Autonomous Quantum Perceptron Neural Network

Alaa Sagheer and Mohammed Zidan
Department of Mathematics
Center for Artificial Intelligence and RObotsics (CAIRO)
Faculty of Science, Aswan University, Aswan, Egypt
Email: alaa@cairo-aswu.edu.eg

Abstract: Recently, with the rapid development of technology, there are a lot of applications require to achieve low-cost learning. However the computational power of classical artificial neural networks, they are not capable to provide low-cost learning. In contrast, quantum neural networks may be representing a good computational alternate to classical neural network approaches, based on the computational power of quantum bit (qubit) over the classical bit. In this paper we present a new computational approach to the quantum perceptron neural network can achieve learning in low-cost computation. The proposed approach has only one neuron can construct self-adaptive activation operators capable to accomplish the learning process in a limited number of iterations and, thereby, reduce the overall computational cost. The proposed approach is capable to construct its own set of activation operators to be applied widely in both quantum and classical applications to overcome the linearity limitation of classical perceptron. The computational power of the proposed approach is illustrated via solving variety of problems where promising and comparable results are given.

1 Introduction

Classical Artificial Neural Networks (CANN) derives its computing power through its massively parallel-distributed structure and the ability to learn and, therefore, generalize. However, CANN may face many difficulties such as the absence of concrete algorithms and rules for specifying optimal design architectures, limited memory capacity, time-consuming training, etc. One of the known classical approaches is the classical Perceptron Neural Network (CPNN), which is applied only for linearly separable learning problems. In other words, CPNN cannot be applied for problems which have inseparable classes, such as XOR problem. These limitations, and others, have been motivated many researchers to investigate new trends in neural computation domain. One of the novel trends in this domain is to evoke properties and techniques of quantum computing into classical neural computation approaches.

Several researchers expect that quantum computing is capable to enhance the performance, and overcoming the above limitations, of classical neural computation. The beginning was in 1995 with Kak who was the first researcher introduced the concept of quantum neural computation. Then,
Menneer [10] defines a class of quantum neural network (QNN) as a superposition of single component networks where each is trained using only one pattern. Ventura et al. in [6] introduced a new associative memory technique based on Grover’s quantum search algorithm can solve the completion problem. The technique restores the full pattern when a part of it is initially presented with just a part of the pattern. Also, an exponential increase in the capacity of the memory is performed when it compared with the CANN capacity.

As long as classical perceptron is concerned, some researchers have been tried to increase the efficiency of perceptron using the power of quantum computation. In 2001, Altaisky [11] developed a simple quantum perceptron that depends on selecting the activation operator. However its simplicity, Altaisky approach consumed much time in order to select an activation operator, especially, when the size of training data is large. Next, Fei et al. [12] introduced a new model of quantum neuron and its learning algorithm based on Altaisky perceptron. Fei model used the delta rule as the learning rule which yields considerable results such as computing XOR-Function using only one neuron and nonlinear mapping property. Unfortunately, Fei model did not provide us a new way for deriving the activation operator. Nevertheless, Fei model is sensitive for the selection of the appropriate activation operator, which was the problem of Altaisky perceptron.

Recently, Zhou et al. [13] developed a quantum perceptron approach based on the quantum phase adequately and could to compute the XOR function using only one neuron. The drawback of Zhou perceptron is that it requires many computation iterations to give a response. Finally, Siomau [14] introduced an autonomous quantum perceptron based on calculating a set of positive valued operators and valued measurements (POVM). However, Siomau perceptron cannot be applied for problems such as quantum-Not gate and Hadamard gate.

In this paper, we propose a novel autonomous quantum perceptron neural network (AQPNN) approach can be used to solve both classical applications and quantum applications. The proposed AQPNN improves the computational cost of Altaisky quantum perceptron as well as the computational cost of Zhou quantum perceptron and its ability to learn the problems that Siomau quantum perceptron can not learn. The proposed perceptron is capable to adapt its activation operator very fast which reduces the overall learning time. In addition, it is capable to overcome the linearity restriction of classical perceptron where AQPNN can be viewed as a non-linear perceptron. To evaluate AQPNN, we solve various problems and the results are compared favorably with Zhou [13].

