Parametric Synthesis of Quantum Circuits for Training Perceptron Neural Networks

A. V. Ostroukh, C. B. Pronin, A. V. Volosova, B. S. Subbotin
Moscow Automobile and Road Construction State Technical University (MADI)
Moscow, Russia
ostroukh@mail.ru; caesarpr12@gmail.com; sas32sa@yandex.ru; nich@nich.madi.ru

P. I. Smirnov
Department of Automobiles and Automotive Industry
Vologda State University
Vologda, Russia
smirnov@vogu35.ru

Abstract— This work contains the analysis of results received after running synthesized quantum circuits for training perceptron neural networks. The training is performed by creating a Grover's algorithm with a custom oracle function. The concept of synthesizing quantum circuits was showcased in the process of generating training circuits for three perceptron topologies, which were designed to test the accuracy of the synthesis process. The test circuits serve to prove that the proposed synthesis approach could be scaled to utilize more complex quantum computing systems and to solve more practical tasks. IBM’s 100-qubit cloud quantum simulator was used as the debugging environment. Quantum circuits for described algorithms are generated by the “Naginata” quantum synthesizer, its source code is published and further documented on GitHub along with the code for the provided example algorithms. The article describes the processes behind the algorithm for synthesizing quantum circuits that perform the training process of single-layer perceptrons by finding their weights by filtering all possible input values through a predefined accuracy criterion. Since quantum computing is still in its early development phase, quantum circuits are created mainly by manual placement of logic elements. Implementing quantum algorithms, especially more use-case specific ones, directly on the quantum circuit level could lead to the circuit easily becoming too complex for human comprehension. Quantum Circuit Synthesizer "Naginata" was created to simplify the development and debugging process of quantum algorithms, by adding better clarity to their development process. In our case, better clarity for the development process is achieved by composing functions for commonly used operations performed in the implemented quantum algorithm. The programmer could now implement the quantum algorithm as a set of functions, instead of manually creating a circuit from single logic elements. After this, the synthesizer would handle the task of creating the data for placing logic elements on the circuit. This enables an opportunity of implementing quantum algorithms with higher-level commands. In the scope of this work, parametrically generated generic blocks for frequently used operations such as: the adder, multiplier and digital comparator were created and utilized to form the training circuits. The test results, proved that with the help of the proposed quantum synthesizer, these compositions could be used efficiently as building blocks for implementing quantum algorithms. And by visually comparing sizes of both code and circuit representations of the synthesized circuits, to the code examples used to synthesize these circuits, it is determined that the proposed approach for implementing quantum circuits greatly simplifies the processes of development and debugging a quantum algorithm.

Keywords— quantum synthesizer; quantum circuit synthesis; quantum machine learning; quantum algorithms; neural networks; quantum computing; Grover’s algorithm

I. INTRODUCTION

The goal of project “Naginata - Quantum Circuit Synthesizer” [1] is to create a prototype system for synthesizing complex quantum circuits. One of the applications of this system will be in parametric synthesis of quantum circuits for finding the weights of a neural network (perceptron) of a given topology. Quantum circuits synthesized by the program are exported to a *.qasm text file in the form of OPENQASM 2.0 code, compatible with IBM Quantum - a cloud quantum computing environment [2].

This paper provides results and descriptions of running synthesized circuits for training perceptrons of three different topologies, which were set as parameters in the file "qnn_generation_tests.py". The circuits were tested on a 100-qubit IBM “simulator mps” quantum simulator in the IBM Quantum cloud environment.

The circuit for finding the weights of a perceptron is synthesized based on Grover’s algorithm [3], for which a custom oracle is created based on given parameters of the perceptron’s topology and input neuron values.

II. MODELS AND METHODS

Algorithm of synthesizing quantum circuits for training a single layer perceptron

The topology of the required neural network is inputted as two lists of dictionaries (python). The first list describes connections between input and hidden neurons. The second list describes connections between hidden and output neurons.

