Journal of
Information Technology & Software Engineering

Wang et al. J Inform Tech Soft Engg 2011, 1:2
DOI; 10.4172/2165-7866.1000104

Review Article Open Access

Recurrent Neural Networks: Associative Memory and Optimization

Hui Wang[1], Yue Wu[1], Biaobiao Zhang[1] and K. -L. Du[1,2]*

[1]Enjoyor Labs, Enjoyor Inc., Hangzhou, China, 310030
[2]Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, H3G 1M8

Abstract

Due to feedback connections, recurrent neural networks (RNNs) are dynamic models. RNNs can provide more compact structure for approximating dynamic systems compared to feedforward neural networks (FNNs). For some RNN models such as the Hopfield model and the Boltzmann machine, the fixed-point property of the dynamic systems can be used for optimization and associative memory. The Hopfield model is the most important RNN model, and the Boltzmann machine as well as some other stochastic dynamic models are proposed as its generalization. These models are especially useful for dealing with combinatorial optimization problems (COPs), which are notorious NP-complete problems. In this paper, we provide a state-of-the-art introduction to these RNN models, their learning algorithms as well as their analog implementations. Associative memory, COPs, simulated annealing (SA), chaotic neural networks and multilevel Hopfield models are also important topics treated in this paper.

I. Introduction

In 1943, McCulloch and Pitts found that the neuron can be modeled as a simple threshold device to perform logic function [85]. In the late 1950s, Rosenblatt [102,103] proposed the first neural network model---the perceptron model as well as its learning algorithm called the Perceptron learning algorithm. Interest in neural networks waned rapidly after Minsky and Papert proved mathematically that the Rosenblatt's simple perceptron model cannot be used for complex logic function [89]. The simple perceptron model cannot correctly classify linearly inseparable patterns.

The modern era of neural network research is commonly deemed to have started with the publication of the Hopfield network in 1982 [50,51,53]. The Hopfield model works at the system level rather than at a single neuron level. It is an RNN working with the Hebbian rule [47]. As a dynamically stable network, the fixed points of this network can be used as associative memories for information storage as well as solutions to optimization problems. The Boltzmann machine was introduced in 1985 as an extension to the Hopfield network by incorporating stochastic neurons [4]. The Boltzmann learning is based on a method called simulated annealing (SA) [62]. SA is a stochastic search method for the global optimum of any objective function. The chaotic neural network was introduced by adding a negative self-coupling term, termed chaotic dynamics, to the Hopfield network [5]. The operation of the chaotic neural network is similar to that of SA, but in a deterministically chaotic way rather than in a stochastic way, hence called chaotic SA (CSA).

Memory is important for transforming a static system into a dynamic one. Memories can be long-term or short-term. A long-term memory is used to store stable system information, while a short-term memory is useful for simulating a dynamic system with a temporal dimension. Feedforward neural networks (FNNs) can become dynamic by embedding memory into the network using time-delay [31]. RNNs are dynamic models due to the feedback connections in the architecture.

The combinatorial optimization problem (COP) and associative memory are the two major applications of the Hopfield network and the Boltzmann machine. In this paper, in addition to the Hopfield network and the Boltzmann machine, the topics of associative memory and COPs are also developed. Some other models for associative memory such as the multilayer perceptron (MLP)-based autoassociative memories and the Hamming network [79] are introduced. SA, a general-purpose global optimization method, is also described. This paper is organized as follows. In Section II, we briefly mention a class of RNNs with the universal approximation capability for dynamic systems. The Hopfield model is introduced in Section III, and its analog implementation is described in Section IV. Various associative memory models and their storage/retrieval algorithms are described in Section V. In Section VI, we deal with SA. Combinatorial optimization problems are treated in Section VII. Chaotic neural networks are introduced in Section VIII. In Section IX, we also mention the application of the Hopfield model to some other optimization and signal processing problems. Multilevel Hopfield networks are described in Section X. Section XI is dedicated to the Boltzmann machine and its learning. A discussion is given in Section XII. Finally, in Section XIII, a brief summary of the paper is drawn, and the cellular neural network model is also mentioned in this section.

II. Recurrent Neural Networks

RNNs are inspired by ideas from statistical physics. There is at least one feedback connection in RNNs. RNNs are dynamical systems with temporal state representations. They are computationally powerful, and can be used in many temporal processing models and applications. The Hopfield model can store information in a dynamically stable structure [50]. The Boltzmann machine [4], which relies on a stochastic algorithm of statistical thermodynamics called SA [62], is a generalization of the Hopfield model.

For a fully-connected RNN of J McCulloch-Pitts neurons, the dynamics of the network with sound biological and electronic motivations are given in [35,43]. The output of a sufficiently large such continuous-time model can approximate any continuous state-space trajectory to any desired degree of accuracy [35]. In other words, the RNN is a universal approximator of dynamical systems. The universal approximation capability of RNNs have also been investigated in a number of papers [75,76]. In [75], a fully-connected discrete-time

***Corresponding author:** K. -L. Du, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, H3G 1M8; E-mail: kldu@ece.concordia.ca

Received November 03, 2011; **Accepted** November 22, 2011; **Published** November 24, 2011

Citation: Wang H, Wu Y, Zhang B, Du KL (2011) Recurrent Neural Networks: Associative Memory and Optimization. J Inform Tech Soft Engg 1:104. doi:10.4172/2165-7866.1000104

RNN using McCulloch-Pitts neurons has been proved to be a universal approximator of discrete- or continuous-time trajectories on compact time intervals. A continuous-time RNN McCulloch-Pitts neurons and external input is shown to be able to approximate any finite-time trajectory of a dynamical time-variant system [76].

When used for approximation of dynamic systems, an RNN is more powerful in representing complex dynamics than an FNN. Given the same approximation capability, an RNN has a compact network size considerably smaller than that of an FNN. RNNs can also be used as associative memories to build attractors $\vec{y}_p$ from input-output association $\{\vec{x}_p, \vec{y}_p\}$. The recurrent backpropagation (BP) [97,8] and the RTRL [121] are usually used for RNN training.

III. The Hopfield Model

The Hopfield model [52] is the most popular dynamic model. It is a fully-interconnected RNN with J McCulloch-Pitts neurons [85], and is usually represented by using a J - J layered architecture, as illustrated in Figure 1. The input layer only collects and distributes feedback signals from the output layer. The network has a symmetric architecture with a symmetric zero-diagonal real weight matrix, that is, $w_{ij} = w_{ji}$ and $w_{ii} = 0$. The Hopfield network is biologically plausible since it functions like the human retina [68].

Architecture of the Hopfield network. Note that $w_{ii} = 0$ is represented by a dashed-line connection. $\vec{\varphi}(\cdot)$ and $\vec{\theta}$ are respectively a vector comprising the activation functions of all the neurons and a vector comprising the biases of all the neurons.

The Hopfield model operates in an unsupervised manner. The dynamics of the network are described by a system of nonlinear ordinary differential equations (ODE). The discrete form of the dynamics is defined by

$$net_i(t+1) = \sum_{j=1}^{J} w_{ji} x_j(t) + \theta_i, \tag{1}$$

$$x_i(t+1) = \varphi(net_i(t+1)), \tag{2}$$

where net_i is the weighted net input of the ith neuron, $x_i(t)$ is the output of the ith neuron, θ_i is a bias to the neuron, and $\varphi(\cdot)$ is the sigmoidal function. The discrete time variable t in (1) and (2) takes values $0, 1, 2, \ldots$.

Correspondingly, the continuous Hopfield model is given by

$$\frac{\mathrm{d}net_i(t)}{\mathrm{d}t} = \sum_{j=1}^{J} w_{ji} x_j(t) + \theta_i, \tag{3}$$

$$x_i(t) = \varphi(net_i(t)). \tag{4}$$

Notice that t in (3) and (4) denotes the continuous time variable.

In order to characterize the performance of the network, the concept of energy is introduced and the following energy function defined [50]:

$$E = -\frac{1}{2}\sum_{i=1}^{J}\sum_{j=1}^{J} w_{ij} x_i x_j - \sum_{i=1}^{J} \theta_i x_i$$

$$= -\frac{1}{2}\vec{x}^T \mathbf{W} \vec{x} - \vec{x}^T \vec{\theta}, \tag{5}$$

where $\vec{x} = (x_1, x_2, \ldots, x_J)^T$ is the input and state vector, and $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_J)^T$ is the bias vector.

1. Stability of the Hopfield model

The stability of the Hopfield network is asserted by Lyapunov's second theorem [31]. The dynamic equations of the Hopfield network actually implement a gradient-descent algorithm based on the cost function E [51]. The Hopfield model is asymptotically stable when running in asynchronous or serial update mode. In asynchronous or serial mode, only one neuron at a time updates itself in the output layer, and the energy function either decreases or stays the same after each iteration. However, if the Hopfield memory is running in the synchronous or parallel update mode, that is, all neurons update themselves at the same time, it may not converge to a fixed point, but may instead become oscillatory between two states [80,15,19]. The Hopfield network with the signum activation has a smaller degree of freedom compared to that using the sigmoidal activation, since it is constrained to changing the states along the the edges of a J-dimensional hypercube $\partial = \{-1, 1\}^J$. The use of sigmoidal functions helps in smoothing out some of the local minima.

2. Applications of the Hopfield model

The Hopfield network can be used for converting analog signals into the digital format, for associative memory, and for solving COPs. The Hopfield network can be used as an effective interface between analog and digital devices, where the input signals to the network are analog and the output signals are discrete values. The neural-based analog-to-digital (A/D) converter [113] adapts to compensate for initial device mismatches or long-term drifts. Associative memory is a major application of the Hopfield network. The fixed points in the network energy function are used to store feature patterns. When a noisy or incomplete pattern is presented to the trained network, a pattern in the memory is retrieved. This property is most useful for pattern recognition or pattern completion. The energy minimization ability of the Hopfield network is used to solve optimization problems. A large class of COPs can be expressed in this form of QP optimization problems, and thus can be solved using the Hopfield network. Signal estimation can also be formulated as QP problems. In this approach, the weights and biases are trained such that the global minimum state of the network energy corresponds to the optimum solution of the optimization problem.

IV. Analog Implementation of Hopfield Networks

The Hopfield network is especially suitable for analog VLSI implementation. The convergence time of the network dynamics

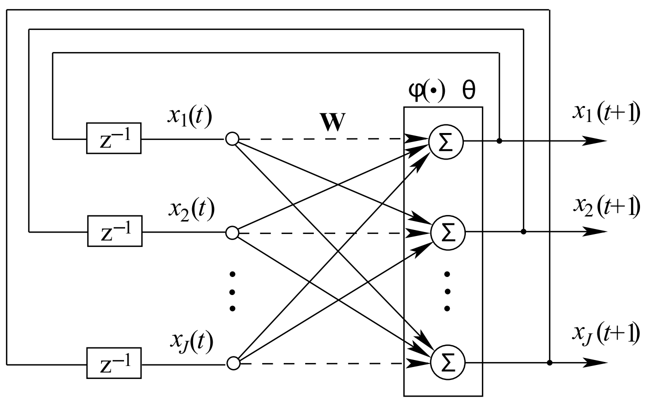

Figure: 1 Architecture of the Hopfield network. Note that $w_{ji} = 0$ is represented by a dashed-line connection. $\vec{\phi}(.)$ and $\vec{\theta}$ are respectively a vector comprising the activation functions of all the neurons and a vector comprising the biases of all the neurons.

is decided by a circuit time constant, which is of the order of a few nanoseconds [51]. The Hopfield network can be implemented by interconnecting an array of resistors, nonlinear operational amplifiers with symmetrical outputs, capacitors, and external bias current sources. Each neuron can be implemented by a capacitor, a resistor, and a nonlinear amplifier. A current source is necessary for representing the bias. The circuit structure of the neuron is shown in Figure 2. A drawback of the Hopfield network is the necessity to update the complete set of network coefficients caused by the signal change, and this causes difficulties in its circuit implementation.

