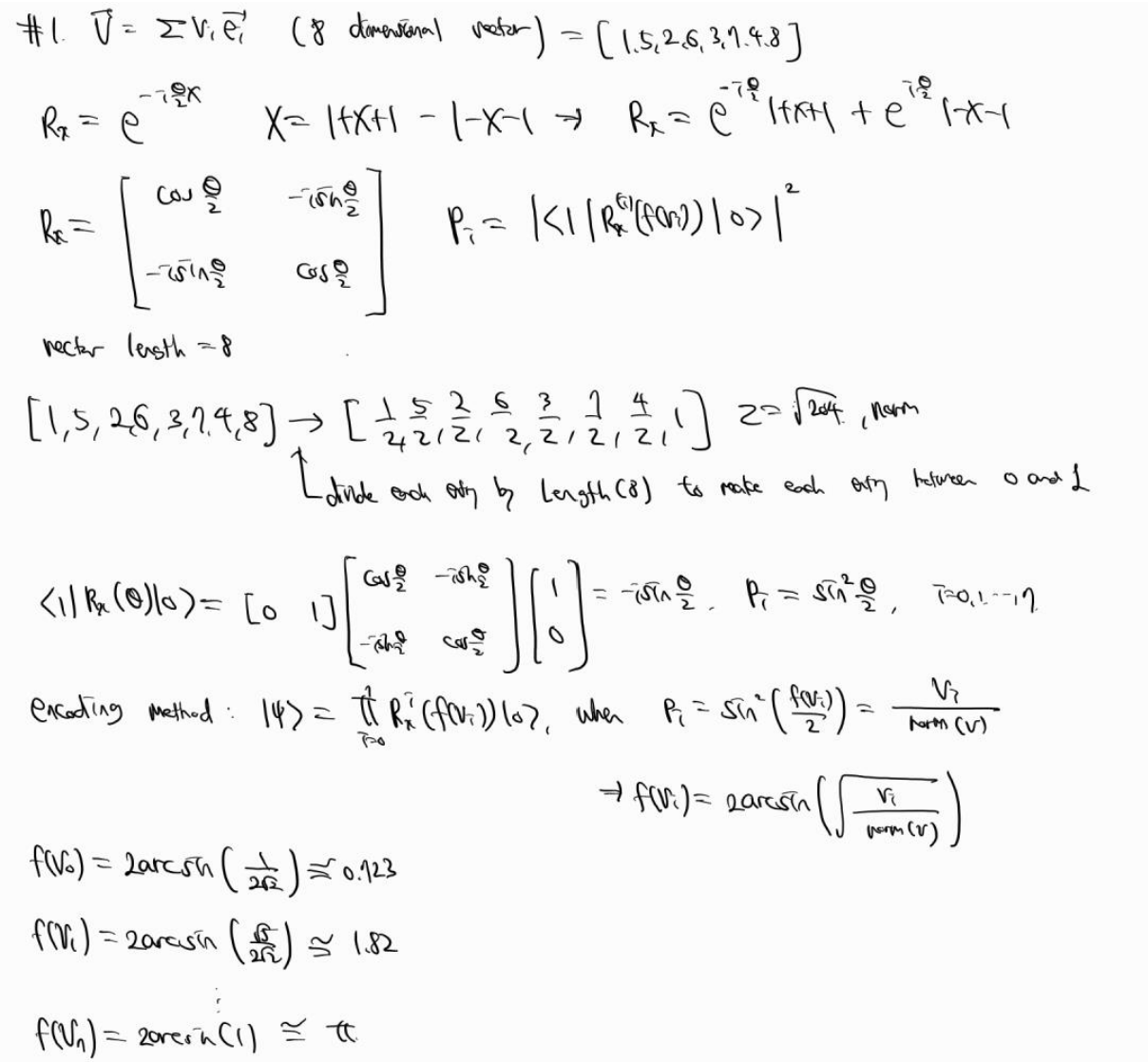
Quantum Challenge 2023 Week 2 Report

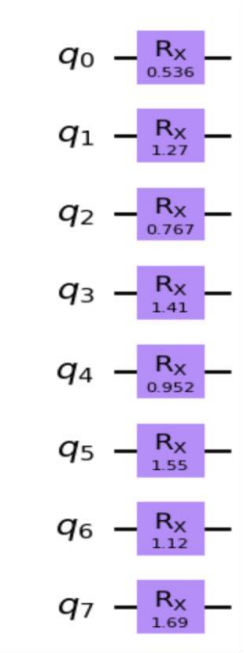
Seoul National University
Electric & Computer Engineering
Sangyeon Lee
이 상 연

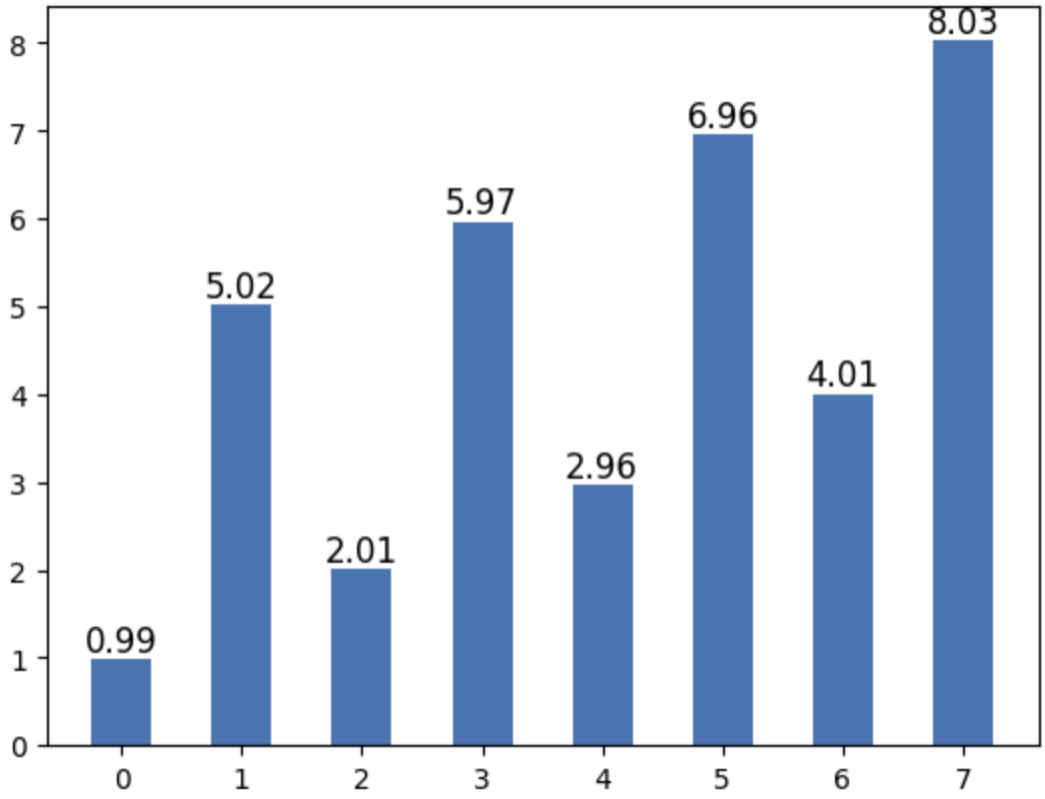
Before introducing my solution, I have some explanation about my work. I tried to use QPU of IONQ. However, there were some issues on my IONQ Cloud API that I couldn't use IONQ Cloud at all. Therefore, I implemented my whole work on IBM QASM Simulator.