The paper is organized as follows: Section 2 describes the operation of AQPNN and its learning algorithm. Section 3 shows the computational power of AQPNN via solving various problems. Section 4 discusses the performance of AQPNN. Section 5 shows the conclusion and our future work.
2 Autonomous Quantum Perceptron Neural Network (AQPNN)

2.1 Description of the AQPNN

The proposed (AQPNN) approach is a quantum neural network approach includes only one neuron with $n$ qubit inputs $|x_1\rangle$, $|x_2\rangle$, ..., $|x_n\rangle$, (for qubit definition, see Appendix A). A set of weight operators $w_1, w_2, ..., w_n$ is assumed such that one weight operator is associated with each input, and $|y_{net}\rangle$ is the final network response; see Figure 1. The operators $F_j$ refers to a set of unique activation operators of the proposed perceptron.

\[ |y_j\rangle = \sum_{i=1}^{n} w_i |x_i\rangle = \begin{bmatrix} \alpha_j \\ \beta_j \end{bmatrix}. \]

(1)

where $\alpha_j$ and $\beta_j$ are the probability amplitudes of the weighted sum qubit for $j^{th}$ pattern in the training set. The weight operators is updated at time $t$ using the following rule:

\[ w_i(t + 1) = w_i(t) + \gamma |e\rangle\langle x_i| . \]

(2)

where $\gamma$ is the learning rate, $|e\rangle = (|d\rangle - |y\rangle)$ is the perceptron error and $|e\rangle\langle x_i|$ denotes the outer product of vectors $|e\rangle$ and $\langle x_i|$. Once the weighted sum is calculated for all the available patterns, then the set of activation operators can be calculated using the form:

\[ F_j = \begin{bmatrix} \cos \theta_j & -\sin \theta_j \\ \sin \phi_j & \cos \phi_j \end{bmatrix} . \]

(3)

where $j = 1, 2, 3, ..., m$, and $m$ is the set of unique activation operators (repeated activation operators are discarded) where $m \leq N$ is the number of training data set. The parameters $\theta_j$ and $\phi_j$ are two real valued angles calculated using the form:

\[ \begin{bmatrix} \cos \theta_j \\ \sin \phi_j \end{bmatrix} = \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} \alpha_{d_i} \\ \beta_{d_i} \end{bmatrix} . \]

(4)
where \((\alpha_j, \beta_j)^T\) is the weighted sum qubit and \((\alpha_d, \beta_d)^T\) is the target qubit. The aim of each activation operator is to transform the weighted sum qubit to be mapped into the given target and make it a normalized qubit Eq.(A2). After calculating the set of all activation operators, the output of the autonomous quantum perceptron is given using the superposition of all activation operators in the following form:

\[
\ket{y_{\text{output}}} = \sum_{j=1}^{m} F_j \sum_{i=1}^{n} \ket{x_i}.
\]

(5)

where \(\ket{y_{\text{output}}}\) is the network output as a superposition of the set of output qubits. This output represents the effect of the activation operators (interference) on the weighted sum qubit resulted when any pattern presented to the network. One qubit only from these qubits will be the response of the network and can be specified by the following form:

\[
\ket{y_{\text{net}}} = L(|\langle y_{\text{output}}| \circ \ket{y_{\text{output}}} - C)|) \circ \ket{D}.
\]

(6)

where \(C = [1, 1, \ldots, 1]^T\) is a good vector and \(D\) is the vector of the target qubits. The operation \(\circ\) achieves the Hadamard product operation [15] (for more details, see Appendix A), where the function \(L\) retain the smallest absolute value and makes it equal one and the rest of values to be equal zeros. Hence, the result of Eq.(6) is only one qubit represents the net response of the AQPNN for the current input.

