J. Phys. A: Math. Gen. **36** (2003) 11173–11186 PII: S0305-4470(03)60850-4

Disorder generated by interacting neural networks: application to econophysics and cryptography

Wolfgang Kinzel[1] **and Ido Kanter**[2]

[1] Institut für Theoretische Physik, Universität Würzburg, Am Hubland, 97074 Würzburg, Germany
[2] Department of Physics, Bar Ilan University, Ramat Gan, Israel

Received 13 March 2003
Published 15 October 2003
Online at stacks.iop.org/JPhysA/36/11173

Abstract
When neural networks are trained on their own output signals they generate disordered time series. In particular, when two neural networks are trained on their mutual output they can synchronize; they relax to a time-dependent state with identical synaptic weights. Two applications of this phenomenon are discussed for (a) econophysics and (b) cryptography. (a) When agents competing in a closed market (minority game) are using neural networks to make their decisions, the total system relaxes to a state of good performance. (b) Two partners communicating over a public channel can find a common secret key.

PACS numbers: 87.18.Sn, 05.20.−y, 75.10.Nr, 89.65.Gh

1. Introduction

Artificial neural networks are adaptive systems which are trained on a set of examples, usually a set of high-dimensional input/output data. Models and methods of statistical physics have greatly contributed to the understanding of such complex systems [1, 2]. Mainly the following scenario has been investigated: a static network—the teacher—is generating a set of examples and a different network—the student—is adapting its couplings to this set. The student net does not only learn the examples, but it also learns to generalize, the student receives an overlap to the teacher.

The examples considered by statistical mechanics are usually pairs of input/output data. The input is a high-dimensional vector and the output is—in the simplest classification problem—a single bit. Typically, in the analytic theory the input vectors used by the teacher are drawn randomly from some distribution. If the student has all the examples available for the training process it tries to minimize the training error. Hence we obtain an optimization problem with quenched disorder which is similar to finding the ground states of spin glasses [3]. However, if the examples are not stored but appear only once during the training process (on-line learning) the student follows each example by a given algorithm. In this case, we obtain a dynamical process for the couplings of the student network. Both kinds of models have been solved exactly in the limit of infinitely large networks [2].

In this paper, we report on a different scenario: all of the participating networks are generating examples and are trained on them. Hence we consider the complex dynamics of interacting adaptive systems. In some cases even the input vectors are generated by the interacting networks, no external randomness is necessary. Nevertheless, it turns out that the system is generating dynamic disorder, it functions as a kind of random number generator. But in contrast to standard pseudorandom number generators, the participating networks are trained on the sequence which they are generating. We find several applications of this scenario: unpredictable time series, competing agents in closed markets (minority game), synchronization by mutual learning and generation of secret keys over a public channel.

2. Self-interaction

We start our discussion on dynamic disorder and its applications with a single neural network. A simple perceptron—or its multilayer extension—is generating an output bit [4]. This bit is used for the next training step, the couplings of the perceptron are changed according to the corresponding input/output pair (Hebbian rule). Then one of the components of the input vector is replaced by the output bit and a new output bit is calculated. By iterating this process the network is generating a bit sequence. Note, that the network is interacting with itself, it is learning examples which are generated by the network itself. In the following, we discuss the bit generator in the context of time series prediction.

2.1. Unpredictable time series

A neural network is an efficient and simple algorithm to predict a given sequence of numbers [5]. It is trained on a part of the sequence and makes predictions on how the sequence will continue in the next few time steps. We want to consider the general question, whether a prediction algorithm, in particular a neural network, is able to predict *any* given time series better than random guessing. Before we address this question we define the neural network.

The simplest mathematical neural network is the perceptron [1, 2]. It consists of a single layer of N synaptic weights $\mathbf{w} = (w_1, \ldots, w_N)$. For a given input vector $\mathbf{x}$, the output bit is given by

$$\sigma = \operatorname{sign}\left(\sum_{i=1}^{N} w_i x_i\right). \tag{1}$$

The decision surface of the perceptron is just a hyperplane in the N-dimensional input space, $\mathbf{w} \cdot \mathbf{x} = 0$. A perceptron may also have a continuous output y as

$$y = \tanh\left(\sum_{i=1}^{N} w_i x_i\right). \tag{2}$$

A perceptron may be considered as an elementary unit of a more complex network such as an attractor network or a multilayer network. In fact any function can be approximated by a multilayer network if the number of hidden units is large enough.

A neural network learns from examples. For our perceptron, examples are pairs of input vectors and output bits,

$$(\mathbf{x}(t), \sigma(t)) \qquad t = 1, \ldots, \alpha N. \tag{3}$$

On-line training means that at each time step t the weights of the perceptron adapt to a new example, for instance by the rule

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \frac{\eta}{N}\sigma(t)\mathbf{x}(t)F(\sigma(t)\mathbf{x}(t)\cdot\mathbf{w}(t)). \tag{4}$$

$F(z) = 1$ is usually called—after the corresponding biological mechanism—the Hebbian rule, each synapse w_i responds to the activities $\sigma(t)x_i(t)$ at its two ends. $F(z) = \Theta(-z)$ is called the Rosenblatt rule: a training step occurs only if the example is misclassified. For the Rosenblatt rule a mathematical theorem can be proven: if the examples can be classified by any perceptron at all, then this rule will find a solution.