1. Product State Encoding to 8 Qubits



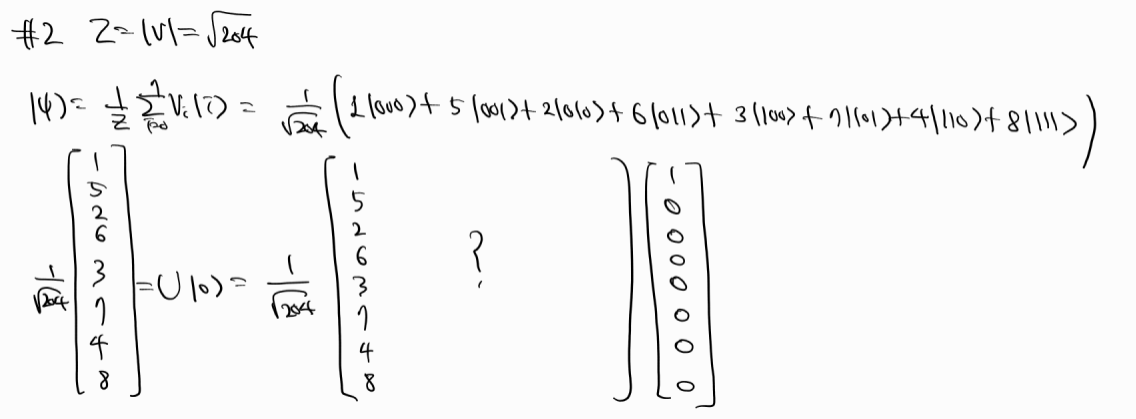
Rx Gate was implemented on each qubits that Probability of measurement 1 is proportional to each vector entry.



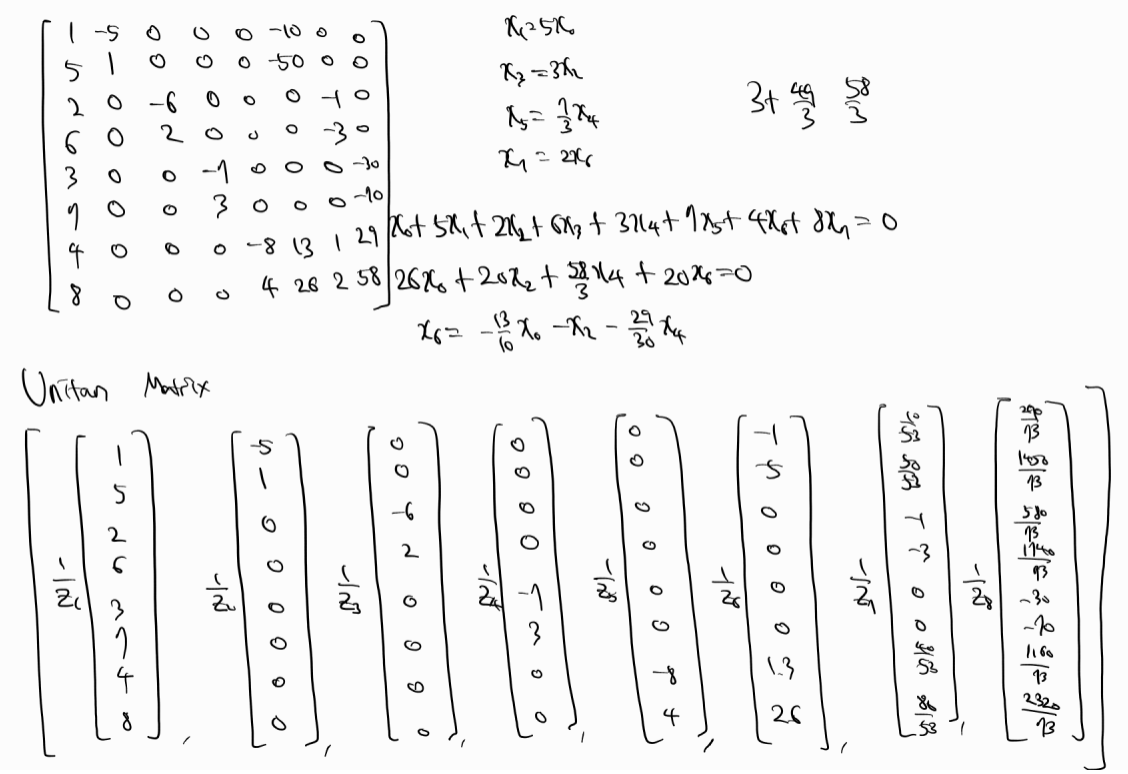


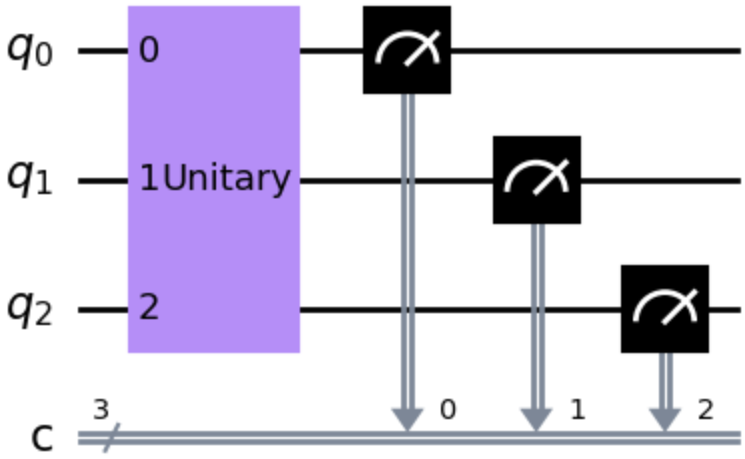
1 Qubit Mid-Circuit Measurement could recover original vector.

