Two-layer tree-connected feed-forward neural network model for neural cryptography

Xinyu Lei and Xiaofeng Liao[*]
College of Computer Science, Chongqing University, Chongqing 400044, People's Republic of China

Fei Chen[†]
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong

Tingwen Huang[‡]
Texas A&M University at Qatar, Doha, P.O. Box 23874, Qatar

(Received 29 November 2012; published 18 March 2013)

Neural synchronization by means of mutual learning provides an avenue to design public key exchange protocols, bringing about what is known as neural cryptography. Two identically structured neural networks learn from each other and reach full synchronization eventually. The full synchronization enables two networks to have the same weight, which can be used as a secret key for many subsequent cryptographic purposes. It is striking to observe that after the first decade of neural cryptography, the tree parity machine (TPM) network with hidden unit $K = 3$ appears to be the sole network that is suitable for a neural protocol. No convincingly secure neural protocol is well designed by using other network structures despite considerable research efforts. With the goal of overcoming the limitations of a suitable network structure, in this paper we develop a two-layer tree-connected feed-forward neural network (TTFNN) model for a neural protocol. The TTFNN model captures the notion that two partners are capable of exchanging a vector with multiple bits in each time step. An in-depth study of the dynamic process of TTFNN-based protocols is then undertaken, based upon which a feasible condition is theoretically obtained to seek applicable protocols. Afterward, according to two analytically derived heuristic rules, a complete methodology for designing feasible TTFNN-based protocols is elaborated. A variety of feasible neural protocols are constructed, which exhibit the effectiveness and benefits of the proposed model. With another look from the perspective of application, TTFNN-based instances, which can outperform the conventional TPM-based protocol with respect to synchronization speed, are also experimentally confirmed.

DOI: 10.1103/PhysRevE.87.032811 PACS number(s): 84.35.+i, 87.18.Sn, 89.70.−a

I. INTRODUCTION

A. Application background

The public key exchange algorithm, initially introduced by Diffie and Hellman in their groundbreaking paper [1], is still a major concern in the field of modern cryptography. A public key exchange protocol enables two partners, named A and B, to communicate over an untrusted and unsecure channel to come up with a common secret value called a *secret key*, whereas an attacker E should be unable to retrieve the key even with the ability to eavesdrop in the communication channel. The common secret key is subsequently used to provide privacy, authentication, data integrity, or for other cryptographic purposes.

In classical cryptography, the solutions for the key exchange problem are mainly based on algebraic number theory (see [2]). The search for new key exchange mechanisms still remains a great challenge. Nevertheless, the synchronization phenomenon of interacting artificial neural networks (ANNs) offers an additional avenue to solve the key exchange problem. A remarkable feature of such a neural-based key exchange protocol is the absence of concrete *one-way trapdoor functions*. This situation is similar to a secret key agreement by using public discussion [3].

B. Fundamental principle

Synchronization of ANNs is a special case of online learning. Two ANNs start with randomly chosen weight vectors. In each time step they receive a common input vector, compute their outputs, and exchange them through a public channel. If they agree on the mapping between the current input and output, their weight vectors are updated according to a suitable mutual learning rule. In the case of discrete weight vectors, this process leads to full synchronization in a finite number of steps. The full synchronization enables the two ANNs to have identical weight vectors, which can be used as the common secret key. This synchronization process is involved in interacting two ANNs. The output of one ANN influences the update behavior of the other, therefore it corresponds to a *mutual learning*. But on the contrary, a third ANN can be trained using the examples, input vectors, and output values generated in the process of synchronization. Note that this third ANN cannot influence the update behavior of the others; it corresponds to a *unidirectional learning*.

In the case of a one-layer perceptron, which is a simple ANN, one finds that the average number of steps needed for mutual learning and unidirectional learning is the same [4,5]. But in the case of a more complex tree parity machine (TPM) network with hidden unit $K = 3$, a crucial scaling law is observed: the average synchronization steps of the

[*]xy-lei@qq.com, xfliao@cqu.edu.cn
[†]chenfeiorange@163.com
[‡]tingwen.huang@qatar.tamu.edu

mutual learning grow polynomially with the synaptic depth L, whereas the average synchronization steps of the unidirectional learning grow exponentially with L [6–8]. This scaling law provides the theoretical foundation for us to construct a TPM ($K = 3$) -based key exchange protocol. In such a protocol, the partners with mutual learning synchronize much faster than the attackers with unidirectional learning. Consequently, the difference in synchronization speed is essential for the security of the neural key exchange protocol.

In addition to the synchronization of ANNs, key exchange can also been achieved by using other interesting systems, including chaotic maps and coupled lasers [9–12]. They use almost the same cryptographic protocol by just substituting the neural networks. We will not elaborate on these works, as this paper is written from the perspective of neural cryptography.

C. Related work

The innovative idea of using neural synchronization for a cryptographic key exchange protocol has stimulated much research in this area. An excellent review of neural cryptography can be found in Ref. [13]. In the following, we will briefly summarize notable works, encompassing very recent research.

Numerous studies have been based on the TPM ($K = 3$) -based protocol. Three kinds of learning rules are analyzed [14], and the dynamics of such a protocol are studied [15]. In addition, the model of the classical ruin problem is used to examine the average synchronization time of the protocol [16]. Four common attacks, including a simple attack [17], a geometric attack [18], a majority attack [19], and a genetic attack [14], are also experimentally investigated in detail. Furthermore, the hardware implementation of the TPM ($K = 3$) -based protocol in embedded systems is realized [20]. Some informative comments about previous research are embedded in this paper whenever they are needed.

As for the TPM ($K = 3$) -based protocol, efforts to enhance it have been ongoing. At a high level, these efforts can be generally divided into two distinct types.

(1) The first type focuses exclusively on the public information (the public generated input vectors or the public exchanged output bits) in the neural protocol. Among this type of effort, we note three representative mechanisms. (i) The feedback mechanism tries to keep input vectors partially secret from attackers by means of adopting a linear feedback shift register (LFSR) [21]. This approach only achieves a small improvement in security. (ii) The queries mechanism replaces the random input vectors by generated queries [22]. It is shown that in this way, the communicating partners can accelerate the synchronization process, whereas the attacker cannot. (iii) The error prediction mechanism introduces an algorithm called "do not trust my partner" (DTMP), which relies on one party sending erroneous output bits, with the other party being capable of predicting and correcting this error [23]. The error prediction mechanism disrupts the attacker confidence in the exchanged outputs and thus increases the attack difficulty.

(2) The second type concentrates on constructing new network structures as well as new learning rules on which the neural protocol is based. In the first decade of the neural protocol, a few such attempts failed eventually. For instance, the permutation parity machines (PPMs) -based protocol [24,25] is among the few to consider designing a new network structure and learning rule. Unfortunately, such a PPM-based protocol is soon broken in the presence of a probabilistic attack [26].

D. Research motivations and road map

Previous papers mainly explored the TPM-based protocol itself. We realized that up to now there has been no successful construction of a new network structure other than the conventional TPM ($K = 3$) network. For application significance, neural protocols based on other networks may considerably improve the conventional one. We are therefore motivated to seek such neural protocols.

Note that the conventional protocol computes and exchanges only one bit in each time step. We start, therefore, by extending the number of output bits. Then the network structure is extended to more general two-layer tree-connected feed-forward neural networks (TTFNNs) (we also refer to our model as the TTFNN model in this paper). In TTFNN-based protocols, the learning rules should be adaptively modified to capture the multiple exchanged bits as well. Intuitively, the TTFNN model captures the notion that two partners exchange more information in each time step, which may lead to an improvement of the neural protocol with respect to efficiency and security.

E. Main contributions

We regard our main contributions as threefold:

(i) By writing a Markovian process of weights in the synchronization process, the dynamics of the TTFNN-based protocol are investigated. According to the theoretically derived feasible condition, a "trial and verification" methodology for seeking feasible protocols is clearly presented.

(ii) By introducing the notion of the Hamming distance, a completed methodology for designing feasible TTFNN-based protocols is elaborated upon based on two analytically derived heuristic rules.

(iii) Numerous feasible protocols are designed in this paper. Better TTFNN-based instances with regard to synchronization speed are also experimentally confirmed.

F. Paper organization

The remainder of this paper is organized as follows. Section II describes the TTFNN model. In Sec. III, we investigate the dynamics of weights in the learning process, based upon which the feasible condition is derived. Afterward, we exemplify the methodology to apply the feasible condition to search for additional feasible protocols in Sec. IV. In Sec. V, a completed methodology for designing feasible protocols is detailed. Finally, some conclusions are drawn in Sec. VI.

II. MODEL DESCRIPTION

We consider the TTFNN model consisting of two elements, namely a network and a learning algorithm, which are formulated below.