Now we consider time series prediction. Consider some arbitrary prediction algorithm. It may contain all the knowledge of mankind, many experts may have developed it. For a given bit sequence $S_1, S_2, \ldots,$ the algorithm has been trained on the first t bits $S_1, \ldots, S_t$. Can it predict the next bit S_{t+1}? Is the prediction error, averaged over a large t interval, less than 50%?

If the bit sequence is random then every algorithm will give a prediction error of 50%. But if there are some correlations in the sequence then a clever algorithm should be able to reduce this error. In fact, for the most powerful algorithm one is tempted to say that for *any* sequence it should perform better than 50% error. However, this is not true [6]. To see this just generate a sequence $S_1, S_2, S_3, \ldots$ using the following algorithm:

> Define S_{t+1} to be the opposite of the prediction of this algorithm which has been trained on $S_1, \ldots, S_t$.

Now, if the same algorithm is trained on this sequence, it will always predict the following bit with 100% error. Hence there is no general prediction machine; to be successful the algorithm needs some pre-knowledge about the class of problems it is applied to.

The Boolean perceptron (1) is a very simple prediction algorithm for a bit sequence, in particular with the Hebbian on-line training algorithm (4). What does the bit sequence look like for which the perceptron fails completely?

Following (4), we just have to take the negative value

$$S_t = -\mathrm{sign}\left(\sum_{j=1}^{N} w_j\, S_{t-j}\right) \tag{5}$$

and then train the network on this new bit:

$$\Delta w_j = +\frac{1}{N} S_t\, S_{t-j}. \tag{6}$$

The perceptron is trained on the opposite (negative) of its own prediction. Starting from (say) random initial states $S_1, \ldots, S_N$ and weights $\mathbf{w}$, this procedure generates a sequence of bits $S_1, S_2, \ldots, S_t, \ldots$ and of vectors $\mathbf{w}, \mathbf{w}(1), \mathbf{w}(2), \ldots, \mathbf{w}(t), \ldots$ as well. Given this sequence and the same initial state, the perceptron which is trained on it yields a prediction error of 100%.

It turns out that this simple algorithm produces a rather complex bit sequence which comes close to a random one [7]. After a transient time the weight vector $\mathbf{w}(t)$ seems to perform a kind of random walk on an N-dimensional hypersphere. The bit sequence runs to a cycle whose average length L scales exponentially with N,

$$L \simeq 2.2^N. \tag{7}$$

The autocorrelation function of the sequence shows complex properties: it is close to zero up to N, oscillates between N and $3N$ and it is similar to random noise for larger distances.

Its entropy is smaller than the one of a random sequence since the frequency of some patterns is suppressed. Of course, it is not random since the prediction error is 100% instead of 50% for a random bit sequence.

When a second perceptron (student) with different initial state $\mathbf{w}^S$ is trained on such a anti-predictable sequence generated by equation (5) it can perform somewhat better than the teacher: the prediction error goes down to about 78% but it is still larger than 50% for random guessing. Related to this, the student obtains knowledge about the teacher: the angle between the two weight vectors relaxes to about 45° [6, 7]. Hence the complex anti-predictable sequence still contains enough information for the student to follow the time-dependent teacher.

In fact, the disorder of the generated bit sequence helps to learn the rule. A simple sequence can be learned well but it is difficult to obtain an overlap to the rule. A complex sequence, however, is difficult to predict but the rule can be estimated with great precision [8–10].

2.2. Agents competing in a closed market

We just considered a simple network interacting with itself. Now we extend this model to a multilayer network consisting of several perceptrons. The output bit is just the sign of the majority of the perceptrons, hence this network is called a *committee machine*. As before, the network is trained on the negative of its own output bit, which afterwards is used for the input. This multilayer network may also be considered as a system of many perceptrons interacting with the minority decision of all members. This work was motivated by the following problem of econophysics [11].

Recently a mathematical model of economy has received a lot of attention in the community of statistical physics. It is a simple model of a closed market: there are K agents (K is an odd integer) who have to make a binary decision $\sigma(t) \in \{+1, -1\}$ at each time step. All of the agents who belong to the minority gain one point, the majority has to pay one point (to a cashier which always wins). The global loss is given by

$$G = \left| \sum_{t=1}^{K} \sigma(t) \right| . \tag{8}$$

If the agents come to an agreement before they make a new decision, it is easy to minimize $G : (K - 1)/2$ agents have to choose +1, then $G = 1$. However, this is not the rule of the game; the agents are not allowed to make contracts, and communicate only through the global sum of decisions. Each agent knows only the history of the minority decision, $S_1, S_2, S_3, \ldots,$ but otherwise he/she has no information. Can the agent find an algorithm to maximize his/her profit?