By applying Ohm's law and Kirchhoff's current law to the *i*th neuron, we obtain

$$\tau_i \frac{\mathrm{d}u_i}{\mathrm{d}t} = -\alpha_i u_i + \left(\sum_{j=1}^{J} w_{ji} x_j + \theta_i \right) = -a_i u_i + \vec{w}_i^T \vec{x} + \theta_i, \tag{6}$$

where $x_i = v_i = \varphi(u_i)$ is the input signal, $\varphi(\cdot)$ being the sigmoidal function, $\tau_i = r_i C_i$ is the circuit time constant, r_i is a scaling resistance, $\alpha_i = \frac{r_i}{R_i}$ is a damping coefficient, $\frac{1}{R_i} = G_{i0} + \sum_{j=1}^{J} G_{ij}$ and G_{ij} is the conductance corresponding to R_{ij}, $w_{ji} = \frac{r_i}{R_{ij}} = r_i G_{ij}$ is the synaptic weight, $\theta_i = r_i I_i$ is the external bias signal, and $\vec{w}_i = (w_{1i}, w_{2i}, \ldots, w_{Ji})^T$ is the weight vector feeding neuron i. In Figure 2, the inverting output of each neuron is used to generate negative weights since G_{ij} is always positive.

The dynamics of the whole network can be written as

$$\vec{\tau} \frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = -\vec{\alpha}\vec{u} + \mathbf{W}^T \vec{x} + \vec{\theta}, \tag{7}$$

where the circuit time constant matrix $\vec{\tau} = \mathrm{diag}(\tau_1, \tau_2, \ldots, \tau_J)$, the interconnect-point voltage vector $\vec{u} = (u_1, u_2, \ldots, u_J)^T$, the damping coefficient matrix $\vec{\alpha} = \mathrm{diag}(\alpha_1, \alpha_2, \ldots, \alpha_J)$, the input and state vector $\vec{x} = (x_1, x_2, \ldots, x_J)^T$, the bias vector $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_J)^T$, and the $J \times J$ weight matrix $\mathbf{W} = [\vec{w}_1 \vec{w}_2 \ldots \vec{w}_J]$.

At the equilibrium of the system, $\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \vec{0}$, thus

$$\vec{\alpha}\vec{u} = \mathbf{W}^T \vec{x} + \vec{\theta}. \tag{8}$$

The dynamics of the network are controlled by C_i and R_{ij}. A sufficient condition for the Hopfield network to be stable is that $\mathbf{W}$ is a symmetric matrix with zero diagonal elements [51]. The stable states correspond to the local minima of the Liapunov function [51]

$$E(\vec{x}) = -\frac{1}{2} \vec{x}^T \mathbf{W} \vec{x} - \vec{x}^T \vec{\theta} + \sum_{i=1}^{J} \alpha_i \int_0^{x_i} \varphi^{-1}(\xi) \mathrm{d}\xi, \tag{9}$$

where $\varphi^{-1}(\cdot)$ is the inverse of $\varphi(\cdot)$. Equation (9) is a special case of the Cohen-Grossberg model [24].

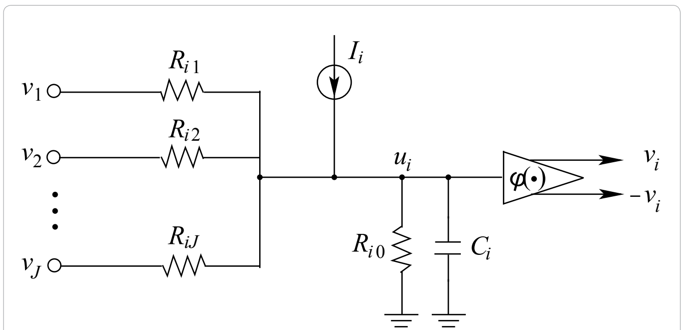

Figure 2: A circuit for neuron i in the Hopfield model. $w_{ji} = 0$ is the output voltage of neuron $\vec{\phi}(.)$ is the external bias current source for neuron $\vec{\theta}$ is the voltage at the interconnection point, C_i is a capacitor, and $R_{ik}, k = 0,1,\ldots,J$ are resistors. The sigmoidal function $\phi(.)$ is used as the transfer function of the amplifiers.

An inspection of (6) and (3) shows that α is zero in the basic Hopfield model. This term corresponds to the integral related to $\varphi^{-1}(\cdot)$ in (9). When the gain of the sigmoidal function $\beta \to \infty$, that is, when the sigmoidal function is selected as the hard-limiter function and the nonlinear amplifiers function as switches, the integral terms are insignificant. In this case, the circuit model approaches the basic Hopfield model. The stable states of the basic Hopfield network are the corners of the hypercube, namely, the local minima of (5) are in $\{-1,+1\}^J$ [51]. For large but finite gains, the sigmoidal function leads to a large positive contribution near the hypercube boundaries, but to a negligible contribution far from the boundaries. This leads to an energy surface that still has its maxima at the corners, but the minima slightly move inward from the corners of the hypercube. As β decreases, each minimum moves further inward and disappears one at a time. When β gets small enough, the energy minima start to disappear.

For the neuron given in Figure 2, the network parameters cannot be independently adjusted, since the coefficient α_i is nonlinearly related to all the weights w_{ij}. Many other circuit implementations of the Hopfield model have been proposed, among which the work in [74] deserves mentioning. In [74], α_i are removed by replacing the integrators and nonlinear amplifiers in the previous model by ideal integrators with saturation. This model is referred to as a linear system in a saturated mode (LSSM), which retains the basic structure of the Hopfield model and is easier to analyze, synthesize and implement than the Hopfield model. The energy function of the model is exactly the same as (5).

The Hopfield model is a network model which is most suitable for hardware implementation. Numerous circuits for the Hopfield model have been proposed, including analog VLSI and optical [33] implementations.

V. Associative Memory Models

Association is a salient feature of human memory. Associative memory is known as content-addressable memory (CAM). A pattern can be stored in memory through a learning process. For an imperfect input pattern, associative memory has the capability to recall the stored pattern correctly by performing a collective relaxation search. Associative memories can be either heteroassociative or autoassociative, where the input and output vectors range over different vector spaces or over the same vector space. Numerous associative memory models are available in the literature, among which linear associative memories [10,64,11], the brain-states-in-a-box (BSB) [12], and bidirectional associative memories [66,14] can be used as both autoassociative and heteroassociative memories, while the Hopfield model [50], the Hamming network [79], the Boltzmann machine [4] can only be used as autoassociative models.

The Hopfield model is a continuous-time, continuous-state dynamic associative memory model. Information retrieval is realized as an evolution of the system state. The binary Hopfield network is a well-known model for nonlinear associative memories. This is done by mapping a fundamental memory $\vec{x}$ onto a stable point of a dynamical system. The states of the neurons can be considered as short-term memories (STMs) while the synaptic weights can be treated as long-term memories (LTMs).

A. Hopfield Model: Storage and Retrieval

As an associative memory, the Hopfield network operates in two phases, namely storage and retrieval. Bipolar coding is often used for associative memory since bipolar vectors have a greater probability of being orthogonal than binary vectors. Given a set, $\{\vec{x}_p\}$, of N bipolar patterns, where $\vec{x}_p = \left(x_{p,1}, x_{p,2}, \ldots, x_{p,J}\right)^T$, $x_{p,i} = \pm 1$, which are to be stored in the network. These patterns are called the fundamental memories. Storage is performed by using a learning algorithm, while retrieval is based on the dynamics of the network.

1. Generalized Hebbian Rule: Conventional algorithms for associative storage are typically local algorithms based on the Hebbian rule [47], known as the outer product rule of storage in connection with associative learning. A generalized Hebbian rule for training the Hopfield network is defined by [50]

$$w_{ij} = \frac{1}{J}\sum_{p=1}^{N} x_{p,i} x_{p,j}, \forall i \neq j, \tag{10}$$

and $w_{ii} = 0$. In matrix form

$$\mathbf{W} = \frac{1}{J}\left(\sum_{p=1}^{N} \vec{x}_p \vec{x}_p^T - N\mathbf{I}_J\right), \tag{11}$$

where $\mathbf{I}_J$ denotes the $J \times J$ identity matrix. Equation (10) can be written in an incremental form

$$w_{ij}(t) = w_{ij}(t-1) + \frac{1}{J} x_{t,i} x_{t,j}, \forall i \neq j, t = 1, \ldots, N, \tag{12}$$

where $w_{ij}(0) = 0$. As such, the learning is completed after each pattern $\vec{x}_t$ in the pattern set is presented exactly once.

The generalized Hebbian rule is both local and incremental. It has an absolute storage capability of $N_{\max} = \frac{J}{2\ln J}$ [86], where the storage capability of an associative memory network is defined by the maximum number of fundamental memories, $N_{\max}$, that can be stored and retrieved reliably. For reliable retrieval, $N_{\max}$ is dropped to approximately $\frac{J}{4\ln J}$ [86]. The generalized Hebbian rule, however, suffers severe degradation and $N_{\max}$ decreases significantly, if the training patterns are correlated. For example, time series usually include significant correlations in the measurements of adjacent samples. Some variants of the Hebbian rule, such as the weighted Hebbian rule [10] and the Hebbian rule with decay [65], can improve the storage capabilities. When training associative memory networks using the classical Hebbian learning, an additional term called *crosstalk* may arise. When crosstalk becomes too large, spurious states other than the negative stored patterns appear [99]. The number of negative stored patterns is always equivalent to the number of stored patterns. Hebbian learning produces good results when the stored patterns are nearly orthogonal. This is the case when N bipolar vectors are randomly selected from R^J, and $N \ll J$. In practice, patterns are usually correlated and the incurred crosstalk may reduce the capacity of the network. The storage capability of the network is expected to decrease if the Hamming distance between the fundamental memories becomes smaller.

2. Pseudoinverse Rule: The pseudoinverse solution targets at minimizing the crosstalk between the stored patterns. The pseudoinverse rule uses the pseudoinverse of the pattern matrix, while classical Hebbian learning uses the correlation matrix of the patterns [94,61,60,99].

Denoting $\mathbf{X} = \left[\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\right]$, the autoassociative memory is defined as

$$\mathbf{X}^T\mathbf{W} = \mathbf{X}^T. \tag{13}$$

The pseudoinverse solution for the weight matrix is thus given as

$$\mathbf{W} = \left(\mathbf{X}^T\right)^{\dagger}\mathbf{X}^T. \tag{14}$$

The pseudoinverse rule, also called the projection learning rule, is neither incremental nor local. It involves inverting an $N \times N$ matrix, thus the training is very slow and impractical. However, there are iterative procedures consisting of successive local corrections [30] and incremental procedures for the calculation of the pseudoinverse [40].

