Emulation of Quantum BP Neural Network using Python

Kim Ki Song, Hwang UiSong, Hong SongHak *, Han HyonSok and Jang YongChol

Faculty of Information Science, ***Kim Il Sung*** *University, Taesong District, Pyongyang, DPR Korea*

Abstract

Recently, the Quantum Neural Network(QNN) is the newly appeared discipline by combining the quantum computing theory and neural network attracts attention. As a matter of fact, the quantum artificial intelligence is no more than the beginning, however the theoretical research and analysis have already been developed for the quantum associative storage, quantum state superposition and quantum parallel learning, etc, in the quantum computing ranges in the world, so the theoretical basis has been laid for development of the quantum neural computing.

In this paper, we described a simulation method of quantum BP neural network constructed with multiple Control-NOT(CNOT) gates in the "Jupyter lab" using python language. This QNN consist of the multiple CNOT gates and phase control gates, and is emulated with the sequence quantum steps in the emulator. In this work, we simulated this QNN using MNIST database, and have got the same results in accuracy as the classical neural network.

Keywords: quantum computing, Jupyter, python, BP neural network, multiple CNOT gate

1. Introduction

Nowadays the rapid development of artificial intelligence(AI) plays an important role in the social economy, and in recent years, the deep neural network technology has been developed by leaps and bounds.

However, in the computer hardware range, the quantum computer has become practical use and entered into the stage of application so the AI technology is developing[1-4, 10]. The literature [2, 3] introduced the quantum computer of D-Wave Company, and its capacity is incomparably superior to the traditional super computer. So in some counties, they are giving further study in development of quantum algorithms, and its education and research work are pushed forward actively. Eladio Gutiérrez introduced the quantum emulation method by using the NVIDIA GPU [11] and Matthew Rocklin provided the method using special quantum emulation library [6]. And the other researchers described some quantum circuit emulation methods by using the python language in the Jupyter plateform [5, 7-9]. In this paper, the immediacy and convenience of the description of python code in the Jupyter environment has been mentioned.

But in these methods, the real methods for emulation of neural network which is the basis of AI were not provided and the methods for emulation of quantum circuits in any program language has not mentioned.

So in this paper, we are going to provide the quantum neural network emulation method by using the sympy library of python in the Jupyter for the students and the beginners who are learning the quantum computing theory and AI.

* Corresponding author

Email addresses: *sh.hong0105@star-co.net.kp* (Song Hak Hong)

In section 2, we introduced the symbolic simulation library "sympy" made by python language. And section 3 described about the quantum circuit of BP neural network. In section 4, we described the simulation method of quantum BP neural network(QBPNN) in the Jupyter environment by using sympy. And section 5 mentioned its' result and conclusion in section 6 and acknowledgment in section 7.

2. The Sympy library of Python

All physical operations consist of symbolic and vector operations. In the view of the our quantum world, it contains the large range of the statistic and algebraic and topological fields. It is the open source library sympy of anaconda that processes these problems with symbolic description. There are many symbolic methods to describe the special problems and implement them. In fact, we introduce the statistic and algebraic operating symbols to solve many quantum computing problems, and it costs enormous expense to develop our projects in the case of using pure program codes without its specialized processing means. (for example: C++, Java ,etc.).

And also, from AI point of view, the "tensorflow" or "numpy" of "Anaconda" are the best ones and the training and implementation of deep neural network has become popular.

However, in the recent computing theory including the quantum computing, the web-based quantum computing online servers such as IBMQ are serving for the heavy computations, these are also some symbolic program language of Python and distributed as the libraries: sympy and qiskit.

So, we describe the library sympy as follows.

This library is the popular symbolic computing one by using the programming language Python. In sympy, the modern algebraic library such as "algebras", "tensor" and "diffgeom", and physical simulation library such as "physics". And the library physics contains the electromagnetic and quantum computational vector libraries.

The library sympy could be implemented in all Python environments. In this paper, we are going to implement the functions of the library by using Jupyter.