If each agent makes a random decision, then $\langle G^2 \rangle = K$. It is possible, but not trivial, to find algorithms which perform better than random [12].

Here we use a perceptron for each agent to make a decision based on the past N steps $\mathbf{S} = (S_{t-N}, \ldots, S_{t-1})$ of the minority decision. The decision of agent $\mathbf{w}(t)$ is given by

$$\sigma(t) = \operatorname{sign}(\mathbf{w}(t) \cdot \mathbf{S}). \tag{9}$$

After the bit S_t of the minority has been determined, each perceptron is trained on this new example $(\mathbf{S}, S_t)$,

$$\Delta \mathbf{w}(t) = \frac{\eta}{N} S_t \, \mathbf{S}. \tag{10}$$

This problem could be solved analytically [13]. The average global loss for $\eta \to 0$ is given by

$$\langle G^2 \rangle = (1 - 2/\pi) K \simeq 0.363 K. \tag{11}$$

Hence, for small enough learning rates the system of interacting neural networks performs better than random decisions. A successful strategy emerges from the cooperation of adaptive perceptrons.

2.3. Cooperation between random walks

For many applications of neural networks, discrete couplings are preferred over continuous ones. For generalization, discrete couplings limit the ability to learn a rule when an on-line algorithm is used [14]. In contrast, synchronization of discrete weights of multilayer networks [17] or fully connected networks [22] can be easily achieved by mutual learning. Hence, multilayer networks with discrete couplings are potential candidates for novel algorithms in public channel cryptography, as discussed later in this paper. Therefore, we have studied such networks in the context of self-learning discussed before. It turns out that such networks may be considered as an ensemble of random walks with reflecting boundaries, triggered by global signals which are generated by the ensemble.

Each synaptic weight of our perceptron, w_i, is now limited to integers between $\pm L$,

$$w_i \in \{-L, -L+1, \ldots, L-1, L\}. \tag{12}$$

For a given input vector $\mathbf{x}$ with binary components the perceptron produces an output bit $\sigma = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$ which is used for the anti-Hebbian training step

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \sigma \mathbf{x}. \tag{13}$$

The weights remain in the interval (12), therefore if any component moves out of the allowed range it is set back to the boundary $w_i = \pm L$.

After the training step the output bit σ replaces the first bit of the shifted input vector, we use the same feedback as before.

Our simulations show that this perceptron being trained on its negative output bit has similar properties to the corresponding network with continuous couplings [15]. It generates a bit sequence which is hard to distinguish from a random one. But a closer investigation of correlations shows similar patterns as before.

Since the input sequence of the network is close to random, each component w_i performs a kind of random walk on $2L + 1$ sites with reflecting boundaries. However, the ensemble itself generates the noise which determines the direction of motion for each walk. Note that the walk is deterministic; except for the random initial state, no random numbers are used for the dynamics.

Later in this paper, discussing cryptography, we use a multilayer network consisting of K perceptrons with discrete weights. The architecture is tree-like, each perceptron receives a different input vector $\mathbf{w}_k$, $k = 1, \ldots, K$. The output bit of the total network is the product of the K hidden units,

$$\tau = \prod_{k=1}^{K} \sigma_k \qquad \sigma_k = \mathrm{sign}(\mathbf{w}_k \cdot \mathbf{x}_k). \tag{14}$$

This network is called a *parity machine*, since its output is determined by the parity of the hidden units in the 0, 1 bit notation.

For each time step, the hidden output bit σ_k is used for the corresponding input vector $\mathbf{x}_k$. However, only if the hidden unit σ_k agrees with the output bit τ are the corresponding weights $\mathbf{w}_k$ changed according to the rule (13). Our numerical simulations show that the generated bit sequence passes many tests of random number generators, we could not detect any correlations [15]. Later in this paper we will see that this property helps to encrypt secret messages.

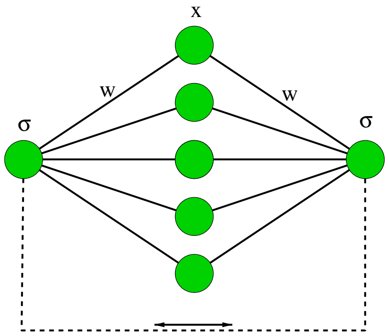

Figure 1. Two perceptrons are trained by their mutual output bits, from [13].

3. Mutual learning

In the previous section, we have investigated the properties of *single* neural networks trained on their own output. Now we study a system of two neural networks, perceptrons or multilayer networks, which learn from their partners. We find a novel phenomenon, a phase transition to synchronization [13], and we use it for a novel algorithm in public channel cryptography [17].

