Received: 8 March 2024 | Revised: 27 April 2024 | Accepted: 14 May 2024 | Published online: 23 May 2024

Journal of Computational and Cognitive Engineering
2024, Vol. 3(4) 404–411
DOI: 10.47852/bonviewJCCE42022789

RESEARCH ARTICLE

Deterministic Versus Nondeterministic Optimization Algorithms for the Restricted Boltzmann Machine

Gengsheng L. Zeng[1,*]

[1]*Department of Computer Science, Utah Valley University, USA*

Abstract: A restricted Boltzmann machine is a fully connected shallow neural network. It can be used to solve many challenging optimization problems. The Boltzmann machines are usually considered probability models. Probability models normally use nondeterministic algorithms to solve their parameters. The Hopfield network which is also known as the Ising model is a special case of a Boltzmann machine, in the sense that the hidden layer is the same as the visible layer. The weights and biases from the visible layer to the hidden layer are the same as the weights and biases from the hidden layer to the visible layer. When the Hopfield network is considered a probabilistic model, everything is treated as stochastic (i.e., random) and nondeterministic. An optimization problem in the Hopfield network is considered searching for the samples that have higher probabilities according to a probability density function. This paper proposes a method to consider the Hopfield network as a deterministic model, in which nothing is random, and no stochastic distribution is used. An optimization problem associated with the Hopfield network thus has a deterministic objective function (also known as loss function or cost function) that is the energy function itself. The purpose of the objective function is to assist the Hopfield network to reach a state that has a lower energy. This study suggests that deterministic optimization algorithms can be used for the associated optimization problems. The deterministic algorithm has the same mathematical form for the calculation of a perceptron that consists of a dot product, a bias, and a nonlinear activation function. This paper uses some examples of searching for stable states to demonstrate that the deterministic optimization method may have a faster convergence rate and smaller errors.

Keywords: restricted Boltzmann machine, Hopfield network, deterministic optimization, nondeterministic optimization

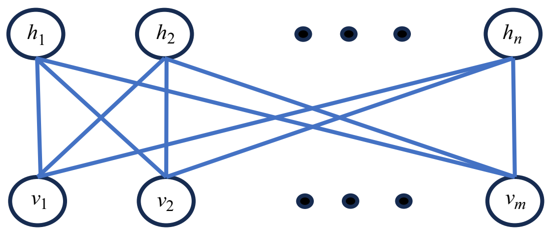
1. Introduction

A restricted Boltzmann machine (RBM) is a fully connected neural network consisting of two layers: the visible layer and hidden layer, respectively, as shown in Figure 1 [1–8]. The neurons in the visible layer are labeled as $v_1, v_2, \dots, v_m$, and the neurons in the hidden layer are labeled as $h_1, h_2, \dots, h_n$. All neurons are fully connected between the two layers. However, the neurons are not connected within each layer, which is an additional requirement dictated by the word "restricted." In unrestricted Boltzmann machines, on the other hand, we allow connections between the neuron connections in the hidden layer [9–11].

A common RBM is binary valued. In other words, $v_1, v_2, \dots, v_m, h_1, h_2, \dots, h_n \in \{0, 1\}$ or $v_1, v_2, \dots, v_m,\ h_1, h_2, \dots, h_n \in \{-1, 1\}$. An RBM is associated with an energy function, which is a quadratic form. To use an RBM, a problem is formulated as an energy function (also known as, an objective function, a cost function, or a loss function). This energy function is formed according to the task to be performed [12–17].

A unique feature of the RBMs is their parallel implementations by, for example, using the Field Programmable Gate Array (FPGA) technology [18–23]. Therefore, they can be used to solve the NP-hard problems once the NP-hard problems are represented by well-defined energy functions. Many NP-hard problems can be formulated in a combinatorial nature.

Figure 1
A restricted Boltzmann machine (RBM) consists of two layers: the visible layer *v* and the hidden layer *h*



In most machine learning tasks, input data and desired output data (known as targets or labels) are given to train the neural network parameters (known as weights and biases), according to a user-specified cost function, which defines the distance between the desired target and the model produced result. However, in a typical RBM application, we do not use the input data and the desired output data to train the network parameters. The network parameters are calculated through a design procedure. If there is no training, there is no need for an objective function.

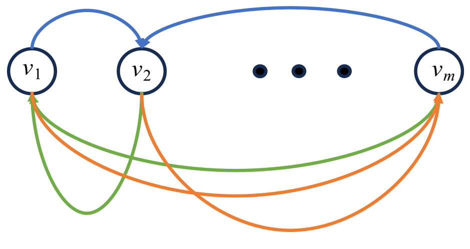
*__Corresponding author:__ Gengsheng L. Zeng, Department of Computer Science, Utah Valley University, USA. Email: larry.zeng@uvu.edu

The energy function in the context of Boltzmann machines is inherent to the network weights and biases; the weights connect the visible layer and the hidden layer. The energy function can be used as an objective function. However, this objective function is not quite the same as the objective function that is used commonly in machine learning. The common objective function in machine learning is used for network training to find the weights and biases. On the other hand, the energy function of a Boltzmann machine indicates the stability of the state of the machine. In a binary Boltzmann machine, the neurons store a $\{-1,\ 1\}$ string, which is referred to as a state. Each state is associated with a value of the energy function. If the Boltzmann machine is initialized with a random state, the tendency of the Boltzmann machine is to move from the current state to a new state that has a lower energy.