2.2 The learning algorithm of AQPNN

According to the description given above, the AQPNN learning algorithm is divided into two main stages: The First stage is imbedded in both Eq.(3) and Eq.(5), where the AQPNN algorithm collects information about the problem in hand by constructing a set of activation operators. In the second stage the AQPNN takes the decision about the networks response according to Eq.(6) based on the gathered information. In the following, we can summarize the AQPNN learning algorithm in the following steps:

Step 1: Set all \(F_i = I\) (identity matrix). Then choose the initial weight operators \(w_i\) randomly, set the learning rate \(0 < \gamma < 1\) and set iteration number \(k = 1\),

Step 2: Calculate the weighted sum qubit for each given pattern using Eq.(1),

Step 3: Compare each weighted sum qubit for the patterns of each class with all other weighted sum qubits for other classes. We have two cases here:

1. If each weighted sum qubit for any class does not equal the same value for any weighted sum in any other classes then go to step 5, else, go to step 4.

2. If the value of any weighted sum qubit is zero then go to step 4, else, go to step 5.
Step 4: Update the weight operators using Eq.(2), set \( k = k + 1 \), go to step 2.

Step 5: Calculate the activation operator for each weighted sum qubit using Eq.(4).

The superposition of output qubits of the network is given by Eq.(5) whereas the net response of the AQPNN is given by Eq.(6).

3 The Computational Power of AQPNN

We proceed now to evaluate practically the computational power of the proposed AQPNN algorithm. In this section, we show the results of using AQPNN in solving four different problems. In the first two problems, we solve the problems of quantum Not-gate and the Hadamard-gate. In the third problem we compute the XOR-function, whereas in the fourth problem we achieve a classification task application.

3.1 The Quantum Not-gate

Class A: The first pattern is \( P_1 = \{ |x_1\rangle = [1 \ 0], |d_1\rangle = [0 \ 1] \} \).

Class B: The second pattern is \( P_2 = \{ |x_2\rangle = [0 \ 1], |d_2\rangle = [1 \ 0] \} \).

The initial weight operators is chosen arbitrary as \( w = [1 \ 0 \ 0 \ 1] \). Once, we introduced the two patterns to the AQPNN network we obtain the weighted sum, according to Eq.(1), as follows:

\[
|y_1\rangle = [1 \ 0], \quad |y_2\rangle = [0 \ 1]
\]

Since \( |y_1\rangle \) has a different value than \( |y_2\rangle \), then we can calculate the set of activation operators as follows:

\[
\begin{bmatrix}
\cos \theta_1 & -\sin \theta_1 \\
\sin \phi_1 & \cos \phi_1
\end{bmatrix}
\begin{bmatrix}
1 \\
0
\end{bmatrix}
= \begin{bmatrix}
0 \\
1
\end{bmatrix}
\]

i.e. \( \theta_1 = -90 \), \( \phi_1 = 90 \) and, thereby, \( F_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \).

Similarly, we can get \( F_2 \) as follows:

\[
\begin{bmatrix}
\cos \theta_2 & -\sin \theta_2 \\
\sin \phi_2 & \cos \phi_2
\end{bmatrix}
\begin{bmatrix}
0 \\
1
\end{bmatrix}
= \begin{bmatrix}
1 \\
0
\end{bmatrix}
\]
\[ \theta_2 = -90, \quad \phi_2 = 90 \] and, thereby, \[ F_2 = F_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \]

Therefore, the superposition output is \[ |y_{output} \rangle = F \sum_{i=1}^{n} w_i |x_i \rangle. \] This means that, the quantum Not gate is trained after only one iteration.