3.1. Synchronization

We consider the model where two perceptrons A and B receive a common random input vector $\mathbf{x}$ and change their weights $\mathbf{w}$ according to their mutual bit σ, as sketched in figure 1. The output bit σ of a single perceptron is given by the equation

$$\sigma = \operatorname{sign}(\mathbf{w}\cdot\mathbf{x}) \tag{15}$$

where $\mathbf{x}$ is an N-dimensional input vector with components which are drawn from a Gaussian with mean 0 and variance 1. $\mathbf{w}$ is an N-dimensional weight vector with continuous components which are normalized,

$$\mathbf{w}\cdot\mathbf{w} = 1. \tag{16}$$

The initial state is a random choice of the components $w_i^{\mathrm{A/B}}$, $i = 1, \ldots, N$ for the two weight vectors $\mathbf{w}^{\mathrm{A}}$ and $\mathbf{w}^{\mathrm{B}}$. At each training step a common random input vector is presented to the two networks which generate two output bits σ^{A} and σ^{B} according to (15). Now the weight vectors are updated by the Rosenblatt learning rule (4):

$$\begin{aligned}\mathbf{w}^{\mathrm{A}}(t+1) &= \mathbf{w}^{\mathrm{A}}(t) + \frac{\eta}{N}\mathbf{x}\sigma^{\mathrm{B}}\Theta(-\sigma^{\mathrm{A}}\sigma^{\mathrm{B}})\\ \mathbf{w}^{\mathrm{B}}(t+1) &= \mathbf{w}^{\mathrm{B}}(t) + \frac{\eta}{N}\mathbf{x}\sigma^{\mathrm{A}}\Theta(-\sigma^{\mathrm{A}}\sigma^{\mathrm{B}})\end{aligned} \tag{17}$$

where $\Theta(x)$ is the step function. Hence, only if the two perceptrons disagree is a training step performed with a learning rate η. After each step (17), the two weight vectors are normalized.

In the limit $N \to \infty$, the overlap

$$R(t) = \mathbf{w}^{\mathrm{A}}(t)\cdot\mathbf{w}^{\mathrm{B}}(t) \tag{18}$$

has been calculated analytically [13]. The number of training steps t is scaled as $\alpha = t/N$, and $R(\alpha)$ follows the equation

$$\frac{\mathrm{d}R}{\mathrm{d}\alpha} = (R+1)\left(\sqrt{\frac{2}{\pi}}\eta(1-R) - \eta^2\frac{\phi}{\pi}\right) \tag{19}$$

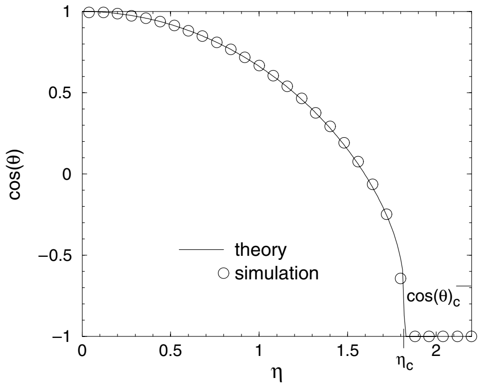

Figure 2. Final overlap R between two perceptrons as a function of learning rate η. Above a critical rate η_c the time-dependent networks are synchronized (from [13]).

where ϕ is the angle between the two weight vectors $\mathbf{w}^A$ and $\mathbf{w}^B$, i.e. $R = \cos\phi$. This equation has fixed points $R = 1$, $R = -1$, and

$$\frac{\eta}{\sqrt{2\pi}} = \frac{1-\cos\phi}{\phi}. \tag{20}$$

Figure 2 shows the attractive fixed point of (19) as a function of the learning rate η. For small values of η the two networks relax to a state of mutual agreement, $R \to 1$ for $\eta \to 0$. With increasing learning rate η the angle between the two weight vectors increases up to the value $\phi = 133°$ for

$$\eta \to \eta_c \cong 1.816. \tag{21}$$

Above the critical rate η_c the networks relax to a state of complete disagreement, $\phi = 180°$, $R = -1$. The two weight vectors are antiparallel to each other, $\mathbf{w}^A = -\mathbf{w}^B$.

As a consequence, the analytic solution shows, well supported by numerical simulations for $N = 100$, that two neural networks can synchronize with each other by mutual learning. Both of the networks are trained to the examples generated by their partners and finally obtain an antiparallel alignment. Even after synchronization the networks keep moving, the motion is a kind of random walk on an N-dimensional hypersphere producing a rather complex bit sequence of output bits $\sigma^A = -\sigma^B$ [7]. In fact, after synchronization the system is identical to the single network learning its opposite output bit, discussed in section 2.1.

3.2. Neural cryptography

In the field of cryptography, one is interested in methods to transmit secret messages between two partners A and B. An attacker E who is able to listen to the communication should not be able to recover the secret message.

Before 1976, all cryptographic methods had to rely on secret keys for encryption which were transmitted between A and B over a secret channel not accessible to any attacker. Such a common secret key can be used, for example, as a seed for a random bit generator by which the bit sequence of the message is added (modulo 2).

In 1976, however, Diffie and Hellmann found that a common secret key could be created over a public channel accessible to any attacker. This method is based on number theory: given limited computer power, it is not possible to calculate the discrete logarithm of sufficiently large numbers [16].

Here we discuss a completely different method of generating secret keys. We have shown that interacting neural networks can produce a common secret key by exchanging bits over a public channel and by learning the bits of their partners [17].