The “build_param_network” function creates all necessary quantum registers to store the desired weights, and also independently combines them into “grover_regs” - a list of registers that store the “body” of the Grover algorithm (Fig. 1)
Then, registers are created and filled to store the values of input neurons and the target value $A_c$. Next, registers are created to store multiplication results, from them, some get selected to hold the total sum of these results for each hidden and for each output neuron.

The first step in constructing the circuit is applying the Hadamard gates to the weight registers $w_n$ (the first step in Grover's algorithm).

The next step, is the construction of the oracle function based on the topology of the neural network:

1) The oracle function contains calculations (Mul - operation of multiplication, Sum - the operation of addition) that are performed during the forward pass of the neural network - calculating the values of hidden and then output neurons (1). With a linear activation function of hidden neurons $H_{\text{input}} = H_{\text{output}} = H_s$:

$$
H_1 = I_1 w_1 + I_2 w_2 \\
H_2 = I_3 w_3 + I_4 w_4 \\
O_1 = H_1 w_3 + H_2 w_6
$$

2) The values of the output neurons are compared with the threshold value $A_c$, which is a condition for selecting suitable values $w_n$.

3) The registers that store the results of comparison operations are rotated by a NCZ gate. N-Controlled-Z Gate (NCZ) – a Pauli-Z gate, with multiple (N) control qubits.

4) Reversed versions of the calculation operations from steps 1 and 2 are performed

The last step is to perform the amplitude amplification function (the third step in Grover's algorithm) on $w_n$ registers, and to measure the values of these registers. Thus, the "body" of Grover's algorithm is placed in registers $w_n$.

For a given neural network (Fig. 2), the condition for selecting suitable values $w_n$ is formed, as inequality (2) [4-6]:

$$(I_1 w_1 + I_2 w_2)^* w_1 + (I_3 w_3 + I_4 w_4)^* w_6 \geq A_c$$

or

$$O_1 \geq A_c$$

(2)

To simplify the analysis of the algorithm's results, we implement only condition (4):

$$(I_1 w_1 + I_2 w_2)^* w_1 = A_c$$

(3)

Generating a quantum circuit for training perceptron example No. 1

Let’s observe an example of finding weights of a perceptron with the topology shown on Fig. 3. The goal of training this neural network is to find the coefficients $w_i$ that satisfy inequality (3).

$$(I_1 w_1 + I_2 w_2)^* w_1 \geq A_c$$

(3)

To simplify the analysis of the algorithm’s results, we implement only condition (4):

$$(I_1 w_1 + I_2 w_2)^* w_1 = A_c$$

(4)
Fig. 4. The spread of measured values of the algorithm for training perceptron No. 1 after 8192 iterations (shots = 8192)

At the input of the algorithm, the values are: $I_1 = 11, I_2 = 3, A_c = 0001102 = 6_{10}$.

At the output of the algorithm, the following values were obtained as solutions: 010010, 011100, 100001, 110100. They correspond to peaks in the diagram on Fig. 4. Based on the order in which the registers are defined for this circuit in file “qnn_generation_tests.py” and the bit numbering order in the IBM Quantum environment, the obtained values should be divided into equal 2-bit registers corresponding to desired values of $w_1, w_2, w_3$, in the manner indicated in Table I.

| measured | $w_1$ | $w_2$ | $w_3$ |
|----------|-------|-------|-------|
| 010010   | 01    | 00    | 10    |
| 011100   | 01    | 11    | 00    |
| 100001   | 10    | 00    | 01    |
| 110100   | 11    | 01    | 00    |

TABLE I. ORDER OF SPLITTING THE MEASURED BIT STRINGS INTO W VALUES, FOR PERCEPTRON #1

Generating a quantum circuit for training perceptron example No. 2

The goal of training the neural network on Fig. 5 is to find the coefficients $w_i$ that satisfy inequality (5).

$$I_1w_1w_3 + I_2w_2w_4 \geq A_c$$  \hspace{1cm} (5)

To simplify the analysis of the algorithm’s results, we implement only condition (6).

$$I_1w_1w_3 + I_1w_2w_4 = A_c$$  \hspace{1cm} (6)

Fig. 5. Topology of example perceptron No. 2

Fig. 6. The spread of measured values of the algorithm for training perceptron No. 2 after 8192 iterations (shots = 8192)

At the input of the algorithm, the values are: $I_1 = 11, I_2 = 3, A_c = 0001102 = 6_{10}$.