### 3.2 The Hadamard-gate

**Class A:** The first pattern is \[ P_1 = \{ |x_1 \rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |d_1 \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \}. \]

**Class B:** The second pattern is \[ P_2 = \{ |x_2 \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix}, |d_2 \rangle = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \}. \]

The initial weight operators is chosen arbitrary as \[ w = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \]

Once, we introduced the two patterns to the AQPNN network we obtain the weighted sum, according to Eq. (1), as follows:

\[ |y_1 \rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |y_2 \rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \]

By the same way, as \[ |y_1 \rangle \] has a different value than \[ |y_2 \rangle \] then we can calculate the set of activation operators, we can get \[ F_1 \] as follows:

\[ \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 \\ \sin \phi_1 & \cos \phi_1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]

i.e. \( \theta_1 = -45, \quad \phi_1 = 135 \) and, thereby, \[ F_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \]

Similarly, we can get \[ F_2 \] as follows:

\[ \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 \\ \sin \theta_2 & \cos \theta_2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \]

i.e. \( \theta_2 = -45, \quad \phi_2 = 135 \) and, thereby, \[ F_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \]

Therefore, the superposition output is \[ |y_{output} \rangle = F \sum_{i=1}^{n} w_i |x_i \rangle. \] This means that, the Hadamard-gate is trained after only one iteration.

### 3.3 The XOR-Function

It is known that, there are two classes in XOR-function:

**Class A:** The first pattern is \[ P_1 = \{ |x_1 \rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |x_2 \rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |d_1 \rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \}. \]
The second pattern is \( P_2 = \{ |x_1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, |x_2\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} |d_2\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \} \)

**Class B:** The third pattern is \( P_3 = \{ |x_1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |x_2\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} |d_3\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \} \)

The fourth pattern is \( P_4 = \{ |x_1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, |x_2\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} |d_4\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \} \)

Assume a random initial weight operator takes the value \( w_1 = w_2 = \begin{bmatrix} 1.1 \\ 1.2 \\ 0 \\ 0 \end{bmatrix} \).

If we introduced the four patterns into the AQPNN network we obtain, after only one iteration, the weighted sum of each pattern as follows:

\[
|y_1\rangle = \begin{bmatrix} 2.2 \\ 0 \end{bmatrix}, |y_2\rangle = \begin{bmatrix} 2.4 \\ 0 \end{bmatrix}, |y_3\rangle = \begin{bmatrix} 2.3 \\ 0 \end{bmatrix}, |y_4\rangle = \begin{bmatrix} 2.3 \\ 0 \end{bmatrix}
\]

It is easy to observe that the first couple of weighted sum qubits has different values than the other couple of weighted sum qubits (i.e. either of \(|y_1\rangle\) or \(|y_2\rangle\)) has a different value than \(|y_3\rangle\) and \(|y_4\rangle\). Also, we may observe that \(|y_3\rangle = |y_4\rangle\).

As these two patterns are in the same class B, this implies that there only three activation operators will take the following forms:

\[
F_1 = \begin{bmatrix} 0.4545 \\ 0 \\ -0.8907 \\ 1 \end{bmatrix}, F_2 = \begin{bmatrix} 0.4167 \\ 0 \\ -0.9091 \\ 1 \end{bmatrix}, F_3 = F_4 = \begin{bmatrix} 0 \\ 0.4348 \\ -1 \\ 0.9005 \end{bmatrix}
\]

Then, the superposition output can be calculated as:

\[
|y_{\text{output}}\rangle = \sum_{j=1}^{m=3} F_j \sum_{i=1}^{n=2} w_i |x_i\rangle
\]

Table 1 shows a comparison between the proposed perceptron AQPNN, Zhouh perceptron [13] and the classical perceptron, the classical perceptron is not applicable in case of using one neuron. The proposed perceptron gives the final output after only one iteration whereas Zhouh perceptron gives the final result after 16 iterations [13]. Then it is clear that AQPNN reduces the computation steps to get the final results.

| Algorithm name | AQPNN | Zhouh Perceptron | Classical Perceptron(One neuron) |
|----------------|-------|------------------|----------------------------------|
| No.of iterations | 1     | 16               | Not applicable by one neuron     |

### 3.4 Two-overlapped classification problem

We proceed now to use the proposed AQPNN approach in a classification application. The application we use here is atypical two-overlapped classes classification problem, which can be regarded as a complex generalization of the...
XOR problem [16]; see Figure 2. It has two classes: the first is a oval-shape class has the target $|0\rangle = \left[ \begin{array}{c} 1 \\ 0 \end{array} \right]$ with arbitrary input patterns given in Table 2. The second class is square-shape class which has the target $|1\rangle = \left[ \begin{array}{c} 0 \\ 1 \end{array} \right]$ with arbitrary input patterns given in Table 3.