We apply synchronization of neural networks to cryptography. In the previous section we have seen that the weight vectors of two perceptrons learning from each other can synchronize. The new idea is to use the common weights $\mathbf{w}^{\mathrm{A}} = -\mathbf{w}^{\mathrm{B}}$ as a key for encryption. But two problems have yet to be solved: (i) can an external observer, recording the exchange of bits, calculate the final $\mathbf{w}^{\mathrm{A}}(t)$, (ii) does this phenomenon exist for discrete weights? Point (i) is essential for cryptography, it will be discussed further below. Point (ii) is important for practical solutions since communication is usually based on bit sequences. It will be investigated in the following.

Synchronization occurs for normalized weights, unnormalized ones do not synchronize [13]. Therefore, for discrete weights, we introduce a restriction in the space of possible vectors and limit the components $w_i^{\mathrm{A/B}}$ to $2L + 1$ different values,

$$w_i^{\mathrm{A/B}} \in \{-L, -L+1, \ldots, L-1, L\}. \tag{22}$$

In order to obtain synchronization to a parallel—instead of an antiparallel—state $\mathbf{w}^{\mathrm{A}} = \mathbf{w}^{\mathrm{B}}$, we modify the learning rule (17) to

$$\begin{aligned} \mathbf{w}^{\mathrm{A}}(t+1) &= \mathbf{w}^{\mathrm{A}}(t) - \mathbf{x}\sigma^{\mathrm{A}}\Theta(\sigma^{\mathrm{A}}\sigma^{\mathrm{B}}) \\ \mathbf{w}^{\mathrm{B}}(t+1) &= \mathbf{w}^{\mathrm{B}}(t) - \mathbf{x}\sigma^{\mathrm{B}}\Theta(\sigma^{\mathrm{A}}\sigma^{\mathrm{B}}). \end{aligned} \tag{23}$$

Now the components of the random input vector $\mathbf{x}$ are binary $x_i \in \{+1, -1\}$. If the two networks produce an identical output bit $\sigma^{\mathrm{A}} = \sigma^{\mathrm{B}}$, then their weights move one step in the direction of $-x_i\sigma^{\mathrm{A}}$. But the weights should remain in the interval (22), therefore if any component moves out of this interval, it is set back to the boundary $w_i = \pm L$.

Each component of the weight vectors performs a kind of random walk with reflecting boundary. Two corresponding components w_i^{A} and w_i^{B} receive the same random number ± 1. After each hit at the boundary the distance $|w_i^{\mathrm{A}} - w_i^{\mathrm{B}}|$ is reduced until it has reached zero. For two perceptrons with an N-dimensional weight space we have two ensembles of N random walks on the internal $\{-L, \ldots, L\}$. We expect that after some characteristic time scale $\tau = \mathcal{O}(L^2)$ the probability of two random walks being in different states decreases as

$$P(t) \sim P(0)\,\mathrm{e}^{-t/\tau}. \tag{24}$$

Hence the total synchronization time should be given by $N \cdot P(t) \simeq 1$ which gives

$$t_{\mathrm{sync}} \sim \tau \ln N. \tag{25}$$

In fact, the simulations for $N = 100$ show that two perceptrons with $L = 3$ synchronize in about 100 time steps and the synchronization time increases logarithmically with N. However, the simulations also showed that an attacker, recording the sequence of $(\sigma^{\mathrm{A}}, \sigma^{\mathrm{B}}, \mathbf{x})_t$ is able to synchronize, too. Therefore, a single perceptron does not allow the generation of a secret key.

Obviously, a single perceptron transmits too much information. An attacker, who knows the set of input/output pairs, can derive the weights of the two partners after synchronization. Therefore, one has to hide so much information that the attacker cannot calculate the weights, but on the other side one has to transmit enough information that the two partners can synchronize.

In fact, it was shown that multilayer networks with hidden units may be candidates for such a task [17–19]. More precisely, we consider parity machines with three hidden units as shown in figure 3. Each hidden unit is a perceptron (1) with discrete weights (22). The output bit τ of the total network is the product of the three bits of the hidden units

$$\tau^{\mathrm{A}} = \sigma_1^{\mathrm{A}}\sigma_2^{\mathrm{A}}\sigma_3^{\mathrm{A}} \qquad \tau^{\mathrm{B}} = \sigma_1^{\mathrm{B}}\sigma_2^{\mathrm{B}}\sigma_3^{\mathrm{B}}. \tag{26}$$

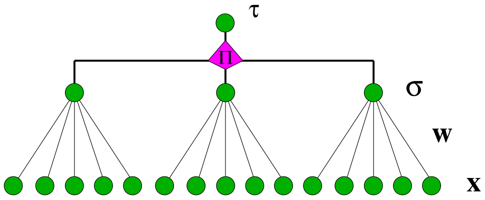

Figure 3. Parity machine with three hidden units.