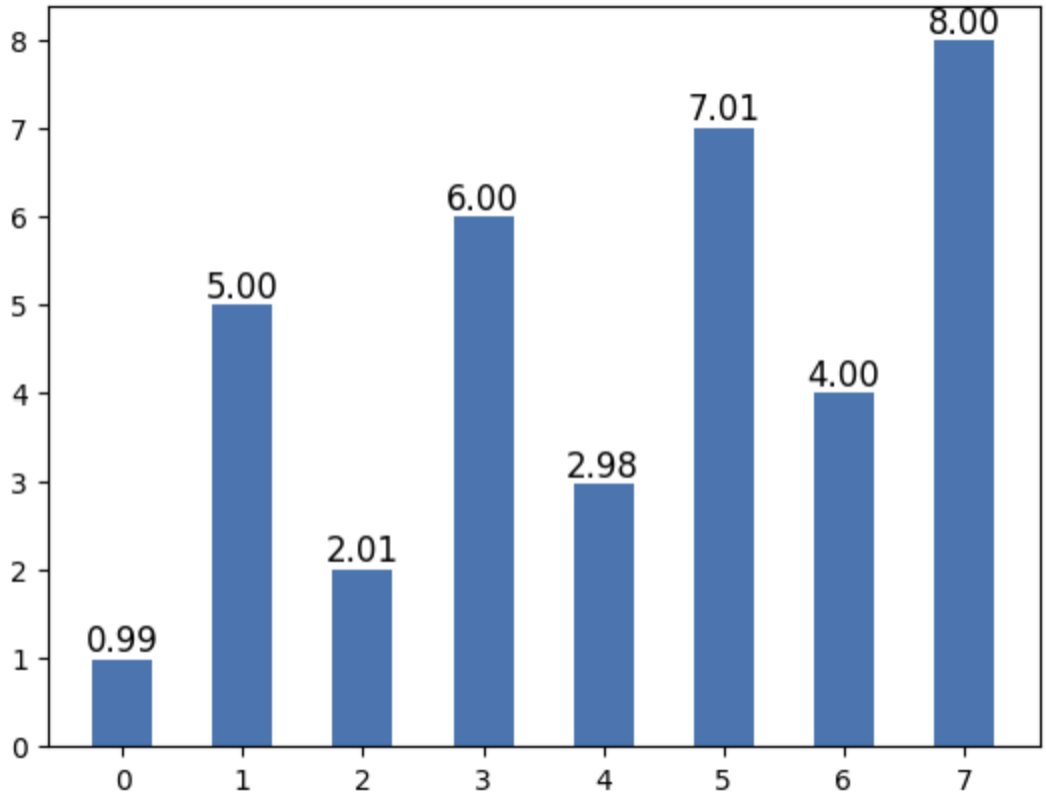
2. Encoding to 3 Qubits



Normalized vector entry is the coefficient of each composite state. I had to find Unitary Operator whose first column is same as normalized vector. I found basis of column space and used Gram-Schmidt to make orthogonal set, which produce Unitary Matrix.







I implemented Unitary Matrix, and After Measurement of each qubit and multiplying norm, I could decode original vector.

3. Enhanced Encoding Method

1) Task1 vs Task2

Task 1

Pros : Only 8 single rotating gates. Low cost in gate number.

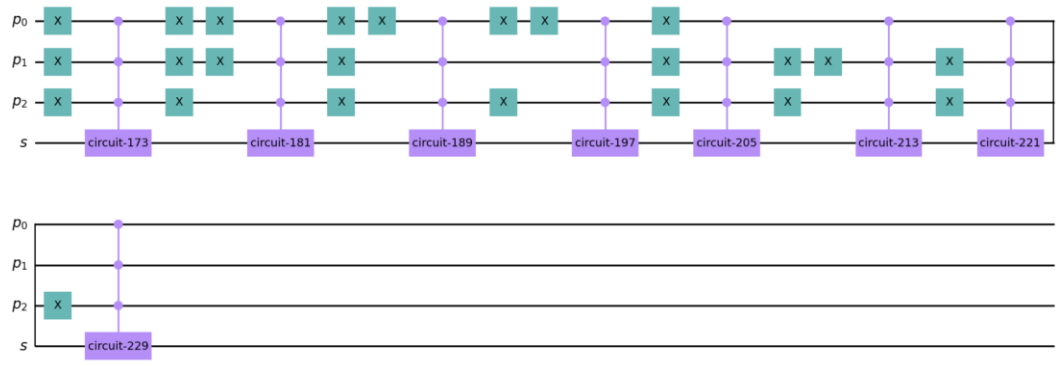
Cons : One-Hot encoding. 8 qubits are used. High cost in qubit number.