![Figure 2: Two overlapped classes classification problem](image)

| Table 2: Training input patterns of the oval-shape class |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| P1  | P2  | P3  | P4  | P5  | P6  | P7  |
| (0.1,0) | (0.1,0.2) | (0.0,0.1) | (-0.1,0.2) | (-0.1,0) | (0,-0.1) | (0.1,-0.2) |

| Table 3: Training input patterns of the square-shape class |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| P9  | P10 | P11 | P12 | P13 | P14 | P15 |
| (0.1,0.1) | (0,0) | (0,0.2) | (-0.1,0.1) | (0.1,-0.1) | (-0.1,-0.1) | (0,-0.2) |

It’s clear that the values of input patterns are classical data, i.e. real values, so it must be transformed into qubits using qubit normalization equation Eq. (A2). For example, for the pattern $P1= (0.1,0)$, where $a=0.1$ and, then, $b = \sqrt{1-(0.1)^2}$. Thus, we may have $P1 = \{|x_1\rangle = \left[ \begin{array}{c} 0.1 \\ 0.9950 \end{array} \right], |d_1\rangle = \left[ \begin{array}{c} 1 \\ 0 \end{array} \right] \}$. In this experiment, we chose only 15 patterns as training data whereas the testing data is generalized over 176 patterns. If the learning rate is chosen, randomly, to be 0.1, we will find the classification rate approaches 97.73% after only one iteration for the learning process.
4 Discussion

It is worth now to discuss the performance of the proposed algorithm. It is clear from the above examples that the computational power of the proposed AQPNN is high, however, many observations may be one will record. First observation is that under equal weight operators, the AQPNN model, in some applications, does not utilize all the training data like other perceptron algorithms [2, 13, 14]. For example, in the first two situations, i.e. Not-gate and Hadamard-gate, we need only one training input (because we use only the unique activation operator), whereas in the XOR function situation, it is required three training inputs in order to accomplish the learning process. In the three situations, the AQPNN is capable to reduce both the computation time and the number of activation operators.

Second observation is concerned with the relation between the initial weight operators and the activation operators. In the situations of the quantum Not-gate and the quantum Hadamard-gate, the initial weight operator was the unitary operator, whereas in case of the XOR-function it was not the unitary operator. The reason for this is due to the nature of the unitary operator \( U \) where \( UU^\dagger = I \). Then, using Eq. (3), that includes the formula of the activation operators, we have:

\[
\begin{bmatrix}
\cos \theta_j - \sin \theta_j \\
\sin \phi_j & \cos \phi_j
\end{bmatrix}
\begin{bmatrix}
\cos \theta_j & \sin \phi_j \\
-\sin \theta_j & \cos \phi_j
\end{bmatrix}
= 
\begin{bmatrix}
1 & \sin (\theta_j - \phi_j) \\
\sin (\theta_j - \phi_j) & 1
\end{bmatrix}
\]

Obviously, the value of activation operator depends on the values of \( \theta_j \) and \( \phi_j \), which in turn depend on the initial weight operators and the training data.

5 Conclusion and Future Work

This paper presented a novel algorithm achieves autonomous quantum perceptron neural network (AQPNN) to enable real time computations. The developed algorithm represents a good computational alternate to the classical perceptron neural network approach. AQPNN constructs self-adaptive activation operators capable to accomplish the learning process in limited number of iterations and reduces the overall computational cost. These activation operators can be applied in both quantum and classical applications. The efficiency of the proposed algorithm is evaluated via solving four different problems where promising and comparable results are given. In addition, to train the AQPNN algorithm, it uses a limited number of training data samples, and in testing, it accomplishes a well generalization. Using the proposed perceptron in real world applications is one of our future aims.
References

[1] M. Hagan, H. Demuth and M. Beale (1996), *Neural Network Design*, PWS publishing Company (USA).