A Boltzmann machine is considered already trained or already designed. The weights and biases are fixed. This machine "remembers" some "good" stable states. The "bad" states are unstable, and the Boltzmann machine tries to avoid them. The machine will transition to a stable state every time the current state somehow becomes unstable. In this sense, the energy function for a Boltzmann machine is an objective function, which pushes the neuron state to the closest local minimum. This point will be illustrated by examples later in this paper.

Let us consider a special case of the RBM, in which the visible layer and the hidden layer are the same, that is, $h = v$. This special case is referred to as the Hopfield network [24] or the Ising model [20] as shown in Figure 2, where the visible layer and the hidden layer merge into one layer. The Hopfield network is fully interconnected, meaning that each neuron (i.e., unit) is connected to every other neuron (unit). These connections have values of w_{ij} from unit i to unit j. The weights typically have the following restrictions:

Figure 2
A Hopfield network consists of only one layer



$$w_{ii} = 0, \tag{1}$$

$$w_{ij} = w_{ji}. \tag{2}$$

In other words, no unit has a connection with itself, and connections are symmetric. Each neuron has its own bias and an activation function. We can use a symmetric matrix W to represent the weights in Equations (1) and (2). The diagonal elements of W are zeros.

The network is continuously updating itself. The value of each unit v_j is determined by the current values of other units. Unlike a usual perceptron, in which the output is evaluated as an activation function of the inner product of the neuron values from the previous layer and the layer weights plus a bias, the next value of a particular neuron is updated by a not-so-easy algorithm that will be discussed in the next section of this paper. The aim of this algorithm is to select the neuron value such that the energy function of the network is reduced.

For each set of weights w_{ij} and biases b_i, an energy function E is defined as

$$E = -\frac{1}{2}\sum_{i,j} w_{ij}v_iv_j - \sum_i b_iv_i \tag{3}$$

When the weights and biases are given and fixed, the neuron values will get updated by an algorithm, and the network will eventually converge to a local minimum of the energy function E. In a Hopfield network, we do not see the usual activation functions; instead, we use an algorithm to generate a new state of the neurons.

When we say that the network will converge to a lower-energy state, we do not mean to update the weights and biases. The network reaches a lower-energy state by updating the values of the neurons. The weights and biases cannot be changed. The determination (i.e., training) of the weights and biases is not easy and is beyond the scope of this paper [25].

2. Methods

A Hopfield network is fully determined by the weights and biases. The weights and biases then uniquely determine an energy function E as given by Equation (6). The local minima of the energy function E are the stable "solutions" that are stored in the Hopfield network [26, 27]. After Hopfield network neurons are initialized by a random binary string, the network will gradually converge to its closest local "solution" by an algorithm. We are now deriving such an algorithm as follows. This algorithm is going to be nondeterministic.

Traditionally, finding a local minimum in a Hopfield network is first to represent its energy function E as a joint probability density function of a certain stochastic distribution, which is called Boltzmann probability distribution as

$$P(state) = \frac{1}{Z}e^{-\frac{E}{T}}, \tag{4}$$

where T is the "temperature" and Z is a normalization constant so that the sum of $P(state)$ is 1 for all possible states. The term "temperature" is a parameter that is used in some optimization algorithms related to the Boltzmann distribution and artificial annealing. This term is borrowed from thermal dynamics. A higher "temperature" indicates a parameter that encourages randomness. The function defined in Equation (4) is monotonically decreasing. It is important to notice that a large E value corresponds to a small P value and a small E value corresponds to a large P value. In other words, minimizing E is equivalent to maximizing P. A state is a possible binary string of $V = [v_1, v_2, \ldots, v_m]$. If E is a non-negative real number, then $P(state)$ is in $(0,\ 1]$ and

$$\sum_{state} P(state\ V) = 1. \tag{5}$$

It is thus justified to treat P as a probability density function of a certain stochastic distribution. Every state is assigned a P value, which is a probability. Minimizing the energy function E is equivalent to finding a state that maximizes the probability density function P locally.

Let v_k^+ denote the state which is equal to V in all positions except at position k (i.e., v_i with $i \neq k$) and equal to $+1$ at position k (i.e., $v_k = +1$):

$$v_k^+ = (v_1, \ldots, v_{k-1}, 1, v_{k+1}, \ldots, v_m). \tag{6}$$

Similarly, v_k^- is defined as

$$v_k^- = (v_1, \ldots, v_{k-1}, -1, v_{k+1}, \ldots, v_m). \tag{7}$$

We now define the ratio p of two probabilities as

$$p = \frac{P(v_k^+)}{P(v_k^-)} = \frac{e^{-E(v_k^+)}}{e^{-E(v_k^-)}} = e^{-E(v_k^+)+E(v_k^-)}. \tag{8}$$

Here, the uppercase P is a probability, and the lowercase p is the ratio of two probabilities. Let us consider the following conditional probability of neuron v_k being $+1$ given the current state of other neurons

$$\begin{aligned} P(v_k = 1|v_{i\neq k}) &= \frac{P(v_k^+)}{P(v_{i\neq k})} \\ &= \frac{P(v_k^+)}{P(v_k^-) + P(v_k^+)} \\ &= \frac{P(v_k^+)/P(v_k^-)}{1 + P(v_k^+)/P(v_k^-)} \\ &= \frac{p}{1+p} \\ &= \frac{e^{-E(v_k^+)+E(v_k^-)}}{1 + e^{-E(v_k^+)+E(v_k^-)}} \\ &= \frac{e^{-E(v_k^+)+E(v_k^-)}}{1 + e^{-E(v_k^+)+E(v_k^-)}}. \end{aligned} \tag{9}$$