The Jupyter Applications, such as Jupyter Notebook, JupyterLab and JupyterHub, provide interactive computing environments across the wide range of programming languages for scientific computing, data science and analytics[8]. Jupyter Notebook and JupyterLab are the single user notebook web applications and both are based on a server-client structure [9, 10]. Jupyter Notebooks provide scientist with a single environment where not only their code, but also their equations, figures, explanatory text and even interactive visualizations can be stored. Moreover, the Jupyter Notebook and JupyterLab web applications provide an environment where scientists can construct the deep neural networks and train them and the libraries for physical simulation such as quantum calculation.

The figure 1 shows the sympy implementation in the Jupyter lab.

Like this, Jupyter and sympy have powerful ability to help many real computations by using Python language. Matthew Rocklin and Andy showed the methods implementing quantum gate circuits [6, 8, 9].

But the basic examples of AI technology using quantum computation program code are not provided yet enough. It gives limitation to the intention of the researchers and education for implementing AI technology by using quantum computation theory.

Thus, in this paper, we describe the simulation method of QBPNN that is the basis of quantum AI using the sympy library in the Python-Jupyter environment.

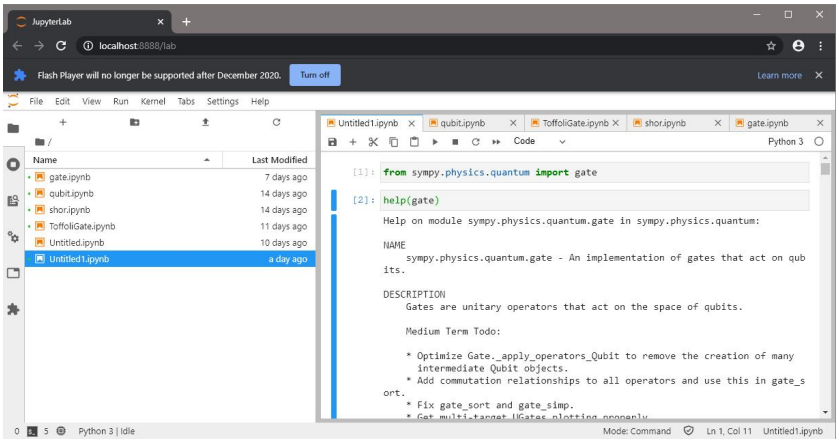

Fig 1: Implementation of sympy in Jupyter lab

3. Quantum Circuit of BP Neural Network

In this section, we introduce the QBPNN using Multiple Control-NOT(MCNOT) gates which is a type of multiple qubit logic gate.

CNOT gate has two input qubits. One is the control qubit and the other is the target qubit.

It's action is like as follows.

If the control qubit is $|0\rangle$, the target qubit maintains one's state. And the control qubit is $|1\rangle$, the target qubit should be reversed. It's expression is as follows.

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

Double-qubit CNOT gate could be extended to multiple-qubit gate. For example, if there are $n+1$ qubits and X is a single qubit CNOT gate, the MCNOT gate's operation $C^n(X)$ may be defined as follows.

$$C^n(|x_1x_2...x_n\rangle|\phi\rangle) \equiv |x_1x_2\cdots x_n\rangle X^{x_1x_2\cdots x_n}|\phi\rangle \quad (1)$$

Here the exponent $x_1x_2...x_n$ of X is the multiplication of qubits $x_1,\ x_2, \cdots, x_n$. So, if all n qubits are the $|1\rangle$, the NOT gate X reverses the last one qubit, and in the other case the last qubit should be kept.

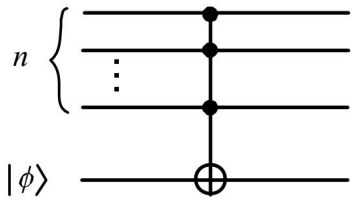

Fig 2: MCNOT gate

If the n control qubits are $|x_i\rangle = a_i|0\rangle + b_i|1\rangle\ (i = 1,2, ..., n)$ and the target qubit is $|\phi\rangle = |0\rangle$, the output of MCNOT gate is represented as follows.

$$C^n(|x_1\rangle\otimes\cdots\otimes|x_n\rangle\otimes|0\rangle) = |x_1\rangle\otimes\cdots\otimes|x_n\rangle\otimes|0\rangle - \\ - b_1b_2\cdots b_n\left|\overbrace{11\ldots10}^{n}\right\rangle + b_1b_2\cdots b_n\left|\overbrace{11\ldots11}^{n}\right\rangle \quad (2)$$

As we know, this output lays in entanglement of $n+1$ qubits and the probability of the target qubit $|\phi\rangle = |1\rangle$ is $|b_1b_2\cdots b_n|^2$.

