Method Article

Single qubit neural quantum circuit for solving Exclusive-OR

I.V. Grossu
University of Bucharest, Faculty of Physics, Bucharest-Magurele, P.O. Box MG 11, 077125, Romania

Abstract

The main goal of this work was designing a single qubit neural quantum circuit for performing Exclusive-OR, a concrete example of an operation requiring multiple layers in classical neural networks. The corresponding Qiskit code was tested on both simulators and IBM's “ibmqx2” five-qubit quantum processor. Further analyses along training the proposed neural quantum circuit are currently in progress.

A single qubit neural quantum circuit for performing XOR was tested on IBM Q Experience
XOR is a concrete example of an operation requiring multiple layers in classical neural networks

Article history: Received 21 April 2021; Accepted 31 October 2021; Available online 6 November 2021

Introduction

A perceptron [1,2] is a mathematical model inspired by neural cells. It accepts $n$ input values “connected” to a single output. Each input channel is characterized by a weight parameter. The output
is calculated using the activation function:

\[
y = \begin{cases} 
1, & \text{if } \sum_{i=1}^{n} x_i w_i > t, \\
0, & \text{else}
\end{cases}
\]  

(1)

where \( y \) is the output, \( x \) the input vector, \( w \) the trainable weight vector, and \( t \) a threshold trainable parameter.

Although of high interest in both neurosciences and artificial intelligence, the model was proven to be limited to linearly separable functions [2]. Thus, for solving the Exclusive-OR (XOR) function a multi-layer perceptron should be considered instead (the connectionist paradigm).

In the field of Quantum Neural Networks there are various attempts [3–7] at implementing quantum perceptrons. One important direction is based on the qubit neuron concept ("quron"). Most models are exploiting the nonlinearity of the measurement process for implementing the threshold function. Another direction is based on mimicking the classical Rosenblatt perceptron. As opposed to the classical case, quantum perceptrons are not limited to linearly separable problems.

Hybrid quantum-classical neural networks are described in [8]. The approach is based on implementing hidden layers with parameterized quantum circuits (circuits which rotation angles for each gate are specified by the components of a classical input vector). The measurement statistics of each quantum layer is collected and used as input for the following layer. One could calculate the gradient as the difference between the circuit evaluated at \( \theta + s \) and \( \theta - s \), where \( \theta \) represents the circuit parameters, and \( s \) is a macroscopic shift. Thus, the circuit could be trained using gradient descent methods, such as backpropagation. The detailed method, including PyTorch code examples, is also discussed in Ref. [8].

The ability of single quantum neuron to compute linearly non-separable functions is directly connected with the basic property of quantum mechanics that a wave function is invariant under application of global phase [3,6]. For the same reason and, as opposed to classical neurons, AND and OR operations are non-trivial for single quantum neurons [7]. Thus, in Siomau’s model [3] is discussed the possibility of computing XOR with a two-qubits perceptron. In Ref. [9], it was noticed that XOR can also be performed by a particular single qubit parametrized circuit.

**Circuit implementation**

A single qubit neural circuit is presented in [8]. It contains a Hadamard followed by a rotation \( RY(\theta) \) gate, where \( \theta \) is a trainable parameter. The output in z-basis is given by the expectation:

\[
\sigma = \sum_{i} z_i p(z_i)
\]  

(2)

where \( p(z_i) \) represents the probability of measuring the value \( z_i \).

Starting from this example, a particular parametrized quantum circuit with two rotation gates was considered (Fig. 1): \( RZ(\theta_1 \times x_1 + \alpha) \), and \( RX(\theta_2 \times x_2 + \alpha) \), where \( (x_1, x_2) \) is the input vector, \( (\theta_1, \theta_2) \) are the “dendrites” trainable weights, and \( \alpha \) is also a trainable parameter. The Hadamard gate is
used for initializing the qubit to the $|+\rangle$ diagonal state, while the angles of the two rotation gates are controlled by the $(x_1, x_2)$ input vector. The output is obtained by collecting the measurement statistics for calculating the expectation $\sigma$ (Eq. (2)). The corresponding Qiskit code is also presented in Appendix A.

The circuit could be trained using gradient descent methods as it was designed following the parametrized quantum circuits principles described in [8]. However, for the XOR problem, one solution could be immediately observed on the Bloch Sphere [10]: $\theta_1 = \theta_2 = \pi$, for $\alpha = -\pi/2$. This case is illustrated, for each input vector, in (Fig. 2).