The pseudoinverse solution performs better than the Hebbian learning when the patterns are correlated. Both the Hebbian and pseudoinverse rules are general-purpose methods for training associative memory networks that can be represented as $\mathbf{X}^T\mathbf{W} = \overline{\mathbf{X}}^T$, where $\mathbf{X}$ and $\overline{\mathbf{X}}$ are respectively the stored and associated pattern matrices. For an autoassociated pattern $\vec{x}_i$, the weights generated from the Hebbian learning projects the whole input space into the linear subspace spanned by $\vec{x}_i$. The projection, however, is not orthogonal. Instead, the pseudoinverse solution provides orthogonal projection to the linear subspace spanned by the stored patterns [99]. Theoretically, for $N < J$ and uncorrelated patterns, the pseudoinverse solution has a zero error, and the storage capability in this case is $N_{\max} = J - 1$ [60], [99].

The pseudoinverse rule is also adapted to sorting sequences of prototypes, where an input $\vec{x}_i$ leads to an output $\vec{x}_{i+1}$. The MLP with BP learning can be used to compute the pseudoinverse solution when the dimension J is large [60,99].

3. Improved Hebbian Learning Rule: An alternative local and incremental learning rule based on the Hebbian rule is given by [108], [109]

$$w_{ij}(t) = w_{ij}(t-1) + \frac{1}{J}\left[x_{t,i}x_{t,j} - h_{ji}(t)x_{t,i} - h_{ij}(t)x_{t,j}\right], \tag{15}$$

$$h_{ij}(t) = \sum_{u=1, u\neq i,j}^{J} w_{iu}(t-1)x_{t,u} \tag{16}$$

for $t = 1, 2, \ldots, N$, where $w_{ij}(0) = 0$ for all *i* and *j*, and h_{ij} is a form of local field at neuron *i*.

The improved Hebbian rule given by (15) and (16) has an absolute capacity of $\frac{J}{\sqrt{2\ln J}}$ for uncorrelated patterns. It also performs better than the generalized Hebbian rule for correlated patterns [108]. It does not suffer significant capacity loss when patterns with medium correlation are stored. It is shown in [109] that the Hebbian rule is the zeroth-order expansion of the pseudoinverse rule, and the improved Hebbian rule given by (15) and (16) is one form of the first-order expansion of the pseudo-inverse rule.

4. Perceptron-type Learning Rule: Training of the above rules can be completed in a single epoch. A learning problem in a Hopfield network with J units can be transformed into a learning problem for a perceptron of dimension $\frac{J(J+1)}{2}$ [99], and thus every learning algorithm for perceptrons can be transformed into a learning algorithm for Hopfield networks. Perceptron learning-based learning algorithms for storing bipolar patterns in Hopfield networks are discussed in [60]. They are simple, online, local algorithms. Perceptron learning-based algorithms work over multiple epochs and often reduce the error nonmonotonically over the epochs. The perceptron-type learning algorithm is given by [56,60]

$$w_{ij}(t) = w_{ij}(t-1) + \eta\left[x_{t,i}x_{t,j} - \frac{1}{2}(y_{t,i}x_{t,j} + y_{t,j}x_{t,i})\right], \quad (17)$$

where $\vec{y}_t = \mathrm{sgn}(\mathbf{W}\vec{x}_t)$, $t = 1,\ldots,N$, the learning rate η is a small positive number, and $w_{ji}(0)$'s are small random numbers or zero. From now on, the signum function sgn (x) takes 1 for $x \geq 0$ and −1 for $x < 0$. If all $w_{ji}(0)$'s are selected as zero, η can be selected as any positive number; otherwise, η can be selected as a number of the same order-of-magnitude or larger than the weights. This accelerates the convergence process. Note that $w_{ji}(t) = w_{ij}(t)$.

However, when the signum vector is not realizable, the perceptron-type algorithm does not converge but oscillates indefinitely. The perceptron-type rule can be viewed as a supervised extension of the Hebbian rule by incorporating a term for correcting unstable bits. For an RNN, the storage capability of the perceptron-type algorithm can reach the upper bound $N_{\max} = J$, for uncorrelated patterns. In [92], a complex perceptron learning algorithm has also been studied for associative memory by using complex weights and a decision circle in the complex plane for the output function.

An extensive experimental comparison between a perceptron-type learning rule [60] and the generalized Hebbian rule [50] has been made in [56] on a wide range of conditions on the library patterns: the number of patterns *N*, the pattern-density *p*, and the amount of correlation of the bits in a pattern, decided by block size *B*. The perceptron-type rule is found to be perfect in ensuring stability of the stored library patterns under all the evaluated conditions. In many cases, the perceptron-type rule works much better than the generalized Hebbian rule in correcting pattern errors. The uniformly random case (p = 0.5, B = 1) is the main exception, when the generalized Hebbian rule systematically equals or outperforms the perceptron-type rule in the error-correction experiments. An experimental comparison between the storage capacities of generalized Hebbian learning, improved Hebbian learning, and perceptron-type learning is given in [31].

5. Retrieval Stage: After the bipolar words have been stored, the network can be used for information retrieval. When a J-dimensional vector (bipolar word) $\vec{x}$, representing a corrupted or incomplete memory of the network, is presented to the network as its state, information retrieval is performed automatically according to the network dynamics given by (1) and (2), or (3) and (4). For hard-limiting activation function, the discrete form of the network dynamics can be written as

$$x_i(t+1) = \mathrm{sgn}\left(\sum_{j=1}^{J} w_{ji}x_j(t) + \theta_i\right), \quad i = 1,2,\ldots,J, \quad (18)$$

or in matrix form

$$\vec{x}(t+1) = \mathrm{sgn}(\mathbf{W}\vec{x}(t) + \vec{\theta}), \quad (19)$$

where $\vec{x}(0)$ is the input corrupted memory, and $\vec{x}(t)$ represents the retrieved memory at time *t*. The retrieval process continues until the state vector $\vec{x}$ remains unchanged. The convergent $\vec{x}$ is a fixed point or the retrieved memory.

B. Storage Capability

An upper bound on the storage capability of a class of RNNs with zero-diagonal weight matrix is derived deterministically in [3].

Theorem 5.1 (Upper Bound): *For any subset of N binary J-vectors, in order to find a corresponding zero-diagonal weight matrix* **W** *and a bias vector $\vec{\theta}$ such that these vectors are fixed points of the network*

$$\vec{x}_i = \mathrm{sgn}(\mathbf{W}\vec{x}_i + \vec{\theta}), \quad i = 1,2,\ldots,N, \quad (20)$$

one needs to have $N \leq J$.

Thus, the upper bound on the storage capability is $N_{\max} = J$. This bound is valid for any learning algorithm for RNNs with a zero-diagonal weight matrix. The Hopfield network, having a symmetric zero-diagonal weight matrix, is one such network, and as a result, the Hopfield network can at most stably store J patterns. This upper bound is too tight, since it requires that all the *N*-tuple subsets of bipolar J-vectors are retrievable. It is also noted in [115] that any two datums differing in precisely one component cannot be jointly stored as stable states in the Hopfield network.

When permitting a small fraction ε of a set of *N* bipolar J-vectors irretrievable, the upper bound approximates 2J when $J \to \infty$. This is given by a theorem derived from the function counting theorem [27,37,115,60]. This theorem is more general than Theorem 5.1, since there is no constraint on **W**. This RNN is sometimes referred to as the generalized Hopfield network. Both the theorems hold true irrespective of the updating mode, be it synchronous or asynchronous. The generalized Hopfield network with a general, zero-diagonal weight matrix has stable states in randomly asynchronous mode [82]. The asymptotic storage capacity of such a network using the perceptron learning scheme [103,83] gives a lower and an upper bound of the asymptotic storage capacity as $J-1$ and 2J, respectively. In a special case of the generalized Hopfield network with zero bias vector, some spectral strategies are used for constructing **W** [115]. All the spectral storage algorithms have a storage capacity of J for uncorrelated patterns [115].

When the Hopfield network is used as associative memory, there are cases where the fundamental memories are not stable. In addition, spurious states, which are other stable states different from the fundamental memories and their negative counterparts, may arise [6,99]. The Hopfield network trained with the generalized Hebbian rule can have a large number of spurious states, depending exponentially on *N*, even in the case when these vectors are orthogonal [16]. These spurious states are the corners of the unit hypercube that lie on or near the subspace spanned by the *N* fundamental memories. Presence of spurious states and limited storage capacity are the two major restrictions for the Hopfield network being used as associative memory. As long as *N* is small compared to the number of neurons J, the fundamental memories are proved to be stable in a probabilistic sense [6].

The Gardner conditions [37] are often used as a measure of the stability of the patterns. Associative learning can be designed to enhance the basin of attraction for every pattern to be stored by optimizing these conditions. The Gardner algorithm [37] combines maximal storage with a pre-defined level of stability for the patterns. Based on the Gardner conditions, the inverse Hebbian rule [26,25] is given by

$$w_{ij} = -\left(\mathbf{C}^{-1}\right)_{ij}, \quad (21)$$

where the correlation matrix $\mathbf{C} = \frac{1}{N}\sum_{p=1}^{N}\vec{x}_p\vec{x}_p^T$. Unlike the generalized Hebbian rule, which can only store unbiased patterns, the inverse-Hebbian rule is capable of storing N patterns, biased or unbiased, in a Hopfield network of N neurons. The patterns have zero basins of attraction, and **C** must be nonsingular. Matrix inversion can be implemented using a local learning algorithm [57]. The inverse-Hebbian rule provides ideal initial conditions for any algorithm capable of increasing the pattern stability. The quadratic Oja algorithm is a non-local generalization of the Oja's algorithm [93], and can be applied to adapt the weights so as to enhance the size of the basin of attraction of a subset of the stored patterns until the storage of the pattern subset is compromised. In [63], by using the Gardner algorithm for training the weights [37] and using a nonmonotonic activation function, the storage capacity of the network can be made to be always larger than $2J$ and reach its maximum value of $10.5J$. In [90,127], a continuous nonmonotonic activation function is used to greatly improve the storage capacity of the Hopfield network to approximately $0.4J$, and spurious states can be totally eliminated. When it fails to recall a memory, a chaotic behavior will occur.

The eigenstructure learning rule [74] is developed for continuous-time Hopfield models in linear saturated mode. The design method allows linear combinations of the prototype vectors to be stored as asymptotically stable equilibrium points as well. The storage capacity is better than those of the pseudoinverse solution [94] and the generalized Hebbian rule [50]. The method has been extended to discrete-time neural networks in [88]. A quantum computational learning algorithm that combines quantum computation with the Hopfield network is given in [116]. The quantum associative memory offers a storage capacity of $O(2^J)$. It employs simple spin-1/2 (two-state) quantum systems and represents patterns as quantum operators.

C. Multilayer Perceptrons as Associative Memories

Most RNN-based associative memories have low storage capacity as well as poor retrieval ability. RNNs exhibit asymptotic behavior and as such are difficult to analyze. MLP-based autoassociative memories with equal number of input and output nodes have been introduced to overcome these limitations [19,122].