A. Network

The structure of a general TTFNN is shown in Fig. 1. Another similar structure is called a two-layer fully-connected

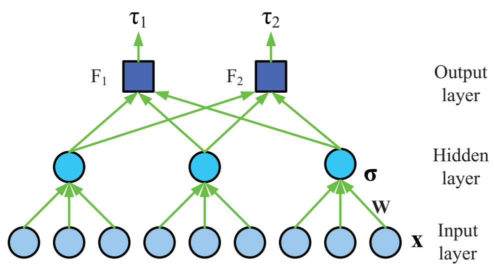

FIG. 1. (Color online) A 3-3-2 TTFNN.

feed-forward neural network [27]. A TTFNN consists of three layers. It has N inputs in the input layer, K hidden units in the hidden layer, and O outputs in the output layer. For the sake of brevity, the network is referred to as N-K-O TTFNN. Each hidden unit works like a perceptron with independent local fields. The input values are binary, $x_{ij} \in \{-1,+1\}$. The weights are discrete numbers between $-L$ and $+L$, $w_{ij} \in \{-L,\ldots,+L\}$. Here index $i = 1,\ldots,K$ denotes the ith hidden unit of a TTFNN and $j = 1,\ldots,N$ denotes the jth element of a weight vector. The local field h_i of the ith hidden unit is defined as

$$h_i = \frac{1}{\sqrt{N}}\mathbf{w}_i\mathbf{x}_i = \frac{1}{\sqrt{N}}\sum_{j=1}^{N} w_{ij}x_{ij}. \quad (1)$$

The output σ_i of the ith hidden unit is defined as the sign of h_i,

$$\sigma_i = \mathrm{sgn}(h_i). \quad (2)$$

The special case $h_i = 0$ is mapped to $\sigma_i = -1$ in order to ensure a binary output value, i.e., $\sigma_i \in \{-1,+1\}$. Generally, the mappings from the internal representation $\Sigma = (\sigma_1,\ldots,\sigma_K)$ to the total output vector $\mathbf{T} = (\tau_1,\ldots,\tau_O)$ are coded by Boolean functions,

$$\tau_q = F_q(\Sigma) = F_q(\sigma_1,\ldots,\sigma_K), \quad (3)$$

where q is indexed from 1 to O and $\tau_q \in \{-1,+1\}$.

As two special cases of TTFNNs, the TPM network and the tree committee machine (TCM) network use N-K-1 structure. The total output τ_{TPM} of the TPM network is

$$\tau_{\mathrm{TPM}} = F_1(\Sigma) = \mathrm{sgn}\left(\prod_{i=1}^{K}\sigma_i\right). \quad (4)$$

The total output τ_{TCM} of the TCM network is

$$\tau_{\mathrm{TCM}} = F_1(\Sigma) = \mathrm{sgn}\left(\sum_{i=1}^{K}\sigma_i\right). \quad (5)$$

B. Learning algorithm

Algorithm 1: Learning algorithm.

Step 1: A and B start with an identically structured TTFNN and they select random initial weight vectors $\mathbf{w}_i^A(t=0)$ and $\mathbf{w}_i^B(t=0)$, respectively, both of which are kept secret.

Step 2: A and B receive identical random input vectors $\mathbf{x}_i$ in each time step, where $\mathbf{x}_i$ are generated publicly. Afterward, A and B compute and exchange their output vectors $\mathbf{T}^A$ and $\mathbf{T}^B$, respectively.

Step 3: Upon receipt of $\mathbf{T}^B$ for A ($\mathbf{T}^A$ for B), both partners update their weight vectors $\mathbf{w}^A$ and $\mathbf{w}^B$ according to a certain learning rule.

Step 4: Repeat step 2 to step 3 until $\mathbf{w}^A(t) = \mathbf{w}^B(t)$.

End Algorithm 1

The TTFNN-based learning algorithm is presented in Algorithm 1. In this algorithm, three learning rules, which are natural extensions of learning rules for the TPM-based protocol, can be described as follows:

(1) *Hebbian learning rule*:

$$w_{ij}^{A/B}(t+1) = g\left(w_{ij}^{A/B}(t) + x_{ij}\sigma_i^{A/B}\Theta\left(\sigma_i^{A/B}\tau_q\right)\prod_{i=1}^{O}\Theta\left(\tau_i^A\tau_i^B\right)\right). \quad (6)$$

(2) *Anti-Hebbian learning rule*:

$$w_{ij}^{A/B}(t+1) = g\left(w_{ij}^{A/B}(t) - x_{ij}\sigma_i^{A/B}\Theta\left(\sigma_i^{A/B}\tau_q\right)\prod_{i=1}^{O}\Theta\left(\tau_i^A\tau_i^B\right)\right). \quad (7)$$

(3) *Random-walk learning rule*:

$$w_{ij}^{A/B}(t+1) = g\left(w_{ij}^{A/B}(t) + x_{ij}\Theta\left(\sigma_i^{A/B}\tau_q\right)\prod_{i=1}^{O}\Theta\left(\tau_i^A\tau_i^B\right)\right). \quad (8)$$

In these learning rules, q can take values from $\{1,\ldots,O\}$. $\Theta(x)$ is the Heaviside function, i.e., $\Theta(x) = 0$ for $x < 0$ and $\Theta(x) = 1$ otherwise. If any component of the weight vectors $\mathbf{w}^A$ and $\mathbf{w}^B$ moves outside the range $\{-L,\ldots,+L\}$, it is replaced by the nearest boundary, either $-L$ or $+L$. This is achieved by

$$g(w) = \begin{cases}\mathrm{sgn}(w)L & \text{for} \quad |w| > L,\\ w & \text{otherwise.}\end{cases} \quad (9)$$

Two types of learning rules are taken into account in the TTFNN model.

(L_1) Random-walk learning rule with sufficiently large N.

(L_2) Hebbian or anti-Hebbian learning rules with $N \to +\infty$.

C. Characteristic parameters

A TTFNN-based protocol is parametrized by its characteristic parameters, as sketched below.

(i) The structure of a TTFNN, i.e., the values of N-K-O. The choice of N is determined by the chosen type of learning rule, either (L_1) or (L_2). Therefore, without the consideration of N, the first factor is determined by K and O.

(ii) Boolean functions F_q.

(iii) The learning rule.

D. Secure definition

A *computationally secure system* [1] requires a neural protocol to maintain the following two conditions:

(C_1) The communicating partners take only polynomial (P) time in terms of the security parameter L to come up with a secret key.

(C_2) All currently known attacks take nondeterministic polynomial (NP) time in terms of the security parameter L.

Definition 1. If a TTFNN-based protocol maintains condition (C_1), we say it is feasible; otherwise, it is infeasible.

It can be deduced from Definition 1 that TTFNN-based protocols can be divided into two classes: feasible class and infeasible class.

III. FEASIBILITY ANALYSIS

Recalling that a TTFNN-based protocol is parametrized by its characteristic parameters, it is desirable to reveal how the characteristic parameters of a TTFNN-based protocol influence its feasibility. Toward that end, we will examine the dynamic process of such protocols in this section.

A. Related parameters

For the sake of describing the correlations between two TTFNNs in the synchronization process, one can look at the probability distribution of the weight values in each hidden unit. It is given by $(2L+1)\times(2L+1)$ variables,

$$p_{a,b} = P\big(w_{i,j}^A = a \wedge w_{i,j}^B = b\big), \tag{10}$$

which are defined as the probability to find a weight with $w_{i,j}^A = a$ in A's TTFNN and $w_{i,j}^B = b$ in B's TTFNN.

The standard order parameters [27], initially used for the analysis of online learning, can be calculated as functions of $p_{a,b}^i$,

$$\begin{aligned} Q_i^A &= \frac{1}{N}\mathbf{w}_i^A\mathbf{w}_i^A = \sum_{a=-L}^{L}\sum_{b=-L}^{L} a^2 p_{a,b}^i,\\ Q_i^B &= \frac{1}{N}\mathbf{w}_i^B\mathbf{w}_i^B = \sum_{a=-L}^{L}\sum_{b=-L}^{L} b^2 p_{a,b}^i, \\ R_i^{AB} &= \frac{1}{N}\mathbf{w}_i^A\mathbf{w}_i^B = \sum_{a=-L}^{L}\sum_{b=-L}^{L} ab p_{a,b}^i. \end{aligned} \tag{11}$$

The synchronization level is then represented by the normalized overlap between two corresponding hidden units [27],

$$\rho_i = \frac{\mathbf{w}_i^A\mathbf{w}_i^B}{\sqrt{\mathbf{w}_i^A\mathbf{w}_i^A}\sqrt{\mathbf{w}_i^B\mathbf{w}_i^B}} = \frac{R_i^{AB}}{\sqrt{Q_i^A Q_i^B}}, \tag{12}$$

where $0 \leqslant \rho_i \leqslant 1$, and the subscript i denotes the ith pair of corresponding hidden units. Uncorrelated weight vectors at the beginning of the synchronization process have $\rho = 0$, whereas $\rho = 1$ represents that full synchronization is reached.

Another important parameter is the well-known *generation error* ε of the perceptron. With the random input vectors, the generation error ε is the probability of the event that two corresponding hidden units have different σ. It can be computed from the overlap ρ [27,28],

$$\varepsilon = \frac{1}{\pi}\arccos(\rho). \tag{13}$$

B. Average step size

We first consider TTFNN-based protocols with the learning rule (L_1): the random-walk learning rule with sufficiently large N. Then (8) can be rewritten as

$$w_{ij}^{A/B}(t+1) = g\big[w_{ij}^{A/B}(t) + f^{A/B}x_{ij}\big], \tag{14}$$

with $f^{A/B} = \Theta(\sigma_i^{A/B}\tau_q)\prod_{i=1}^{O}\Theta(\tau_i^A\tau_i^B) \in \{0,+1\}$. In each step, three cases are possible.

Case 1: If $f^A = f^B = 1$, the weights in both corresponding hidden units are moved in the same direction. This is an *attractive step*, which increases the overlap on average. It can be described as anisotropic diffusion with reflecting boundary conditions, as shown in Eq. (A1) in the Appendix.

Case 2: If $f^A + f^B = 1$, only the weights of one hidden unit are updated. This is a *repulsive step*, which decreases the overlap on average. This step performs a normal diffusion on a $(2L+1)\times(2L+1)$ square lattice with boundary conditions, as shown in Eq. (A2) in the Appendix.

