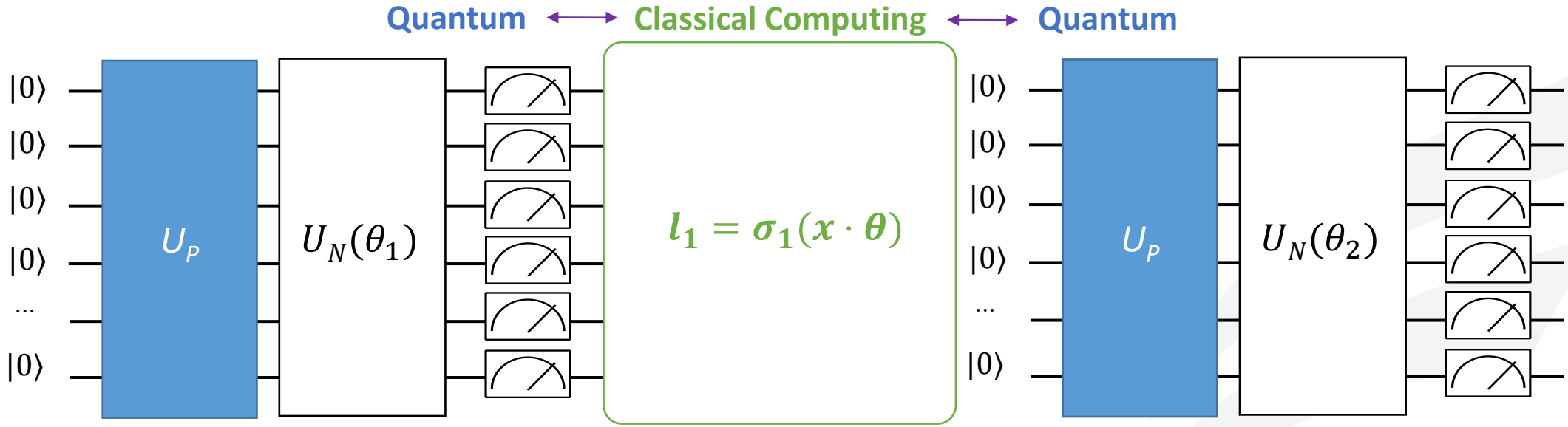
Challenge 2: Quantum-Classical Interface is Expensive



Ref [1]

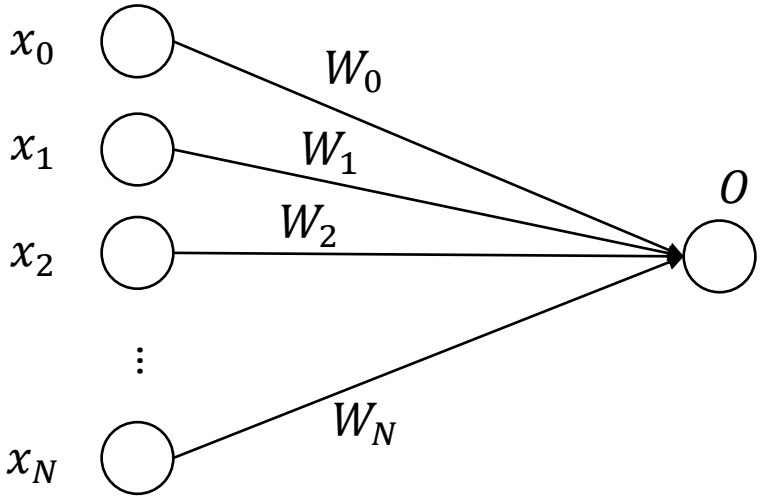
Table 2 Complexity of each step in hybrid quantum-classical computing for deep neural network with U-LYR.

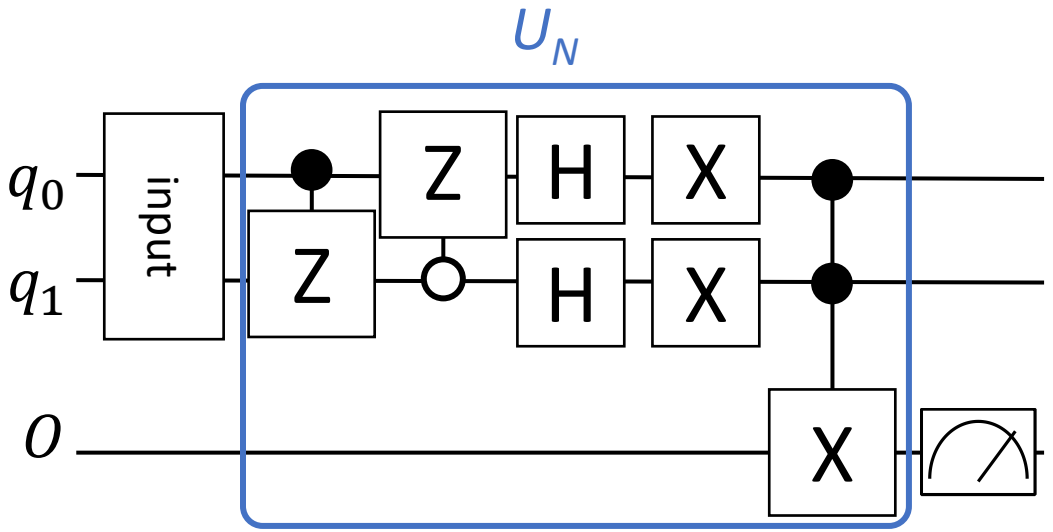
Complexity	State-preparation	Computation	Measurement
Depth (T)	$O(d \cdot \sqrt{n})$	$O(d \cdot \log^2 n)$	$O(d)$
Qubits (S)	$O(n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$
Cost (TS)	$O(d \cdot n^{\frac{3}{2}})$	$O(d \cdot n \cdot \log^3 n)$	$O(d \cdot n \cdot \log n)$
Total (TS)	$O(d \cdot n^{\frac{3}{2}})$ **dominate**		



[1] W. Jiang, et al. A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage, Nature Communications