**Testing the circuit**

The circuit was implemented on IBM Quantum Composer [11] which provides a user-friendly graphical development environment, and which allows running the generated OpenQASM code on several quantum computers (an example of code is available in Appendix B). For a better understanding, both the “ibmq_qasm_simulator” simulator and the “ibmqx2” five-qubit quantum
Table 1
The XOR circuit test results.

| x\_1 | x\_2 | 32q ibmq\_qasm\_simulator | 5q ibmqx2 |
|------|------|---------------------------|-----------|
| 0.0  | 0.0  | 0.0000                    | 0.0585    |
| 0.0  | 1.0  | 1.0000                    | 0.8805    |
| 1.0  | 0.0  | 1.0000                    | 0.8875    |
| 1.0  | 1.0  | 0.0000                    | 0.0585    |

Table 2
The XOR circuit test results for input data affected by errors.

| X\_1 | X\_2 | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|------|------|-----|-----|-----|-----|-----|-----|
| 0.0  | 0.0  | 0.00| 0.11| 0.35| 0.65| 0.91| 1.00|
| 0.2  | 0.11| 0.18| 0.38| 0.61| 0.83| 0.91| 1.00|
| 0.4  | 0.35| 0.38| 0.45| 0.55| 0.61| 0.65| 1.00|
| 0.6  | 0.65| 0.61| 0.55| 0.44| 0.38| 0.34| 1.00|
| 0.8  | 0.91| 0.83| 0.61| 0.38| 0.14| 0.09| 1.00|
| 1.0  | 1.00| 0.91| 0.65| 0.34| 0.09| 0.00| 1.00|

The processor were chosen. The results are presented in Table 1. The differences between the simulator and quantum computer outputs are due to quantum errors [8]. For evaluating the circuit behavior on input data affected by errors, the attached Qiskit code (see Appendix A) was employed on IBM Quantum Lab. One can notice that, within a reasonable error, the circuit succeeded also to classify noisy data (Table 2).

Conclusion

A particular single qubit parametrized quantum circuit for performing XOR was successfully tested on both simulators and the “ibmqx2” five-qubit quantum processor. Further analyses along training the proposed neural quantum circuit are currently in progress.

Direct submission or co-submission

Co-submissions are papers that have been submitted alongside an original research paper accepted for publication by another Elsevier journal.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: The author declares no conflict of interest.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.mex.2021.101573.

Appendix

A. The Qiskit code used for testing the XOR circuit on IBM Quantum Lab:

```python
# importing standard Qiskit libraries and configuring account
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
```
from qiskit.visualization import *
from numpy import pi
qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

# the input vector is manually changed on each run
x1 = 0.0
x2 = 1.0

# hardcoded trainable parameters
theta1 = pi
theta2 = pi
alpha = -pi/2

# parametrized quantum circuit
circuit.reset(qreg_q[0])
circuit.h(qreg_q[0])
circuit.rz(x1 * theta1 + alpha, qreg_q[0])
circuit.rx(x2 * theta2 + alpha, qreg_q[0])
circuit.measure(qreg_q[0], creg_c[0])

# the simulator to be used
sim = Aer.get_backend('aer_simulator')
# turns the circuit into an object the backend can run
qobj = assemble(circuit, shots=1024)
# run the experiment and get the result from that experiment
result = sim.run(qobj).result()
# get a dictionary containing the output
counts = result.get_counts()
# display output on a histogram
plot_histogram(counts)

B. An example of OpenQASM 2.0 code generated with IBM Quantum Composer:

OPENQASM 2.0;
include "qelib1.inc";
qreg q[1];
creg c[1];
reset q[0];
h q[0];
rz(-0.5*pi) q[0];
rx(-0.5*pi) q[0];
measure q[0] --> c[0];

References
[1] D. Dumitrescu, H. Costin, in: Retele Neuronale Teorie si Aplicatii, Teora, Romania, 1996, pp. 192–227.
[2] M. Minsky, S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, Cambridge, 1969.
[3] M. Siomau, Quantum Inf. Process. 13, Issue 5, 1211-1221 (2014). https://link.springer.com/article/10.1007%2Fs11128-013-0723-5
[4] M. Schuld, I. Sinayskiy, F. Petruccione, The quest for a quantum neural network, Quantum Inf. Process 13 (2014) 2567–2586 https://link.springer.com/article/10.1007/s11128-014-0809-8.
[5] M. Schuld, I. Sinayskiy, F. Petruccione, Simulating a perceptron on a quantum computer, Phys. Lett. A 379 (7) (2015) 660–663 20 MarchPages, doi:10.1016/j.physleta.2014.11.061.
[6] F. Tacchino, C. Macchiavello, D. Gerace, D Bajoni, An artificial neuron implemented on an actual quantum processor, NPJ Quantum Inf. 526 (2019) https://www.nature.com/articles/s41534-019-0480-4.pdf?origin=ppub.
[7] S. Mangini, et al., Mach. Learn. Sci. Technol. 1 (2020) 045008 https://iopscience.iop.org/article/10.1088/2632-2153/aba98.
[8] A. Asfaw, L. Bello, Y. Ben-Haim et al., Learn quantum computation using Qiskit, 2020, http://community.qiskit.org/textbook
[9] I.V. Grossu, Research gate, Preprint, August 2020, DOI: 10.13140/RG.2.2.23971.07205
[10] R Portugal, Quantum Walks and Search Algorithms, Quantum Science and Technology, Springer, New York, 2013 https://link.springer.com/book/10.1007/978-1-4614-6336-8.
[11] IBM Quantum Composer, https://quantum-computing.ibm.com/composer