Case 3: If $f^A = f^B = 0$, the weights stay at their position.

The above discussion indicates that the TTFNN-based neural synchronization is a stochastic process consisting of attractive steps and repulsive steps. The average step size of an attractive step is denoted by $\langle\Delta\rho_{\mathrm{a}}\rangle$, while we use $\langle\Delta\rho_{\mathrm{r}}\rangle$ in the case of a repulsive step. Additionally, we denote by P_{a} and P_{r} the probability of an attractive step and a repulsive step, respectively. The *transition probabilities* P_{a} and P_{r} are functions of the generation error ε, and hence functions of ρ by Eq. (13).

We explicitly note that the dynamics of $p_{a,b}$ is a Markovian process, which can be described by the state moving equations (A1) and (A2) in the Appendix. With the help of the state moving equations, the average step sizes $\langle\Delta\rho_{\mathrm{a}}\rangle$ and $\langle\Delta\rho_{\mathrm{r}}\rangle$ at different synchronization levels can be estimated by Algorithm 2.

Notations in Algorithm 2:

$\rho[] \triangleq$ a variable length array, each element records current overlap ρ.

$\rho_{\mathrm{a}}[] \triangleq$ a variable length array, each element records current overlap ρ if an attractive step occurs.

$\Delta\rho_{\mathrm{a}} \triangleq$ an array with the identical length as ρ_{a}, where $\Delta\rho_{\mathrm{a}}[i]$ denotes $\langle\Delta\rho_{\mathrm{a}}(\rho)\rangle$ at $\rho = \rho_{\mathrm{a}}[i]$.

$\rho_{\mathrm{r}}[] \triangleq$ a variable length array, each element records current overlap ρ if a repulsive step occurs.

$\Delta\rho_{\mathrm{r}}[] \triangleq$ an array with the identical length as ρ_{r}, where $\Delta\rho_{\mathrm{r}}[i]$ denotes $\langle\Delta\rho_{\mathrm{r}}(\rho)\rangle$ at $\rho = \rho_{\mathrm{r}}[i]$.

$Current \triangleq$ counter of the current step.

$Count_{\mathrm{a}}(Count_{\mathrm{r}}) \triangleq$ counter of the attractive (repulsive) step.

$Ran \xleftarrow{\mathrm{r}} [0,1]$ denotes a real number. Ran is randomly chosen in the interval $[0,1]$.

```
Algorithm 2: Estimation of $\langle\Delta\rho_{\mathrm{a}}(\rho)\rangle$ and $\langle\Delta\rho_{\mathrm{r}}(\rho)\rangle$.
Step 1: Initialize $p_{a,b} = \frac{1}{(2L+1)^2}$ by Eq. (15). $Current = Count_{\mathrm{a}} = Count_{\mathrm{r}} = 1$, apply $p_{a,b}$ to compute $\rho[Current]$ by Eqs. (11) and (12).
Step 2: while ($\rho[Current] < 0.99$) do
Step 3: $Ran \xleftarrow{\mathrm{r}} [0,1]$.
Step 4: if [$Ran < P_{\mathrm{a}}(\rho[Current])$] then
```

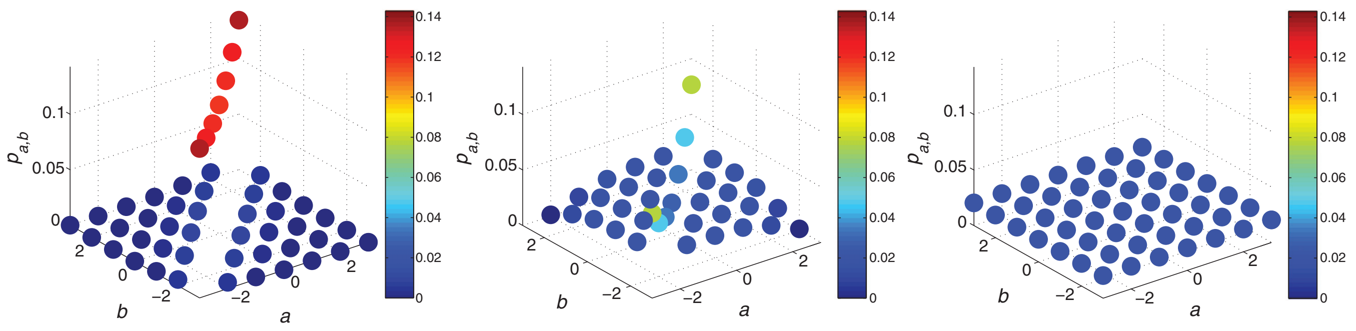

FIG. 2. (Color online) Dynamics of $p_{a,b}$ for TTFNN-based protocols with $L = 3$: (a) At the beginning ($\rho = 0$), $p_{a,b}$ is given by Eq. (15). (b) In the middle ($\rho = 0.57$). (c) Toward full synchronization ($\rho = 0.97$), $p_{a,b}$ is close to the distribution in Eq. (16).

Step 5: Update $p_{a,b} = M_\mathrm{a}(p_{a,b})$, where M_a is given by the moving functions in Eq. (A1).

Step 6: $\rho_\mathrm{a}[Count_\mathrm{a}] = \rho[Current]$.

Step 7: Apply $p_{a,b}$ to compute $\rho[Current + 1]$ by Eqs. (11) and (12).

Step 8: $\Delta\rho_\mathrm{a}[Count_\mathrm{a}] = \rho[Current + 1] - \rho[Current]$.

Step 9: $Count_\mathrm{a} = Count_\mathrm{a} + 1$.

Step 10: **else if** $[Ran > 1 - P_\mathrm{r}(\rho[Current])]$ **then**

Step 11: Update $p_{a,b} = M_\mathrm{r}(p_{a,b})$, where M_r is given by the moving functions in Eq. (A2).

Step 12: $\rho_\mathrm{r}[Count_\mathrm{r}] = \rho[Current]$.

Step 13: Apply $p_{a,b}$ to compute $\rho[Current + 1]$ by Eqs. (11) and (12).

Step 14: $\Delta\rho_\mathrm{r}[Count_\mathrm{r}] = \rho[Current + 1] - \rho[Current]$.

Step 15: $Count_\mathrm{r} = Count_\mathrm{r} + 1$.

Step 16: **end if**

Step 17: $Current = Current + 1$.

Step 18: **end while**

Step 19: **Outputs:** $(\rho_\mathrm{a}, \Delta\rho_\mathrm{a})$, $(\rho_\mathrm{r}, \Delta\rho_\mathrm{r})$. Several discrete values of $\langle\Delta\rho_\mathrm{a}(\rho)\rangle$, $\langle\Delta\rho_\mathrm{r}(\rho)\rangle$ are obtained. By interpolation, we can get the estimation of $\langle\Delta\rho_\mathrm{a}(\rho)\rangle$, $\langle\Delta\rho_\mathrm{r}(\rho)\rangle$.

End Algorithm 2

In Algorithm 2, the dynamics of $p_{a,b}$ can be obtained by recording $p_{a,b}$ in each update step. It is observed that the dynamics of $p_{a,b}$ share a similar moving tendency in the learning process, as graphically illustrated in Fig. 2, which helps us to gain insight into the dynamics of weights.

(i) At the beginning of the synchronization, the weights are randomly chosen and uncorrelated,

$$p_{a,b}(\rho = 0) = \frac{1}{(2L+1)^2}. \tag{15}$$

A sufficiently large N ensures that (15) can better match the frequencies of weights in reality. Substituting (15) into Eqs. (11) and (12), it is then easily checked that $\rho = 0$.

(ii) In the middle of the learning process, the distribution of $p_{a,b}$ is shown in Fig. 2(b).

(iii) When full synchronization is reached,

$$p_{a,b}(\rho = 1) = \begin{cases} \frac{1}{2L+1} & \text{for} \quad a = b, \\ 0 & \text{for} \quad a \neq b. \end{cases} \tag{16}$$

Putting (16) into Eqs. (11) and (12) yields $\rho = 1$.

In Algorithm 2, the estimation of $\langle\Delta\rho_\mathrm{a}(\rho)\rangle$ and $\langle\Delta\rho_\mathrm{r}(\rho)\rangle$ highly depends on the fact that the overlap ρ can be computed from the knowledge of $p_{a,b}$. The inputs of Algorithm 2 involve the transition probabilities $P_\mathrm{a}(\rho)$ and $P_\mathrm{r}(\rho)$, which are case-specific for different TTFNN-based protocols. However, through simulations using Algorithm 2, the following is observed:

Property 1. For any TTFNN-based protocol with identical L, $\langle\Delta\rho_\mathrm{a}\rangle$ and $\langle\Delta\rho_\mathrm{r}\rangle$ are generally identical as a function of ρ itself and independent of the learning rule, as graphically depicted in Fig. 3.

The experimental confirmation of Property 1 can be found in the Appendix.

C. Feasible condition

For a given L, $\langle\Delta\rho_\mathrm{a}\rangle$, $\langle\Delta\rho_\mathrm{r}\rangle$, P_a, and P_r are functions of ρ. Accordingly, the average change of overlap $\langle\Delta\rho\rangle$ can be computed by

$$\langle\Delta\rho(\rho)\rangle = P_\mathrm{a}(\rho)\langle\Delta\rho_\mathrm{a}(\rho)\rangle + P_\mathrm{r}(\rho)\langle\Delta\rho_\mathrm{r}(\rho)\rangle, \quad 0 \leqslant \rho \leqslant 1. \tag{17}$$

The quantity $\langle\Delta\rho(\rho)\rangle$ plays an important role in depicting the dynamics of overlap in the synchronization process, which takes the total effect of both types of steps into consideration.