Challenge 3: High Complexity in the Previous Design



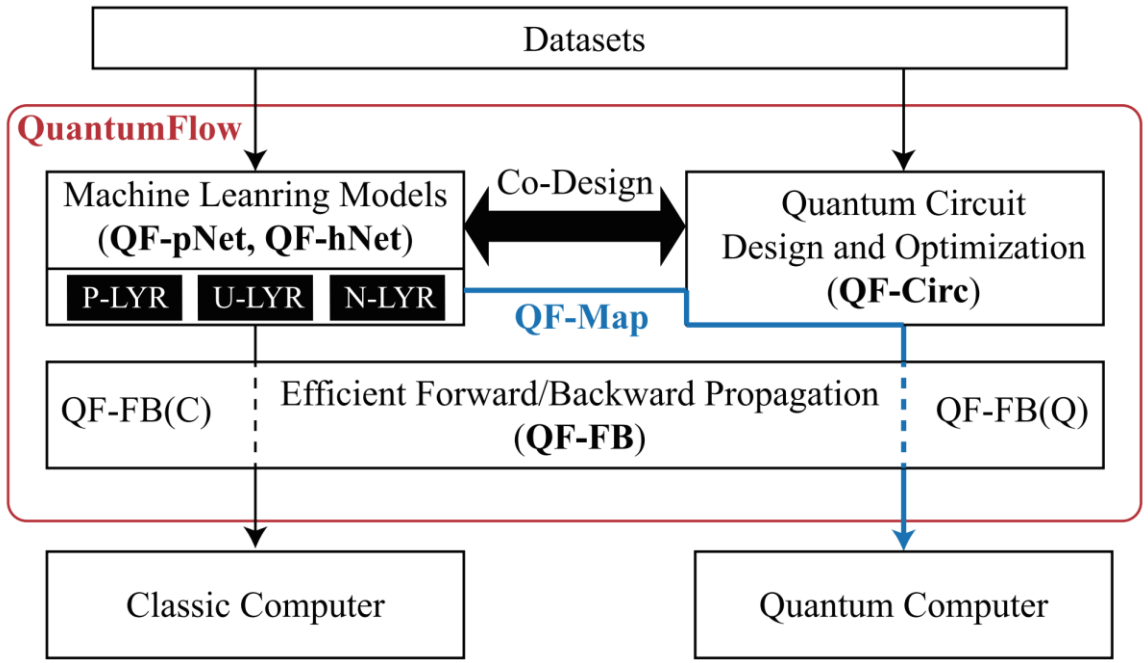


Cost Complexity

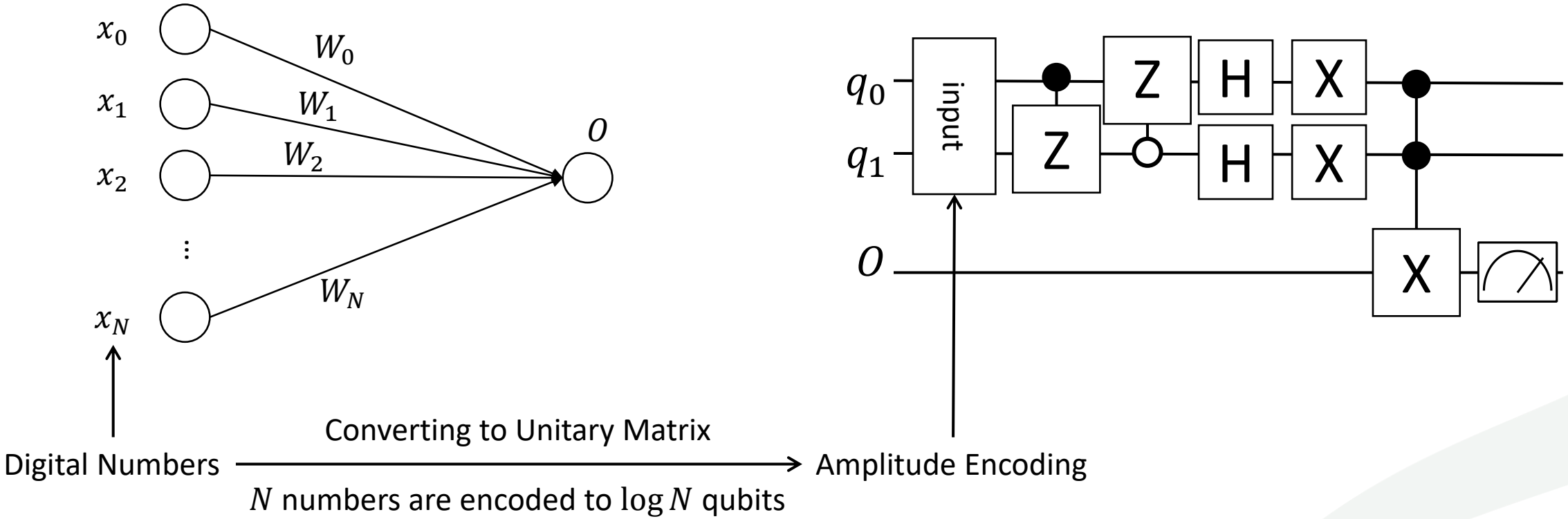
Classical Computing		
	No Parallelism	Full Parallelism
Time (T)	O(N)	O(1)
Space (S)	O(1)	O(N)
Cost (TS)	O(N)	O(N)

Quantum Computing		
	Previous Design	Optimization
Circuit Depth (T)	O(N)	???
Qubits (S)	O($\log N$)	O(N)
Cost (TS)	O($N \cdot \log N$)	**target** O(ploylog N)