Now let's see the quantum neural network using quantum gate circuit and update formula of network parameters.

The QNN model-based quantum gates is like the figure 3. Here $|x_1\rangle, \cdots, |x_n\rangle$ is input, $|h_1\rangle, \cdots, |h_p\rangle$ is the output of middle layer, $|y_1\rangle, \cdots, |y_m\rangle$ is the output of neural network.

Let's derivate the input-output relation of the QNN.

The training pattern $\bar{X} = (\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n)^T$ is represented through the real value vector of n-dimensional Hilbert space. Here $\bar{x}_i \in [a_i, b_i]$, and the transforming equation is as follows:

$$X = (|x_1\rangle, |x_2\rangle, \cdots, |x_n\rangle)^T \tag{3}$$

And use follow equation.

$$|x_i\rangle = \cos\frac{2\pi(\bar{x}_i - a_i)}{b_i - a_i}|0\rangle + \sin\frac{2\pi(\bar{x}_i - a_i)}{b_i - a_i}|1\rangle = \left(\cos\frac{2\pi(\bar{x}_i - a_i)}{b_i - a_i}, \sin\frac{2\pi(\bar{x}_i - a_i)}{b_i - a_i}\right)^T \tag{4}$$

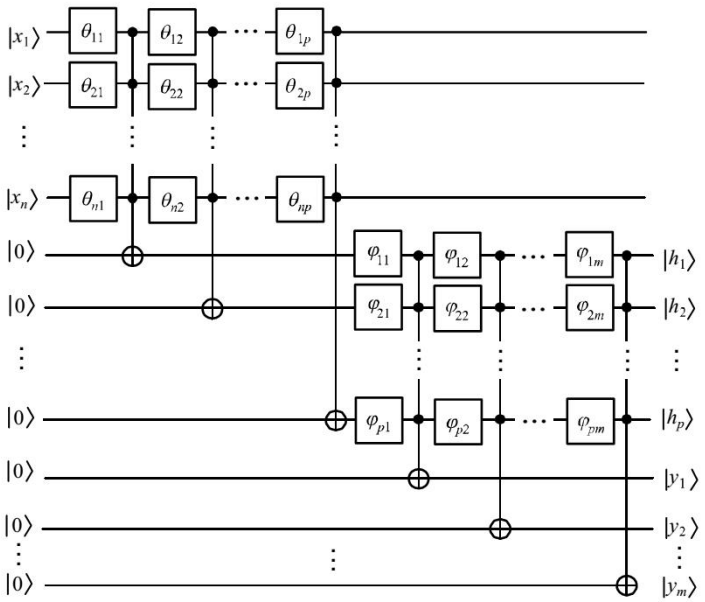

Fig 3: QBPNN model build with phase control gates and MCNOT gates

If $|x_i\rangle = cos\,\theta_i|0\rangle + sin\,\theta_i|1\rangle$, we can get the input-output equation from the single qubit phase rotation gate and MCNOT gate.

Middle layer output:

$$|h_j\rangle = \cos\left(\varphi_j + \sum_{k=1}^{m}\varphi_{jk}\right)|0\rangle + \sin\left(\varphi_j + \sum_{k=1}^{m}\varphi_{jk}\right)|1\rangle$$

Output layer output:

$$|y_k\rangle = \cos\xi_k|0\rangle + \sin\xi_k|1\rangle$$

Here

$$\varphi_j = \arcsin\left(\prod_{i=1}^{n} sin\left(\theta_i + \sum_{l=1}^{j}\theta_{il}\right)\right), \xi_k = \arcsin\left(\prod_{j=1}^{p} sin\left(\varphi_j + \sum_{q=1}^{k}\varphi_{jq}\right)\right)$$