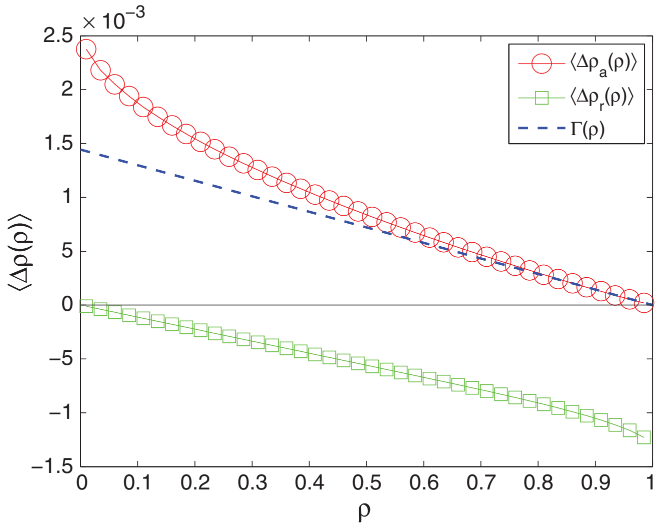

FIG. 3. (Color online) $\langle\Delta\rho_\mathrm{a}(\rho)\rangle$ and $\langle\Delta\rho_\mathrm{r}(\rho)\rangle$ for TTFNN-based protocols, $L = 10$. The linear fitting equation $\Gamma(\rho)$ is given by Eq. (25).

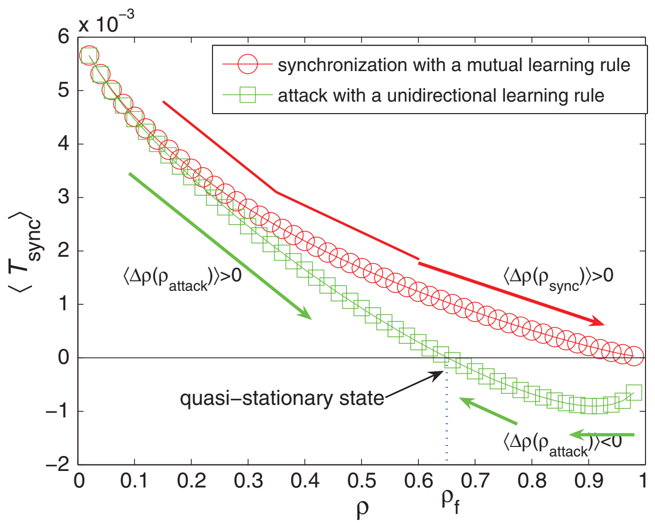

FIG. 4. (Color online) Two typical types of dynamics of $\langle\Delta\rho(\rho)\rangle$ for TTFNN-based protocols. The arrow lines display the moving directions on average.

Let us concentrate on two typical types of dynamics of $\langle\Delta\rho(\rho)\rangle$ for TTFNN-based protocols, as shown in Fig. 4.

(i) With regard to synchronization with a mutual learning rule, $\langle\Delta\rho(\rho_{\rm sync})\rangle > 0$, $0 \leqslant \rho < 1$, there is only one absorbing state at $\rho = 1$; the overlap $\rho_{\rm sync}$ keeps increasing on average in the whole process of synchronization.

(ii) With regard to attack with a unidirectional learning rule, there is a fixed point of dynamics at $0 < \rho_{\rm f} < 1$, such that

$$\langle\Delta\rho(\rho_{\rm attack})\rangle \begin{cases} >0, & 0 < \rho_{\rm attack} < \rho_{\rm f}, \\ =0, & \rho_{\rm attack} = \rho_{\rm f}, \\ <0, & \rho_{\rm f} < \rho_{\rm attack} < 1. \end{cases} \quad (18)$$

As long as $0 < \rho_{\rm attack} < \rho_{\rm f}$, $\langle\Delta\rho(\rho_{\rm attack})\rangle > 0$, the overlap ρ keeps increasing on average. But when $\rho_{\rm f} < \rho_{\rm attack} < 1$, $\Delta\rho(\rho_{\rm attack})\rangle < 0$, the overlap ρ keeps decreasing on average. The point $\rho_{\rm f}$ is a *quasistationary* state. When $\rho_{\rm attack}$ reaches the quasistationary state, $\rho_{\rm attack}$ displays an oscillatory behavior around the point $\rho_{\rm f}$. A further absorbing state $\rho = 1$ can only be reached by fluctuation. With regard to $\rho_{\rm attack}$, it is explicitly observed that time is wasted in reaching $\rho = 1$ if $\rho_{\rm attack}$ oscillates around $\rho_{\rm f}$.

Property 2. For TTFNN-based protocols, if the overlap ρ keeps increasing on average in the whole process of synchronization, the average synchronization time $\langle T_{\rm sync}\rangle$ grows polynomially with L, whereas $\langle T_{\rm sync}\rangle$ for fluctuation grows exponentially with L.

As a special case of TTFNN-based protocols, the TPM ($K = 3$)-based protocol maintains this property [15]. Based on Property 2, a feasible condition for a TTFNN-based protocol is

$$\langle\Delta\rho(\rho)\rangle > 0, \quad 0 \leqslant \rho < 1. \quad (19)$$

By a feasible condition, we mean that if a TTFNN-based protocol maintains (19), it is feasible; otherwise, it is infeasible.

We now consider $\langle\Delta\rho(\rho)\rangle$ for TTFNN-based protocols with the learning rule (L_2): Hebbian or anti-Hebbian learning rules with $N \to +\infty$.

Property 3. For any TTFNN-based protocol, $\langle\Delta\rho(\rho)\rangle$ of Hebbian and anti-Hebbian learning rules converges to that of their corresponding random-walk learning rule in the limit $N \to +\infty$.

Readers may refer to the Appendix for a detailed analysis.

Due to Property 3 and (19), it can be deduced that the feasibility of a TTFNN-based protocol with learning rule (L_1) and its corresponding learning rule (L_2) is consistent. Consequently, we will only consider TTFNN-based protocols with learning rule (L_1) in the rest of this paper.

D. Further analysis

Substituting (17) into Eq. (19), and applying ε to replace ρ according to Eq. (13), the feasible condition (19) can be rewritten as

$$R(\varepsilon) < U(\varepsilon), \quad 0 < \varepsilon \leqslant 0.5, \quad (20)$$

where

$$R(\varepsilon) = \frac{P_{\rm r}(\varepsilon)}{P_{\rm a}(\varepsilon)} \quad (21)$$

and

$$U(\varepsilon) = -\frac{\langle\Delta\rho_{\rm a}(\varepsilon)\rangle}{\langle\Delta\rho_{\rm r}(\varepsilon)\rangle}. \quad (22)$$

Figure 3 shows that when it is closed to full synchronization, an attractive step size is approximated to 0, whereas a repulsive step size reaches its maximum effect,

$$\langle\Delta\rho_{\rm r}(\rho = 1)\rangle = -\frac{3}{(L+1)(2L+1)}. \quad (23)$$

The derivation of Eq. (23) is similar to that of the TPM ($K = 3$) -based protocol [15].

With the goal of maintaining feasible condition (20), $P_{\rm a}(\varepsilon)$ should be dominant in comparison with $P_{\rm r}(\varepsilon)$ when $\varepsilon \to 0^+$. Therefore, we can expect and then observe in simulations that $R(\varepsilon)$ meets its strictest requirement when $\varepsilon \to 0^+$. This can be formally stated as follows:

Property 4. If $R(\varepsilon)$ of a TTFNN-based protocol satisfies

$$R(\varepsilon) < U(\varepsilon) \text{ in the limit } \varepsilon \to 0^+, \quad (24)$$

then it satisfies the feasible condition (20).

When it is closed to full synchronization, $\langle\Delta\rho_{\rm a}(\rho)\rangle$ can be approximated as a straight line $\Gamma(\rho)$, as displayed in Fig. 3,

$$\begin{aligned}\langle\Delta\rho_{\rm a}(\rho)\rangle &\approx \Gamma(\rho) \\ &= \frac{-7L}{(L+1)(2L+1)^2}(\rho - 1), \quad 0.8 \leqslant \rho \leqslant 1. \end{aligned} \quad (25)$$

The following equality always holds:

$$\lim_{\varepsilon\to 0^+} U(\varepsilon) = -\frac{\lim_{\rho\to 1^-}\langle\Delta\rho_{\rm a}(\rho)\rangle}{\langle\Delta\rho_{\rm r}(\rho = 1)\rangle}. \quad (26)$$

Ignoring the fitting error in Eq. (25) and combining (23) and (26), then we have

$$U(\varepsilon) \sim I(\varepsilon)(\varepsilon \to 0^+), \quad (27)$$

where $I(\varepsilon) = \frac{7L}{12L+6}\pi^2\varepsilon^2$. For the purpose of capturing the scaling law of $\langle T_{\rm sync}\rangle$ with large L, we have $I(\varepsilon) = \frac{7}{12}\pi^2\varepsilon^2$ for sufficiently large L.

To maintain (24), $R(\varepsilon)$ needs to satisfy $R(\varepsilon) \sim a\varepsilon^2$, $a < \frac{7}{12}\pi^2 (\varepsilon \to 0^+)$, or $R(\varepsilon) = o(\frac{7}{12}\pi^2\varepsilon^2)$ [$R(\varepsilon)$ is a higher-order infinitesimal quantity of $\frac{7}{12}\pi^2\varepsilon^2$]. This can be described by a single inequality,

$$C < 1, \tag{28}$$

where

$$C = \lim_{\varepsilon \to 0^+} \frac{R(\varepsilon)}{\frac{7}{12}\pi^2\varepsilon^2}. \tag{29}$$

According to Property 4, feasible conditions (19), (20), (24), and (28) are equivalent to each other, among which (28) will be most frequently used in the following.