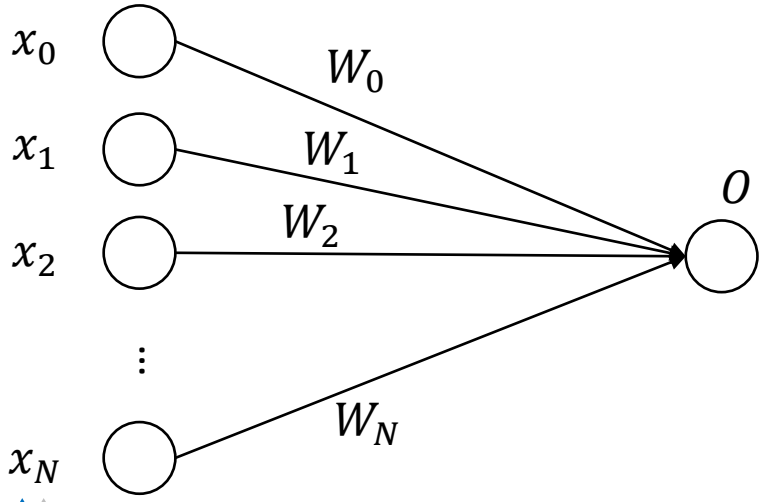
Co-Design Framework

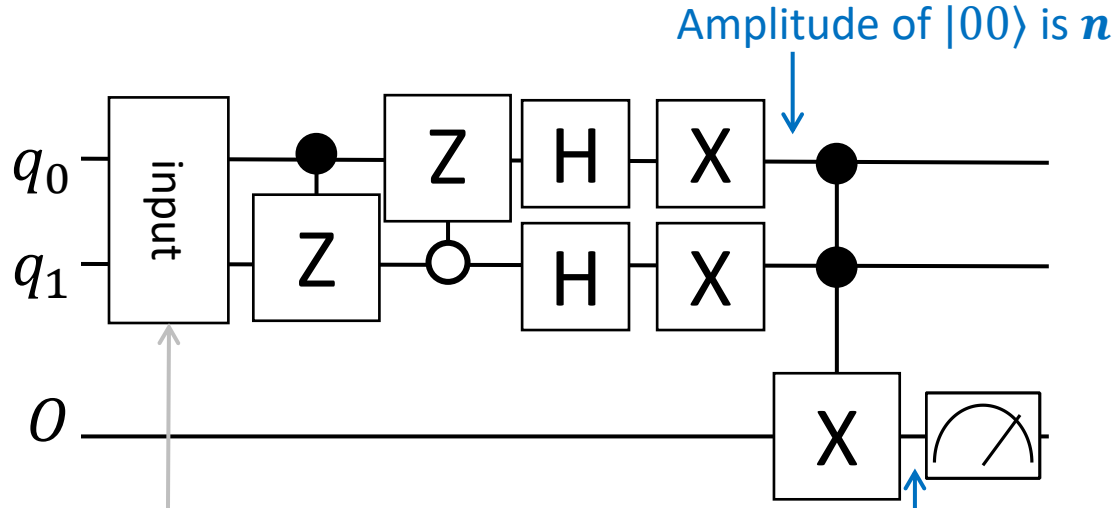


Design Direction 1: NN → Quantum Circuit



Design Direction 2: Quantum Circuit → NN





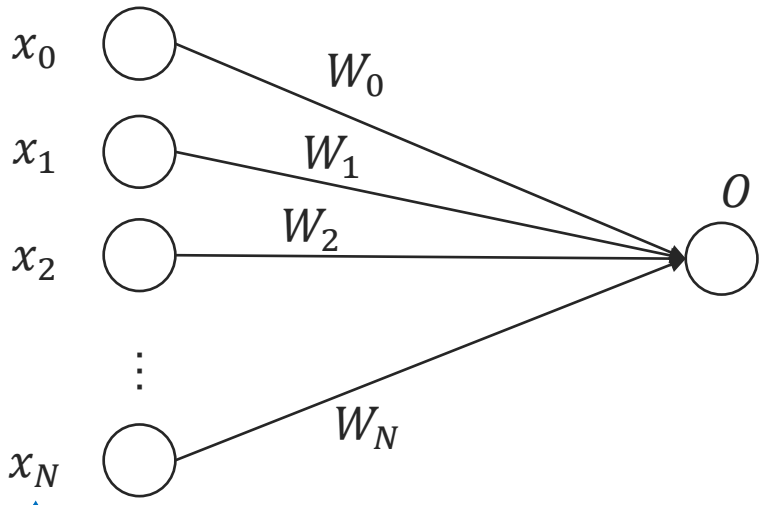
Converting to Unitary Matrix

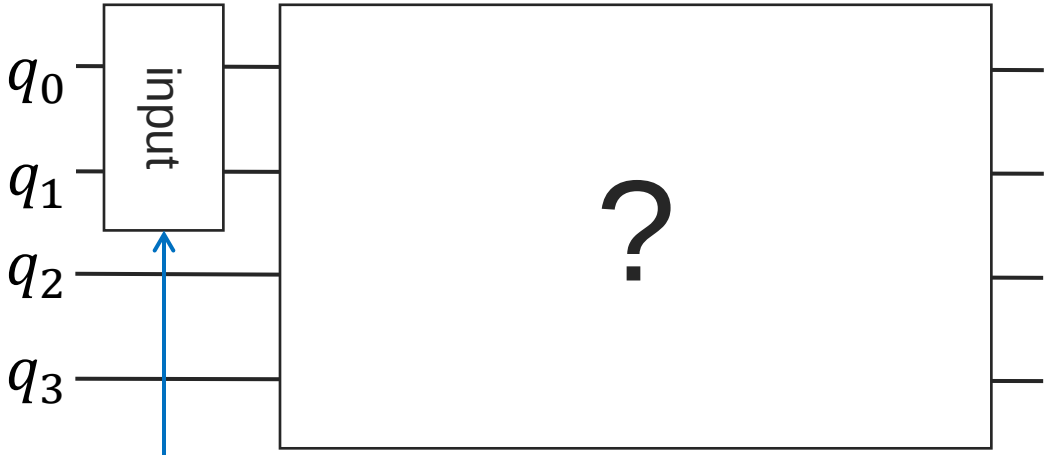
Digital Numbers → Amplitude Encoding

N numbers are encoded to $\log N$ qubits

Random Variable ← Probability of $O = |1\rangle$ is $\boldsymbol{n^2}$

Design Direction 3: NN → Quantum Circuit



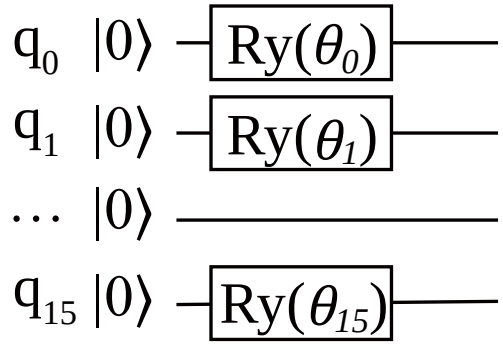


Random Variable → **Rotation Gate, say Ry** → Angle Encoding

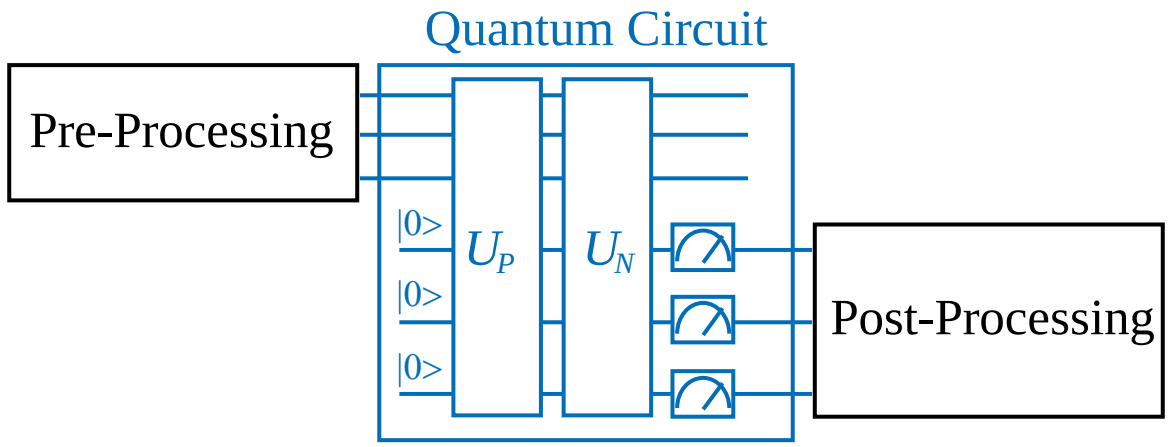
***N* numbers are encoded to *N* qubits**

0.0	0.9	0.0	0.0
0.0	0.1	0.0	0.0
0.0	1.0	0.5	0.5
0.0	0.0	0.0	0.0

r.v.	-1	+1
x_0	0.0	1.0
x_1	0.9	0.1
…	…	
x_{15}	0.0	1.0



Apply Our Framework to Address Challenges 1 & 2 (non-linear & Q-C comm.)



(1) Data Pre-Processing (*PreP*)

(2) HW/Quantum Acceleration

(2.1) rvU_p Quantum-State-Preparation

(2.2) rvU_N Quantum Neural Computation

(2.3) *M* Measurement

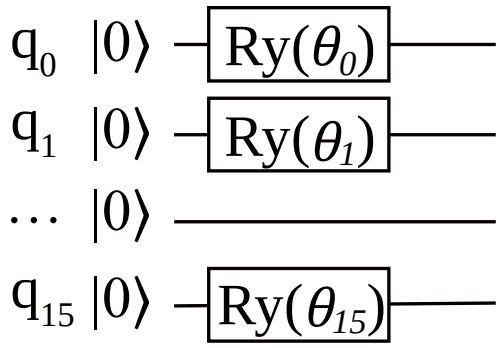
(3) Data Post-Processing (*PostP*)

rvU_P --- Data Encoding / Quantum State Preparation

- **Given:** A vector of input data, ranging from [0,1] (do scaling in *PreP* if range out of [0,1])
- **Do:** Applying rotation gate to encode each data to one qubits
- **Output:** A quantum circuit, where the probability of each qubit to be $|1\rangle$ is the same as the corresponding input data

0.0	0.9	0.0	0.0
0.0	0.1	0.0	0.0
0.0	1.0	0.5	0.5
0.0	0.0	0.0	0.0

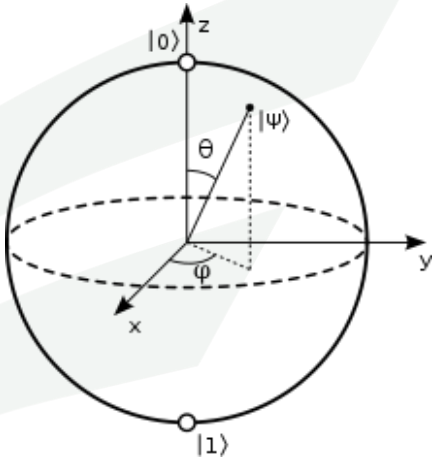
r.v.	-1	+1
x_0	0.0	1.0
x_1	0.9	0.1
…	…	
x_{15}	0.0	1.0



Determination of θ_i:

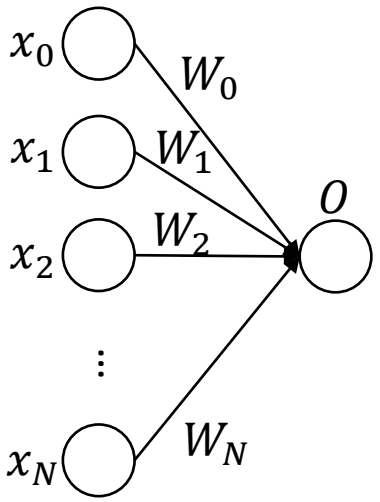
$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + (\cos\phi + i\cdot\sin\phi)\cdot\sin\frac{\theta}{2}|1\rangle$$

$$\theta_i = 2 \times \arcsin(\sqrt{x_i})$$



[1] W. Jiang, et al. A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage, Nature Communications

rvU_N --- Neural Computation



- **Given:** (1) A circuit with encoded input data $\boldsymbol{x}$; (2) the trained binary weights $\boldsymbol{w}$ for one neural computation, which will be associated to each data.
- **Do:** Place quantum gates on qubits, such that it performs $\frac{(\boldsymbol{x}*\boldsymbol{w})^2}{\|\boldsymbol{x}\|^2}$, where $\boldsymbol{x}$ are random variables

Target: $O = \left[\frac{\sum_i (x_i \times w_i)}{\|\boldsymbol{x}\|}\right]^2$

- **Assumption 1:** Parameters/weights (W_0 --- W_N) are binary weight, either +1 or -1
- **Assumption 2:** The weight $W_0 = +1$, otherwise we can use $-\boldsymbol{w}$ (quadratic func.)

Step 1: $m_i = x_i \times w_i$

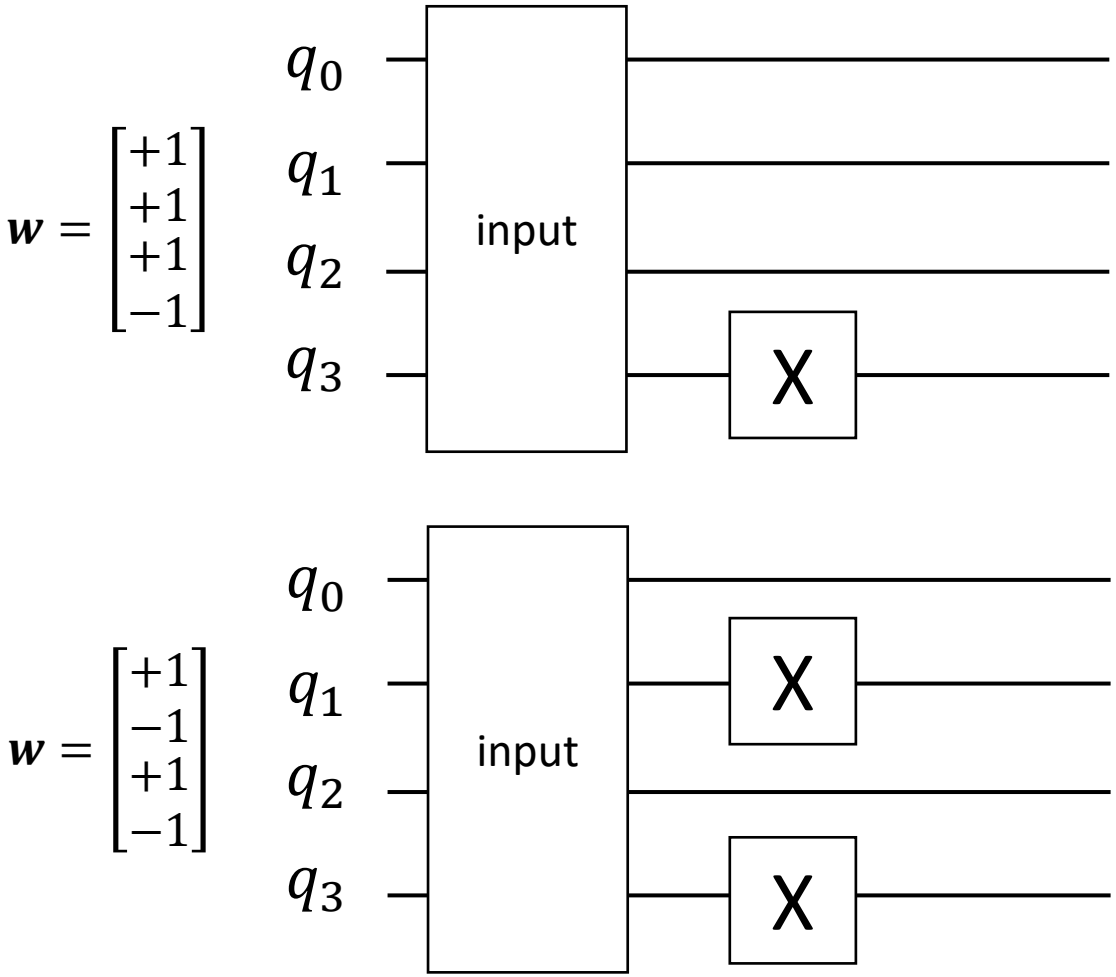
Step 2: $n = \left[\frac{\sum_i (m_i)}{\|\boldsymbol{x}\|}\right]$