At each training step the two machines A and B receive identical input vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. The training algorithm is the following: only if the two output bits are identical, $\tau^{\mathrm{A}} = \tau^{\mathrm{B}}$, can the weights be changed. In this case, only the hidden unit σ_i which is identical to τ changes its weights using the Hebbian rule

$$\mathbf{w}_i^{\mathrm{A}}(t+1) = \mathbf{w}_i^{\mathrm{A}}(t) - \mathbf{x}_i \tau^{\mathrm{A}}. \tag{27}$$

For example, if $\tau^{\mathrm{A}} = \tau^{\mathrm{B}} = 1$ there are four possible configurations of the hidden units in each network:

$$(+1, +1, +1), (+1, -1, -1), (-1, +1, -1), (-1, -1, +1).$$

In the first case, all three weight vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ are changed, in all the other three cases only one weight vector is changed. The partner as well as any attacker does not know which one of the weight vectors is updated.

The partners A and B react to their mutual stop and move signals τ^{A} and τ^{B}, whereas an attacker can only receive these signals but not influence the partners with its own output bit. This is the essential mechanism which allows synchronization but prohibits learning. Numerical [17] as well as analytical [18] calculations of the dynamic process show that the partners can synchronize in a short time whereas an attacker needs a much longer time to lock into the partners.

This observation holds for an observer who uses the same algorithm (27) as the two partners A and B. Note that the observer knows (1) the algorithm of A and B, (2) the input vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ at each time step and (3) the output bits τ^{A} and τ^{B} at each time step. Nevertheless, it does not succeed in synchronizing with A and B within the communication period.

Since for each run the two partners draw random initial weights and since the input vectors are random, one obtains a distribution of synchronization times as shown in figure 4 for $N = 100$ and $L = 3$. The mean value of this distribution is shown as a function of system size N in figure 5. Even a very large network needs only a relatively small number of exchanged bits—about 400 in this case—to synchronize. The figure even seems to indicate that the synchronization time is finite for $N \to \infty$, but similar arguments as given above show $t_{\mathrm{sync}} \sim \log N$.

If the communication continues after synchronization, an attacker has a chance to lock into the moving weights of A and B. Figure 6 shows the distribution of the ratio between the synchronization time of A and B and the learning time of the attacker. In the simulations for $N = 100$, this ratio never exceeded the value $r = 0.1$, and the average learning time is about 50 000 time steps, much larger than the synchronization time. Hence, the two partners can take their weights $\mathbf{w}_i^{\mathrm{A}}(t) = \mathbf{w}_i^{\mathrm{B}}(t)$ at a time step t where synchronization most probably occurred as a common secret key. Synchronization of neural networks can be used as a key exchange protocol over a public channel.

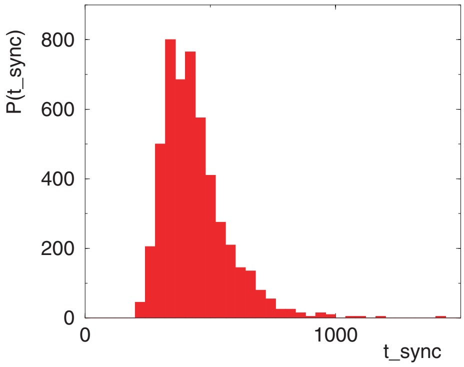

Figure 4. Distribution of synchronization time for $N = 100$, $L = 3$ (from [17]).

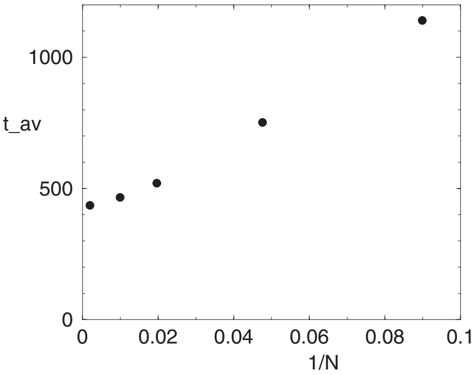

Figure 5. Average synchronization time as a function of inverse system size.

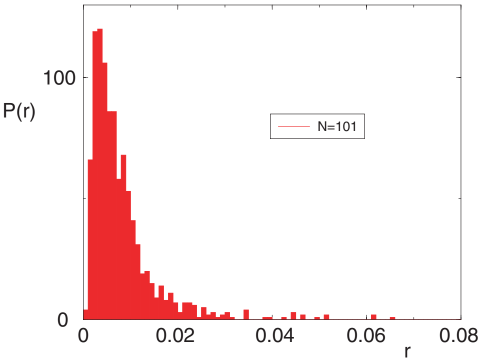

Figure 6. Distribution of the ratio of synchronization time between networks A and B to the learning time of an attacker E.