Task2

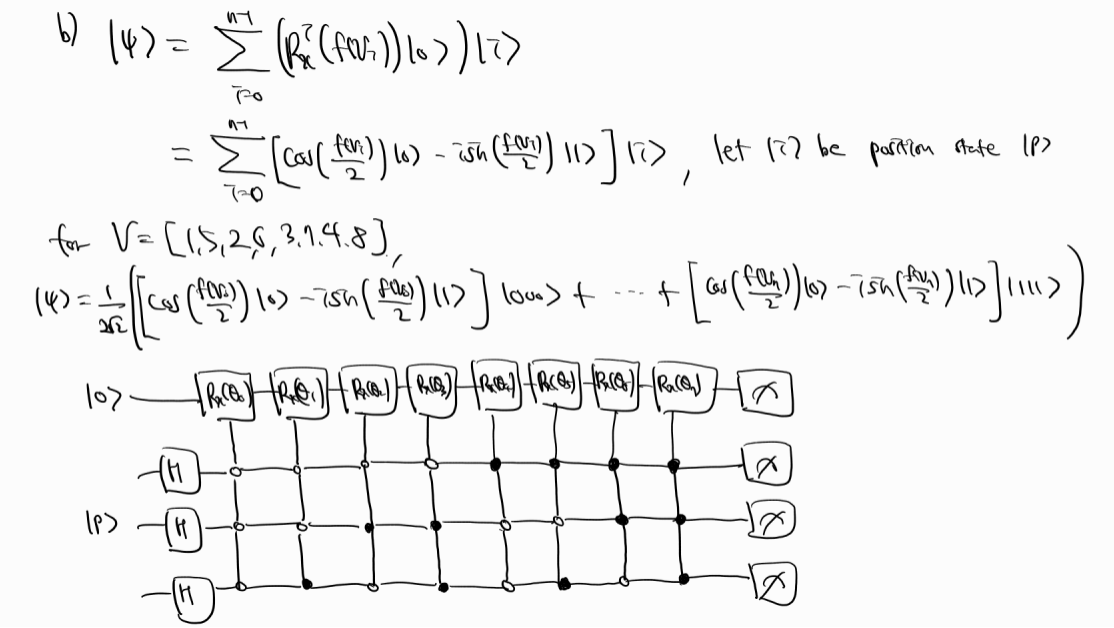
Pros : Binary encoding. Only 3 qubits used for encoding. Low cost in qubit number.

Cons : too much gates to construct accurate unitary matrix whose probability of each state is equal to normalized vector's entry.

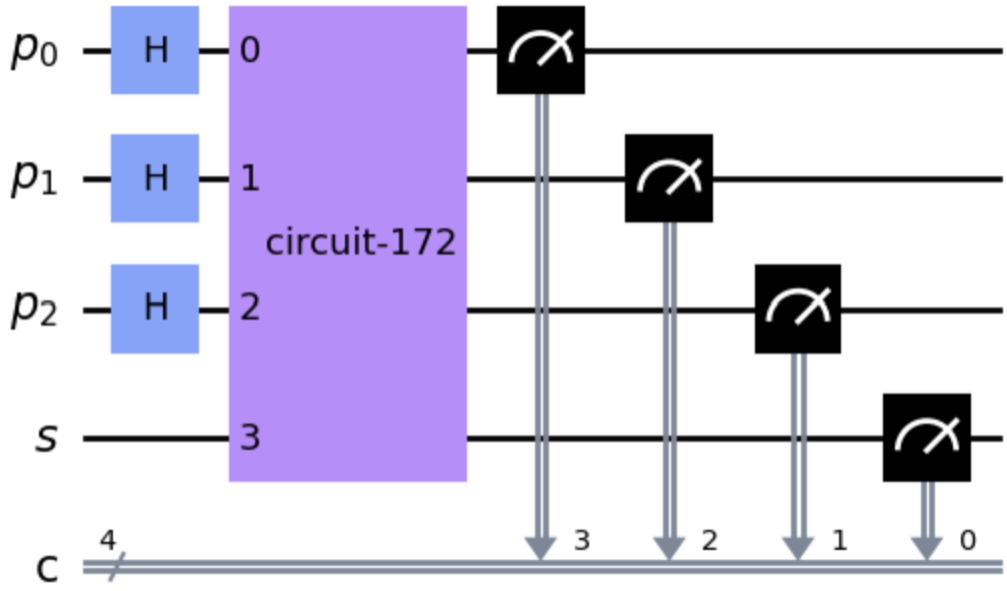
2) Enhanced Encoding

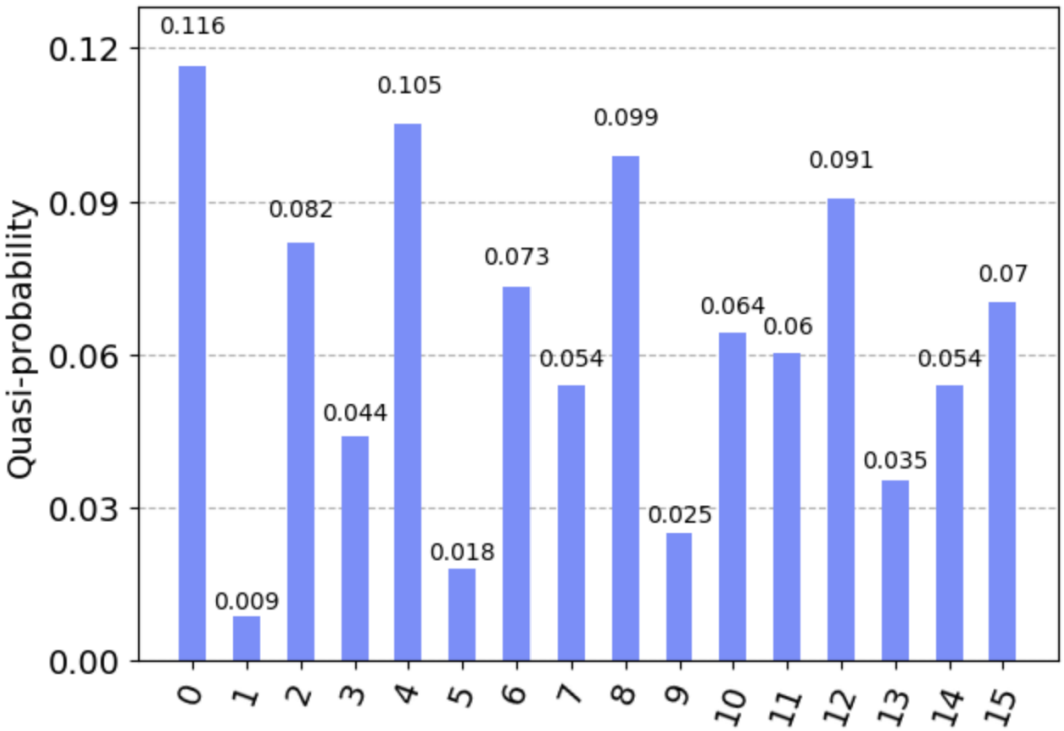


I used CNOT Gates. Why?