But in the case of regarding output inducted from middle layer's output directly:

$$|h_j\rangle = \cos\varphi_j|0\rangle + \sin\varphi_j|1\rangle \ (j = 1,2,\cdots,p) \tag{5}$$

$$|y_k\rangle = \cos\xi_k|0\rangle + \sin\xi_k|1\rangle \ (k = 1,2,\cdots,m) \tag{6}$$

In each layers, if we set the probability amplitudes of the state $|1\rangle$ of qubits to real output values of the layer, the real outputs of each layers of neural network are as follows.

$$h_j = \sin\varphi_j = \prod_{i=1}^{n} \sin\left(\theta_j + \sum_{l=1}^{j} \theta_{il}\right) \quad (k = \overline{1,p}) \tag{7}$$

$$y_k = \sin\xi_k = \prod_{j=1}^{p} \sin\left(\arcsin\left(\prod_{i=1}^{n} sin\left(\theta_i + \sum_{l=1}^{j} \theta_{il}\right)\right) + \sum_{q=1}^{k} \varphi_{jq}\right) \quad (k = \overline{1,m}) \tag{8}$$

The control parameters of the proposed model are rotated phases of the single phase rotation gate for the middle and output layers. If the expected outputs after regularization are $\tilde{y}_1, \tilde{y}_2, \cdots, \tilde{y}_n$, then the error function is represented as follows.

$$E = \frac{1}{2}\sum_{k=1}^{m} (\tilde{y}_k - y_k)^2 \tag{9}$$

Using the gradient descent method, the gradient descent computing equation is represented as follow.

$$\frac{\partial E}{\partial \theta_{ij}} = -\sum_{k=1}^{m} \frac{(\tilde{y}_k - y_k) y_k \cot\left(\varphi_j + \sum_{q=1}^{k} \varphi_{jq}\right) h_j \cot\left(\theta_i + \sum_{l=1}^{j} \theta_{il}\right)}{\sqrt{1 - h_j^2}} \tag{10}$$

$$\frac{\partial E}{\partial \varphi_{jk}} = -(\tilde{y}_k - y_k) y_k \cot\left(\varphi_j + \sum_{q=1}^{k} \varphi_{jq}\right) \tag{11}$$

Thus, the updating equation about the angles of rotations of the middle and output layers is:

$$\theta_{ij}(t+1) = \theta_{ij}(t) - \eta \frac{\partial E}{\partial \theta_{jk}} \tag{12}$$

$$\varphi_{jk}(t+1) = \varphi_{jk}(t) - \eta \frac{\partial E}{\partial \varphi_{jk}} \tag{13}$$

Here t is the number of steps and η is the learning rate.

4. Emulation of QBPNN in JupyterLab using Sympy

BP neural network could be consisted of quantum circuit by several methods, here we made it using MCNOT gates and phase rotation gates. But in the library sympy, only the double-qubit CNOT gate. So we build a MCNOT gate from the double-qubit CNOT gate. In this paper, we implemented the triple-qubit CNOT gate known as toffoli gate by sympy, and made the BP neural network in the case of $2^3 = 8$ input data (N=8).

As the phase rotation gate, we use the T gate and S gate, and also use the CNOT gate.

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix}, \; CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

These gate circuits are defined in the library “sympy.physics.quantum.gate”.

The toffoli gate is defined as follow[12].