Step 3: $O = n^2$

rvU_N --- Neural Computation: Step 1

Step 1: $m_i = x_i \times w_i$

EX: 4 input data on 4 qubits



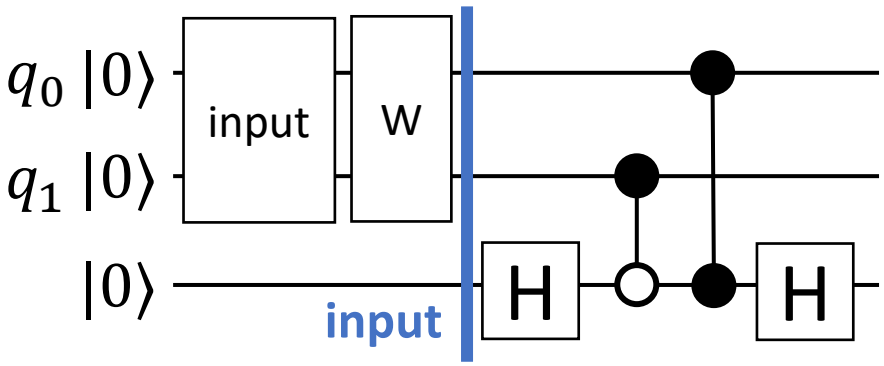
rvU_N --- Neural Computation: Step 2

Step 2: $n = \left[\frac{\sum_i (m_i)}{\|x\|}\right]$

r.v.	-1 ($\|1\rangle$)	+1 ($\|0\rangle$)
m_0	p_0	q_0
m_1	p_1	q_1

r.v.	-1	0	+1
n	$p_0 p_1$	$p_0 q_1 + p_1 q_0$	$q_0 q_1$

EX: 2 input data on 2 qubits



Input

$\sqrt{q_0 q_1}$	$\|000\rangle$
0	$\|001\rangle$
$\sqrt{q_0 p_1}$	$\|010\rangle$
0	$\|011\rangle$
$\sqrt{p_0 q_1}$	$\|100\rangle$
0	$\|101\rangle$
$\sqrt{p_0 p_1}$	$\|110\rangle$
0	$\|111\rangle$

IIH+CZs

$\sqrt{q_0 q_1}/\sqrt{2}$	$\|000\rangle$
$\sqrt{q_0 q_1}/\sqrt{2}$	$\|001\rangle$
$-\sqrt{q_0 p_1}/\sqrt{2}$	$\|010\rangle$
$\sqrt{q_0 p_1}/\sqrt{2}$	$\|011\rangle$
$\sqrt{p_0 q_1}/\sqrt{2}$	$\|100\rangle$
$-\sqrt{p_0 q_1}/\sqrt{2}$	$\|101\rangle$
$-\sqrt{p_0 p_1}/\sqrt{2}$	$\|110\rangle$
$-\sqrt{p_0 p_1}/\sqrt{2}$	$\|111\rangle$

IIH

$\sqrt{q_0 q_1}$	$\|000\rangle$
---	$\|100\rangle$
0	$\|001\rangle$
---	$\|101\rangle$
0	$\|010\rangle$
---	$\|110\rangle$
$-\sqrt{p_0 p_1}$	$\|011\rangle$
---	$\|111\rangle$

rvU_N --- Neural Computation: Step 3

Classical:

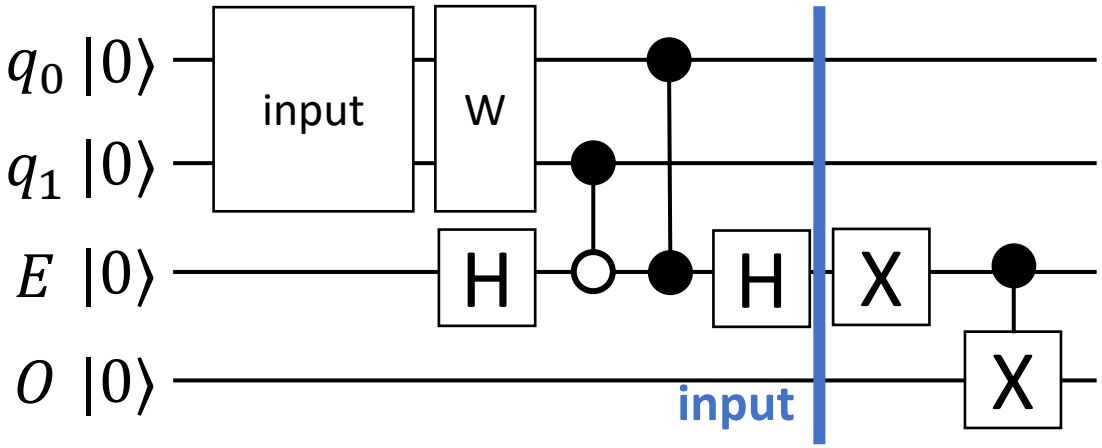
$$E(O) = E(n^2)$$
$$= 0 \times (p_0q_1 + p_1q_0) + 1 \times (q_0q_1 + p_0p_1)$$

Step 3: $O = n^2$

r.v.	-1	0	+1
n	p_0p_1	$p_0q_1 + p_1q_0$	q_0q_1

r.v.	0	+1
n^2	$p_0q_1 + p_1q_0$	$q_0q_1 + p_0p_1$

EX: 2 input data on 2 qubits



Input

$\sqrt{q_0q_1}$	$\|000\rangle$
---	$\|001\rangle$
0	$\|010\rangle$
---	$\|011\rangle$
0	$\|100\rangle$
---	$\|101\rangle$
$-\sqrt{p_0p_1}$	$\|110\rangle$
---	$\|111\rangle$

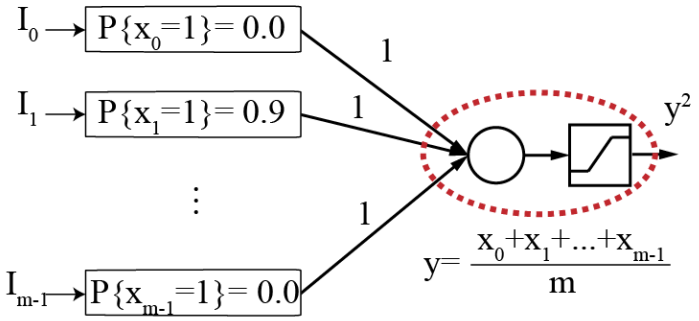
IIX

---	$\|000\rangle$
$\sqrt{q_0q_1}$	$\|001\rangle$
---	$\|010\rangle$
0	$\|011\rangle$
---	$\|100\rangle$
0	$\|101\rangle$
---	$\|110\rangle$
$-\sqrt{p_0p_1}$	$\|111\rangle$

Quantum:

$$P(E = |1\rangle)$$
$$= \sqrt{q_1q_0}^2 + (-\sqrt{p_1p_0})^2$$
$$= q_0q_1 + p_0p_1$$