IV. APPLICATION OF FEASIBLE CONDITION

In this section, we will exemplify how to apply the feasible condition to search for additional feasible TTFNN-based protocols. The search methodology is graphically depicted in Fig. 5.

This methodology involves computing the transition probabilities,

$$\begin{aligned} P_{\rm u}(\varepsilon) &= P\left[\bigwedge_{i=1}^{O}\left(\tau_i^A = \tau_i^B\right)\right], \\ P_{\rm a}(\varepsilon) &= P\left[\sigma_i^A = \sigma_i^B = \tau_q \,\middle|\, \bigwedge_{i=1}^{O}\left(\tau_i^A = \tau_i^B\right)\right], \\ P_{\rm r}(\varepsilon) &= P\left[\sigma_i^A \neq \sigma_i^B \,\middle|\, \bigwedge_{i=1}^{O}\left(\tau_i^A = \tau_i^B\right)\right]. \end{aligned} \tag{30}$$

We denote by $P_{\rm u}$, $P_{\rm a}$, and $P_{\rm r}$ the probability of an update step, an attractive step, and a repulsive step, respectively. These transition probabilities can be computed from the knowledge of the generation error ε. An example of how to compute these transition probabilities is displayed in the Appendix. See [29] for further details.

A. Case of success

Case (i): For the conventional TPM-based protocol: $K = 3$, $O = 1$, F_1 is given by Eq. (4), and the random-walk learning rule can be described as

$$w_{ij}^{A/B}(t+1) = g\left[w_{ij}^{A/B}(t) + x_{ij}\Theta\left(\sigma_i^{A/B}\tau\right)\Theta\left(\tau_{\rm TPM}^A\tau_{\rm TPM}^B\right)\right]. \tag{31}$$

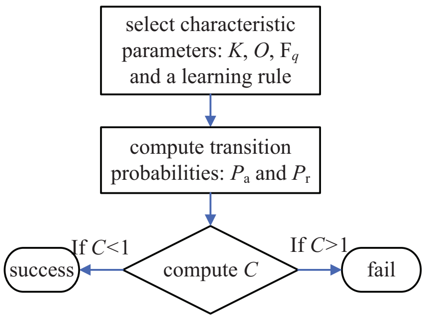
select characteristic parameters: K, O, F_q and a learning rule
compute transition probabilities: P_a and P_r
compute C
If C<1
success
If C>1
fail

FIG. 5. (Color online) The methodology for seeking additional feasible protocols.

One finds [15,21]

$$\begin{aligned} P_{\rm u}^{{\rm TPM}(K=3)}(\varepsilon) &= (1-\varepsilon)^3 + 3(1-\varepsilon)\varepsilon^2, \\ P_{\rm a}^{{\rm TPM}(K=3)}(\varepsilon) &= \frac{\frac{1}{2}(1-\varepsilon)^3 + \frac{1}{2}(1-\varepsilon)\varepsilon^2}{P_{\rm u}^{{\rm TPM}(K=3)}(\varepsilon)}, \\ P_{\rm r}^{{\rm TPM}(K=3)}(\varepsilon) &= \frac{2(1-\varepsilon)\varepsilon^2}{P_{\rm u}^{{\rm TPM}(K=3)}(\varepsilon)}, \\ R^{{\rm TPM}(K=3)}(\varepsilon) &= \frac{4(1-\varepsilon)\varepsilon^2}{(1-\varepsilon)^3 + (1-\varepsilon)\varepsilon^2}. \end{aligned} \tag{32}$$

It follows that $C^{{\rm TPM}(K=3)} = \frac{48}{7\pi^2} < 1$; the feasible condition (28) indicates it is feasible.

Case (ii): We now attempt to extend the number of outputs. A straightforward method is to let $O = 2$. F_1 uses the same function as the TPM network in Eq. (4), and F_2 uses the same function as the TCM network in Eq. (5). There are two different random learning rules for the above construction,

$$\begin{aligned} w_{ij}(t+1) = g\big[&w_{ij}(t) + x_{ij}\Theta\left(\sigma_i\tau_{\rm TPM}^{A/B}\right) \\ &\times\Theta\left(\tau_{\rm TPM}^A\tau_{\rm TPM}^B\right)\Theta\left(\tau_{\rm TCM}^A\tau_{\rm TCM}^B\right)\big], \quad q = 1, \end{aligned} \tag{33}$$

$$\begin{aligned} w_{ij}(t+1) = g\big[&w_{ij}(t) + x_{ij}\Theta\left(\sigma_i\tau_{\rm TCM}^{A/B}\right) \\ &\times\Theta\left(\tau_{\rm TPM}^A\tau_{\rm TPM}^B\right)\Theta\left(\tau_{\rm TCM}^A\tau_{\rm TCM}^B\right)\big], \quad q = 2. \end{aligned} \tag{34}$$

We refer here to neural protocols using the learning rules (33) and (34) as tree parity committee machine (TPCM) -based protocols and tree committee parity machine (TCPM) -based protocols, respectively.

With regard to the TPCM-based protocol: $K = 3$, $O = 2$, F_1 is given by Eq. (4), F_2 is given by Eq. (5), and the random-walk learning rule is described by Eq. (33). It is obtained that

$$\begin{aligned} P_{\rm u}^{{\rm TPCM}(K=3)}(\varepsilon) &= (1-\varepsilon)^3 + \frac{3}{2}(1-\varepsilon)\varepsilon^2, \\ P_{\rm a}^{{\rm TPCM}(K=3)}(\varepsilon) &= \frac{\frac{1}{2}(1-\varepsilon)^3}{P_{\rm u}^{{\rm TPCM}(K=3)}(\varepsilon)}, \\ P_{\rm r}^{{\rm TPCM}(K=3)}(\varepsilon) &= \frac{(1-\varepsilon)\varepsilon^2}{P_{\rm u}^{{\rm TPCM}(K=3)}(\varepsilon)}, \\ R^{{\rm TPCM}(K=3)}(\varepsilon) &= \frac{2(1-\varepsilon)\varepsilon^2}{(1-\varepsilon)^3}. \end{aligned} \tag{35}$$

Since $C^{{\rm TPCM}(K=3)} = \frac{24}{7\pi^2} < 1$, it belongs to the feasible class by Eq. (28).

Case (iii): With regard to the TCPM-based protocol: $K = 3$, $O = 2$, F_1 is given by Eq. (4), F_2 is given by Eq. (5), and the random-walk learning rule is described by Eq. (34). We can obtain

$$\begin{aligned} P_{\rm u}^{{\rm TCPM}(K=3)}(\varepsilon) &= (1-\varepsilon)^3 + \frac{3}{2}(1-\varepsilon)\varepsilon^2, \\ P_{\rm a}^{{\rm TCPM}(K=3)}(\varepsilon) &= \frac{\frac{3}{4}(1-\varepsilon)^3 + \frac{1}{2}(1-\varepsilon)\varepsilon^2}{P_{\rm u}^{{\rm TCPM}(K=3)}(\varepsilon)}, \\ P_{\rm r}^{{\rm TCPM}(K=3)}(\varepsilon) &= \frac{(1-\varepsilon)\varepsilon^2}{P_{\rm u}^{{\rm TCPM}(K=3)}(\varepsilon)}, \\ R^{{\rm TCPM}(K=3)}(\varepsilon) &= \frac{(1-\varepsilon)\varepsilon^2}{\frac{3}{4}(1-\varepsilon)^3 + \frac{1}{2}(1-\varepsilon)\varepsilon^2}. \end{aligned} \tag{36}$$

Because $C^{\mathrm{TCPM}(K=3)} = \frac{16}{7\pi^2} < 1$, from Eq. (28), this one appears to be feasible.

B. Case of failure

The TCM ($K = 3$) -based neural protocol is analyzed to be infeasible [29]. This result can be clearly verified by our methodology. As to the TCM-based protocol: $K = 3$, $O = 1$, F_1 is given by Eq. (5), and the random-walk learning rule is $w_{ij}^{A/B}(t+1) = g[w_{ij}^{A/B}(t) + x_{ij}\Theta(\sigma_i^{A/B}\tau)\Theta(\tau_{\mathrm{TCM}}^A\tau_{\mathrm{TCM}}^B)]$. One finds [29]

$$\begin{aligned}
P_{\mathrm{u}}^{\mathrm{TCM}(K=3)}(\varepsilon) &= (1-\varepsilon)^3 + \frac{3}{2}(1-\varepsilon)^2\varepsilon + \frac{3}{2}(1-\varepsilon)\varepsilon^2,\\
P_{\mathrm{a}}^{\mathrm{TCM}(K=3)}(\varepsilon) &= \frac{\frac{3}{4}(1-\varepsilon)^3 + \frac{1}{2}(1-\varepsilon)\varepsilon^2 + (1-\varepsilon)^2\varepsilon}{P_{\mathrm{u}}^{\mathrm{TCM}(K=3)}(\varepsilon)},\\
P_{\mathrm{r}}^{\mathrm{TCM}(K=3)}(\varepsilon) &= \frac{(1-\varepsilon)\varepsilon^2 + \frac{1}{2}(1-\varepsilon)^2\varepsilon}{P_{\mathrm{u}}^{\mathrm{TCM}(K=3)}(\varepsilon)},\\
R^{\mathrm{TCM}(K=3)}(\varepsilon) &= \frac{(1-\varepsilon)\varepsilon^2 + \frac{1}{2}(1-\varepsilon)^2\varepsilon}{\frac{3}{4}(1-\varepsilon)^3 + \frac{1}{2}(1-\varepsilon)\varepsilon^2 + (1-\varepsilon)^2\varepsilon}.
\end{aligned} \tag{37}$$

Due to $C^{\mathrm{TCM}(K=3)} = +\infty > 1$, this protocol unfortunately falls into the infeasible class by Eq. (28).