At the output of the algorithm, the values obtained after measurement are given in Table II. They correspond to peaks in the diagram on Fig. 6. Based on the order in which the registers are defined for this circuit in file “qnn_generation_tests.py” and the bit numbering order in the IBM Quantum environment, the obtained values should be divided into equal 2-bit registers corresponding to desired values of $w_1, w_2, w_3$ and $w_4$, in the manner indicated in Table II.

| measured | $w_4$ | $w_3$ | $w_2$ | $w_1$ |
|----------|-------|-------|-------|-------|
| 00010010 | 00    | 01    | 00    | 10    |
| 00101100 | 00    | 01    | 10    | 01    |
| 00010100 | 00    | 10    | 00    | 01    |
| 00101001 | 00    | 10    | 10    | 01    |
| 00101101 | 00    | 10    | 11    | 01    |
| 01001000 | 01    | 01    | 00    | 10    |
| 01001010 | 01    | 01    | 10    | 01    |
| 01001101 | 01    | 01    | 11    | 01    |
| 01010001 | 01    | 01    | 01    | 01    |
| 01010010 | 01    | 01    | 01    | 01    |
| 01010100 | 01    | 01    | 10    | 01    |
| 01010101 | 01    | 01    | 10    | 01    |
| 01100001 | 01    | 01    | 10    | 00    |
| 01100100 | 01    | 10    | 00    | 01    |
| 01110000 | 01    | 10    | 00    | 01    |
| 10000100 | 10    | 00    | 00    | 01    |
| 10000101 | 10    | 00    | 01    | 01    |
| 10000110 | 10    | 00    | 01    | 10    |
| 10000111 | 10    | 00    | 01    | 11    |
| 10010010 | 10    | 01    | 00    | 10    |
| 10010100 | 10    | 01    | 01    | 00    |
| 10100001 | 10    | 10    | 00    | 01    |
| 10100100 | 10    | 10    | 00    | 01    |
| 10101000 | 10    | 11    | 00    | 00    |
| 11000010 | 11    | 01    | 00    | 10    |
| 11000011 | 11    | 01    | 00    | 01    |

Generating a quantum circuit for training perceptron example No. 3

The goal of training the neural network on Fig. 7 is to find the coefficients $w_i$ that satisfy inequality (7).

$$I_1w_1w_3 + I_2w_2w_4 \geq A_c$$  \hspace{1cm} (7)
To simplify the analysis of the algorithm’s results, we implement only condition (8).

\[ I_1 w_3 + I_2 w_4 = Ac. \] (8)  

Fig. 7. Topology of example perceptron No. 3

Fig. 8. The spread of measured values of the algorithm for training perceptron No. 3 after 8192 iterations (shots = 8192)

At the input of the algorithm, the values are: \( I_1 = 11_2 = 3_{10}; I_2 = 10_2 = 2_{10}; Ac = 01011_2 = 22_{10}. \)

At the output of the algorithm, the values obtained after measurement are given in Table III. They correspond to peaks in the diagram on Fig. 8. Based on the order in which the registers are defined for this circuit in file “qnn_generation_tests.py” and the bit numbering order in the IBM Quantum environment, the obtained values should be divided into equal 2-bit registers corresponding to desired values of \( w_1, w_2, w_3, \) and \( w_4, \) in the manner indicated in Table III.