If we introduce the logistic sigmoid function $\varphi(z)$ as

$$\varphi(z) = \frac{1}{1+e^{-z}}, \tag{10}$$

then

$$P(v_k = 1|\text{other } v_{i\neq k}) = \varphi(E(v_k^-) - E(v_k^+)). \tag{11}$$

Consequently, the conditional probability of neuron v_k being -1 is given as

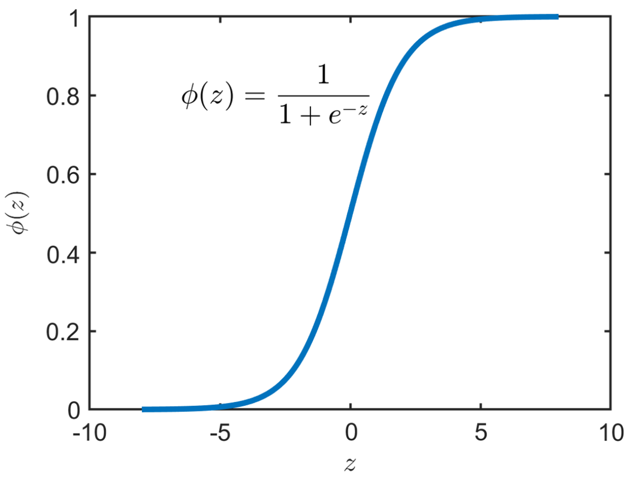
$$P(v_k = -1|\text{other } v_{i\neq k}) = 1 - \varphi(E(v_k^-) - E(v_k^+)). \tag{12}$$

Clearly, according to Equation (10), the values of the logistic sigmoid function are between 0 and 1. A plot of the logistic sigmoid function is shown in Figure 3. It is reasonable to treat $P(v_k = 1|v_{i\neq k})$ as a probability.

It is straightforward to observe that if $P(v_k = 1|\text{other } v_{i\neq k}) > 0.5$, then $P(v_k = -1|\text{other } v_{i\neq k}) < 0.5$. In this case, the network has a higher tendency to choose $v_k = 1$ than to choose $v_k = -1$. According to Figure 3 and Equation (10), $P(v_k = 1|\text{other } v_{i\neq k}) > 0.5$ implies that $E(v_k^+) < E(v_k^-)$. Therefore, the choice of $v_k = 1$ reduces the energy E.

The nondeterministic algorithm can be explained with a small example as follows. We are trying to update the value of the kth neuron v_k. We first calculate $P(v_k = 1|\text{other } v_{i\neq k})$ according to Equation (11). Let us assume that this value happens to be 0.7. We then run a random number generator that generates a random number uniformly distributed on [0, 1]. If this random number is < 0.7, we set $v_k = 1$; otherwise, we set $v_k = -1$. The nondeterministic nature of this algorithm is created by the [0, 1] uniform number generator. It allows a probability of 0.7 to set v_k to be 1 and a probability of 0.3 to set v_k to be -1.

Figure 3
The logistic sigmoid function



In fact, we do not have to convert the energy function E into a Boltzmann distribution (4). Then we can stay away from any random variables. The updating algorithm will become deterministic as explained as follows.

We now explain the deterministic algorithm using the same small example as above. We are still trying to update the value of the kth neuron v_k. We first calculate the energy values $E(v_k^+)$ and $E(v_k^-)$ according to Equation (3), where the exponential function is never used and the energy is never converted into the Boltzmann distribution. If $E(v_k^+) < E(v_k^-)$, we set $v_k = 1$; otherwise, we set $v_k = -1$.

As a matter of fact, there is an easier way. It can be shown that the energy difference can be easily calculated as [27]

$$E(v_k^-) - E(v_k^+) = 2(W_k V + B) \tag{13}$$

where, using an N-neuron Hopfield network example

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} \tag{14}$$

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{15}$$

$$W = \begin{bmatrix} 0 & w_{12} & \cdots & w_{1N} \\ w_{12} & 0 & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1N} & w_{2N} & \cdots & 0 \end{bmatrix} \tag{16}$$

and W_k is the kth row of W. The deterministic algorithm can be readily written as

$$v_k = \sigma\left(\sum_i w_{ik} v_i + b_j\right) \tag{17}$$

where σ is an activation sign function

$$\sigma(x) = \begin{cases} 1 & if \quad x > 0 \\ -1 & if \quad x \leq 0. \end{cases} \tag{18}$$

It is interesting to notice that the deterministic algorithm (17) happens to be the usual expression for a perceptron, which involves a dot product and bias term and a nonlinear activation function.

The experiment details are given in Tables 1 and 3, which show the logical relationship between the inputs and the output. The inverse solutions (i.e., the inputs) for a given output in general is not unique. Mathematically speaking, the inverse function really does not exist because the function value is not unique. If the "inverse function" has three values, these three values should have an equal chance to reach. The multiple solutions are the local minima of an objective function which is defined by the matrices W and B. The proposed deterministic optimization algorithm is iteratively updating the variables (v_1 and v_2, in our examples) according to algorithm (17).