V. HOW TO DESIGN FEASIBLE PROTOCOLS

Our approach to search for additional feasible protocols is based on the "trial and verification" methodology. This methodology, however, does not instruct us how to design a feasible protocol. Consequently, we proceed to study how to design feasible protocols.

A. Heuristic rules

Let us consider a TTFNN-based protocol with K hidden units and O outputs. According to the values of $(\tau_1,\ldots,\tau_O)$, with $\tau_i = F_i(\Sigma)$, 2^K configurations of internal representation Σ can be divided into 2^O classes.

Definition 2. For any internal representation Σ, we define Σ as belonging to class number $(c_1\cdots c_O)_2$, where subscript 2 denotes binary representation of a number, and

$$c_i = \begin{cases} 1 & \text{if} \quad \tau_i = F_i(\Sigma) = 1,\\ 0 & \text{if} \quad \tau_i = F_i(\Sigma) = -1. \end{cases} \tag{38}$$

We note from Definition 2 that $0 \leqslant (c_1\cdots c_O)_2 \leqslant 2^O - 1$. The following descriptions will make it clear.

(i) 2^K configurations of Σ are divided into 2^O classes.

(ii) Each class contains zero or several configurations, which are arranged by a certain sequence.

(iii) Each configuration of Σ consists of K items.

Let $v_{ijl} \in \{-1,+1\}$ stand for the value of the class number i, the jth configuration, and the lth item. We define

$$h_{ip,q} = \sum_{l=1}^{K}(v_{ipl} \oplus v_{iql}), \tag{39}$$

where $h_{ip,q}$ denotes the Hamming distance between the pth internal representation and the qth internal representation in class number i. In particular, if $p = q$, $h_{ip,q} = 0$. Suppose that there are totally N_i different configurations for class number i. Then h_i and H are defined as

$$h_i = \begin{cases} +\infty, & N_i = 0,1,\\ \min_{1\leqslant p,q\leqslant N_i, p\neq q}\{h_{ip,q}\}, & N_i \geqslant 2, \end{cases} \tag{40}$$

$$H = \min_{0\leqslant i\leqslant 2^O-1}\{h_i\}. \tag{41}$$

The variable H, indeed, plays an important role in computing the transition probabilities, which can be explained by the following analysis.

Under the precondition that an update step occurs, let $P_{\mathrm{r}i}(\varepsilon)$ denote the probability of a repulsive step if there are i pairs of corresponding hidden units disagreeing. Recalling the definition of P_{u} in Eq. (30), one can deduce that an update step occurs iff Σ^A and Σ^B belong to an identical class. Note that if $0 \leqslant i \leqslant H-1$, an update step cannot occur. This finally results in

$$P_{\mathrm{r}i}(\varepsilon) = 0, \quad 0 \leqslant i \leqslant H-1. \tag{42}$$

But when $H \leqslant i \leqslant K$, the following equality always holds:

$$P_{\mathrm{r}i}(\varepsilon) = \frac{d_i(1-\varepsilon)^{K-i}\varepsilon^i}{P_{\mathrm{u}}(\varepsilon)}, \quad H \leqslant i \leqslant K, \tag{43}$$

where $d_i \geqslant 0$. It is not hard to find that

$$\lim_{\varepsilon\to 0^+} P_{\mathrm{u}}(\varepsilon) = 1. \tag{44}$$

Suppose that $d_H > 0$. Apply (42)–(44) and, together with $P_{\mathrm{r}}(\varepsilon) = \sum_{i=0}^{K} P_{\mathrm{r}i}(\varepsilon)$, we can obtain

$$P_{\mathrm{r}}(\varepsilon) \sim d_H\varepsilon^H (\varepsilon \to 0^+). \tag{45}$$

Further, suppose that $\lim_{\varepsilon\to 0^+} P_{\mathrm{a}}(\varepsilon) = e > 0$. This leads to

$$R(\varepsilon) = \frac{P_{\mathrm{r}}(\varepsilon)}{P_{\mathrm{a}}(\varepsilon)} \sim \frac{d_H}{e}\varepsilon^H (\varepsilon \to 0^+). \tag{46}$$

We proceed to consider the choice of H:

Case (i) $H = 1$ implies that (28) cannot be maintained.

Case (ii) $H = 2$ indicates that (28) may be maintained, depending on the coefficient $\frac{d_H}{e}$.

Case (iii) $H \geqslant 3$, and then (28) is maintained immediately. Accordingly, two heuristic rules to design feasible TTFNN-based protocols are as follows:

(R_1) For the constructed protocol, it is preferable to let $H \geqslant 3$. If we let $H = 2$, the feasibility should be further checked by applying (28).

(R_2) Meanwhile, it should be ensured that $d_H > 0$ and $\lim_{\varepsilon\to 0^+} P_{\mathrm{a}}(\varepsilon) = e > 0$.

B. Applications

1. Design a feasible protocol

Following the instructions of the two rules, we design two protocols: $K = 3$, $O = 2$, F_1 and F_2 are given by Table I, and the learning rules are

$$\begin{aligned} w_{ij}^{A/B}(t+1) = g\big[&w_{ij}^{A/B}(t) + x_{ij}\Theta\big(\sigma_i^{A/B}\tau_q\big)\\ &\times\Theta\big(\tau_1^A\tau_1^B\big)\Theta\big(\tau_2^A\tau_2^B\big)\big], \quad q = 1,2. \end{aligned} \tag{47}$$

In Classification I, $h_i = 3$; for $0 \leqslant i \leqslant 3$, $H = 3$. This implies that rule (R_1) is maintained. It can be easily found that rule (R_2) is also maintained. We refer to classification I

TABLE I. Classification I.

$\Sigma^{A/B}$	$\tau_1 = F_1(\Sigma^{A/B})$	$\tau_2 = F_2(\Sigma^{A/B})$	Class number $(c_1c_2)_2$
(−1,−1,−1) (+1,+1,+1)	−1	−1	$(00)_2$
(−1,−1,+1) (+1,+1,−1)	−1	+1	$(01)_2$
(−1,+1,−1) (+1,−1,+1)	+1	−1	$(10)_2$
(+1,−1,−1) (−1,+1,+1)	+1	+1	$(11)_2$

with learning rule (47), $q = 1$ as the CI ($q = 1$) neural protocol, and (47), $q = 2$ as the CI ($q = 2$) neural protocol. Based on our heuristic rules, they are believed to be feasible. Their feasibilities are further checked below. For CI ($q = 1$) and CI ($q = 2$) neural protocols, we obtain

$$\begin{aligned} P_{\mathrm{u}}^{\mathrm{CI}(q=1,2)}(\varepsilon) &= (1-\varepsilon)^3 + \varepsilon^3, \\ P_{\mathrm{a}}^{\mathrm{CI}(q=1,2)}(\varepsilon) &= \frac{\frac{1}{2}(1-\varepsilon)^3}{P_{\mathrm{u}}^{\mathrm{CI}(q=1,2)}(\varepsilon)}, \\ P_{\mathrm{r}}^{\mathrm{CI}(q=1,2)}(\varepsilon) &= \frac{\varepsilon^3}{P_{\mathrm{u}}^{\mathrm{CI}(q=1,2)}(\varepsilon)}, \\ R^{\mathrm{CI}(q=1,2)}(\varepsilon) &= \frac{2\varepsilon^3}{(1-\varepsilon)^3}. \end{aligned} \tag{48}$$

By Eq. (28) again, $C^{\mathrm{CI}(q=1,2)} = 0 < 1$, they appear to be feasible, in agreement with our expectation. Because of symmetry, the above two protocols can be viewed as one.

More feasible instances will not be enumerated. We remark, however, that the two heuristic rules allow us to design a variety of feasible protocols, even with $K \geqslant 4$ and $O \geqslant 3$. Readers can gain a profound understanding of the two rules if they try to design a feasible protocol by themselves.

2. *Feasibility from a classification perspective*

The feasibility of TTFNN-based protocols can be explained from the perspective of classification. Applying the classification of the TCM ($K = 3$) -based protocol as an example, it is clearly illustrated in Table II that $h_0 = h_1 = 1$, $H = 1$. This results in $R(\varepsilon) \sim f\varepsilon(\varepsilon \to 0^+)$, with $f > 0$. Accordingly, it turns out to be infeasible immediately. In the same manner, from the classification of TPM ($K = 3$), TPCM ($K = 3$), and

TABLE II. Classification TCM ($K = 3$).