### TABLE III. ORDER OF SPLITTING THE MEASURED BIT STRINGS INTO W VALUES, FOR PERCEPTRON №3

| measured     | \( w_1 \) | \( w_2 \) | \( w_3 \) | \( w_4 \) |
|--------------|-----------|-----------|-----------|-----------|
| 01101011     | 01        | 10        | 10        | 11        |
| 01111100     | 01        | 11        | 10        | 10        |
| 01001011     | 10        | 10        | 01        | 11        |
| 10110101     | 10        | 11        | 01        | 10        |

IV. CONCLUSIONS

Parametrically generated compositions for frequently used operations such as: the adder, multiplier and digital comparator were created and utilized to form perceptron training circuits. The test results, proved that with the help of the proposed quantum synthesizer, these compositions could be used efficiently as building blocks for implementing quantum algorithms. And by visually comparing sizes of both code and circuit representations of the synthesized circuits, to the code examples used to synthesize these circuits, it is determined that the proposed approach to implementing quantum circuits greatly simplifies the processes of development and debugging a quantum algorithm.

Circuits generated for 3 example perceptrons were run on “simulator_mps” - a 100-qubit IBM quantum simulator, which is available in IBM’s Quantum cloud environment. The results of running synthesized circuits for training perceptrons of three different topologies were provided, described and checked for correctness. The code for generating these circuits is provided in file “qnn_generation_tests.py” of the “Naginata” Quantum Circuit Synthesizer repository.

The development of systems for synthesizing quantum algorithms could become one of the main directions for development and further implementation of quantum computing in various spheres of economy, because they help to simplify the process of creating and debugging complex quantum circuits [7 – 10].

The use of computational advantages of quantum computers in machine learning could significantly optimize artificial intelligence models and improve the accuracy of their training, which in turn will make these models more reliable and versatile.

REFERENCES

[1] C. Pronin, Naginata-QCS. 2022. Accessed: Sep. 20, 2022. [Online]. Available: https://github.com/MexIvanov/Naginata-QCS
[2] IBM Quantum Computing, Oct. 01, 2015. URL: https://www.ibm.com/www.ibm.com/quantum (accessed Sep. 18, 2022).
[3] L. K. Grover, “A fast quantum mechanical algorithm for database search,” Proceedings of the Annual ACM Symposium on Theory of Computing, vol. Part F129452, pp. 212-219, 1996, doi: 10.1145/237814.237866.
[4] C. B. Pronin, O. I. Maksimychev, A. V. Ostroukh, A. V. Volosova, and E. N. Matukhina, “Creating Quantum Circuits for Training Perceptron Neural Networks on the Principles of Grover’s Algorithm”, 2022 Systems of Signals Generating and Processing in the Field of on Board Communications, SSGI 2022 - Conference Proceedings, pp. 1-5, 2022, doi: 10.1109/IEEECONF53456.2022.9744279.
[5] N. Benchasatrabue, P. Chongstitvatana, and C. Apontewan, “Quantum Rough Counting and Its Application to Grover's Search Algorithm,” 2018 3rd International Conference on Computer and Communication Systems (ICCCS), pp. 21-24, 2018, doi: 10.1109/COMMS.2018.8463331.
[6] R. Wang, “Comparing Grover’s Quantum Search Algorithm with Classical Algorithm on Solving Satisfiability Problem,” 2021 IEEE Integrated STEM Education Conference (ISEC), 2021, pp. 204-204, doi: 10.1109/ISEC52395.2021.9764017.
[7] S. Adarsh, and M. Möller, “Resource Optimal Executable Quantum Circuit Generation Using Approximate Computing,” 2021 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 225-231, 2021, doi: 10.1109/QCE52317.2021.00040.
[8] Y. Kondo, R. Moria, and R. Movassagh, “Quantum supremacy and hardness of estimating output probabilities of quantum circuits,” 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 1296-1307, 2022, doi: 10.1109/FOCSS2979.2021.00126.
[9] N. Thumwanit, C. Lortanprasert, H. Yano, and R. Raymond, “Trainable Discrete Feature Embeddings for Quantum Machine Learning,” 2021 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 479-480, 2021, doi: 10.1109/QCE52317.2021.00087.
[10] T. G. d. Brugières, M. Baboulin, B. Valiron, S. Martel, and C. Allouche, “Reducing the Depth of Linear Reversible Quantum Circuits,” IEEE Transactions on Quantum Engineering, vol. 2, pp. 1-22, 2021, art no. 3102422, 2021, doi: 10.1109/TQE.2021.3091648.