The recurrent correlation associative memory (RCAM) [19] is A J - N - J MLP whose outputs are fed back to their respective inputs. At each instant, the hidden layer computes an intermediate mapping, while the output layer completes an association of the input pattern to an approximate prototype pattern. The approximated pattern is fed back to the network and the process continues until convergence to a prototype is achieved. The weight matrix between the input and hidden layers $\mathbf{W}^{(1)}$, a $J \times N$ matrix, is made up of the N J-bit bipolar memory patterns $\vec{x}_i, i = 1, 2, \ldots, N$, that is, $\mathbf{W}^{(1)} = \left[\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\right]$. The weight matrix between the hidden and output layers $\mathbf{W}^{(2)} = \left[\mathbf{W}^{(1)}\right]^T$.

The activation function for the ith neuron in the hidden layer is $\varphi_i(\cdot)$, and the activation function at the output layer is the signum function. At the presentation of pattern $\vec{x}$, the network evolution is given by

$$\vec{x}(t+1) = \mathrm{sgn}\left(\sum_{j=1}^{N} \varphi_j\left(\vec{x}_j^T \vec{x}(t)\right) \cdot \vec{x}_j\right). \tag{22}$$

The correlation of two patterns, $\vec{x}_1^T \vec{x}_2 = J - 2d_{\mathrm{H}}\left(\vec{x}_1, \vec{x}_2\right)$, where $d_{\mathrm{H}}(\cdot)$ is the Hamming distance between two binary vectors within, which is the number of bits in the two vectors that do not match each other.

When all $\varphi_i(net) = \varphi(net)$, $\varphi(net)$ being any continuous, monotonic nondecreasing weighting function over $[-J, J]$, the RCAM (22) is proved to be asymptotically stable in both the synchronous and asynchronous update modes [19]. This property is especially suitable for hardware implementation, since there are faults in the manufacture of any physical device. Based on this, a family of RCAMs are proposed which possess the asympototical stability. When all $\varphi_i(net) = net$, the RCAM model is equivalent to the correlation-matrix associative memory [64,11], that is, the connection can be written as $\mathbf{W} = \sum_{p=1}^{N} \vec{x}_p \vec{x}_p^T$. By suitably selecting $\varphi_i(\cdot)$, the model is reduced to some existing associative memories, which have a storage capacity that grows polynomially or exponentially with J [19]. In particular, when all $\varphi_i(net) = a^{net}$ with radix $a > 1$, an exponential correlation associative memory model (ECAM) [19] is obtained. The exponential activation function stretches the ratios among the weights and makes the largest weight more overwhelming. This significantly increases the storage capacity. The ECAM model exhibits an asymptotic storage capacity that scales exponentially with J [19]. Under the noise-free condition, this storage capacity is 2^J patterns [19]. A VLSI chip for the ECAM model has been fabricated and tested [19]. The multi-valued RCAM [20] can increase the error correction capability with large storage capability and less interconnection complexity.

The local identical index (LII) [122] is an autoassociative memory model that uses the J - N - J MLP architecture. The weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ are the same as those defined in the RCAM. It utilizes the signum activation function and biases in both the hidden and output layers. The LII model utilizes the local characteristics of the fundamental memories through two metrics, namely, the global identical index (GII) and the LII. Based on the minimum Hamming distance as the underlying association principle, the scheme can be viewed as an approximate Hamming decoder. The LII model exhibits low structural as well as operational complexity. It is a one-shot associative memory, and can accommodate up to 2^J prototype patterns. The LII model outperforms the LSSM [74] and its discrete version [88] in recognition accuracy at the presentation of the corrupted patterns, controlled by using the Hamming distance. It can successfully associate input patterns that are even loosely correlated with the corresponding prototype pattern.

For a J - J_2- J MLP-based autoassociative memory, the hidden layer is a bottleneck layer with less nodes, $J_2 < J$. This bottleneck layer is used to discover a limited set of unique prototypes that cluster the training set. When a linear activation function is employed, the MLP-based autoassociative memory has a serious limitation, namely, it does not allow the user to control the granularity of the clusters formed. Due to lack of cluster competition mechanism, different clusters which are close to one another in the input space may merge. This problem can be overcome by using the sigmoidal activation function.

D. The Hamming Network

The Hamming network [79] is a straightforward associative memory. It calculates the Hamming distance between the input pattern and each memory pattern, and selects the memory with the smallest Hamming distance. The network output is the index of a prototype pattern and thus the network can be used as a pattern classifier. The Hamming network is used as classical Hamming decoder or Hamming associative memory. It provides the minimum-Hamming-distance solution.

The Hamming network has J - N - N layered architecture, as illustrated in Figure 3. The third layer is called *memory layer*, each of whose neurons corresponds to a prototype pattern. The input and hidden layers are feedforward, fully-connected, while each hidden node has a feedforward connection to its corresponding node in the memory layer. Neurons in the memory layer are fully interconnected, and form a competitive, winner-take-all (WTA) subnetwork known as *MAXNET*. The MAXNET responds to an input pattern by generating a winner-neuron through iterative competitions. The Hamming network is implicitly recurrent due to the interconnections in the memory layer.

Architecture of the J - N - N Hamming network. The activation function at all the units is the signum function. The number of neurons in the memory layer, N, corresponds to the number of stored patterns. $\mathbf{T} = \left[t_{kl}\right]$, $k, l = 1, \ldots, N$.

The second layer generates matching scores $J - d_{\mathrm{H}}\left(\vec{x}, \vec{x}_i\right)$, $i = 1, \ldots, N$, for pattern $\vec{x}$. These matching scores ranges from 0 to J. The unit with the highest matching score corresponds to the stored pattern that best matches the input. The weights between the input and hidden layers and the biases of the hidden layer are respectively set as $w_{ij} = \frac{x_{j,i}}{2}$, and $\theta_j^{(2)} = \frac{J}{2}$, $j = 1, \ldots, N$, $i = 1, \ldots, J$. All the thresholds and the weights t_{kl} in the MAXNET are fixed. The thresholds are set as zero. The weights from each node to itself are set as unity and weights between nodes are inhibitory, that is, $t_{kl} = 1$ for $k = l$ and $-\varepsilon$ otherwise, where $\varepsilon < \frac{1}{N}$.

When a binary pattern is presented to the network, the network first generates an initial input for the MAXNET

$$y_j(0) = \varphi\left(\sum_{i=1}^{N} w_{ij} x_i - \theta_j^{(2)}\right), \quad j = 1, \ldots, N, \tag{23}$$

where $\varphi(\cdot)$ is a threshold-logic nonlinear function. The input pattern is then removed and the MAXNET continues the iteration

$$y_j(t+1) = \varphi\left(y_j(t) - \varepsilon \sum_{k=1, k \neq j}^{N} y_k(t)\right), \quad j = 1, \ldots, N, \tag{24}$$

until the output of only one node is positive. This node corresponds to the selected class.

The Hamming network implements the minimum error classifier,

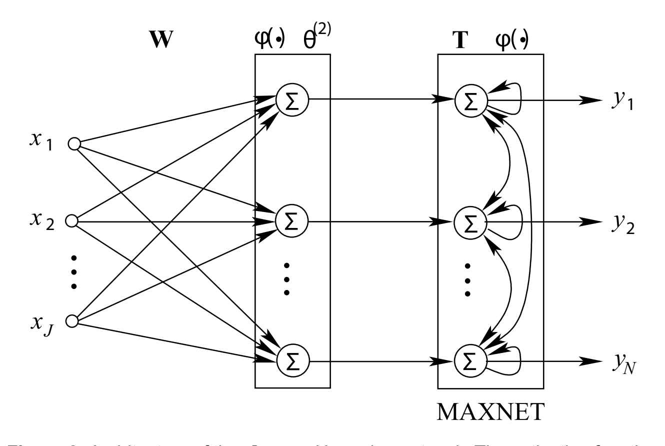

Figure 3: Architecture of the J - N - N Hamming network. The activation function at all the units is the signum function. The number of neurons in the memory layer, N, corresponds to the number of stored patterns. $\mathbf{T} = [t_{kl}]$, k, $l = 1, \ldots, N$.

when the bit errors are random and independent. For the J - N - N Hamming network, there are $J \times N + N^2$ connections, while for the Hopfield network the number of connections is J^2. When $J \gg N$, the number of connections in the Hamming network is significantly less than that in the Hopfield network. In addition, the Hamming network offers a storage capacity that is exponential in the input dimension [44], and it does not have any spurious state. Under the noise-free condition, the Hamming network has a storage capacity of 2^J patterns [44]. For a sufficiently large but finite radix α, the ECAM operates as a Hamming associative memory [44]. However, the Hamming network suffers from difficult hardware implementations and slow retrieval speed. Based on the correspondence between the Hamming network and the ECAM [44], the ECAM can be used to compute the minimum Hamming distance, in a distributed fashion by analog exponentiation and thresholding devices. The two-level Hamming network [54] generalizes the Hamming memory by providing for local Hamming distance computations in the first level and a voting mechanism in the second level. It allows for a much more practical hardware implementation and a faster retrieval.

VI. Simulated Annealing

SA is a process simulating the anealing of certain alloys of metal. The Metropolis algorithm is a simple method for simulating the evolution to the thermal equilibrium of a solid for a given temperature [87], and the SA algorithm [62] extends the Metropolis algorithm by changing the temperature from high to low. SA is a general, serial algorithm for finding a global minimum for a continuous function [62]. The solutions by this technique are close to the global minimum within a polynomial upper bound for the computational time. Some parallel algorithms for SA have been proposed aiming to improve the accuracy of the solutions [29]. SA is a successful method, for example, used for the layout of integrated circuits [105].

A. Classic Simulated Annealing

According to statistical thermodynamics, the probability of a physical system being in state with energy E at absolute temperature T satisfies the Boltzmann or Boltzmann-Gibbs distribution. At high T, the Boltzmann distribution exhibits uniform preference for all the states, regardless of the energy. When T approaches zero, only the states with minimum energy have nonzero probability of occurrence. In SA, at high T, the system ignores small changes in the energy and approaches the thermal equilibrium rapidly, that is, it performs a coarse search of the space of global states and finds a good minimum. As T is lowered, the system responds to small changes in the energy, and performs a fine search in the neighborhood of the already determined minimum and finds a better minimum. At T = 0, any change in the system states does not increase the energy, and thus, the system must reach equilibrium if T = 0.