Table 1
The logic OR gate in {−1, 1}

v_1	v_2	v_3
−1	−1	−1
−1	1	1
1	−1	1
1	1	1

Instead of using Equation (17), an alternative way to update a variable v_k is the direct usage of the objective function E as follows

$$v_k = \begin{cases} 1 & \text{if} \quad E(v_k = 1) < E(v_k = -1) \\ -1 & \text{if} \quad E(v_k = 1) > E(v_k = -1) \end{cases} \tag{19}$$

where the objective function E is evaluated using the current variable values other than v_k.

In the deterministic algorithm, the concept of the probability distribution is never used. The entire Hopfield network updating procedure is deterministic. Some numeric examples are presented to compare these two methods in the next section.

3. Results

This section uses two examples to illustrate the performance differences between a nondeterministic algorithm and a deterministic algorithm in Hopfield network applications. In these two examples, the Hopfield networks are assumed to be already designed. In other words, the weights W and biases B are given. The goal is to find the closest local minimum of the energy function.

In the first example, we use a Hopfield network to remember the stable OR gate states using $\{-1, 1\}$ as variables. The logic OR gate relations are defined in Table 1.

The four stable states listed in Table 1 are the four lower-energy states with negative energy values in an energy function E defined in Equation (3), where the weights and biases are the elements of W and B given as

$$W = \begin{bmatrix} 0 & -1 & 2 \\ -1 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \tag{20}$$

and

$$B = \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix} \tag{21}$$

The stable states are (−1, −1, −1), (−1, 1, 1), (1, −1, 1), and (1, 1, 1).

We now verify that the energy function defined by Equations (20) and (21) indeed has the local minima for the desired stable OR gate states and has unstable larger energy values for the states that do not satisfy the OR gate relationship. There are eight possible states. Four of the states are stable, and the other four states are unstable as listed in Table 2, as calculated by using Equation (3).

Table 2
The energy values for all possible states by using the logic OR gate relationship in {−1, 1}

	v_1	v_2	v_3	E	
An OR gate state	−1	−1	−1	−3	Stable
An OR gate state	−1	1	1	−3	Stable
An OR gate state	1	−1	1	−3	Stable
An OR gate state	1	1	1	−3	Stable
A non-OR gate state	−1	−1	1	1	Unstable
A non-OR gate state	−1	1	−1	1	Unstable
A non-OR gate state	1	−1	−1	1	Unstable
A non-OR gate state	1	1	−1	9	Unstable

We first clamp the value of v_3 to +1. In this case, according to Table 1, there are three solutions for (v_1, v_2); they are $(-1, 1)$, $(1, -1)$, and $(1, 1)$. With v_3 clamped as +1, we generate 100,000 random initial values of (v_1, v_2) and run the two energy minimization algorithms (one deterministic and one nondeterministic), respectively. For each algorithm, we obtain 100,000 converged states. The converged states are summarized in a histogram. The histogram results are shown in Figures 4 and 5, respectively. The ideal result should show the equal likelihood of the solutions of $(-1, 1)$, $(1, -1)$, and $(1, 1)$. There should be very little occurrence elsewhere. It can be observed that the deterministic algorithm gives much better results than the nondeterministic algorithm. In the figures, "−1" is labeled as "0" along the horizontal axes, and $v_1v_2v_3$ is labeled as ABC with $A = v_1$, $B = v_2$, and $C = v_3$.

Next, we clamp the value of v_3 to −1. In this case, according to Table 1, there is only one solution for (v_1, v_2), that is, $(-1, -1)$. With v_3 clamped to −1, we generate 100,000 random initial values of (v_1, v_2) and run the two energy minimization algorithms (one deterministic and one nondeterministic), respectively. The histogram

Figure 4
The histogram of the Hopfield network converged states for the OR gate model with v_3 clamped to +1 using the deterministic algorithm

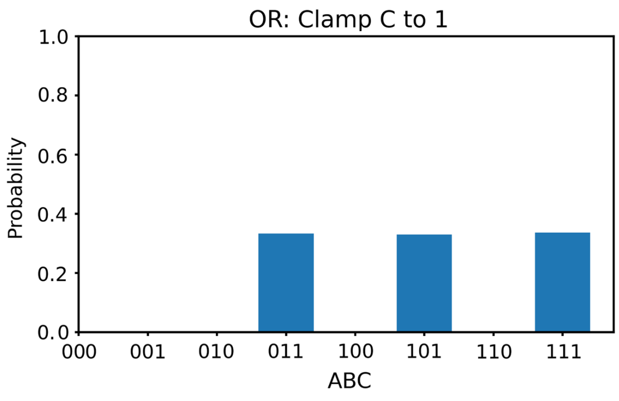
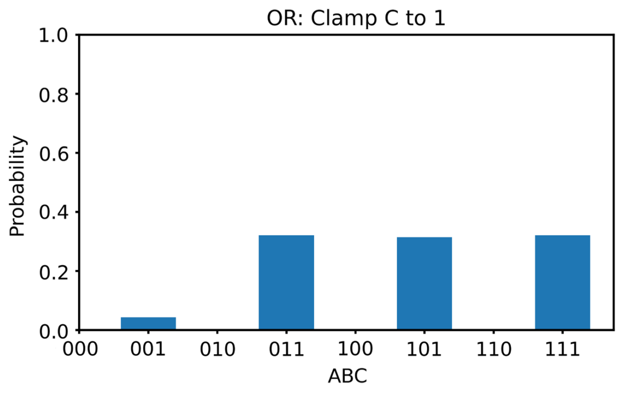


Figure 5
The histogram of the Hopfield network converged states for the OR gate model with v_3 clamped to +1 using the nondeterministic algorithm



results are shown in Figures 6 and 7, respectively. The ideal result should only show $(-1,\ -1)$ with nothing elsewhere. Once again, the deterministic algorithm gives much better results than the nondeterministic algorithm.