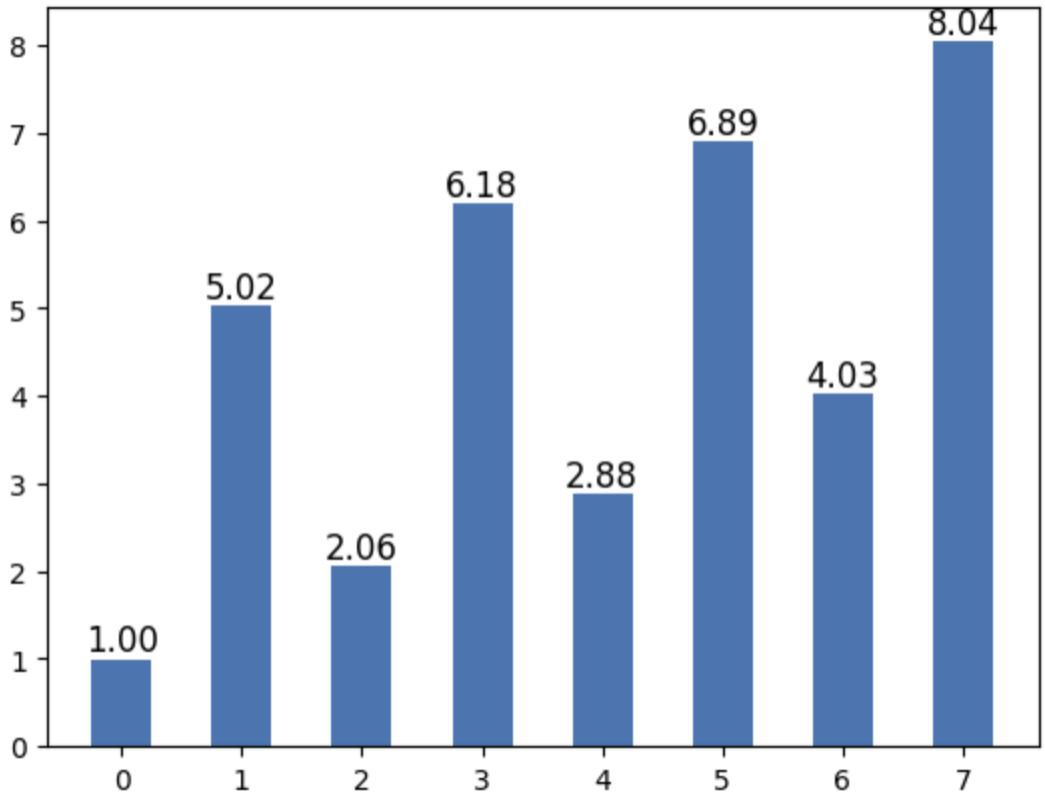


There's position qubit register & probability qubit register. Position qubit register encodes each index, and probability qubit register encodes each entry. Entry appears on probability of measuring the position state corresponding to the index and probability qubit register "1".



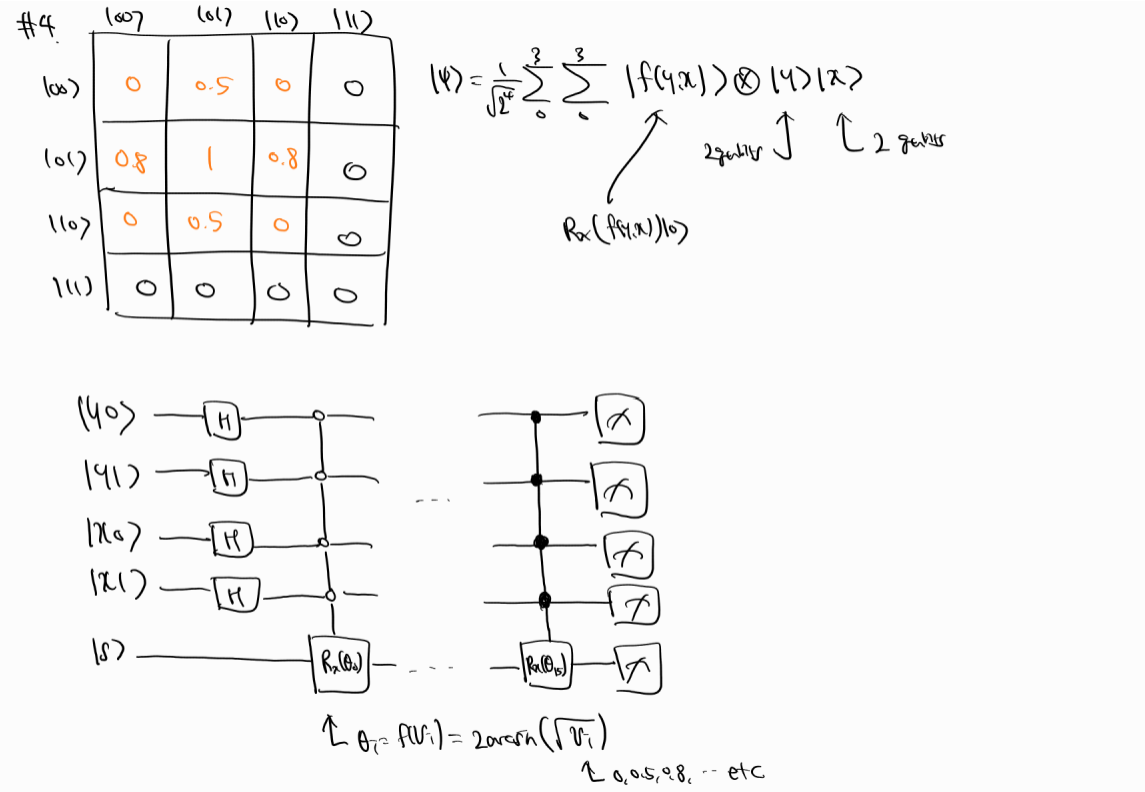


After measuring each composite states, I only need probability of composite states that contains probability qubit register "1". Therefore, I need state 1, 3, 5, 7, … Each vector entry comes from multiplying norm and some power of sqrt 2 to probability of state 1, 3, 5, 7, …