$\Sigma^{A/B}$	$\tau_{\mathrm{TCM}} = F_1(\Sigma^{A/B})$	Class number $(c_1)_2$
(−1,−1,−1) (−1,−1,+1) (−1,+1,−1) (+1,−1,−1)	−1	$(0)_2$
(−1,+1,+1) (+1,−1,+1) (+1,+1,−1) (+1,+1,+1)	+1	$(1)_2$

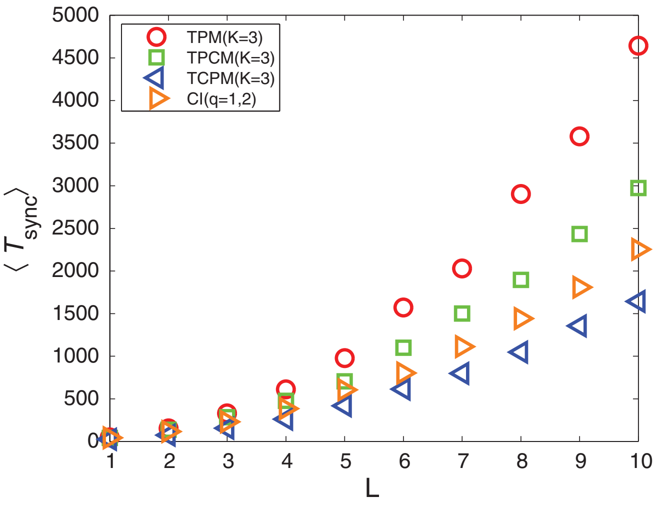

FIG. 6. (Color online) The average synchronization time $\langle T_{\mathrm{sync}}\rangle$ as a function of L for different protocols. Symbols denote averages over 1000 simulations with learning rule (L_1) using $N = 100$.

TCPM ($K = 3$), it can be found that $H = 2$. Therefore, their feasibilities should be further checked by applying (28).

VI. SYNCHRONIZATION SPEED AND SECURITY

On the basis of our results, a question arises naturally: is there any TTFNN-based protocol that is superior to the conventional TPM-based protocol with regard to synchronization speed and security? For speed considerations, it can be observed from Fig. 6 that three additional available protocols considerably accelerate the synchronization speed. For security considerations, a secure protocol should further maintain Condition (C_2). It is easy to observe that some instances in the TTFNN model cannot satisfy this condition. For example, the trivial solution in which a neural protocol with each configuration is a class is a feasible protocol since no repulsive steps will occur. However, in this case, the attacker E knows all the internal representations of partners, which leads to this protocol being insecure. In the TTFNN model, there are many instances that are more secure than the TPM-based protocol. Because the majority attack and the genetic attack apply a geometric attack or a simple attack as an element and because the geometric attack cannot directly work on neural protocols with multiple bits, we only consider the simple attack in this paper. It is shown in Table III that the TCPM and the CI ($q = 1,2$) protocol are better at resisting a simple attack. In Table III, following [30], success probability P_{E} is defined as

TABLE III. Success probability P_{E} of a simple attack as a function of L. Results are obtained in 10 000 simulations with learning rule (L_1) using $N = 1000$.

	$L = 1$	$L = 2$	$L = 3$
TPM	$P_{\mathrm{E}} = 0.4390$	$P_{\mathrm{E}} = 0.0320$	$P_{\mathrm{E}} = 0.0005$
TPCM	$P_{\mathrm{E}} = 0.5180$	$P_{\mathrm{E}} = 0.0843$	$P_{\mathrm{E}} = 0.0008$
TCPM	$P_{\mathrm{E}} = 0.3240$	$P_{\mathrm{E}} = 0.0129$	$P_{\mathrm{E}} = 0.0002$
CI ($q = 1,2$)	$P_{\mathrm{E}} = 0.3176$	$P_{\mathrm{E}} = 0.0073$	$P_{\mathrm{E}} = 0.0000$

the probability that the attacker knows 98% of the weights at synchronization time.

Based on our simulations, we come to the conclusion that the TCMP and the CI ($q = 1,2$) protocol can outperform the TPM-based protocol in terms of both synchronization speed and security. It is worth noting that despite the fact that the original geometric attack currently cannot be used for neural networks with multiple outputs, a generalization of the geometric attack may be possible, and we leave this as a future research issue. Note that we can design many different neural protocols in the TTFNN model. Finding the best one is another interesting future research issue.

VII. CONCLUSION

This paper has formulated and studied the TTFNN model for neural cryptography. A completed methodology for designing a TTFNN-based protocol satisfying Condition (C_1) has been elaborated on, based on two analytically provided heuristic rules. Several TTFNN instances that can outperform the TPM-based protocol were presented. Simulations were also conducted to validate our results. Our model in this paper has shown that further development of neural cryptography is indeed possible.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (No. 60973114 and No. 61170249), the Natural Science Foundation project of CQCSTC (2009BA2024), in part by the State Key Laboratory of Power Transmission Equipment & System Security and New Technology, Chongqing University (2007DA10512711206), in part by the program for Changjiang scholars, and supported by Specialized Research Fund for priority areas for the Doctoral Program of Higher Education. This work was also supported by NPRP Grant No. 4-1162-1-181 and Grant No. 4-451-2-168 from the Qatar National Research Fund (a member of the Qatar Foundation).

APPENDIX

1. State moving equations

These state moving equations are consistent with those of the TPM ($K = 3$) -based protocol [15]. They are presented here to keep this paper self-contained.

a. Attractive steps

In an attractive step, corresponding weights move in the same direction. The weights move according to the following state moving equations, where $-L < a,b < L$:

$$\begin{aligned}
p^+_{a,b} &= \tfrac{1}{2}(p_{a+1,b+1} + p_{a-1,b-1}),\\
p^+_{a,L} &= \tfrac{1}{2}(p_{a-1,L} + p_{a-1,L-1}),\\
p^+_{a,-L} &= \tfrac{1}{2}(p_{a+1,-L} + p_{a+1,-L+1}),\\
p^+_{L,b} &= \tfrac{1}{2}(p_{L,b-1} + p_{L-1,b-1}),\\
p^+_{-L,b} &= \tfrac{1}{2}(p_{-L,b+1} + p_{-L+1,b+1}),\\
p^+_{L,L} &= \tfrac{1}{2}(p_{L-1,L-1} + p_{L-1,L} + p_{L,L-1} + p_{L,L}),\\
p^+_{-L,-L} &= \tfrac{1}{2}(p_{-L+1,-L+1} + p_{-L+1,-L} + p_{-L,-L+1} + p_{-L,-L}),\\
p^+_{L,-L} &= 0,\\
p^+_{-L,L} &= 0.
\end{aligned} \tag{A1}$$

b. Repulsive steps

In a repulsive step, only the weights in one hidden unit move, either A's or B's TTFNN. The weights move according to the following state moving equations, where $-L < a,b < L$:

$$\begin{aligned}
p^+_{a,b} &= \tfrac{1}{4}(p_{a+1,b} + p_{a-1,b} + p_{a,b+1} + p_{a,b-1}),\\
p^+_{a,L} &= \tfrac{1}{4}(p_{a+1,L} + p_{a-1,L} + p_{a,L} + p_{a,L-1}),\\
p^+_{a,-L} &= \tfrac{1}{4}(p_{a+1,-L} + p_{a-1,-L} + p_{a,-L+1} + p_{a,-L}),\\
p^+_{L,b} &= \tfrac{1}{4}(p_{L,b} + p_{L-1,b} + p_{L,b+1} + p_{L,b-1}),\\
p^+_{-L,b} &= \tfrac{1}{4}(p_{-L+1,b} + p_{-L,b} + p_{-L,b+1} + p_{-L,b-1}),\\
p^+_{L,L} &= \tfrac{1}{4}\big(2p_{L,L} + p^i_{L-1,-L} + p_{L,L-1}\big),\\
p^+_{-L,-L} &= \tfrac{1}{4}\big(2p_{-L,-L} + p^i_{-L+1,-L} + p_{-L,-L+1}\big),\\
p^+_{L,-L} &= \tfrac{1}{4}\big(2p_{L,-L} + p^i_{L-1,-L} + p_{L,-L+1}\big),\\
p^+_{-L,L} &= \tfrac{1}{4}\big(2p_{-L,L} + p^i_{-L+1,L} + p_{-L,L-1}\big).
\end{aligned} \tag{A2}$$

2. Experimental results for Property 1

Figure 7 illustrates that $\langle\Delta\rho_{\mathrm{a}}(\rho)\rangle$ and $\langle\Delta\rho_{\mathrm{r}}(\rho)\rangle$ are generally identical for different feasible TTFNN-based protocols, especially when $0.8 \leqslant \rho \leqslant 1$.

3. Analysis of Property 3

Following the methodology in Ref. [14], we briefly introduce how to derive Property 3. Three learning rules (6)–(8) can be described by a uniform equation,

$$w^{A/B}_{ij}(t+1) = g\big(w^{A/B}_{ij}(t) + f^{A/B}x_{ij}\big). \tag{A3}$$

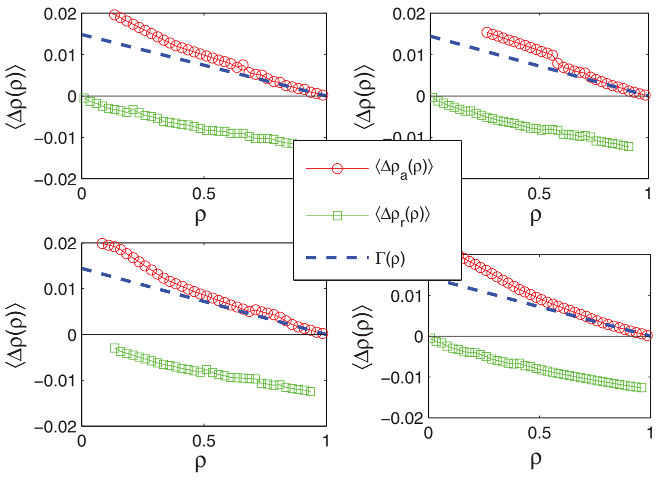

FIG. 7. (Color online) $\langle\Delta\rho_{\mathrm{a}}(\rho)\rangle$ and $\langle\Delta\rho_{\mathrm{r}}(\rho)\rangle$ of (a) TPM ($K = 3$), (b) TPCM ($K = 3$), (c) TCPM ($K = 3$), and (d) CI ($q = 1,2$). $L = 10$. The linear fitting equation $\Gamma(\rho)$ is given by Eq. (25).