[2] F. Rosenblatt (1957), *The Perceptron - a perceiving and recognizing automaton*, Tech. report 85-460-1, Aeronautical Lab., Cornell Univ.

[3] F. Shafee (2007), *Neural networks with quantum gated nodes*, Engineering Applications of Artificial Intelligence , 20,4, pp. 429-437.

[4] R. Zhou (2010), *Quantum Competitive Neural Network*, International Journal of Theoretical Physics, 49, pp. 110-119.

[5] A. Sagheer and N. Metwally (2010), *Communication via Quantum Neural Networks*, the 2nd world congress on nature and biologically inspired computing, NaBIC, IEEE, pp. 418-422.

[6] D. Ventura and T. Martinez (2000), *Quantum Associative Memory*, Information Sciences, 5124, pp. 273-296.

[7] C. Li and S. Li (2008), *Learning algorithm and application of quantum BP neural networks based on universal quantum gates*, 19, 1, pp. 167-174.

[8] M. Nielsen and I. Chuang (2000.), *Quantum Computation and Quantum Information*, Cambridge University Press (Cambridge).

[9] S. C. Kak (1995), *Quantum Neural Computing*, Advances in Imaging and Electron Physics, 94, pp. 259-314.

[10] T. Menneer (1998), *Quantum Artificial Neural Networks*, Ph. D. thesis of The Univ. of Exeter, UK.

[11] M. V. Altaisky (2001), *Quantum neural network*, quant-ph/0107012.

[12] L. Fei and Z. Baoyu (2008), *A study of quantum neural networks*, Neural Networks and Signal Processing, IEEE, 1, pp. 539-542.

[13] R. Zhou, L. Qin and N. Jiang (2006), *Quantum Perceptron Network*, The 16th International Conference on Artificial Neural Networks, ICANN, LNCS, 4131, pp. 651-657.

[14] M. Siomau (2013), *A Quantum Model for Autonomous Learning Automata*, quant-ph/1210.6626 v3.

[15] R. Horn and C. Johnson (1999), *Topics in matrix analysis*, Cambridge University Press (Cambridge).

[16] H. Xiao and M. Cao (2009), *Hybrid Quantum Neural Networks Model Algorithm and Simulation*, The 5th International Conference on Natural Computation, 1, pp. 164-168.
Appendix A

**Qubit:** The smallest element store information in quantum computer is called quantum-bit (qubit). The qubit takes either value of 0 or 1 or a superposition of these states in the form:

\[ |\psi\rangle = a|0\rangle + b|1\rangle. \quad (A1) \]

Where \(a, b\) are complex numbers called the probability amplitudes. The qubit state \(|\psi\rangle\) is collapse into either basis state \(|0\rangle\) or \(|1\rangle\) with probability \(|a|^2\) or \(|b|^2\) respectively where

\[ |a|^2 + |b|^2 = 1. \quad (A2) \]

**Hadamard product (matrices)** In mathematics, the Hadamard product is a binary operation that takes two matrices of the same dimensions, and produces another matrix where each element \(i,j\) is the product of elements \(i,j\) of the original two matrices. For two matrices, \(A, B\) of the same dimension, \(m \times n\) the Hadamard product, \(A \circ B\), is a matrix, of the same dimension as the operands, with elements given by

\[ (A \circ B)_{i,j} = (A)_{i,j}(B)_{i,j}. \quad (A3) \]

**Example:** Suppose two matrices

\[ A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \text{and} \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \]

then the hadamard product is

\[ A \circ B = \begin{bmatrix} 1(5) & 2(6) \\ 3(7) & 4(8) \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix} \]