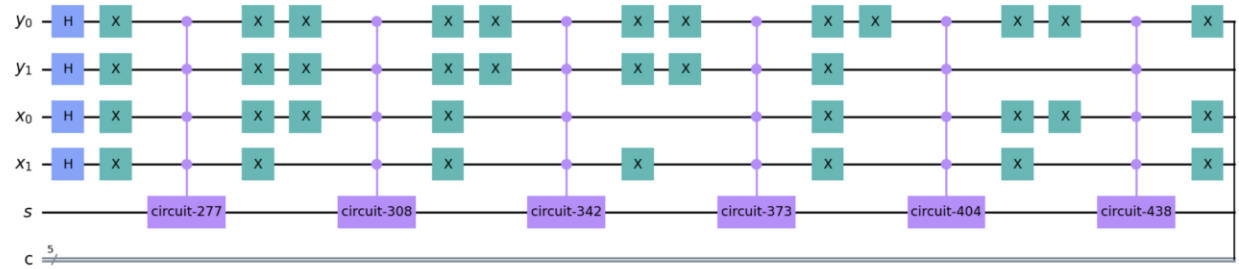


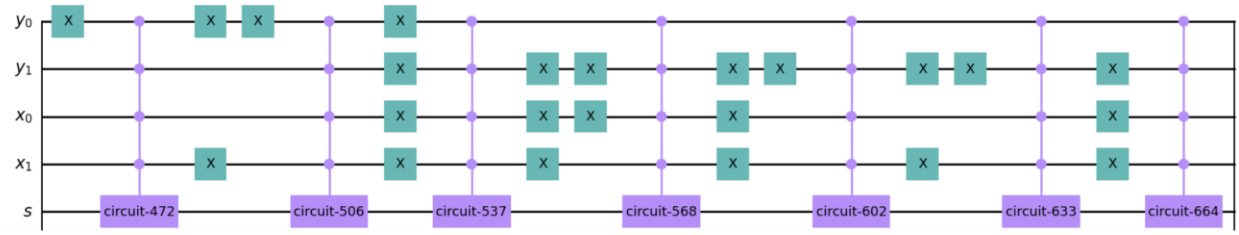
Enhanced Encoding only needs 4 qubits, and 3 Hadamard Gate + 8 CNOT Gate.

4. Encoding 3 X 3 Image

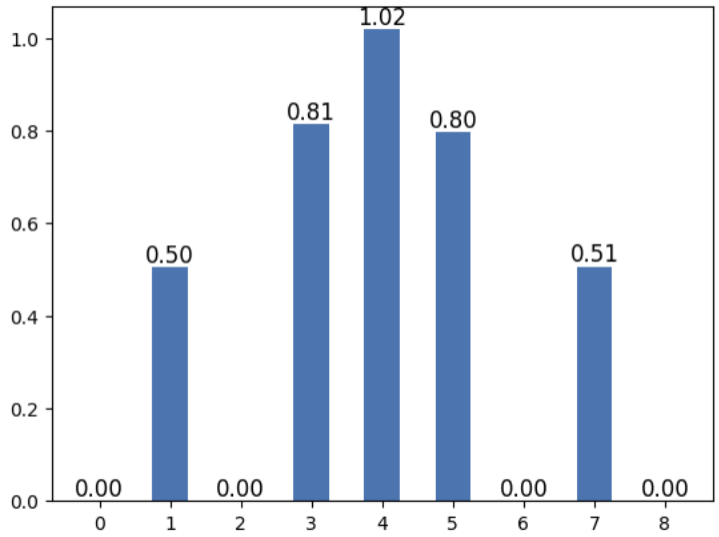


First, add dummy pixels 0 to make 4 X 4 Image because of dimension. We have to use 2 qubits for x axis, and 2 qubits for y axis. Additionally, we need "probability qubit register" that encodes each pixel entry. Methodology is as same as Enhanced Encoding Method in Task 3.





4 Hadamard gates + 16 CNOT gates are needed to make superposition state of position quantum register and probability quantum register.



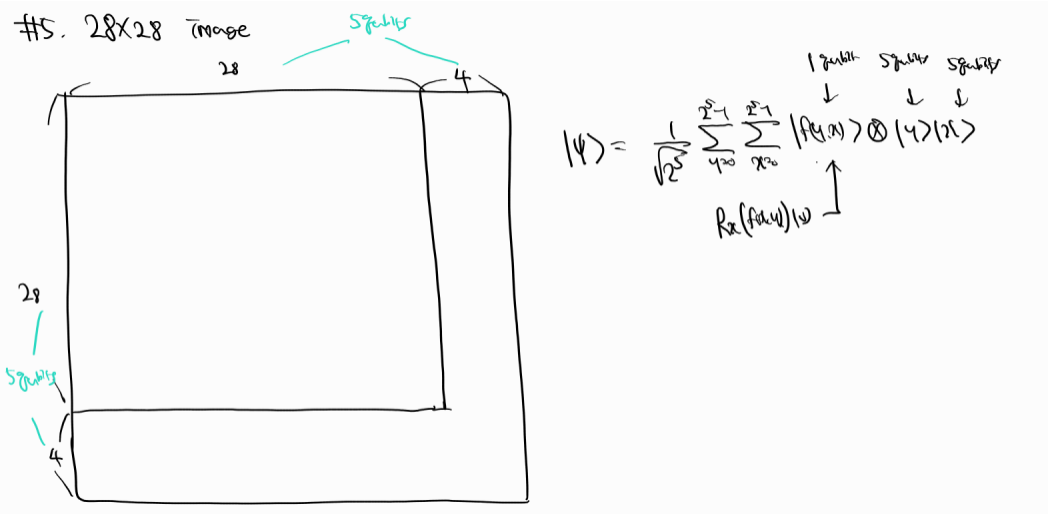
I could decode the encoded states by measurement and multiplying norm and some power of sqrt 2.

5. Encoding 28 X 28 Image

First, I added some dummy pixels to make 32 X 32 Image.