When performing SA, theoretically a global minimum is guaranteed to be reached with a high probability. The artificial thermal noise is gradually decreased in time. The computational temperature T controls the magnitude of the perturbations of the energy function $E(\vec{x})$. The probability of a state change is determined by the Boltzmann distributions of the energy difference between the two states, $P = \mathrm{e}^{-\frac{\Delta E}{T}}$. The probability of uphill moves in the energy function ($\Delta E > 0$) is large at a high T, and is low at a low T. SA allows uphill moves in a controlled fashion: it attempts to improve on greedy local search by occasionally taking a risk and accepting a worse solution. SA can be performed as given by Algorithm 1 [62], [23].

```
Algorithm 1 (SA)
1. Initialize the system configuration.
2. Randomize $\vec{x}(0)$.
3. Initialize $T$ with a large value.
4. Apply random perturbations to the output state of neurons $\vec{x} = \vec{x} + \Delta\vec{x}$.
5. Evaluate $\Delta E(\vec{x}) = E(\vec{x} + \Delta\vec{x}) - E(\vec{x})$;
(a) If $\Delta E(\vec{x}) < 0$, keep the new state;
(b) Otherwise, accept the new state with the probability $P = e^{-\frac{\Delta E}{T}}$.
6. Repeat Steps 4 and 5 until the number of accepted transitions becomes below a threshold level.
7. Set $T = T - \Delta T$.
8. Repeat Steps 4 through 7 until $T$ is small enough.
```

The cooling schedule for T is critical to the efficiency of SA. If T cools too fast, a premature convergence to a local minimum may occur. In contrast, if it is too slow, the algorithm is very slow to converge. Based on a Markov chain analysis on the SA process, T must be decreased according to $T(t) \ge \frac{T_0}{\ln(1+t)}, t = 1, 2, \ldots$ to ensure convergence to the global minimum with probability one [38], where T_0 is the initial temperature. This is practically too slow. In practice, one usually applies, in Step 7, a fast schedule $T(t) = \alpha T(t-1)$ with $0.85 \le \alpha \le 0.96$, to achieve a suboptimal solution.

B. Variants of Simulated Annealing

Boltzmann annealing is too slow for a reliable cooling schedule. Many methods, such as Cauchy annealing [111], simulated re-annealing [55], generalized SA [114], and the SA algorithm with known global value [81], have been proposed to accelerate the SA search. There are also global optimization methods that make use of the idea of annealing [98,100]. Some VLSI designs of SA are also available [68].

In Cauchy annealing [111], the Cauchy or Cauchy-Lorentz distribution is used to replace the Boltzmann distribution. The infinite variance provides a better ability to escape from local minima and allows for the use of faster schedules, such as T decreasing according to $T(t) = T_0/t$. A stochastic neural network trained with Cauchy annealing is called the Cauchy machine. In simulated re-annealing [55], T decreases exponentially with t, that is, $T = T_0 e^{-c_1 t^{1/J}}$, where $\boldsymbol{c}_1 > 0$ is a constant and J is the dimension of the input space. The introduction of re-annealing also permits adaptation to changing insensitivities in the multi-dimensional parameter space. The generalized SA [114] generalizes both Cauchy annealing [111] and Boltzmann annealing [62] within a unified framework inspired by the generalized thermostatistics. An SA algorithm under the assumption of known global value $\boldsymbol{E}^*$ has been investigated in [81]. The algorithm is the same as the classical SA except that at each iteration a uniform random point is generated over a sphere, whose radius depends on the difference $E(\vec{x}(t)) - E^*$, and T is also decided by this difference. The algorithm has guaranteed convergence and an upper bound for the expected first hitting time, i.e. the expected number of iterations before reaching the global optimum value within a given accuracy ε, is established. The idea of annealing is a general optimization principle, which can be extended by using fuzzy logic. In the fuzzy annealing scheme [98], fuzzification is performed by adding an entropy term. The fuzziness at the beginning of the entire procedure is used to prevent the optimization process getting stuck at an inferior local optimum. The fuzzy annealing scheme results in an increase in the computation speed by a factor of one hundred or more compared to SA [98].

SA makes a random search on the energy surface. Deterministic annealing [101,100] is a method where randomness is incorporated into the energy or cost function, which is then deterministically optimized at a sequence of decreasing temperature. The approach is derived within the framework of information theory and probability theory. Deterministic annealing has been used for nonconvex optimization problems such as clustering, MLP training, and RBFN training [101,100].

VII. Combinatorial Optimization Problems

Any problem that has a large set of discrete solutions and a cost function for rating those solutions relative to one another is a COP. COPs are known to be NP-complete [110]. In COPs, the number of solutions grows exponentially with n, the size of the problem, at $O(n!)$ or $O\left(e^n\right)$ so that no algorithm can find the global minimum solution in a polynomial computational time. The goal for COPs is to find an optimal solution or sometimes a nearly optimal solution.

The traveling salesman problem (TSP) is a well-known COP [50]: Assuming there are n cities, find the shortest possible tour such that a salesman visits every city exactly once and then returns to the starting point. There are $(n-1)!/2$ possible tours. The Hopfield network was the first neural network used for the TSP, and it achieves a near-optimum solution [52]. Routing of wires on a printed circuit board (PCB) is a typical TSP. The location-allocation problem is another COP.

The Hopfield network can be effectively used to deal with COPs with the objective functions of the linear or quadratic form, linear equalities and/or inequalities as the constraints, and binary variable values so that the constructed energy function can be of quadratic form. For example, a class of COPs including the location-allocation problem is formulated in [84,31]. To make use of the Hopfield network, one needs first to convert the COP into a constrained real optimization problem and solve the latter using the penalty method. The total cost E is the weighted sum of the objective term and terms associated with the constraints. When all the constraint terms are all zeros, the solution is a feasible one. The weights for individual constraints can be tuned for an optimal or good solution. The cost E should the same form as that of the energy function of the Hopfield network, given by (5). For the penalty method, there is always a compromise between good-quality solution and convergence. For a feasible solution, the weighting factors for the penalty terms should be sufficiently large, which however causes the constraints on the original problem to become relatively weaker, resulting in a deterioration of the quality of the solution. A trial-and-error process for choosing some of the penalty parameters is inevitable in order to obtain feasible solutions. Moreover, the gradient-descent method often leads to local minima of the energy landscape. An inequality constraint can be expressed as an equality constraint by introducing some slack variables (slack neurons).

A. Escaping Local Minima for Combinatorial Optimization Problems

SA [62] is a popular method for any optimization problem including COPs. However, due to its Monte Carlo nature, SA would require even more iterations than complete enumeration, for some problems, in order to guarantee convergence to an exact solution. For example, for an n-city TSP, SA using the logarithmic cooling schedule needs a computational complexity of $O\left(n^{n^{2n-1}}\right)$, which is far more than $O((n-1)!)$ for complete enumeration and $O\left(n^2 2^n\right)$ for dynamic programming [18,1]. Thus, one has to apply heuristic fast cooling schedules to improve the convergence speed.

The Hopfield network is more desirable for solving COPs that can be formulated into quadratic functions. The Hopfield network converges very fast, and it can also been easily implemented using RC circuits. However, due to its gradient-descent nature, it always gets trapped at the nearest local minimum of the initial random state. Some strategies are necessary for escaping from the local minima.

1. Gain Annealing: To escape from the local minima, a popular strategy is to change the sigmoidal gain β, by starting from a low gain and gradually increasing it. When β is low, the energy landscape is smooth, and the algorithm can easily find a good local minimum. As β increases, more details of the energy landscape is revealed, and the algorithm can find a better solution. This is analogous to the cooling process of SA [62], and this process is usually called *gain annealing*. In the limit, when $\beta \to \infty$, the hypobolic tangent function becomes the signum function.

2. Balancing Objective and Constraint Terms: In order to use the Hopfield network for solving optimization problems, the cost function can be constructed as

$$E(\vec{x}) = \lambda_o E_o(\vec{x}) + \lambda_c E_c(\vec{x}), \tag{25}$$

where E_0 and E_c represent the objective and constraint terms, respectively, and λ_o, $\lambda_c > 0$ are their corresponding weights.

By adaptively adjusting the balance between the constraint and objective terms, the network can avoid falling into a local minimum and continue to update in a gradient-descent direction of energy [119]. At a local minimum of *E*, one always has

$$\frac{\partial E}{\partial x_i}\Delta x_i = \left(\lambda_o \frac{\partial E_o}{\partial x_i} + \lambda_c \frac{\partial E_c}{\partial x_i}\right)\Delta x_i \geq 0, \quad i = 1, \ldots, J. \tag{26}$$

The method for escaping from the local minimum is to adjust λ_o and/or λ_c so that (26) is not satisfied for at least one neuron. The energy of the network decreases with the state change of the *i*th neuron. Thus, the learning eliminates the local minimum that the network would fall into. The minimum found is not only a minimum of the total energy function, but also the minima of both the constraint and objective terms. This minimum is always a global or a near-global one.

B. Combinatorial Optimization Problems with Equality and Inequality Constraints

The Hopfield network can be used to solve COPs under equality as well as inequality constraints, as long as the constructed energy function is of the form (5). Some extensions to the Hopfield model are necessary in order to handle both equality and inequality constraints [113,2]. In the extended Hopfield model [2], each inequality constraint is converted to an equality constraint by introducing an additional variable managed by a new neuron, known as the *slack neuron*. Each slack neuron is connected to the initial neurons, where their corresponding variables occur in its linear combination. The extended Hopfield model has the drawback of being frequently stabilized in neuron states far from the suitable ones, i.e., zero and one. To deal with this drawback, a new penalty energy term is derived to significantly reduce the number of neurons with unsuitable states [71]. The derived rules introduce competitions between the variables involved into the same constraint. The competitive mechanism also deals with the upper bounded inequality constraints. This mechanism has the capacity to distribute the neurons into the two states.

The *k*-out-of-*n* design rule [112] is used to facilitate the construction of the network energy functions for multiple *k*-out-of-*n* equality constraints, $\sum_{i=1}^{n} x_i = k$, and inequality constraints, $\sum_{i=1}^{n} x_i \leq k$. For *k*-out-of-*n* inequality constraints, slack neurons are used. A generalized architecture for the Hopfield network with *k*-out-of-*n* design is achieved by adding to the original J neurons one adjustable neuron associated with each given constraint [77] so as to improve the quality of the solutions. This architecture also applies when slack neurons are used for inequality constraints.

VIII. Chaotic Neural Networks

An RNN such as the Hopfield network, when introduced with chaotic dynamics, is sometimes called a *chaotic neural network*. The chaotic dynamics are temporarily generated for searching and self-organizing, and eventually vanish with autonomous decrease of a bifurcation parameter corresponding to the temperature in the SA process. Thus, the chaotic neural network gradually approaches to a dynamical structure of the RNN. Since the operation of the chaotic neural network is similar to that of SA, not in a stochastic way but in a deterministically chaotic way, the operation is known as *chaotic SA (CSA)*. More specifically, the transiently chaotic dynamics are used for searching a basin containing the global optimum, followed by a stable and convergent phase when the chaotic noise decreases to zero. As a result, the chaotic neural network has a high ability for searching globally optimal or near-optimal solutions [18]. SA searches all the possible states by temporally changing the probability distributions, while CSA searches a possible fractal subspace with continuous states by temporally changing invariant measures that are determined by its dynamics. Thus, the search region in CSA is very small compared with the state space, and CSA can perform an efficient search.