Figure 6
The histogram of the Hopfield network converged states for the OR gate model with v_3 clamped to -1 using the deterministic algorithm

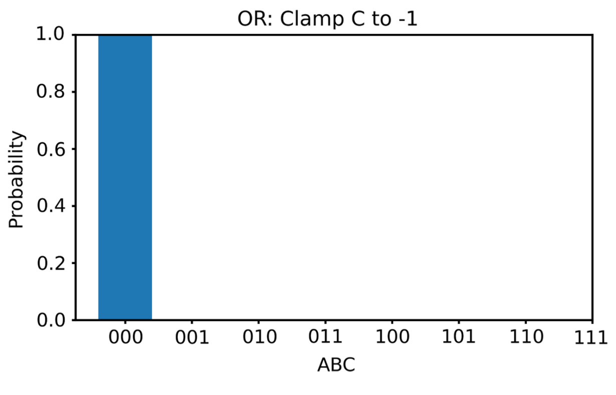
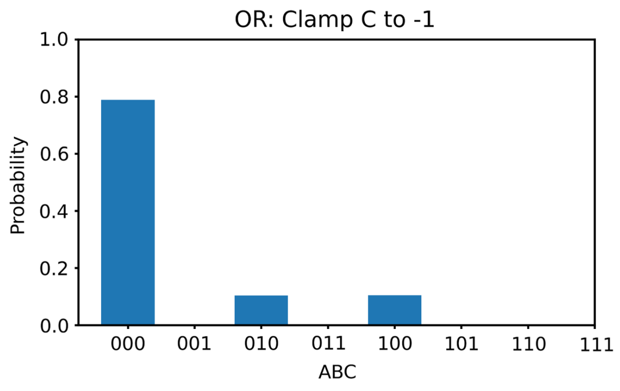


Figure 7
The histogram of the Hopfield network converged states for the OR gate model with v_3 clamped to -1 using the nondeterministic algorithm



We can repeat the same studies for the logic AND gate using $\{-1,\ 1\}$. The AND gate logic relationships are listed in Table 3.

The four stable states listed in Table 3 are the four lower-energy states with negative energy values in an energy function E defined in Equation (3) whose W and B are given as

Table 3
The logic AND gate in {−1, 1}

v_1	v_2	v_3
−1	−1	−1
−1	1	−1
1	−1	−1
1	1	1

$$W = \begin{bmatrix} 0 & -1 & 2 \\ -1 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \tag{22}$$

and

$$B = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}. \tag{23}$$

The stable states are $(-1,\ -1,\ -1)$, $(-1,\ 1,\ 1)$, $(1,\ -1,\ 1)$, and $(1,\ 1,\ 1)$.

It is interesting to notice that the W matrices in Equations (20) and (22) are the same. However, the B vector for the AND gate is the negative counterpart of the B vector for the OR gate.

We now verify that the energy function defined by Equations (22) and (23) indeed has the local minima for the desired stable AND gate states and has unstable larger energy values for the states that do not satisfy the AND gate relationship. There are eight possible states. Four of the states are stable, and the other four states are unstable as listed in Table 4, as calculated by using Equation (3).

Similar to the OR gate study, we first clamp the value of v_3 to -1. In this case, according to Table 3, there are three solutions for $(v_1,\ v_2)$; they are $(-1,\ -1)$, $(1,\ -1)$, and $(-1,\ 1)$. With v_3 clamped to -1, we generate 100,000 random initial values of

Table 4
The energy values for all possible states by using the logic AND gate relationship in {−1, 1}

	v_1	v_2	v_3	E	
An AND gate state	−1	−1	−1	−3	Stable
An AND gate state	−1	1	−1	−3	Stable
An AND gate state	1	−1	−1	−3	Stable
An AND gate state	1	1	1	−3	Stable
A non-AND gate state	−1	−1	1	9	Unstable
A non-AND gate state	−1	1	1	1	Unstable
A non-AND gate state	1	−1	1	1	Unstable
A non-AND gate state	1	1	−1	1	Unstable

$(v_1,\ v_2)$ and run the two energy minimization algorithms (one deterministic and one nondeterministic), respectively. The histogram results are shown in Figures 8 and 9, respectively. The ideal result should show the equal likelihood of the solutions of $(-1,-1)$, $(1,\ -1)$, and $(-1,\ 1)$. There should be very little occurrence elsewhere. The deterministic algorithm gives much better results than the nondeterministic algorithm.