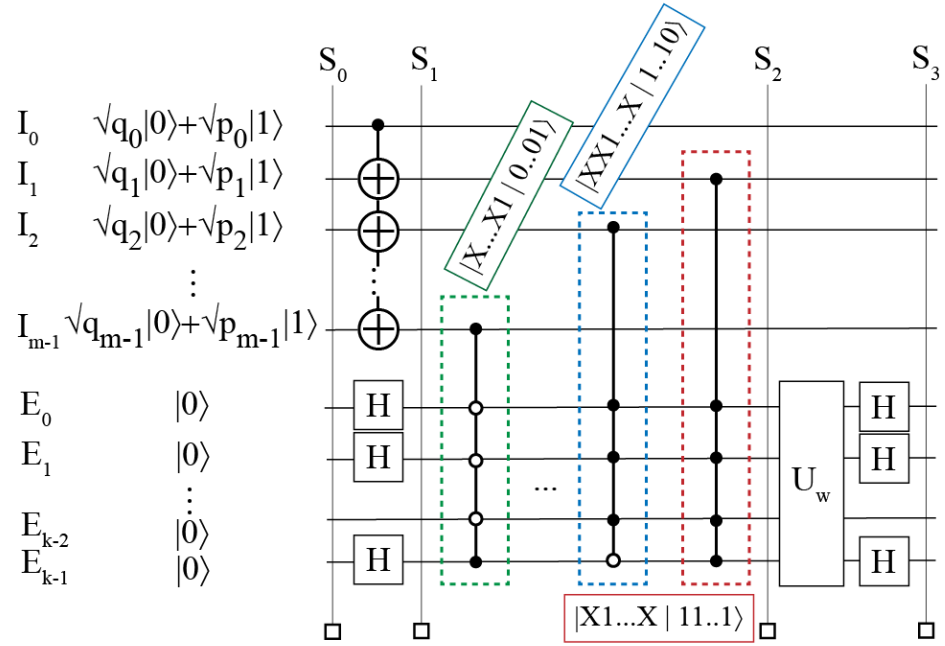
rvU_N --- Neural Computation



	|1⟩	|0⟩
	-1	+1
x_0	p_0	q_0
x_1	p_1	q_1
⋮		
x_{m-1}	p_{m-1}	q_{m-1}

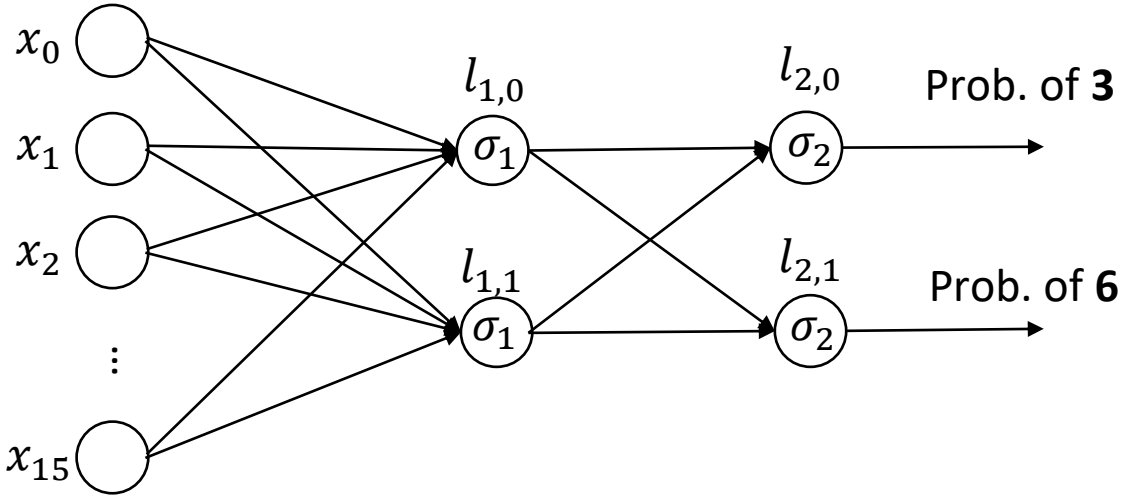
y	-1	$\frac{-m+2}{m}$	⋯	0	⋯	$\frac{m-2}{m}$	1
	Πp_i	$p_{m-1}...p_1q_0 + p_{m-1}...q_1p_0 + \vdots + q_{m-1}...p_1p_0$		…		$q_{m-1}...q_1p_0 + q_{m-1}...p_1q_0 + \vdots + p_{m-1}...q_1q_0$	Πq_i

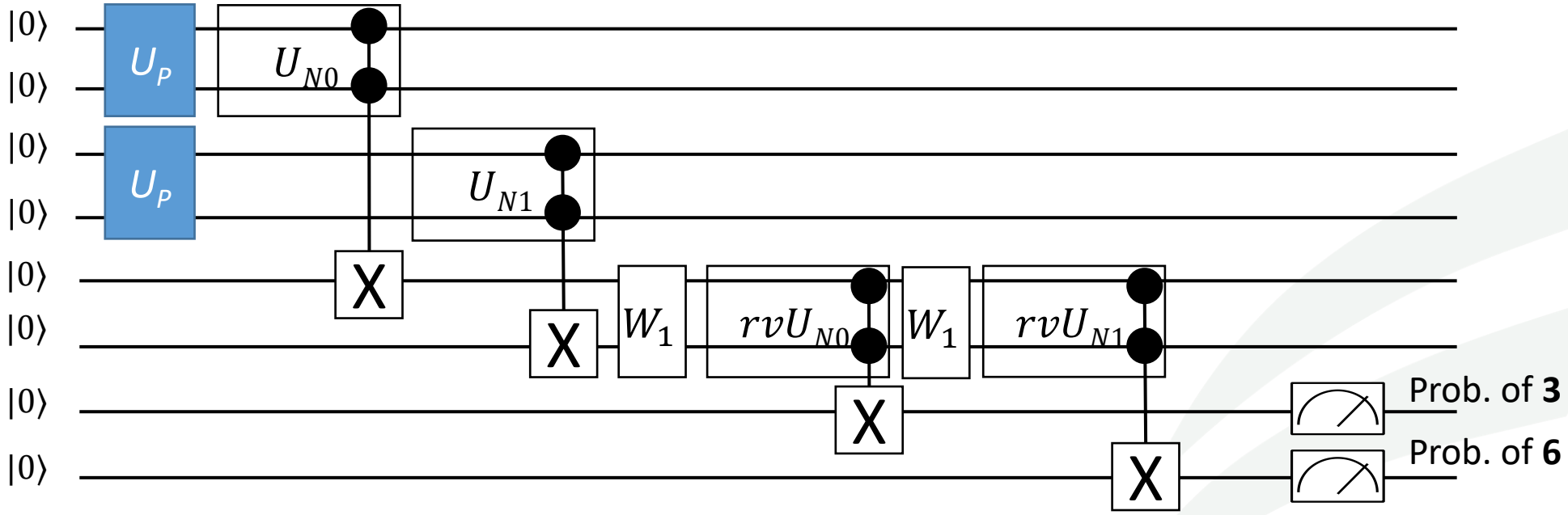
y^2	0 ⋯	$(\frac{m-2}{m})^2$	1
		$p_{m-1}...p_1q_0 + p_{m-1}...q_1p_0 + \vdots + q_{m-1}...p_1p_0$ + $q_{m-1}...q_1p_0 + q_{m-1}...p_1q_0 + \vdots + p_{m-1}...q_1q_0$	$\Pi q_i + \Pi p_i$



m-k Encoder States	Amplitude			
	S_0	S_1	S_2	S_3
|00...0⟩⊗|0..0⟩	$\sqrt{q_{m-1}q_{m-2}...q_0}$	$\sqrt{q_{m-1}q_{m-2}...q_0}$	$\sqrt{q_{m-1}q_{m-2}...q_0}$	$\sqrt{q_{m-1}q_{m-2}...q_0}$
|00...0⟩⊗|0..1⟩	0	$\frac{1}{2^{k/2}}$ $\sqrt{q_{m-1}q_{m-2}...q_0}$	$\frac{1}{2^{k/2}}$ $\sqrt{q_{m-1}q_{m-2}...q_0}$	xxxxxxxxxx
...	...	...	...	...
|00...0⟩⊗|1..1⟩	0	$\sqrt{q_{m-1}q_{k-1}...q_0}$	$\sqrt{q_{m-1}q_{m-2}...q_0}$	xxxxxxxxxx
|00...1⟩⊗|0..0⟩	$\sqrt{q_{m-1}q_{m-2}...p_0}$	$\sqrt{q_{m-1}q_{m-2}...p_0}$	$\sqrt{q_{m-1}q_{m-2}...p_0}$	(m-2)/m $\sqrt{q_{m-1}q_{m-2}...p_0}$
|00...1⟩⊗|0..1⟩	0	$\frac{1}{2^{k/2}}$ $\sqrt{q_{m-1}q_{m-2}...p_0}$	$\frac{1}{2^{k/2}}$ $-\sqrt{q_{m-1}q_{m-2}...p_0}$	xxxxxxxxxx
...	...	...	...	...
|00...1⟩⊗|1..1⟩	0	$\sqrt{q_{m-1}q_{m-2}...p_0}$	$\sqrt{q_{m-1}q_{m-2}...p_0}$	xxxxxxxxxx
...	...	...	...	...
|11...1⟩⊗|0..0⟩	$\sqrt{p_{m-1}p_{m-2}...p_0}$	$\sqrt{p_{m-1}q_{m-2}...q_0}$	$\sqrt{p_{m-1}q_{m-2}...q_0}$	(2-m)/m $\sqrt{q_{m-1}q_{m-2}...p_0}$
|11...1⟩⊗|0..1⟩	0	$\frac{1}{2^{k/2}}$ $\sqrt{p_{m-1}q_{m-2}...q_0}$	$\frac{1}{2^{k/2}}$ $-\sqrt{p_{m-1}q_{m-2}...q_0}$	xxxxxxxxxx
...	...	...	...	...
|11...1⟩⊗|1..1⟩	0	$\sqrt{p_{m-1}q_{m-2}...q_0}$	$-\sqrt{p_{m-1}q_{m-2}...q_0}$	xxxxxxxxxx

Implementing Feedforward Net w/ Non-Linearity, w/o Measurement!