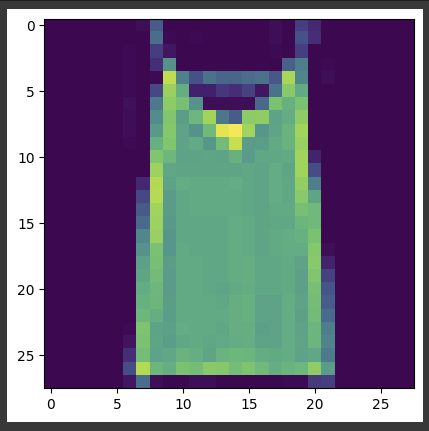
Second, I used 5 qubits for x axis, 5 qubits for y axis and 1 qubit for probability quantum register.

Main idea is same as Task 3.



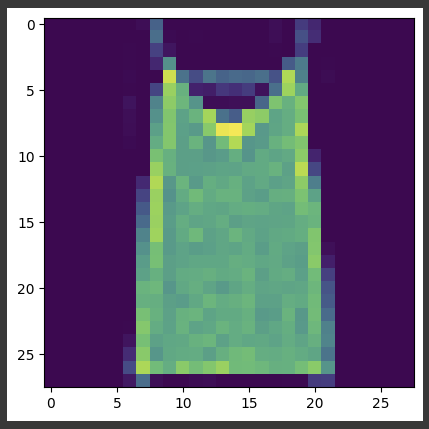
4) Original Image

```
[63] plt.imshow(test, interpolation='nearest')
     plt.show()
```



5) Recovered Image

```
[64] plt.imshow(Decoded_Matrix, interpolation='nearest')
     plt.show()
```

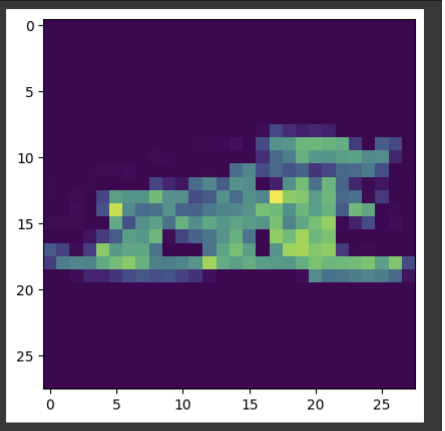


I encoded the first numpy array in npy file and decoded by measurement of 11 qubits.

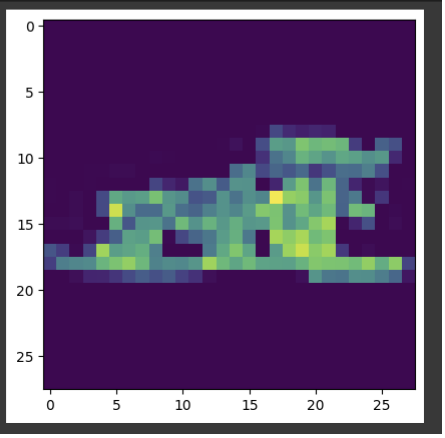
6. Using QPU Encoding 28 X 28 Image

I couldn't access IONQ Cloud, so I just implemented the first five numpy array in npy file, by IBMQ QASM Simulator. First image is introduced in Task 5, and below images are the other four images that I encoded and decoded by same methodology.

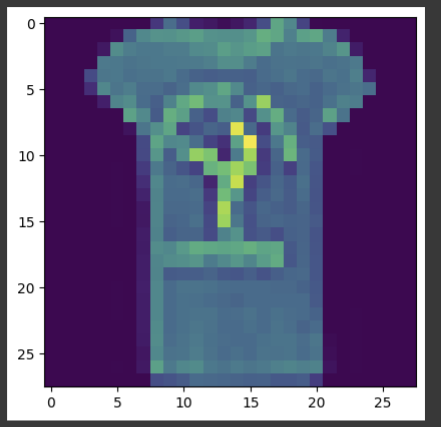
```
# Original Image
plt.imshow(test, interpolation='nearest')
plt.show()
```



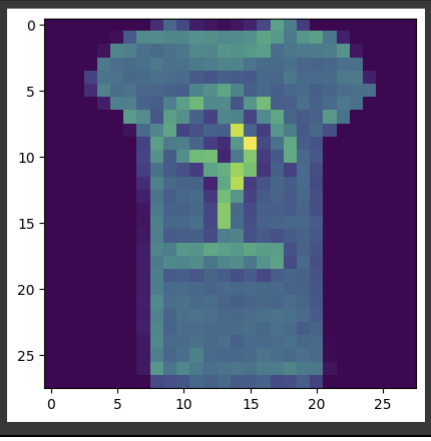

```
[74] # Recovered Image
plt.imshow(Decoded_Matrix, interpolation='nearest')
plt.show()
```