Figure 8
The histogram of the Hopfield network converged states for the AND gate model with v_3 clamped to -1 using the deterministic algorithm

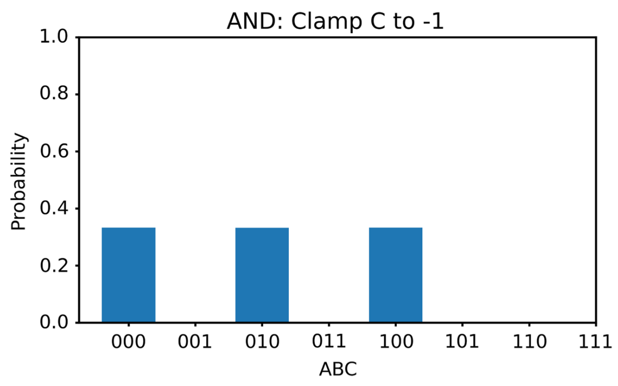
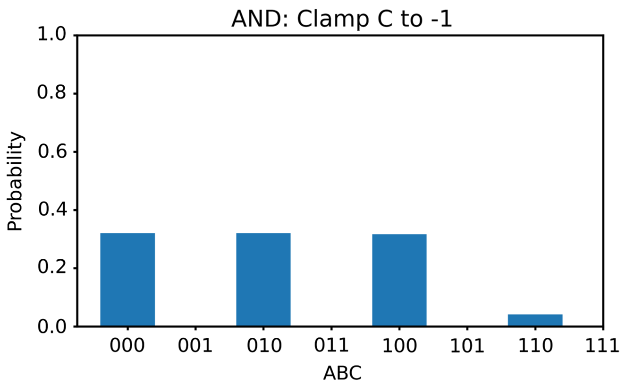


Figure 9
The histogram of the Hopfield network converged states for the AND gate model with v_3 clamped to -1 using the nondeterministic algorithm



Next, we clamp the value of v_3 to +1. In this case, according to Table 2, there is only one solution for $(v_1,\ v_2)$, that is, $(1,\ 1)$. With v_3 clamped as +1, we generate 100,000 random initial values of $(v_1,\ v_2)$ and run the two energy minimization algorithms (one deterministic and one nondeterministic), respectively. The histogram results are shown in Figures 10 and 11, respectively. The ideal result should only show $(1,\ 1)$ with nothing elsewhere. Once again, the deterministic algorithm gives much better results than the nondeterministic algorithm.

Figure 10
The histogram of the Hopfield network converged states for the AND gate model with v_3 clamped to +1 using the deterministic algorithm

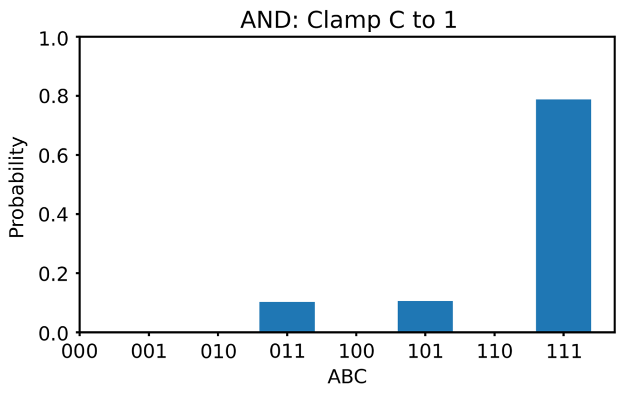
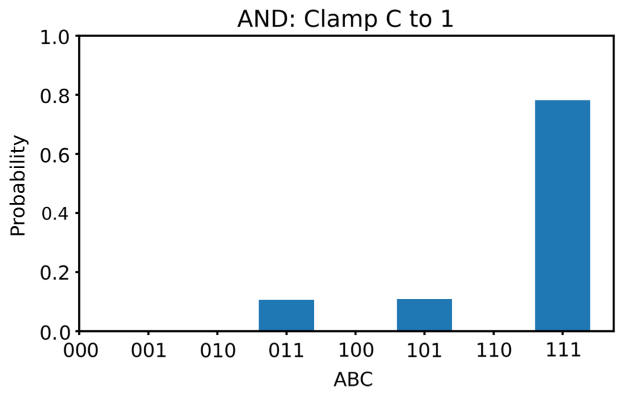


Figure 11
The histogram of the Hopfield network converged states for the AND gate model with v_3 clamped to +1 using the nondeterministic algorithm



4. Discussion

This paper focuses on the theoretical problem of local optimization. In the two examples, we have two problems. One problem is to find the inverse function of an AND gate logic. The other problem is to find the inverse function of an OR gate logic. These two inverse functions are used to form two objective functions, which have multiple local minima. These two objective functions are distinct, and their local minima are different. Our computer simulations demonstrate that for an objective function with multiple local minima, the proposed method is not biased if the initial condition is uniformly sampled.

An NP-hard problem is to search for the global solution. Our method is a greedy method, which looks for the nearest local minimum. In this paper, both deterministic and nondeterministic methods are greedy methods. To search for the local minimum, the iterative algorithms (similar to the gradient descent algorithms) take many steps to reach the local minimum. Since the nondeterministic algorithm allows a percentage of the total number of steps to move in the uphill direction, it will take more steps to converge. An error in the context of the current paper is referred to the situation where the algorithm converges to a local minimum that is not the nearest one.

The implementation of the proposed deterministic algorithm is fairly straightforward for a binary problem, in which the variables can only take two values. On the other hand, in an optimization algorithm with continuous variables, the update step size is tricky to select. If the step size is too large, the algorithm may diverge. If the step size is too small, the algorithm may take too long to converge. For a binary optimization algorithm, a variable may stay at its current value or change to the other value. The most difficult part of using an RBM to solve a practical problem is to set up the objective function.

For deterministic optimization, the objective function is the same as the energy function. No conversion is needed. On the other hand, for the nondeterministic optimization, the energy function must be mapped to the range of [0,1] to treat it as a probability distribution.