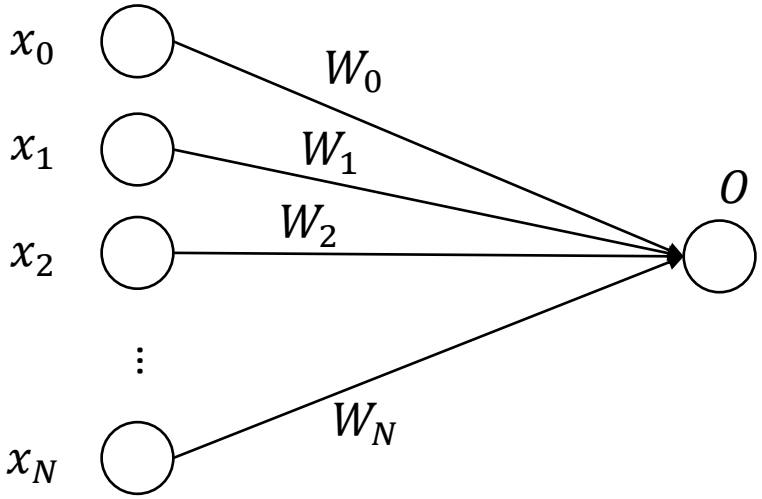


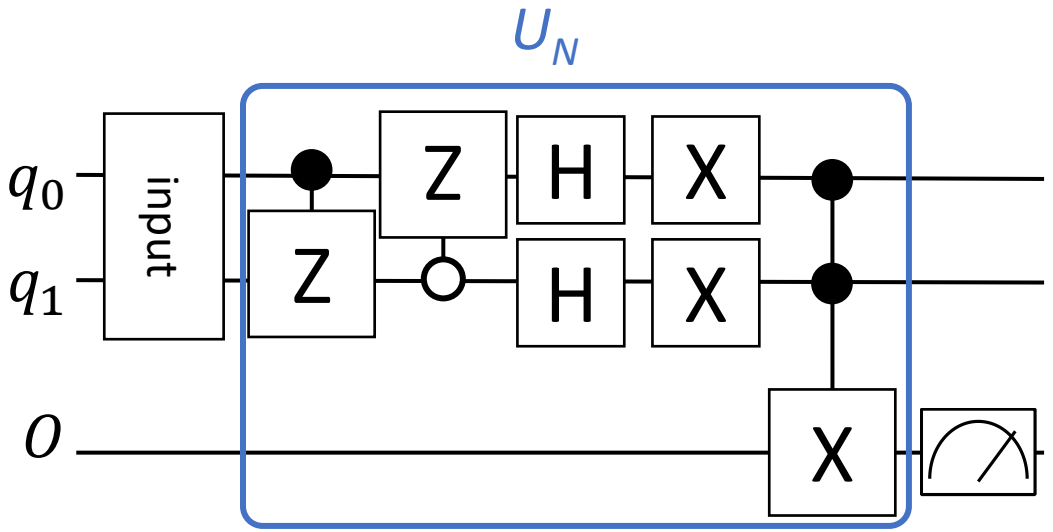


Hands-On

Tutorial 3: *PreP + U_P + U_N+ M+PostP (MNIST)*

Challenge 3: High Complexity in the Previous Design





Cost Complexity

Classical Computing		
	No Parallelism	Full Parallelism
Time (T)	O(N)	O(1)
Space (S)	O(1)	O(N)
Cost (TS)	O(N)	O(N)

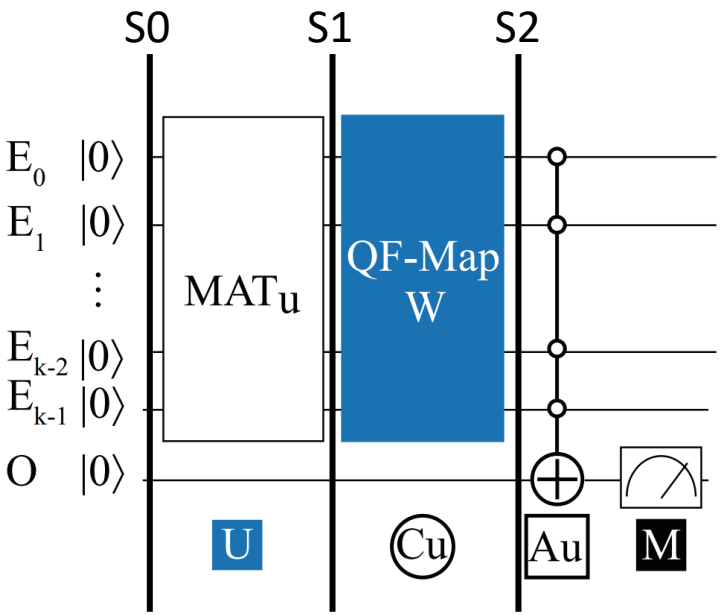
Quantum Computing		
	Previous Design	Optimization
Circuit Depth (T)	O(N)	???
Qubits (S)	O($\log N$)	O(N)
Cost (TS)	O($N \cdot \log N$)	**target** O(ploylog N)

QuantumFlow: Taking NN Property to Design QC

$[0, 0.9, 0, 0, 0, 0.1, 0, 0, 1.0, 0.5, 0.5, 0, 0, 0, 0]^T$

U ↓

$[0, 0.59, 0, 0, 0, 0.07, 0, 0, 0.66, 0.33, 0.33, 0, 0, 0, 0]^T$



S0 -> S1:

$$(v_o; v_{x1}; v_{x2}; \ldots; v_{xn}) \times \begin{pmatrix} 1 \\ 0 \\ \ldots \\ 0 \end{pmatrix} = (v_0)$$

$$S1 = [0, 0.59, 0, 0, 0, 0.07, 0, 0, 0.66, 0.33, 0.33, 0, 0, 0, 0]^T$$

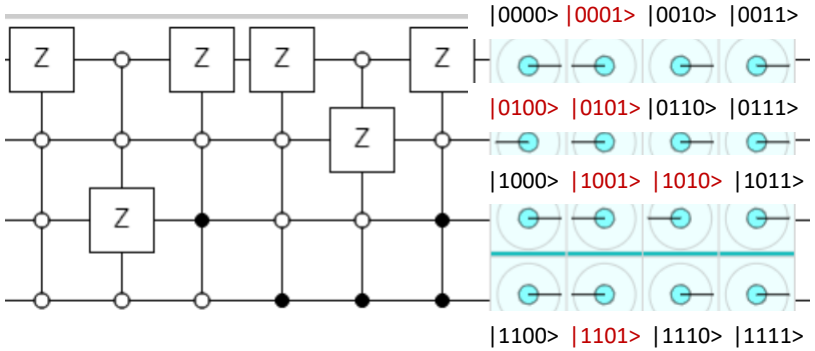
S1 -> S2:

$$W = [+1, -1, +1, +1, -1, -1, +1, +1, +1, -1, -1, +1, +1, -1, +1, +1]^T$$

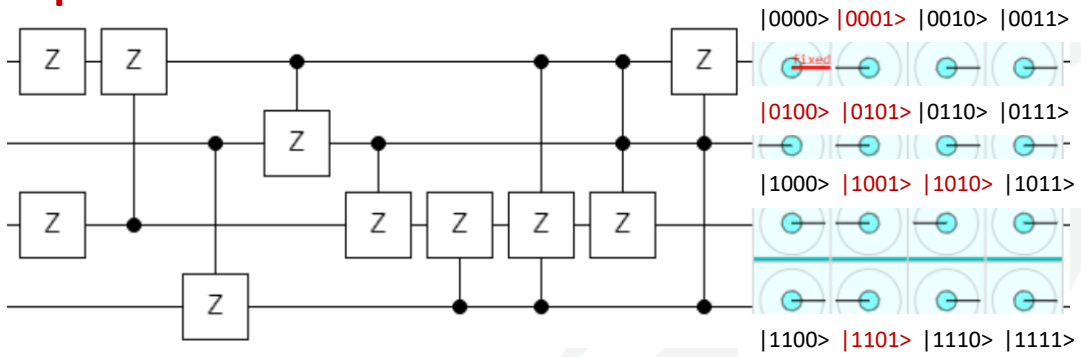
|0000> |0001> |0010> |0011> |0100> |0101> |0110> |0111> |1000> |1001> |1010> |1011> |1100> |1101> |1110> |1111>

$$S2 = [0, -0.59, 0, 0, -0, -0.07, 0, 0, 0, -0.66, -0.33, 0.33, 0, -0, 0, 0]^T$$

Implementation 1 (example in Quirk):

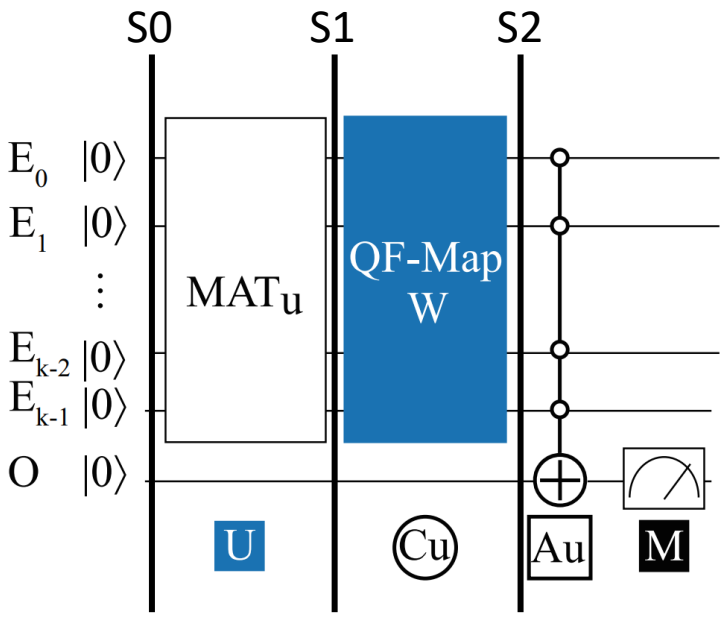


Implementation 2:



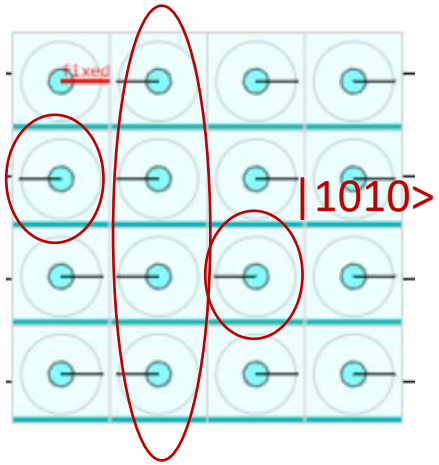
[ref] Tacchino, F., et al., 2019. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, *5*(1), pp.1-8.