A small amount of chaotic noise can be injected to the output of the neurons and/or to the weights during the operation of the Hopfield network. In [46], a chaotic neural network is obtained by adding chaotic noise to each neuron of the discrete-time continuous-output Hopfield network and gradually reducing the noise so that it is initially chaotic, but eventually convergent. The chaotic neural network introduced in [18,5] is obtained by adding a negative self-coupling to the Hopfield network. By gradually removing the self-coupling, the transient chaos is used for searching and self-organizing. The updating rule for the chaotic neural network is given by [18]

$$net_i(t+1) = \left(1 - \frac{\alpha_i}{\tau_i}\right) net_i(t) + \frac{1}{\tau_i}\left(\vec{w}_i^T \vec{x} + \theta_i\right) - c(t)\left(x_i - \gamma\right), \tag{27}$$

$$x_i(t) = \varphi\left(net_i(t)\right), \tag{28}$$

where $c(t+1) = \beta c(t)$, $\beta \in [0,1]$, the bias $\gamma > 0$, and other parameters are the same for (6). A large initial value of $c(t)$ is used so that the self-coupling is strong enough to generate chaotic dynamics for searching the global minima. The damping of $c(t)$ produces successive bifurcations so that the neurodynamics eventually converge from strange attractors to a stable equilibrium point.

In [118], the CSA approach is derived by varying the time step Δt of an Euler discretization of the Hopfield network. The time step is analogous to the temperature parameter in SA, and the method starts with a large Δt, where the dynamics are chaotic, and gradually decreases it. When $\Delta t \to 0$, the system approaches the Hopfield model (6) and minimizes its energy function. When $\Delta t = 1$, the Eulerdiscretized Hopfield network is identical to the chaotic neural network given in [18]. The simulation results for COPs are comparable to that of the method proposed in [18]. Many chaotic approaches [18,118,46] can be unified and compared under the framework of adding an extra energy term E_{CSA} into the original computational energy (9) of the Hopfield model [67]. The extra energy term modifies the original Hopfield energy landscape to accommodate transient chaos. This driving force is diminished as $E_{\text{CSA}} \to 0$ when $\lambda(t) \to 0$.

CSA has a better search ability for solving COPs compared to SA. However, a number of network parameters must be subtly adjusted so as to guarantee the convergence of the chaotic network. Unlike SA, CSA may not find a globally optimal solution no matter how slowly the annealing is carried out, because the chaotic dynamics are completely deterministic. Stochastic CSA [120] is proposed as a combination of the SA and the CSA [18] by using a noisy chaotic neural network. Stochastic CSA restricts the random search to a subspace of chaotic attracting sets, and this subspace is much smaller than the entire state space searched by SA. Simulation results show that stochastic CSA performs more efficiently than SA and CSA [18] for the TSP and the channel assignment problem.

IX. Hopfield Networks for Other Optimization and Sign

The least squares (LS) problem is a typical method for optimization and signal processing. Matrix inversion can be performed using the Hopfield network [57]. Given a nonsingular $n \times n$ matrix **A**, the energy function can be defined by $\| \mathbf{AV} - \mathbf{I} \|_F^2$, where **V** denotes the inverse of **A** and the subscript F denotes the Frobenius norm. This energy function can be decomposed into *n* energy functions, and *n* similar networks are required, each optimizing an energy function. This method can be used to solve a system of *n* linear equations with *n* variables, $\mathbf{A}\vec{x} = \vec{b}$, where $\mathbf{A} \in R^{n \times n}$ and $\vec{x}$, $\vec{b} \in R^n$, if the set of linear equations (SLE) has a unique solution, that is, **A** is nonsingular. In [17], this SLE is solved by using a continuous Hopfield network with *n* nodes. The Hopfield network is designed to minimize the energy function $E = \frac{1}{2} \| \mathbf{A}\vec{x} - \vec{b} \|^2$, and the activation function is selected as a linear transfer function. This method is also applicable when there exists infinitely many solutions and **A** is singular. Another neural LS estimator that uses continuous Hopfield network and a nonlinear activation function has been proposed in [36]. A Hopfield network with linear transfer functions augmented by an additional feedforward layer can be used to solve an SLE [117] and to compute the pseudoinverse of a matrix [72]. The resultant augmented linear Hopfield network can be used to solve constrained LS optimization problems.

The linear programming (LP) network [113] is designed based on the Hopfield model for solving LP problems

$$\min \vec{a}^T \vec{x} \tag{29}$$

subject to

$$\vec{d}_j^T \vec{x} \ge h_j \tag{30}$$

for $j = 1, \ldots, M$, where $\vec{d}_j = \left(d_{j,1}, d_{j,2}, \ldots, d_{j,J} \right)^T$, $\mathbf{D} = \left[d_{j,i} \right]$ is an $M \times J$ matrix, and h_j is a constant. Each inequality constraint is modeled by a slack neuron. The network contains a signal plane with J neurons and a constraint plane with *k* neurons. The energy function decreases until the net reaches a state where all time derivatives are zero. With some modifications, the LP network [113] can be used to solve LSE problems [124]. In [28], a circuit based on a modification of the LP network [113] is designed for computing the discrete Hartley transform (DHT). A circuit for computing the discrete Fourier transform (DFT) is obtained by simply adding a few adders to the DHT circuit. The circuits can compute the DHT and DFT within circuit time constants of the order of nanoseconds. The stability, computational speed, and computational accuracy of the LP network depends substantially on the location of the eigenvalues of the matrix product $\mathbf{D}_g^T \mathbf{D}_f$ (or $\mathbf{D}_f \mathbf{D}_g^T$), where $\mathbf{D}_g$ and $\mathbf{D}_f$ are, respectively, approximations to **D** at the signal and constraint planes of the network in the nonideal case [124,41].

X. Multistate Hopfield Networks

The multilevel Hopfield network [34] and the complex-valued multistate Hopfield network [58,91] are two direct generalizations of the Hopfield network. The multilevel Hopfield network uses neurons with an increasing multistep function as the activation function, while the complex-valued multistate Hopfield network uses a multivalued complex-signum function as the activation function. The complex-valued Hopfield-like network [49], like the complex-valued multistate Hopfield network [58,91], uniformly quantizes the phase of the net input of each neuron and disregards the corresponding amplitude, but uses different dynamic equations.

The use of multistate neurons leads to a network architecture that is significantly smaller than that of the conventional Hopfield network, and hence, a simple hardware implementation. The reduction in the network size is highly desirable in large-scale applications such as image restoration and the TSP. In addition, the complex-valued multistate Hopfield network is also more efficient and convenient than the Hopfield network in the manipulation of complex-valued signals.

A. Multilevel Hopfield Networks

In [34], a multilevel Hopfield network for associative memory is obtained by replacing the threshold activation function with an increasing multistep function and modifying the generalized Hebbian rule. The storage capability of the J-neuron multilevel Hopfield network is proved to be $O\left(J^3\right)$ bits, which is of the same order as that of the Hopfield network [3]. For a network of J neurons, the number of patterns that the multilevel network can reliably store and retrieve may be considerably less than that for the Hopfield network, since each codeword typically contains more bits.

The multilevel sigmoidal function is typically used as the activation function in the multilevel Hopfield network [128,107,129]. In [107], a storage procedure for the multilevel Hopfield network in the synchronous mode has been developed based on the LS solution, and also examined by using an image restoration example. A retrieval procedure for the multilevel Hopfield network has been proposed and applied to COPs such as the TSP in [32]. In [128], a multilevel Hopfield-like network is obtained by using a new neuron with self-feedback and the multilevel sigmoidal activation function. The multilevel model has been applied for A/D conversion, and a circuit implementation for the neural A/D converter has been fabricated [128].

B. Complex-valued Multistate Hopfield Networks

The complex-valued multistate Hopfield network [58,91] adopts the multivalued complex-signum activation function, defined as an L-stage phase quantizer for complex numbers

$$\widehat{\text{csign}}(u)=\begin{cases} z^0, & \arg(u)\in[0,\phi_0) \\ z^1, & \arg(u)\in[\phi_0,2\phi_0) \\ \vdots & \\ z^{L-1}, & \arg(u)\in[(L-1)\phi_0,L\phi_0) \end{cases}, \tag{31}$$

where $z=\mathrm{e}^{j\phi_0}$ with $\phi_0=\frac{2\pi}{L}$ is the Lth root of unity. Each state takes one of the equally spaced L points on the unit circle of the complex plane.

Similar to the Hopfield network, the system dynamics for the J-neuron network are defined by

$$net_i(t)=\sum_{k=1}^{J} w_{ki}x_k(t), \tag{32}$$

$$x_i(t+1)=\text{csign}\left(net_i(t)\cdot z^{\frac{1}{2}}\right) \tag{33}$$

for $i=1,\ldots,J$, where the factor $z^{\frac{1}{2}}=\mathrm{e}^{j\frac{\phi_0}{2}}$ places the resulting states in angular centers of each sector. A sufficient condition for the stability of the dynamics is that the weight matrix is Hermitian with nonnegative diagonal entries, that is, $\mathbf{W}=\mathbf{W}^H$, $w_{ii}\geq 0$ [58]. The energy can be defined as

$$E(\vec{x})=-\frac{1}{2}\vec{x}^H\mathbf{W}\vec{x}. \tag{34}$$

In order to store a set of N patterns, $\{\vec{x}_i\}\subset\{0,1,\ldots,L-1\}^J$, $\vec{x}_i$ is first encoded to its complex memory state $\vec{\varepsilon}_i=\left(\varepsilon_{i,1},\ldots,\varepsilon_{i,J}\right)^T$ with

$$\varepsilon_{i,j}=z^{x_{i,j}}. \tag{35}$$

The decoding of a memory state to a pattern is the inverse of (35). The complex-valued pattern set $\{\vec{\varepsilon}_i\}$ can be stored in weights by the generalized Hebbian rule [50]

$$w_{ji}=\frac{1}{J}\sum_{\mu=1}^{N}\varepsilon_{\mu,i}\varepsilon^*_{\mu,j},\quad i,j=1,2,\ldots,J, \tag{36}$$

where superscript* denotes the conjugate operation. Thus, $\mathbf{W}$ is Hermitian.

The storage capability of the memory, $N_{\max}$, is dependent upon the resolution L for an acceptable level of the error probability P_{max}. As L is increased, $N_{\max}$ decreases, but each pattern contains more information.

Due to the use of the generalized Hebbian rule, the storage capacity of the network is very low and the problem of spurious memories is very pronounced. In [70], a gradient descent-based learning rule has been proposed to enhance the storage capacity and also reduce the number of spurious memories. In [91], an LP method has been proposed for storing into the network each pattern in an integral set $M\subset\{0,1,2,\ldots,L-1\}^J$ as a fixed point. The LP method significantly reduces the number of spurious memories, and provides better results in case of noisy gray-level image reconstruction.

Since gray-scale images can be represented by integral vectors, reconstruction of such images from their distorted versions constitutes a straightforward application of multistate associative memory. The complex-valued Hopfield network is particularly suitable for interpreting images transformed by two-dimensional Fourier transform and two-dimensional autocorrelation functions [58].

The complex-valued Hopfield-like network [49] processes input vectors fully in the complex space using complex weights. Real and imaginary parts of the data are treated with equal significance in nondegenerate complex space. A modification of this network has been successfully applied to the DoA estimation of antenna array [125].

XI. Boltzmann Machines and Learning

Boltzmann machines are a class of stochastic RNNs based on physical systems [4,48]. It has the same network architecture as that of the Hopfield model, that is, it is highly recurrent with $w_{ij}=w_{ji}$ and $w_{ii}=0$, $i,j=1,\ldots,J$. In contrast to the Hopfield network, the Boltzmann machine can have hidden units. The Hopfield network operates in an unsupervised manner, while the Boltzmann machine can also be trained in a supervised manner.