For any discrete problem, for example, integer programming and binary programming, a local minimum is perfectly reached. However, to find the global minimum requires a brute-force search and is well-known to be NP-hard.

The deterministic optimization finds the nearest local minimum from the initial condition. In order to find the global optimum, one must start with a large number of different initial conditions and compare their results.

For binary Boltzmann machines, the variables can only take two values. Therefore, the activation function can only take two values. Thus, the step function is a natural choice for the activation function.

5. Conclusions

An RBM is a fully connected neural network consisting of two layers: the visible layer and the hidden layer, respectively. A common RBM is binary valued and associated with an energy function, which is a quadratic form. When these two layers are the same, the RBM is reduced to a Hopfield network. In this paper, we consider a binary Hopfield network, where the neurons take the values of $\{-1, 1\}$. However, the weights and biases can take any real numbers.

In a Hopfield network, the weights and biases are designed by a problem at hand. The Hopfield network is not trained. For example, the weights can be calculated by the Hebbian learning rule [28]. Once the Hopfield network is designed, it can remember some good stable solutions. Algorithms are available to update the state of the Hopfield network so that a random state can converge to a local minimum of the energy function.

The traditional method is to represent this energy function as a joint probability distribution by using an exponential function conversion. Minimizing the energy function is equivalent to iteratively increasing the conditional probability for each unit (i.e., neuron).

For a binary Hopfield network, at each iteration step, a unit has a probability, say q, to change to 1. The state of a unit is a random variable, and its update procedure is implemented randomly according to the conditional probability q. Therefore, the iterative algorithm is nondeterministic.

This paper uses a different method to update a binary Hopfield network's neuron value. We do not convert the energy function into a joint probability distribution. We use the original definition of the network energy. The optimization of the energy function is now a deterministic problem. The optimization procedure is still iterative but no longer nondeterministic. The proposed deterministic procedure has advantages over the nondeterministic procedure: faster convergence and smaller errors. Best of all, the deterministic update algorithm is in the standard form of a typical perceptron, which consists of a dot product, a bias, and a nonlinear activation function.

The main goal of a Hopfield network is to search for local minima of an energy function. Finding the local minima of an energy function is a mathematical problem. In our manuscript, the proposed deterministic method is applied to four problems: an inversion of the AND gate function with the output clamped to 1, an inversion of the AND gate function with the output clamped to -1, an inversion of the OR gate function with the output clamped to 1, and an inversion of the OR gate function with the output clamped to -1. The proposed deterministic method is compared with the more popular nondeterministic method using these four applications in the form of histograms in the "Results" section. The deterministic method can be applied to many other optimization problems, especially the NP-hard problems, such as number partition, graph partitioning, cliques, satisfiability, minimal maximal matching, graph coloring, tree problems, knapsack with integer weights, and binary integer linear programming, to name a few [25]. This paper presents some computer simulations to compare these two methods. The deterministic algorithm demonstrates more advantages than the nondeterministic algorithm.

Funding Support

This work is sponsored in part by a grant from the National Institutes of Health 2R15EB024283.

Conflicts of Interest

The author declares that he has no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Gengsheng L. Zeng: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition.

References

[1] Alphonse, A. S., Shankar, K., Jeyasheela Rakkini, M. J., Ananthakrishnan, S., Athisayamani, S., Robert Singh, A., & Gobi, R. (2021). A multi-scale and rotation-invariant phase pattern (MRIPP) and a stack of restricted Boltzmann machine

(RBM) with preprocessing for facial expression classification. *Journal of Ambient Intelligence & Humanized Computing*, *12*(3), 3447–3463. https://doi.org/10.1007/s12652-020-02517-7

[2] Aslan, N., Dogan, S., & Koca, G. O. (2023). Automated classification of brain diseases using the Restricted Boltzmann Machine and the Generative Adversarial Network. *Engineering Applications of Artificial Intelligence*, *126*, 106794. https://doi.org/10.1016/j.engappai.2023.106794

[3] Biamonte, J. D. (2008). Nonperturbative *k*-body to two-body commuting conversion Hamiltonians and embedding problem instances into Ising spins. *Physical Review A*, *77*(5), 052331. https://doi.org/10.1103/PhysRevA.77.052331

[4] Fischer, A., & Igel, C. (2012). An introduction to Restricted Boltzmann Machines. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 17th Iberoamerican Congress,* 14–36. https://doi.org/10.1007/978-3-642-33275-3_2

[5] Kirubahari, R., & Amali, S. M. J. (2024). An improved restricted Boltzmann Machine using Bayesian Optimization for Recommender Systems. *Evolving Systems, 15*, 1099–1111. https://doi.org/10.1007/s12530-023-09520-1

[6] Patel, S., Canoza, P., & Salahuddin, S. (2022). Logically synthesized and hardware-accelerated restricted Boltzmann machines for combinatorial optimization and integer factorization. *Nature Electronics*, *5*(2), 92–101. https://doi.org/10.1038/s41928-022-00714-0

[7] Savitha, R., Ambikapathi, A., & Rajaraman, K. (2020). Online RBM: Growing Restricted Boltzmann Machine on the fly for unsupervised representation. *Applied Soft Computing*, *92*, 106278. https://doi.org/10.1016/j.asoc.2020.106278

[8] Zhang, N., Ding, S., Zhang, J., & Xue, Y. (2018). An overview on Restricted Boltzmann Machines. *Neurocomputing*, *275*, 1186–1199. https://doi.org/10.1016/j.neucom.2017.09.065