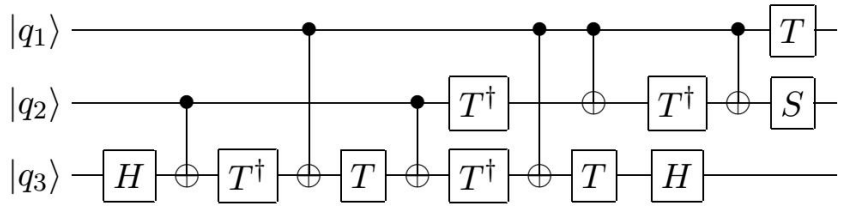

Fig 4: toffoli gate consists of CNOT, Hadamard, phase gates

Now, we define the BPNN by triple-qubit CNOT (toffoli) gate. The construction of quantum circuit is based on the equations of section 3. The Python code of the toffoli gate by using sympy is as follows.

```
#Python code 1 in sympy
    cnot1 = CNOT(1,0)
    cnot2 = CNOT(2,1)
    cnot3 = CNOT(2,0)
    a = 1 / sqrt(2)
    b = 1 / sqrt(2)
    psi1 = a * psi00 + b * psi01
    psi2 = qapply(HadamardGate(0) * psi1)
    psi3 = qapply(cnot1 * psi2)
    psi4 = qapply(DaggerTGate(0) * psi3)
    psi = qapply(cnot3 * psi4)
    psi = qapply(TGate(0) * psi)
    psi = qapply(cnot1 * psi)
    psi = qapply(DaggerTGate(0) * TGate(1) * psi)
    psi = qapply(cnot3 * psi)
    psi = qapply(TGate(0) * cnot2 * psi)
    psi = qapply(HadamardGate(0) * DaggerTGate(0) * psi)
    psi = qapply(cnot2 * psi)
    psi = qapply(SGate(1) * TGate(0) * psi)
```

This neural network is tested in the famous MNIST database.

The operations about the images are carried out by the use of tensorflow's in-out method, and quantized in sympy.

5. Results

In the result of simulation of QBPNN, the values of loss function to the epochs are resulted as follows. Figure 5 shows the similarity of QNN simulation and classical neural network simulation.

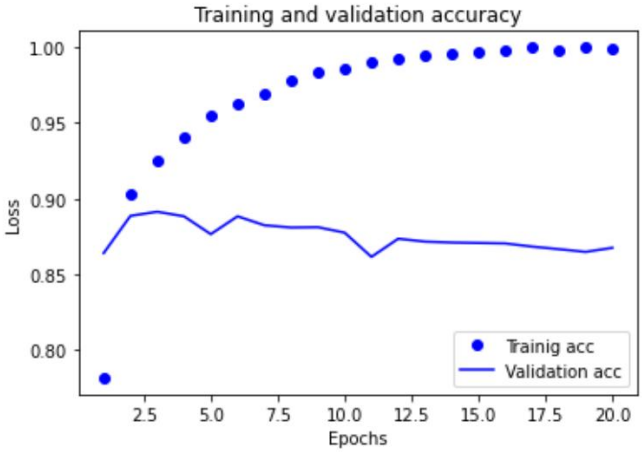

Fig 5: Accuracy curves of learning process

The description of toffoli gate and neural network using toffoli gates is as follow.