A. The Boltzmann Machine

Unlike the Hopfield model, neurons of a Boltzmann machine are divided into visible and hidden units. The visible units are clamped onto specific states determined by the environment. The hidden units always operate freely. By capturing high-order statistical correlations in the clamping vector, the hidden units simulate the underlying constraints contained in the input vectors. This type of Boltzmann machine uses the unsupervised learning, and can perform pattern completion. When the visible units are further divided into input and output neurons, this type of Boltzmann machine can be trained in a supervised manner. The recurrence eliminates the difference in input and output cells. The Boltzmann machine, operated in sequential or synchronous mode, is a universal approximator for arbitrary functions defined on finite sets [126].

Instead of using a sigmoidal function in the Hopfield network, the activation at each neuron takes the value of 0 or 1, depending on the probability of temperature T

$$net_i=\sum_{j=1,j\neq i}^{J} w_{ji}x_j=\vec{w}_i\vec{x}, \tag{37}$$

$$P_i=\frac{1}{1+\mathrm{e}^{-\frac{net_i}{T}}}, \tag{38}$$

$$x_i=\begin{cases} 1, & \text{with probability } p_i \\ -1, & \text{with probability } 1-p_i \end{cases}. \tag{39}$$

When $net_i=0$ or T is very large, x_i is either 1 or 0 with equal probability. For very small T, x_i is deterministically 1. The input and output states can be fixed or variable.

The search for all the possible states is performed at a temperature T in Boltzmann machines. At the steady state, the relative probability of two states in a Boltzmann machine is determined by the Boltzmann distribution of the energy difference between the two states $\frac{p_\alpha}{p_\beta}=\mathrm{e}^{-\frac{E_\alpha-E_\beta}{T}}$, where E_α and E_β are the corresponding energy levels of the two states. The energy can be computed by the same formula as for the Hopfield model given by (5).

For the Boltzmann machine with hidden units, the generalized Hebbian rule cannot be used as in an unsupervised manner. For the supervised learning of the Boltzmann machine, the BP is not applicable due to the different network architecture. SA is used by Boltzmann machines to learn weights corresponding to the global optimum. Nevertheless, the Boltzmann learning is significantly slower than the BP. For constraint-satisfaction problems, some of the neurons are externally clamped to some input patterns, and we then find the global minimum for these particular input patterns. The integration of SA into the Boltzmann learning rule makes the Boltzmann machine especially suitable for constraint satisfaction tasks involving a large number of weak constraints [48].

The original Boltzmann learning algorithm [48,4] is based on counting occurrences. The Boltzmann learning algorithm based on correlations [95] provides a better performance than the original algorithm. For each update of the algorithms, the algorithm first estimates correlation in the clamped condition by performing SA procedure for all the training set, ρ_{ij}^{+}, $i, j = 1, 2, \ldots, J, i \neq j$, then estimates correlation in the free-running condition, ρ_{ij}^{-}. Finally the weight updated is performed by

$$\Delta w_{ij} = \eta\left(\rho_{ij}^{+} - \rho_{ij}^{-}\right), \quad i, j = 1, 2, \ldots, J, i \neq j, \tag{40}$$

where the learning rate $\eta = \frac{\varepsilon}{T}$, and ε is a small positive constant. Equation (40) is called the *Boltzmann learning rule*. This procedure is repeated until there is no change in w_{ij}, for all i, j. The correlation-based learning procedure for the Boltzmann machine is given in [31,95,45].

Like the multistate Hopfield model [58], a multi-valued Boltzmann machine has been proposed as an extension of the two-valued Boltzmann machine [78]. Each neuron of the multi-valued Boltzmann machine can only take L discrete stable states, and the angle between two adjacent directions is given by $\phi_0 = \frac{2\pi}{L}$. The probability of state change is according to the Boltzmann distribution of the energy difference between the two states. A synchronous Boltzmann machine as well as its learning algorithm has been introduced to facilitate parallel implementations [13].

B. The Mean-field-theory Machine

Mean-field approximation is a well-known method in statistical physics [39]. The mean-field annealing algorithm was proposed to accelerate the convergence of the Boltzmann machine [95,96]. The Boltzmann machine with such an algorithm is also termed the *mean-field-theory (MFT) machine* or *deterministic Boltzmann machine.*

The mean-field annealing algorithm is a deterministic method. It generates continuous neuron outputs, which are calculated as the average of the probability of the binary neuron variables at temperature T, are used. The average of state x_i is calculated for a specific value of activation net_i according to (39), (37) and (38)

$$\mathrm{E}[x_i] = \tanh(\frac{net_i}{T}). \tag{41}$$

The correlations in the Boltzmann learning rule is replaced by the mean-field approximation $\mathrm{E}\left[x_i x_j\right] \simeq \mathrm{E}\left[x_i\right]\mathrm{E}\left[x_j\right]$.

The MFT machine provides a substantial speedup over the Boltzmann machine, and is one to two orders-of-magnitude faster than the Boltzmann machine [42,95].

The mean-field annealing algorithm can be derived following the optimization of the Kullback-Leibler divergence between the factorial approximating distribution and the ideal joint distribution of the binary neural variables in terms of the mean activations. In [123], two interactive mean-field algorithms are derived by extending the internal representations to include both the mean activations and the mean correlations. The two algorithms improve the mean-field approximation in both the performance and the relaxation efficiency.

The mean-field annealing dynamics are isomorphic to the steady-state equations of an RC-circuit. The mean-field annealing algorithm can be simulated by RC circuits, coupled with the local nature of the Boltzmann machine, which makes the MFT machine suitable for massively parallel VLSI implementation [9,69,106].

XII. Discussion

Both the Boltzmann and MFT machines can be used as associative memory. These models, when using hidden units, have a far higher capacity for storage and error-correcting retrieval of random patterns and improved basins of attraction than the Hopfield network [42,9]. When the Boltzmann machine is trained as associative memory using an adaptive association rule [59], it does not suffer from spurious states. The association rule, which creates a sphere of influence around each stored pattern, is a generalization of the generalized Hebbian rule. Spurious fixed points, whose regions of attraction are not recognized by the rule, are skipped.

The training for the Boltzmann machine and the MFT machine models is given in the preceding sections. The retrieval can be performed as follows [42]. The visible neurons are clamped to a corrupted pattern, the whole network is annealed to a lower temperature, where the state of the hidden neurons approximates the learned internal representation of the stored pattern, and then the visible neurons are released. The annealing process continues until the whole network is settled.

The Boltzmann machine is important historically and theoretically. However, its exponential complexity with the number of neurons restricts it applications in many problems. Likewise, the MFT machine aslo has its limitations. It can only work in the supervised mode with only a single hidden layer [45]. For multiple hidden layers, the probability distribution cannot be well estimated. The mean state is not a sufficient representation for the free-running probability distribution and thus, the mean-field method is ineffective for unsupervised learning.

In addition to the Boltzmann and MFT machines, there are some other implementations of stochastic Hopfield networks. The Gaussian machine [7] is a general framework that includes the Hopfield network, the Boltzmann machine and also other stochastic networks. Stochastic distribution is realized by adding thermal noise, a stochastic external input ε, to each unit, and the network dynamics are the same as that of the Hopfield network. The stochastic term ε obeys a Gaussian distribution with zero mean and variance σ^2, where the deviation $\sigma = kT$, and T is the temperature. The stochastic term ε can occasionally bring the network to states with a higher energy. When $k = \sqrt{\frac{8}{\pi}}$, the distribution of the outputs has the same behavior as a Boltzmann machine. When employing noise obeying a logistic distribution rather than a Gaussian distribution in the original definition, we can obtain a Gaussian machine identical to a Boltzmann machine. When the noise in the Gaussian machine takes a Cauchy distribution with zero as the peak location and the half-width at the maximum $\sigma = T\sqrt{\frac{8}{\pi}}$, we get a Cauchy machine [111]. The Gaussian machine may be more suitable than the Boltzmann machine for some tasks. A similar idea was embodied in the stochastic network given in [73], where in addition to a cooling schedule for temperature Tgain annealing is also applied. The gain $\frac{1}{\beta}$ has to be decreased more slowly than T, and kept bounded away from zero.

XIII. Summary

We have given a comprehensive introduction to some RNNs including the Hopfield model and the Boltzmann machine. These models are mainly used for associative memory and for solving COPs. The storage capabilities of various storage algorithms for associative memory have been discussed. Some other models for associative memory such as the MLP-based autoassociative memories and the Hamming network are also introduced. The COP has been introduced and various strategies for escaping local minima has been described. SA is a stochastic global optimization method that is useful for any optimization problem. The chaotic neural network is obtained by adding chaotic dynamics to the Hopfield network, and this method can easily find the global optimum of COPs by using the CSA method, which is a fast, deterministic alternative to SA. Multilevel Hopfield models can lead to a compact network architecture, and is especially used for such tasks as image restoration. The Boltzmann machine and learning is a generalization of the Hopfield model, and it can also been used for supervised learning. The MFT machine is obtained by applying the mean-field approximation to Boltzmann learning. Mean-field annealing is a deterministic alternative to SA, and provides a significant acceleration to the training of the Boltzmann machine.

While the Hopfield model and the Boltzmann machine are suitable for hardware implementation. The bulky connections between neurons cause difficulties in large-scale implementations. The cellular neural network (CNN) model, proposed in 1988 by Chua and Yang [21,22], has a unique network architecture. CNN is a generalization of the Hopfield network [50], and can be used to solve a more generalized optimization problem.

A CNN is a two- or higher-dimensional array of regularly spaced neurons, called *cells*, which communicate only with the neurons in its immediate neighborhood. Adjacent cells are connected by mutual interconnections [21,22]. Each cell has its own dynamics whose evolution is dependent on its circuit time constant $\tau = RC$. The local interactions can be programmed by a template matrix [21]. CNN overcomes the massive interconnection problem of parallel distributed processing. The key features are asynchronous parallel processing, continuous time dynamics, and local interactions among the network elements. CNN chips can have high-density cells, and some physical implementations such as analog CMOS, emulated digital CMOS and optical implementations, are available. The CNN universal machine [104] is the analog cellular computer for processing analog array signals, and has a computational power of tera (10^{12}) or peta (10^{15}) analog operations per second on a single CMOS chip [22]. CNN with a two-dimensional array architecture is a natural candidate for image processing or simulation of partial differential equations. Using different cloning templates, namely, the representation of the local interconnection patterns, different operations can be conducted on an image. CNN has now become an important method for image processing.