3.3. Security of neural cryptography

In the previous paragraph we have seen that two interacting neural networks approach each other faster than any other network which is just learning the communication by using identical rules. Is there any algorithm which is faster than the attacking network defined above? In any case the attacker should remain synchronized with the partners if she has achieved complete overlap. Therefore, in the case of agreement, $\tau^{\mathrm{E}} = \tau^{\mathrm{A}}$, the attacking network E should use an identical algorithm to A and B. However, when they do not agree the attacking network can indeed improve the probability of synchronization as was shown by the group of Shamir and co-workers [20]. In the case of disagreement at least one hidden unit σ_k^{E} disagrees with the corresponding σ^{A}. Of course, the attacker does not know which one. But she knows that the probability of disagreement is larger for smaller internal fields. Hence E changes the direction of the hidden unit with the smallest absolute value of the field and trains the corresponding weights. Shamir and co-workers have shown that this modification increases the probability for E to synchronize before A and B are synchronized. Hence if the attacker uses an ensemble of many attacking networks, there is a good chance that at least one of them has found the correct key. Reading all of the encrypted messages the attacker is able to find the one with meaningful text.

Of course, if the probability P_{E} of success is so low that it is computationally infeasible to use the required number of attacking networks the system is considered to be secure. Hence we have calculated P_{E} as a function of model parameters [21]. It turns out that the probability of success is sensitive to the depth L of the weights. Note that each weight can take $2L + 1$ integer values. The numerical simulations indicate that the synchronization time increases linearly with L^2 whereas the probability P_{E} decreases exponentially with L^2,

$$t_{\mathrm{sync}} = AL^2 \qquad P_{\mathrm{E}} = B\exp(-yL^2). \tag{28}$$

This means that the security of the network improves when the parameter L is increased. For a sufficiently large value of L, neural cryptography is secure.

It is desirable to increase the probability P_{E} even further, i.e. to find high values of y in (28). The idea is to use self-generated noise to make the attack more difficult without destroying synchronization between the two partners. The usual noise cannot be added, since it does not distinguish between interaction and learning. However, if the influence of noise decreases if the two networks come close to each other it may improve security. Therefore, the participating networks have to generate noise which becomes identical if the networks are almost synchronized.

In fact we have found several kinds of self-generated noise which increases the value y of (28). One method uses the feedback mechanism discussed previously in this contribution [15]. The input of the hidden units is generated by the corresponding perceptron itself, as described in section 2.1. Hence, when the perceptrons are identical they generate identical inputs and remain identical. But when the networks differ by some fraction of components, synchronization becomes difficult. One has to control the repulsive force by another mechanism which resets the input to identical values depending on the number r of steps with disagreement.

This protocol is public again, hence the attacker can make use of all the information. Nevertheless, it turns out that the value y increases with the additional parameter r, as shown in figure 7. As a consequence, the feedback mechanism improves the security of the key generation.

Another method to increase the value of y (decrease the probability P_{E} of a successful attack) is a combination of neural cryptography with chaotic synchronization [23]. In addition to the weights $\mathbf{w}_k$, each hidden unit obtains a chaotic map triggered by its internal

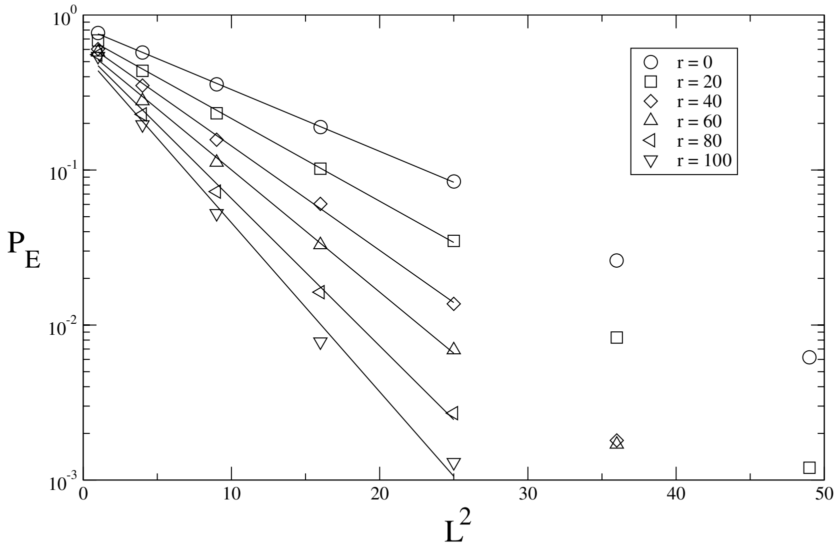

Figure 7. Probability P_E of a successful attack as a function of L^2 for different feedback parameters r. The networks have $K = 3$ hidden units and $N = 1000$ weights per unit.

field. Synchronization of the weights synchronizes the corresponding chaotic maps, and synchronized chaotic maps synchronize the weights. This coupled synchronization process, for suitable values of the corresponding parameters, improves the security of neural cryptography.

4. Conclusions

Artificial neural networks are trained by a sequence of examples. In contrast to previous studies, the examples are generated by the networks themselves. This leads to a rather complex dynamics which is investigated in several different contexts.

If a single-neural network—a simple perceptron or a multilayer parity machine—is trained on the negative of its own output, the network is generating a bit sequence whose statistical properties are close to a random sequence. In fact, for the parity machine with discrete weights, the sequence passed all tests on random numbers. Our networks with discrete weights are identical to an ensemble of random walks with reflecting boundaries and self-generated noise.