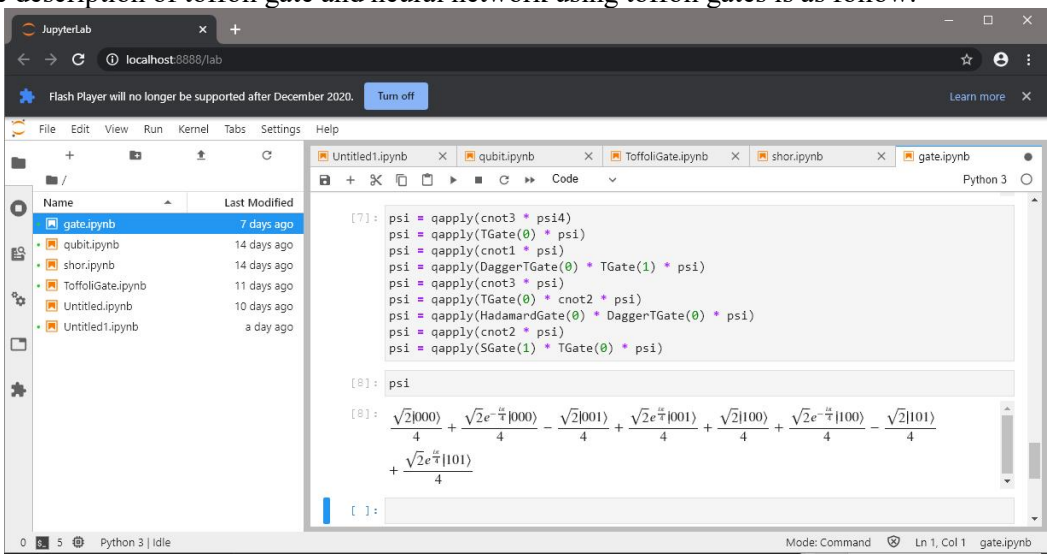

Fig 6: Description of toffoli gate and neural network in Jupyter Lab

We have created the quantum circuit of neural network using the Jupyter and sympy that is Python programming development environment and it's special library, and got the similar results with the classical neural network by using tensorflow.

But, we have not mentioned the calculation rate. Because the rate of QNN in the simulation environment can not exceed the classic personal computer's maximum operation rate as far as it is not real quantum computer.

Thus, this content is suitable for the education and design of methodology about the quantum computing.

6. Conclusion

In this paper, we developed the quantum circuit of BP neural network by using the MCNOT gates and phase rotation gates, and emulated it in the Jupyter of Python and library sympy.

Through the quantum simulation, we get the similar result with classical neural network algorithm, and made a methodology to implement the AI technology with quantum computing theory in quantum computer.

This fact will be used in education, simulation and experiments about quantum computing, and will accelerate the launching to the industrial environment of quantum computer.

Acknowledgment

We thank the following organizations for offering the wonderful basic projects and software:

- The Jupyter team, for providing a great software platform and easy to use resource, for deploying it and for direct support in debugging our issues
- Wiley-IEEE Computer Society Press Publication for providing a large number of useful data and experiments about sympy.
- The Docker team, for providing docker containerization.

References

[1] Ciaran Hughes, Joshua Isaacson, Anastasia Perry, Ranbel F. Sun and Jessica Turner. 2019. "Quantum Computing for the Quantum Curious."Springer. Available from: https://doi.org/10.1007/978-3-030-61601-4_1.

[2] D-Wave:The quantum Computing Company. 2015. "The D-Wave 2X™ Quantum Computer Technology Overview." D-Wave Systems Inc. Available from: https://www.dwavesys.com/.

[3] George Rajna. 2017. "IBM Quantum Computer Test.". Available from: http://phys.org/news/2016-05-ibm-quantum-Computer-Test.html.

[4] Srinivasan Arunachalam, Ronald de Wolf. 2017. "A Survey of Quantum Learning Theory." *arXiv Preprint:1701.06806v3 [quant-ph] 28 Jul 2017*

[5] Raimar Sandner, András Vukics. 2014. "A C++/Python application-programming framework for simulating open quantum dynamics." *Computer Physics Communications 185 (2014) 2380–2382*. Available from: http://www.elsevier.com/locate/cpc

[6] Matthew Rocklin and Andy R. Terrel. 2014. "Symbolic Statistics with SymPy." A Wiley-IEEE Computer Society Press Publication. Available From: http://www.computer.org

[7] Gediminas Kiršanskas, Jonas Nyvold Pedersen. 2017 "An open-source Python package for calculations of transport through quantum dot devices." Available from: http://dx.doi.org/10.1016/j.cpc.2017.07.024.

[8] Project Jupyter. "JupyterLab." (Accessed 10 July 2022). Available from: http://jupyterlab.readthedocs.io/ en/latest/.

[9] Project Jupyter. "The Jupyter Notebook." (Accessed 10 July 2022). Available from: https://jupyter-notebook.readthedocs.io/en/ latest/.

[10] Jialin Chen, Lingli Wang. "A quantum-implementable neural network model". Quantum Inf Process (2017) 16:245. Aailable From: http://doi.org/10.1007/s11128-017-1692-x

[11] Eladio Gutiérrez, Sergio Romero. 2010. "Quantum computer simulation using the CUDA programming model". Computer Physics Communications 181 (2010) 283–300. Available From: http:// doi:10.1016/ j.cpc.2009.09.021

[12] Mark D. Hill. 2011."Quantum Computing for Computer Architects", Second Edition. University of Wisconsin.