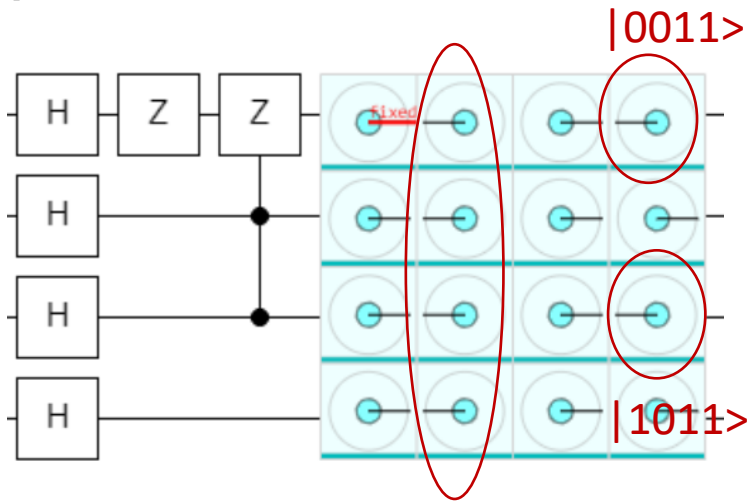
QuantumFlow: Taking NN Property to Design QC



Property from NN

- The **weight order** is not necessary to be fixed, which can be adjusted if the order of inputs are adjusted accordingly
- **Benefit:** No need to require the positions of sign flip are exactly the same with the weights; instead, only need the number of signs are the same.





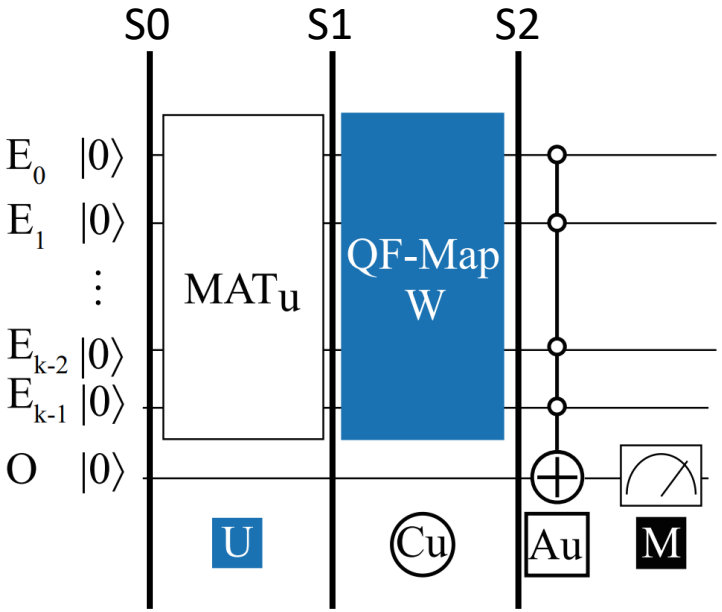
$$S1 = [0, 0.59, 0, \mathbf{0}, \mathbf{0}, 0.07, 0, 0, 0.66, \mathbf{0.33}, \mathbf{0.33}, 0, 0, 0, 0]^T$$

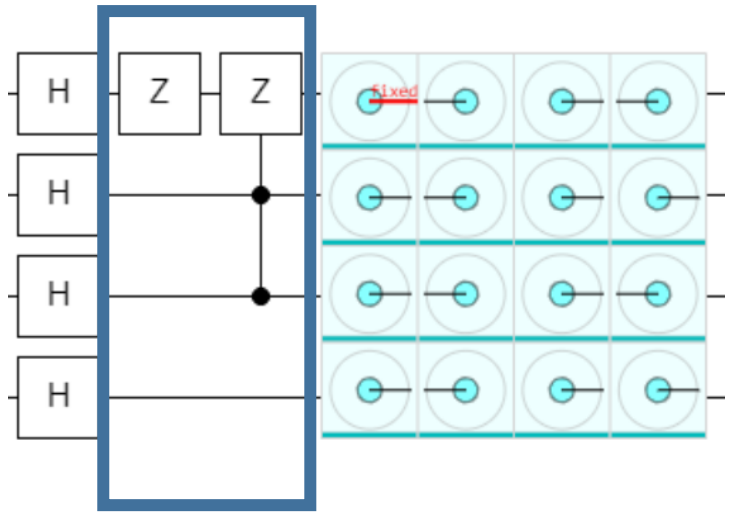
ori + - - +

fin - + + -

$$S1' = [0, 0.59, 0, \mathbf{0.33}, \mathbf{0.33}, 0.07, 0, 0, 0.66, \mathbf{0}, \mathbf{0}, 0, 0, 0, 0]^T$$

QuantumFlow: Taking NN Property to Design QC





Algorithm 4: QF-Map: weight mapping algorithm

Input: (1) An integer $R \in (0, 2^{k-1}]$; (2) number of qbits k;
Output: A set of applied gate G

```
void recursive(G,R,k){
    if (R < 2^{k-2}){
        recursive(G,R,k − 1); // Case 1 in the third step
    }
    else if (R == 2^{k-1}){
        G.append(PG_{2^{k-1}}); // Case 2 in the third step
        return;
    }else{
        G.append(PG_{2^{k-1}});
        recursive(G,2^{k-1} − R,k − 1); // Case 3 in the third step
    }
}
// Entry of weight mapping algorithm
set main(R,k){
    Initialize empty set G;
    recursive(G,R,k);
    return G
}
```