[9] Berrones-Santos, A., & Bagnoli, F. (2023). Biologically plausible Boltzmann machine. *Informatics*, *10*(3), 62. https://doi.org/10.3390/informatics10030062

[10] Chowdhury, S., Niazi, S., & Camsari, K. Y. (2024). Mean-field assisted deep Boltzmann learning with probabilistic computers. *arXiv Preprint: 2401.01996*. https://doi.org/10.48550/arXiv.2401.01996

[11] Viteritti, L. L., Ferrari, F., & Becca, F. (2022). Accuracy of restricted Boltzmann machines for the one-dimensional J_1_J_2 Heisenberg model. *SciPost Physics*, *12*(5), 166. https://doi.org/10.21468/SciPostPhys.12.5.166

[12] Dixit, V., Selvarajan, R., Alam, M. A., Humble, T. S., & Kais, S. (2021). Training Restricted Boltzmann Machines with a d-wave quantum annealer. *Frontiers in Physics*, *9*, 589626. https://doi.org/10.3389/fphy.2021.589626

[13] Fernandez-de-Cossio-Diaz, J., Hardouin, P., du Moutier, F. X. L., Di Gioacchino, A., Marchand, B., Ponty, Y., . . . , & Cocco, S. (2024). Designing molecular RNA switches with Restricted Boltzmann Machines. *bioRxiv Preprint*. https://doi.org/10.1101/2023.05.10.540155

[14] Feng, Z., Winston, E., & Kolter, J. Z. (2023). Monotone deep Boltzmann machines. *arXiv Preprint:2307.04990*. https://doi.org/10.48550/arXiv.2307.04990

[15] Gu, J., & Zhang, K. (2022). Thermodynamics of the Ising model encoded in Restricted Boltzmann Machines. *Entropy*, *24*(12), 1701. https://doi.org/10.3390/e24121701

[16] He, F., Huang, X., Wang, X., Qiu, S., Jiang, F., & Ling, S. H. (2021). A neuron image segmentation method based Deep Boltzmann Machine and CV model. *Computerized Medical Imaging and Graphics*, *89*, 101871. https://doi.org/10.1016/j.compmedimag.2021.101871

[17] Revathi, A., & Santhi, S. G. (2023). Routing-based restricted Boltzmann machine learning and clustering algorithm in wireless sensor network. In *Proceedings of Emerging Trends and Technologies on Intelligent Systems,* 341–357. https://doi.org/10.1007/978-981-19-4182-5_28

[18] Cipra, B. A. (2000). The Ising model is NP-complete. *SIAM News*, *33*(6), 1–3.

[19] Mondal, A., & Srivastava, A. (2021). Ising-FPGA: A spintronics-based reconfigurable Ising model solver. *ACM Transactions on Design Automation of Electronic Systems*, *26*(1), 4. https://doi.org/10.1145/3411511

[20] Singh, S. P. (2020). The Ising model: Brief introduction and its application. In S. Sivasankaran, P. K. Nayak & E. Günay (Eds.), *Solid state physics-Metastable, spintronics materials and mechanics of deformable bodies – Recent progress* (pp. 115–132). IntechOpen. https://doi.org/10.5772/intechopen.90875

[21] Matveev, A. A., Safin, A. R., & Nikitov, S. A. (2023). Exceptional points in coupled vortex-based spin-torque oscillators. *Physical Review B*, *108*(17), 174443. https://doi.org/10.1103/PhysRevB.108.174443

[22] Yasudo, R., Nakano, K., Ito, Y., Katsuki, R., Tabata, Y., Yazane, T., & Hamano, K. (2022). GPU-accelerated scalable solver with bit permutated cyclic-min algorithm for quadratic unconstrained binary optimization. *Journal of Parallel and Distributed Computing*, *167*, 109–122. https://doi.org/10.1016/j.jpdc.2022.04.016

[23] Yasudo, R. (2023). Bandit-based variable fixing for binary optimization on GPU parallel computing. In *31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing,* 154–158. https://doi.org/10.1109/PDP59025.2023.00031

[24] Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., . . . , & Hochreiter, S. (2020). Hopfield networks is all you need. *arXiv Preprint:2008.02217*. https://doi.org/10.48550/arXiv.2008.02217

[25] Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, *2*, 5. https://doi.org/10.3389/fphy.2014.00005

[26] Camsari, K. Y., Faria, R., Sutton, B. M., & Datta, S. (2017). Stochastic *p*-bits for invertible logic. *Physical Review X*, *7*(3), 031014. https://doi.org/10.1103/PhysRevX.7.031014

[27] Christianb93. (2018). *The Ising model and Gibbs sampling.* Retrieved from: https://leftasexercise.com/2018/03/12/the-ising-model-and-gibbs-sampling/

[28] Widrow, B., Kim, Y., Park, D., & Perin, J. K. (2024). Nature's learning rule: The Hebbian-LMS algorithm. In R. Kozma, C. Alippi, Y. Choe & F. C. Morabito (Eds.), *Artificial intelligence in the age of neural networks and brain computing* (pp. 11–40). Elsevier. https://doi.org/10.1016/B978-0-323-96104-2.00012-9

How to Cite: Zeng, G. L. (2024). Deterministic Versus Nondeterministic Optimization Algorithms for the Restricted Boltzmann Machine. *Journal of Computational and Cognitive Engineering*, *3*(4), 404–411. https://doi.org/10.47852/bonviewJCCE42022789