When a network with different initial state is trained on such a sequence, it still receives an overlap to the bit generator. In contrast, the prediction error is higher than for random guessing. Hence, the disordered bit sequence is unpredictable but contains information about the rule which generated it.

The multilayer committee machine which is trained on the negative of its output is an algorithm for competing agents in a closed market (minority game). It may be considered as a system of neural networks adapting to the history of success. We find that an algorithm develops which performs better then random guessing. Successful cooperation emerges from the interaction of neural networks.

When two neural networks are trained on their mutual output bits a novel phenomenon appears: as a function of the parameter values of the models a transition to synchronization occurs. In the synchronized phase the weights still perform a kind of random walk on a high-dimensional hypersphere, but the two weight vectors are identical (up to a sign).

Synchronization by mutual learning is a novel principle whose biological implications still have to be investigated.

We found an application for this phenomenon in the field of cryptography. Synchronization of multilayer networks can be used for generating a secret key over a public channel. Two partners are exchanging their output bits, learning them and arrive at common weight vectors which they use as a key for encryption. Surprisingly, any attacker who knows all the details of the exchanged information as well as the architecture of the participating networks cannot calculate the secret key, at least with limited computer power. Learning by interacting is more efficient than learning by listening.

The security of neural cryptography has been investigated quantitatively. When an ensemble of attacking networks is used the probability of an successful attack can be decreased in several ways: increasing the depth of the synaptic weights, adding self-generated noise by a feedback mechanism and combining networks with synchronization of chaotic maps.

In summary, interacting neural networks have a complex dynamics. They produce disordered bit sequences and synchronize. Applications to econophysics and cryptography have been found.

Acknowledgment

This overview is based on enjoyable collaborations with Richard Metzler, Michal Rosen-Zvi, Andreas Ruttor, Einat Klein and Rachel Mislovaty. This work has been supported by the German Israel Science Foundation (GIF) and the Minerva Center of the Bar Ilan University.

References

[1] Hertz J, Krogh A and Palmer R G 1991 *Introduction to the Theory of Neural Computation* (Redwood City, CA: Addison-Wesley)

[2] Engel A and Van den Broeck C 2001 *Statistical Mechanics of Learning* (Cambridge: Cambridge University Press)

[3] Hartmann A and Rieger H 2002 *Optimization Algorithms in Physics* (Berlin: Wiley-VCH)

[4] Eisenstein E, Kanter I, Kessler D A and Kinzel W 1995 Generation and prediction of time series by a neural network *Phys. Rev. Lett.* **74** 6

[5] Weigand A and Gershenfeld N S 1994 *Time Series Prediction* (Santa Fe, NM: Addison-Wesley)

[6] Zhu H and Kinzel W 1998 Anti-predictable sequences: harder to predict than a random sequence *Neural Comput.* **10** 2219–30

[7] Metzler R, Kinzel W, Ein-Dor L and Kanter I 2001 Generation of unpredictable time series by a neural network *Phys. Rev.* E **63** 056126

[8] Freking A, Kinzel W and Kanter I 2002 *Phys. Rev.* E **65** 050903(R)

[9] Miyazaki Y, Kinzel W and Shinomoto S 2003 *J. Phys. A: Math. Gen.* **36** 1315–22

[10] Rosen-Zvi M, Kanter I and Kinzel W 2003 Time series prediction by feedforward neural networks—is it difficult? *J. Phys. A: Math. Gen.* at press

[11] Econophysics homepage: http://www.unifr.ch/econophysics

[12] Challet D, Marsili M and Zecchina R 2000 Statistical mechanics of systems with heterogeneous agents: minority games *Phys. Rev. Lett.* **84** 1824–7

[13] Metzler R, Kinzel W and Kanter I 2000 Interacting neural networks *Phys. Rev.* E **62** 2555

[14] Kinzel W and Urbanczik R 1998 *J. Phys. A: Math. Gen.* **31** L27–30

[15] Ruttor A, Kinzel W and Kanter I 2003 in preparation

[16] Stinson D R 1995 *Cryptography: Theory and Practice* (Boca Raton, FL: CRC Press)

[17] Kanter I, Kinzel W and Kanter E 2002 Secure exchange of information by synchronisation of neural networks *Europhys. Lett.* **57** 141–7

[18] Rosen-Zvi M, Kanter I and Kinzel W 2002 *J. Phys. A: Math. Gen.* **35** L707–13

[19] Rosen-Zvi M, Klein E, Kanter I and Kinzel W 2002 *Phys. Rev.* E **66** 066135

[20] Klimov A, Mityagin A and Shamir A 2002 Analysis of neural cryptography *ASIACRYPT*

[21] Mislovaty R, Perchenok Y, Kanter I and Kinzel W 2002 *Phys. Rev.* E **66** 066102

[22] Kanter I and Kinzel W unpublished

[23] Mislovaty R, Klein E, Kanter K and Kinzel W Public channel cryptography by synchronization of neural networks and chaotic maps at press