Used gates and Costs

Gates	Cost
Z	1
CZ	1
C^2Z	3
C^3Z	5
C^4Z	6
...	...
C^kZ	2k-1

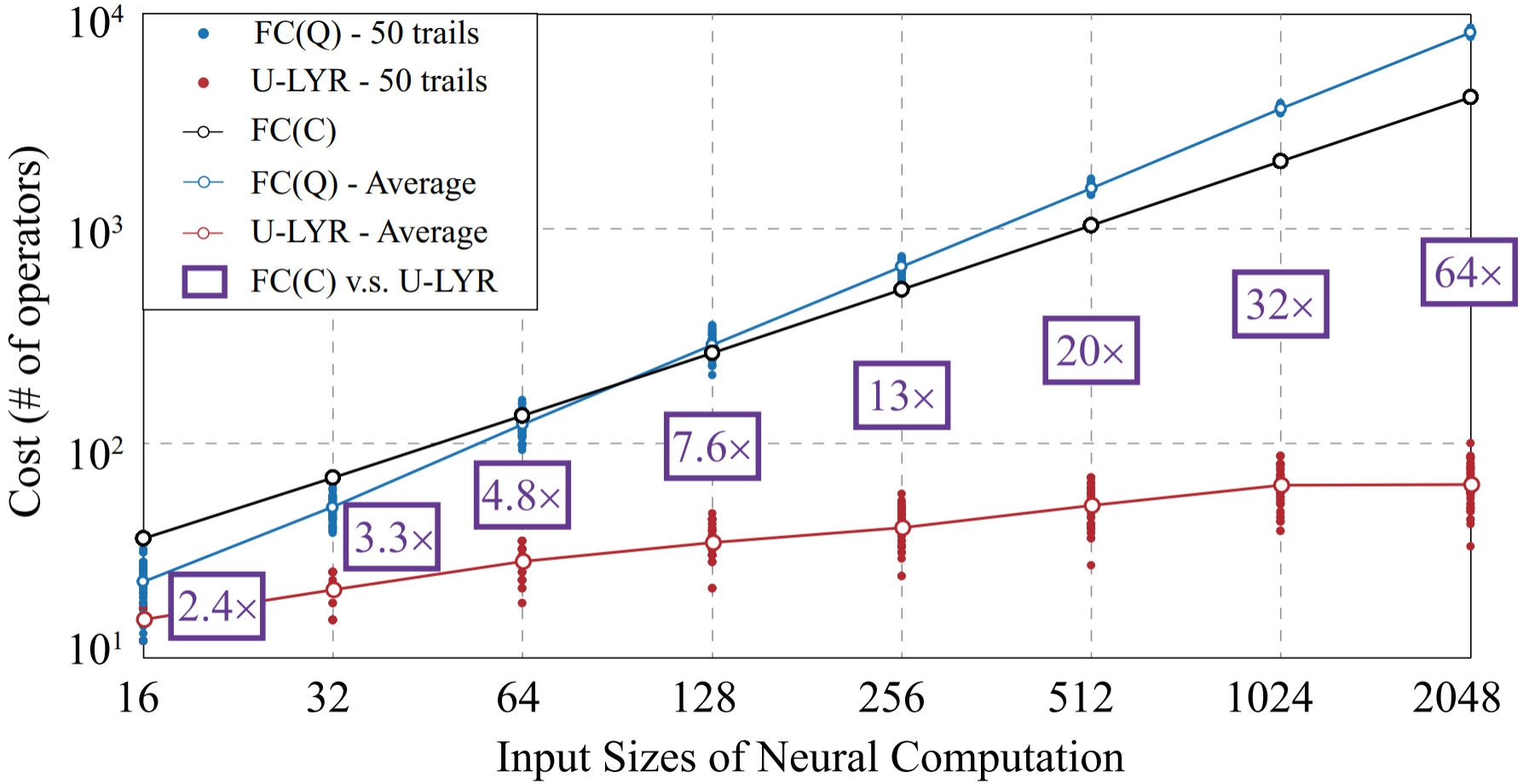
Worst case: all gates

$O(k^2)$

Hands-On

Tutorial 4: *PreP + U_P + Optimized U_N+ M+PostP (MNIST)*

QuantumFlow Results



[ref] Tacchino, F., et al., 2019. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, *5*(1), pp.1-8.

QuantumFlow Achieves Over 10X Cost Reduction

Dataset	Structure			MLP(C)			FFNN(Q)				QF-hNet(Q)			
	In	L1	L2	L1	L2	Tot.	L1	L2	Tot.	**Red.**	L1	L2	Tot.	**Red.**
{1,5}	16	4	2	132	18	150	80	38	118	**1.27**×	74	38	112	**1.34**×
{3,6}	16	4	2				96	38	134	**1.12**×	58	38	96	**1.56**×
{3,8}	16	4	2				76	34	110	**1.36**×	58	34	92	**1.63**×
{3,9}	16	4	2				98	42	140	**1.07**×	68	42	110	**1.36**×
{0,3,6}	16	8	3	264	51	315	173	175	348	**0.91**×	106	175	281	**1.12**×
{1,3,6}	16	8	3				209	161	370	**0.85**×	139	161	300	**1.05**×
{0,3,6,9}	64	16	4	2064	132	2196	1893	572	2465	**0.89**×	434	572	1006	**2.18**×
{0,1,3,6,9}	64	16	5	2064	165	2229	1809	645	2454	**0.91**×	437	645	1082	**2.06**×
{0,1,2,3,4}	64	16	5				1677	669	2346	**0.95**×	445	669	1114	**2.00**×
{0,1,3,6,9}*	256	8	5	4104	85	4189	5030	251	5281	**0.79**×	135	251	386	**10.85**×

*: Model with 16×16 resolution input for dataset {0,1,3,6,9} to test scalability, whose accuracy is 94.09%, which is higher than 8×8 input with accuracy of 92.62%.

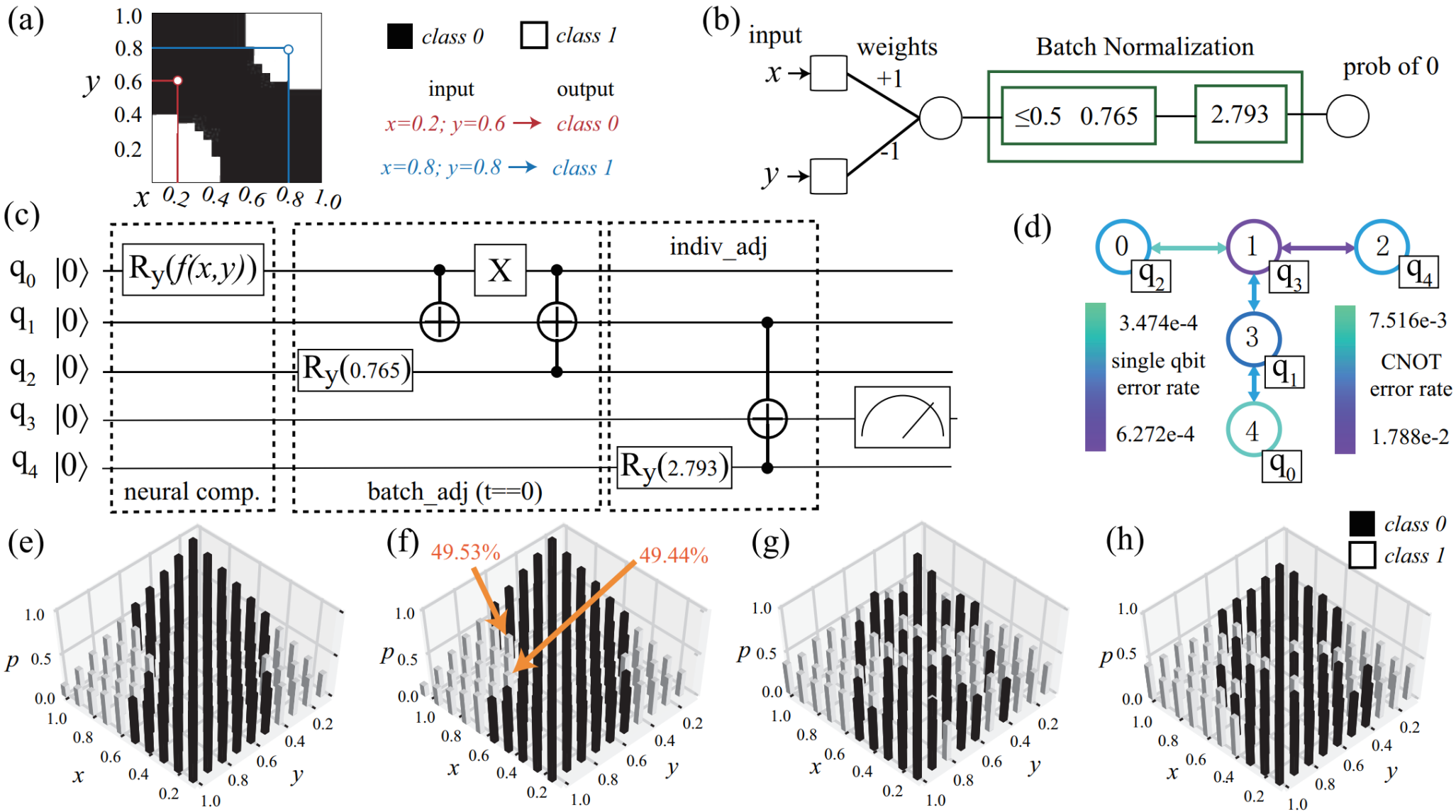
[ref of FFNN] Tacchino, F., et al., 2019. Quantum implementation of an artificial feed-forward neural network. *arXiv preprint arXiv:1912.12486*.

QF-Nets Achieve the Best Accuracy on MNIST

Dataset	w/o BN					w/ BN				
	binMLP(C)	FFNN(Q)	MLP(C)	QF-pNet	QF-hNet	binMLP(C)	FFNN(Q)	MLP(C)	QF-pNet	QF-hNet
1,5	61.47%	61.47%	69.12%	69.12%	90.33%	55.99%	55.99%	85.30%	84.56%	**96.60%**
3,6	72.76%	72.76%	94.21%	91.67%	97.21%	72.76%	72.76%	96.29%	96.39%	**97.66%**
3,8	58.27%	58.27%	82.36%	82.36%	89.77%	58.37%	58.07%	86.74%	86.90%	**87.20%**
3,9	56.71%	56.51%	68.65%	68.30%	95.49%	56.91%	56.71%	80.63%	78.65%	**95.59%**
0,3,6	46.85%	51.63%	49.90%	59.87%	89.65%	50.68%	50.68%	75.37%	78.70%	**90.40%**
1,3,6	60.04%	59.97%	53.69%	53.69%	94.68%	59.59%	59.59%	86.76%	86.50%	**92.30%**
0,3,6,9	72.68%	72.33%	84.28%	87.36%	92.85%	69.95%	68.89%	82.89%	76.78%	**93.63%**
0,1,3,6,9	50.00%	51.10%	49.00%	43.24%	87.96%	60.96%	69.46%	70.19%	71.56%	**92.62%**
0,1,2,3,4	46.96%	50.01%	49.06%	52.95%	83.95%	64.51%	69.66%	71.82%	72.99%	**90.27%**

[ref of FFNN] Tacchino, F., et al., 2019. Quantum implementation of an artificial feed-forward neural network. *arXiv preprint arXiv:1912.12486*.

On Actual IBM "ibmq_essex" Quantum Processor



wjiang8@gmu.edu

George Mason University

4400 University Drive
Fairfax, Virginia 22030

Tel: (703)993-1000