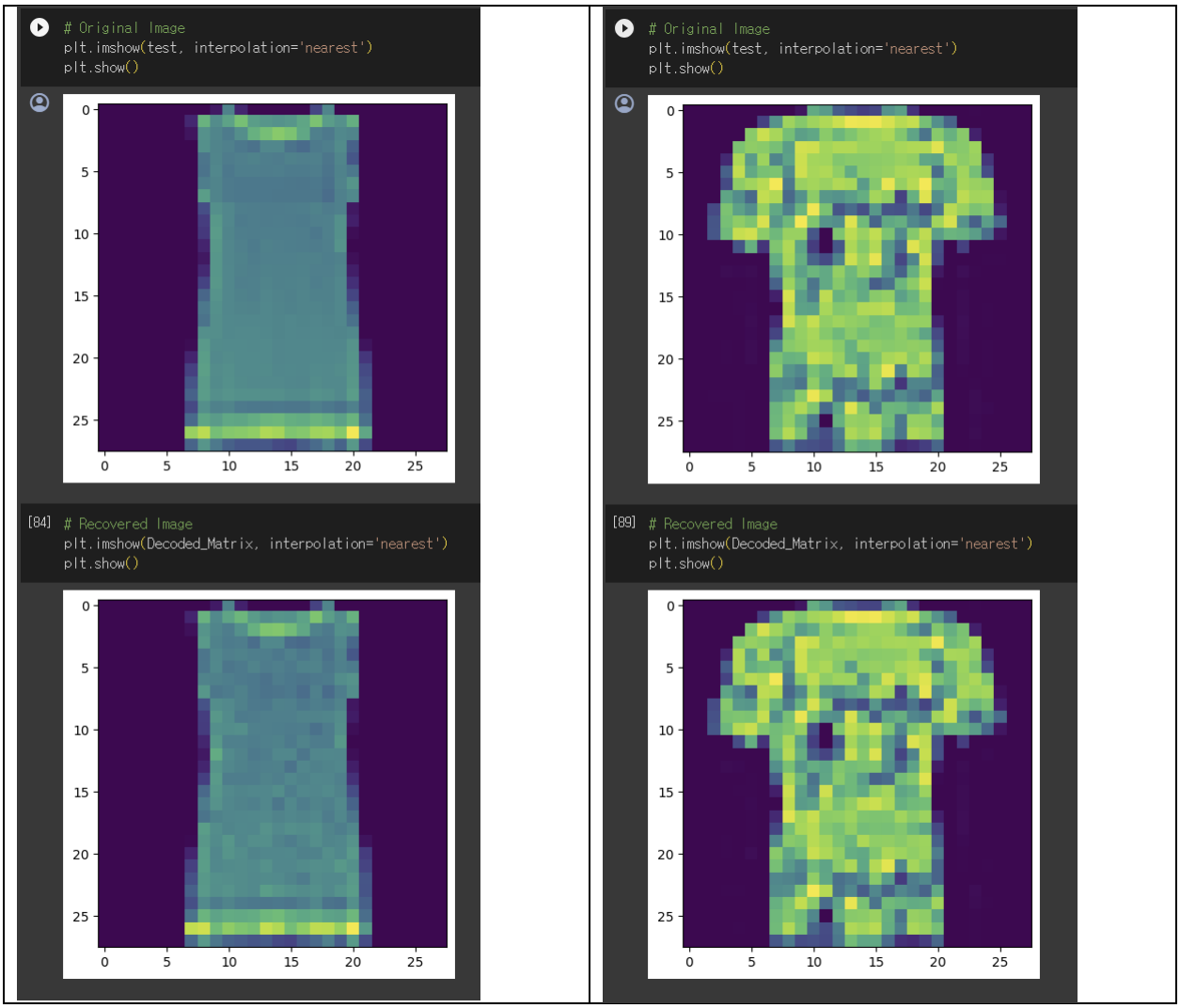



```
# Original Image
plt.imshow(test, interpolation='nearest')
plt.show()
```




```
[79] # Recovered Image
plt.imshow(Decoded_Matrix, interpolation='nearest')
plt.show()
```





7. MNIST Image Classification

I combined encoded circuit and parametrized circuit.

```
# Parametrized Quantum Circuit

# Universal Hadamard Gate
Eleven_Qubits_Encoding.h(y0)
Eleven_Qubits_Encoding.h(y1)
Eleven_Qubits_Encoding.h(y2)
Eleven_Qubits_Encoding.h(y3)
Eleven_Qubits_Encoding.h(y4)
Eleven_Qubits_Encoding.h(x0)
Eleven_Qubits_Encoding.h(x1)
Eleven_Qubits_Encoding.h(x2)
Eleven_Qubits_Encoding.h(x3)
Eleven_Qubits_Encoding.h(x4)

# Single Qubit Rotation & Two Qubits Entanglement Chain
Eleven_Qubits_Encoding.append(controlled_ry(entangle), [y0, y1])
Eleven_Qubits_Encoding.rx(single, y0)
Eleven_Qubits_Encoding.append(CXGate(), [y0, y1])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [y1, y2])
Eleven_Qubits_Encoding.rx(single, y1)
Eleven_Qubits_Encoding.append(CXGate(), [y1, y2])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [y2, y3])
Eleven_Qubits_Encoding.rx(single, y2)
Eleven_Qubits_Encoding.append(CXGate(), [y2, y3])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [y3, y4])
Eleven_Qubits_Encoding.rx(single, y3)
Eleven_Qubits_Encoding.append(CXGate(), [y3, y4])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [y4, s])
Eleven_Qubits_Encoding.rx(single, y4)
Eleven_Qubits_Encoding.append(CXGate(), [y4, s])

Eleven_Qubits_Encoding.append(controlled_ry(entangle_s), [s, y0])
Eleven_Qubits_Encoding.rx(single_s, s)
Eleven_Qubits_Encoding.append(CXGate(), [s, y0])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [x0, x1])
Eleven_Qubits_Encoding.rx(single, x0)
Eleven_Qubits_Encoding.append(CXGate(), [x0, x1])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [x1, x2])
Eleven_Qubits_Encoding.rx(single, x1)
Eleven_Qubits_Encoding.append(CXGate(), [x1, x2])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [x2, x3])
Eleven_Qubits_Encoding.rx(single, x2)
Eleven_Qubits_Encoding.append(CXGate(), [x2, x3])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [x3, x4])
Eleven_Qubits_Encoding.rx(single, x3)
Eleven_Qubits_Encoding.append(CXGate(), [x3, x4])

Eleven_Qubits_Encoding.append(controlled_ry(entangle), [x4, s])
Eleven_Qubits_Encoding.rx(single, x4)
Eleven_Qubits_Encoding.append(CXGate(), [x4, s])

Eleven_Qubits_Encoding.append(controlled_ry(entangle_s), [s, x0])
Eleven_Qubits_Encoding.rx(single_s, s)
Eleven_Qubits_Encoding.append(CXGate(), [s, x0])

Eleven_Qubits_Encoding.barrier()
```

First, Hadamard Gates make superposition state.

Second, Construct Entangled State between y axis and probability quantum register, and x axis and probability quantum register by CX Gate, Single Rx Gate and Controlled Ry Gate. My whole work were done by these 6 step.

1) Training Set / Validation Set Separation

2) Constructing Encoded Circuit List to decrease calculation

3) Constructing Parametrized Circuit

4) Cost Function : # of coincidence between label and

probability difference of 0 and 1 of 1 qubit

5) Training 1st Step : Investigating cost function of some parameters,

investigation is not concentrated on some specified parameters,

uniformly spreaded on bloch sphere.

6) Training 2nd Step : Investigate Cost Function near optimized parameter point of 1st step by investigating gradient of cost function.

However, because of lack of time, I could not finish training. Thank you for reading!