If $f^{A/B} = 0$, no weights move; if $f^{A/B} \neq 0$, it can be described as

$$f^{A/B} = \begin{cases} \sigma_i, & \text{Hebbian learning rule,} \\ -\sigma_i, & \text{Anti-Hebbian learning rule,} \\ 1, & \text{Random-walk learning rule.} \end{cases} \tag{A4}$$

From Eq. (A4) the only difference between these learning rules is whether and how the output σ_i affects the moving direction.

(i) In the case of the random-learning rule, the moving direction is determined by x_{ij}, which is a random variable,

$$P(x_{ij} = +1) = \tfrac{1}{2}, \quad P(x_{ij} = -1) = \tfrac{1}{2}. \tag{A5}$$

(ii) In the case of the Hebbian learning rule, by Eqs. (A3) and (A4), the direction in which the weight w_{ij} moves is determined by the product $\sigma_i x_{ij}$. As the output σ_i is a function of all input values, x_{ij} and σ_i are correlated random variables. Thus the probabilities to observe $\sigma_i x_{ij} = +1$ and $\sigma_i x_{ij} = -1$ are not equal, but they depend on the value of the corresponding weight w_{ij},

$$\begin{aligned} P(\sigma_i x_{ij} = +1) &= \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{w_{ij}}{\sqrt{NQ_i - w_{ij}^2}}\right)\right], \\ P(\sigma_i x_{ij} = -1) &= \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{w_{ij}}{\sqrt{NQ_i - w_{ij}^2}}\right)\right]. \end{aligned} \tag{A6}$$

Note that the error function vanishes in the limit $N \to +\infty$. It holds that

$$\lim_{N\to+\infty} P(\sigma_i x_{ij} = +1) = \tfrac{1}{2}, \quad \lim_{N\to+\infty} P(\sigma_i x_{ij} = -1) = \tfrac{1}{2}. \tag{A7}$$

(iii) In the case of the anti-Hebbian learning rule, the direction in which the weight w_{ij} moves is determined by the product $-\sigma_i x_{ij}$. Similarly, it always holds that

$$\lim_{N\to+\infty} P(-\sigma_i x_{ij} = +1) = \tfrac{1}{2}, \quad \lim_{N\to+\infty} P(-\sigma_i x_{ij} = -1) = \tfrac{1}{2}. \tag{A8}$$

Based on the aforementioned analysis, the moving direction of learning rule (L_2) converges to that of learning rule (L_1). So the state moving equations for learning rule (L_2) can also be described as (A1) and (A2), which implies that the average step sizes $\langle \Delta\rho_a \rangle$ and $\langle \Delta\rho_r \rangle$ for learning rule (L_1) and its corresponding rule (L_2) are identical. Furthermore, note that the transition probabilities $P_a(\rho)$ and $P_r(\rho)$ for learning rule (L_1) and its corresponding rule (L_2) are equal to each other, together with the consequence of Eq. (17), so Property 3 is maintained.

4. Example to compute transition probabilities

With regard to the TPM-based protocol: $K = 3$, $O = 1$, F_1 is given by Eq. (4), and the random-walk learning rule is given by Eq. (31). Then (30) is reduced to

$$\begin{aligned} P_u(\varepsilon) &= P\left(\tau_{\text{TPM}}^A = \tau_{\text{TPM}}^B\right), \\ P_a(\varepsilon) &= P\left(\sigma_i^A = \sigma_i^B = \tau \middle| \tau_{\text{TPM}}^A = \tau_{\text{TPM}}^B\right), \\ P_r(\varepsilon) &= P\left(\sigma_i^A \neq \sigma_i^B \middle| \tau_{\text{TPM}}^A = \tau_{\text{TPM}}^B\right). \end{aligned} \tag{A9}$$

(1) Compute P_u. Let $P_{ui}(\varepsilon)$ denote the probability of an update step if there are i pairs of corresponding hidden units disagreeing. The following equality always holds:

$$P_{ui}(\varepsilon) = \begin{cases} (1-\varepsilon)^3, & i = 0, \\ 0, & i = 1, \\ 3(1-\varepsilon)\varepsilon^2, & i = 2, \\ 0, & i = 3. \end{cases} \tag{A10}$$

It can be computed from Eq. (A10) that

$$P_u^{\text{TPM}(K=3)}(\varepsilon) = \sum_{i=0}^{3} P_{ui}(\varepsilon) = (1-\varepsilon)^3 + 3(1-\varepsilon)\varepsilon^2.$$

(2) Compute P_a. Under the precondition that an update step occurs, let $P_{ai}(\varepsilon)$ denote the probability of an attractive step if there are i pairs of corresponding hidden units disagreeing. It follows that

$$P_{ai}(\varepsilon) = \begin{cases} \frac{\frac{1}{2}(1-\varepsilon)^3}{P_u^{\text{TPM}(K=3)}(\varepsilon)}, & i = 0, \\ 0, & i = 1, \\ \frac{\frac{1}{2}(1-\varepsilon)\varepsilon^2}{P_u^{\text{TPM}(K=3)}(\varepsilon)}, & i = 2, \\ 0, & i = 3. \end{cases} \tag{A11}$$

From Eq. (A11), we have

$$P_a^{\text{TPM}(K=3)}(\varepsilon) = \sum_{i=0}^{3} P_{ai}(\varepsilon) = \frac{\frac{1}{2}(1-\varepsilon)^3 + \frac{1}{2}(1-\varepsilon)\varepsilon^2}{(1-\varepsilon)^3 + 3(1-\varepsilon)\varepsilon^2}.$$

(3) Compute P_r. $P_r(\varepsilon)$ can be computed in a similar manner to $P_a(\varepsilon)$ and is hence omitted.

[1] W. Diffie and M. Hellman, IEEE Trans. Inf. Theory **22**, 644 (1976).

[2] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography* (CRC, Boca Raton, FL, 1996).

[3] U. Maurer, IEEE Trans. Inf. Theory **39**, 733 (1993).

[4] R. Metzler, W. Kinzel, and I. Kanter, Phys. Rev. E **62**, 2555 (2000).

[5] W. Kinzel, R. Metzler, and I. Kanter, J. Phys. A **33**, L141 (2000).

[6] W. Kinzel and I. Kanter, J. Phys. A **36**, 11173 (2003).

[7] W. Kinzel, arXiv:cond-mat/0204054.

[8] W. Kinzel and I. Kanter, Adv. Solid State Phys. **42**, 383 (2002).

[9] T. Gross and U. Feudel, Phys. Rev. E **73**, 016205 (2006).

[10] E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, Phys. Rev. E **72**, 016214 (2005).

[11] E. Klein, N. Gross, E. Kopelowitz, M. Rosenbluh, L. Khaykovich, W. Kinzel, and I. Kanter, Phys. Rev. E **74**, 046201 (2006).

[12] E. Klein, N. Gross, M. Rosenbluh, W. Kinzel, L. Khaykovich, and I. Kanter, Phys. Rev. E **73**, 066214 (2006).

[13] A. Ruttor, Ph.D. dissertation, Wurzburg University, German, 2006.

[14] A. Ruttor, W. Kinzel, R. Naeh, and I. Kanter, Phys. Rev. E **73**, 036121 (2006).
[15] A. Ruttor, W. Kinzel, and I. Kanter, Phys. Rev. E **75**, 056104 (2007).
[16] A. Ruttor, G. Reents, and W. Kinzel, J. Phys. A **37**, 8609 (2004).
[17] I. Kanter, W. Kinzel, and E. Kanter, Europhys. Lett. **57**, 141 (2002).
[18] A. Klimov, A. Mityaguine, and A. Shamir, *Analysis of Neural Cryptography*, Advances in Cryptology-ASIACRYPT 2002 (Springer-Verlag, Berlin, 2002), pp. 823–828.
[19] L. N. Shacham, E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, Phys. Rev. E **69**, 066137 (2004).
[20] M. Volkmer and S. Wallner, IEEE Trans. Comput. **54**, 421 (2005).
[21] A. Ruttor, W. Kinzel, L. Shacham, and I. Kanter, Phys. Rev. E **69**, 046110 (2004).
[22] A. Ruttor, W. Kinzel, and I. Kanter, J. Stat. Mech. (2005) P01009.
[23] A. M. Allam and H. M. Abbas, IEEE Trans. Neural Networks **21**, 1915 (2010).
[24] O. M. Reyes, I. Kopitzke, and K. Zimmermann, J. Phys. A **42**, 195002 (2009).
[25] O. M. Reyes and K.-H. Zimmermann, Phys. Rev. E **81**, 066117 (2010).
[26] L. F. Seoane and A. Ruttor, Phys. Rev. E **85**, 025101 (2012).
[27] A. Engel and C. V. den Broeck, *Statistical Mechanics of Learning* (Cambridge University Press, Cambridge, 2001).
[28] M. G. Blatt, J. Phys. A **29**, 1581 (1996).
[29] M. Rosen-Zvi, E. Klein, I. Kanter, and W. Kinzel, Phys. Rev. E **66**, 066135 (2002).
[30] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel, Phys. Rev. E **66**, 066102 (2002).