References

1. Aarts E, Korst J (1989) Simulated annealing and Boltzmann machines. John Wiley, Chichester.
2. Abe S, Kawakami J, Hirasawa K (1992) Solving inequality constrained combinatorial optimization problems by the Hopfield neural networks. Neural Netw 5 : 663–670.
3. Abu-Mostafa Y, St Jacques J (1985) Information capacity of the Hopfield model. IEEE Trans Inf Theory 31: 461–464.
4. Ackley DH, Hinton GE, Sejnowski TJ (1985) A learning algorithm for Boltzmann machines. Cogn Sci 9:147–169.
5. Aihara K, Takabe T, Toyoda M (1990) Chaotic neural networks. Phys Lett A 144: 333–340.
6. Aiyer SVB, Niranjan N, Fallside F (1990) A theoretical investigation into the performance of the Hopfield model. IEEE Trans Neural Netw 1: 204–215.
7. Akiyama Y, Yamashita A, Kajiura M, Aiso H (1989) Combinatorial optimization with Gaussian machines. In: Proc. IEEE Int. Joint Conf. Neural Netw 533–540.
8. Almeida LB (1987) A learning rule for asynchronous perceptrons with feedback in combinatorial environment. In: Proc. IEEE 1st Int. Conf. Neural Netw San Diego 609–618.
9. Alspector J, Jayakumar A, Ngo B (1992) An electronic parallel neural CAM for decoding. In IEEE Worksh Neural Netw for Signal Process II Amsterdam Denmark 581–587.
10. Amari SI (1972) Learning patterns and pattern sequences by self-organizing nets of threshold elements. IEEE Trans Comput 21: 1197–1206.
11. Anderson JA (1972) A simple neural network generating interactive memory. Math Biosci 14: 197–220.
12. Anderson JA, Silverstein JW, Ritz SA, Jones RS (1977) Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psychological Rev 84: 413–451.
13. Azencott R, Doutriaux A, Younes L (1993) Synchronous Boltzmann machines and curve identification tasks. Network 4: 461–480.
14. Baird B (1990) Associative memory in a simple model of oscillating cortex. In DS Touretzky (ed) Advances in neural information processing systems 2: 68–75. Morgan Kaufmann San Mateo CA
15. Bruck J(1990) On the convergence properties of the Hopfield model. Proc IEEE 78: 579–1585.
16. Bruck J, Roychowdhury WP (1990) On the number of spurious memories in the Hopfield model. IEEE Trans Inf Theory 36: 393–397.
17. Chakraborty K, Mehrotra KG, Mohan CK, Ranka S (1992) An optimization network for solving a set of simultaneous linear equations. In Proc Int Joint Conf Neural Netw Baltimore MD 2: 516–521.
18. Chen L, Aihara K (1995) Chaotic simulated annealing by a neural-network model with transient chaos. Neural Netw 8: 915–930.
19. Chiueh TD, Goodman RM (1991) Recurrent correlation associative memories. IEEE Trans Neural Netw 2: 275–284.
20. Chiueh TD, Tsai HK (1993) Multivalued associative memories based on recurrent networks. IEEE Trans Neural Netw 4: 364–366.
21. Chua LO, Yang L (1988) Cellular neural network–Part I: Theory; Part II: Applications. IEEE Trans Circu Syst 35: 1257–1290.
22. Chua LO, Roska T (2002) Cellular neural network and visual computing–Foundation and applications. Cambridge Univ Press Cambridge UK.
23. Cichocki A, Unbehauen R (1992) Neural networks for optimization and signal processing. Wiley, New York.
24. Cohen MA, Grossberg S (1983) Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. IEEE Trans Syst Man Cybern 13: 815–826.
25. Coombes S, Taylor JG (1994) Using generalized principal component analysis to achieve associative memory in a Hopfield net. Network 5: 75–88.
26. Coombes S, Campbell C (1996) Efficient learning beyond saturation by single-layered neural networks. Bristol Cen App Nonlinear Math Univ of Bristol UK.
27. Cover TM (1965) Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Trans Electron Computers 14: 326–334.
28. Culhane AD, Peckerar MC, Marrian CRK (1989) A neural net approach to discrete Hartley and Fourier transforms. IEEE Trans Circ Syst 36: 695–702.
29. Czech ZJ (2001) Three parallel algorithms for simulated annealing. Springer 2328: 210–217.
30. Diederich S, Opper M (1987) Learning of correlated patterns in spin-glass networks by local learning rules. Phys Rev Lett 58: 949–952.
31. Du KL, Swamy MNS (2006) Neural networks in a softcomputing framework. Springer, London.
32. Erdem MH, Ozturk Y (1996) A new family of multivalued networks. Neural Netw 9: 979–989.

33. Farhat NH, Psaltis D, Prata A, Paek E (1985) Optical implementation of the Hopfield model. Appl Opt 24: 1469–1475.
34. Fleisher M (1988) The Hopfield model with multi-level neurons. In DZ Anderson (ed.) Neural information processing systems American Inst of Physics, New York 278–289.
35. Funahashi KI, Nakamura Y (1993) Approximation of dynamical systems by continuous time recurrent neural networks. Neural Netw 6: 801–806.
36. Gao K, Ahmad MO, Swamy MNS (1990) A neural network least-square estimator. In Proc Int Joint Conf Neural Netw Washington DC 3: 805–810.
37. Gardner E (1988) The space of the interactions in neural network models. J Phys A 21: 257–270.
38. Geman V, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans Pattern Anal Mach Intell 6: 721–741.
39. Glauber RJ (1963) Time-dependent statistics of the Ising model. J Math Phys 4: 294–307.
40. Greville T (1960) Some applications of pseudo-inverse of matrix. SIAM Rev 2: 15–22.
41. Hanna MT (2000) On the stability of a Tank and Hopfield type neural network in the general case of complex eigenvalues. IEEE Trans. Signal Process 48: 289–293.
42. Hartman E (1991) A high storage capacity neural network content-addressable memory. Network 2: 315–334.
43. Hassoun MH (1995) Fundamentals of artificial neural networks. MIT Press, Cambridge, MA, USA
44. Hassoun V, Watta PB (1996) The Hamming associative memory and its relation to the exponential capacity DAM. In Proc IEEE Int Conf Neural Netw Washington DC 1: 583–587.
45. Haykin S (1999) Neural networks: A comprehensive foundation. 2nd ed, Prentice Hall, Upper Saddle River, New Jersey.
46. He Y (2002) Chaotic simulated annealing with decaying chaotic noise. IEEE Trans Neural Netw 13:1526–1531.
47. Hebb V (1949) The organization of behavior. Wiley, New York.
48. Hinton GE, Sejnowski TJ (1986) Learning and relearning in Boltzmann machines. In DE Rumelhart JL McClelland (eds.) Parallel distributed processing: Explorations in microstructure of cognition 1: 282–317.
49. Hirose A (1992) Dynamics of fully complex-valued neural networks. Electron. Lett 28: 1492–1494
50. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Nat Acad Sci 79: 2554–2558.
51. Hopfield JJ (1984) Neurons with graded response have collective computational properties like those of two-state neurons. Proc Nat Acad Sci 81: 3088–3092.
52. Hopfield JJ, Tank DW (1985) Neural computation of decisions in optimization problems. Biol Cybern 52: 141–152.
53. Hopfield JJ, Tank DW (1986) Computing with neural circuits: A model. Science 233: 625–633.
54. Ikeda N, Watta P, Artiklar M, Hassoun MH (2001) A two-level Hamming network for high performance associative memory. Neural Netw 14: 1189–1200.
55. Ingber L (1989) Very fast simulated re-annealing. Math. & Computer Modelling. 12: 967–973.
56. Jagota A, Mandziuk J (1998) Experimental study of Perceptron-type local learning rule for Hopfield associative memory. Inf Sci 111: 65–81.
57. Jang JS, Lee SY, Shin SY (1988) An optimization network for matrix inversion. In DZ Anderson (ed.) Neural information processing systems 397–401.
58. Jankowski S, Lozowski A, Zurada JM (1996) Complex-valued multi-state neural associative memory. IEEE Trans Neural Netw 7: 1491–1496.
59. Kam M, Cheng R (1989) Convergence and pattern stabilization in the Boltzmann machine. In: D.S. Touretzky (ed) Advances in neural information processing systems Morgan Kaufmann San Mateo CA 1: 511–518.
60. Kamp Y, Hasler M (1990) Recursive neural networks for associative memory. Wiley, New York
61. Kanter I, Sompolinsky H (1987) Associative recall of memory without errors. Phys. Rev. A, 35: 380–392.
62. Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Sci 220: 671–680.
63. Kobayashi K (1991) On the capacity of a neuron with a non-monotone output function. Net wor 2: 237–243.
64. Kohonen T (1972) Correlation matrix memories. IEEE Trans. Compu 21: 353–359.
65. Kohonen T (1989) Self-organization and associative memory. Springer Berlin.
66. Kosko B (1987) Adaptive bidirectional associative memories. Appl Opt 26: 4947–4960.
67. Kwok T, Smith KA (1999) A unified framework for chaotic neural-network approaches to combinatorial optimization. IEEE Trans. Neural Netw 10: 978–981.
68. Lee BW, Shen BJ (1992) Design and analysis of analog VLSI neural networks. In B Kosko (ed) Neural networks for signal processing 229–284.
69. Lee BW, Shen BJ (1993) Parallel hardware annealing for optimal solutions on electronic neural networks. IEEE Trans Neural Netw 4: 588–599.
70. Lee DL (2001) Improving the capacity of complex-valued neural networks with a modified gradient descent learning rule. IEEE Trans Neural Netw 12: 439–443.
71. Le Gall A, Zissimopoulos V (1999) Extended Hopfield models for combinatorial optimization. IEEE Trans Neural Netw 10: 72–80.
72. Lendaris GG, Mathia K, Saeks R (1999) Linear Hopfield networks and constrained optimization. IEEE Trans Syst Man Cybern B 29: 114–118.
73. Levy BC, Adams MB (1987) Global optimization with stochastic neural networks. In Proc 1st IEEE Conf Neural Netw San Diego CA 3: 681–689.
74. Li JH, Michel AN, Parod W (1989) Analysis and synthesis of a class of neural networks: Linear systems operating on a closed hypercube. IEEE Trans Circ Syst 36: 1405–1422.
75. Li LK (1992) Approximation theory and recurrent networks. In Proc Int Joint Conf Neural Netw Baltimore MD 2: 266-271.
76. Li XD, Ho JKL, Chow TWS (2005) Approximation of dynamical time-variant systems by continuous-time recurrent neural networks. IEEE Trans Circ Syst 52: 656–660.
77. Liang Y (1996) Combinatorial optimization by Hopfield networks using adjusting neurons. Inf Sci 94: 261–276.
78. Lin CT, Lee CSG (1995) A multi-valued Boltzmann machine. IEEE Trans Syst Man cybern 25: 660–669.
79. Lippman RP (1987) An introduction to computing with neural nets. IEEE ASSP Mag 4: 4–22.
80. Little WA (1974) The existence of persistent states in the brain. Math. Biosci 19: 101–120.
81. Locatelli M (2001) Convergence and first hitting time of simulated annealing algorithms for continuous global optimization. Math Methods of Oper Res 54: 171–199.
82. Ma J (1997) The stability of the generalized Hopfield networks in randomly asynchronous mode. Neural Netw., 10: 1109–1116
83. Ma J (1999) The asymptotic memory capacity of the generalized Hopfield network. Neural Netw 12: 1207–1212.
84. Matsuda S (1998) Optimal Hopfield network for combinatorial optimization with linear cost function. IEEE Trans. Neural Netw 9: 1319–1330.
85. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys 5: 115–133.
86. McEliece RJ, Posner EC, Rodemich ER, Venkatesh SS (1987) The capacity of the Hopfield associative memory. IEEE Trans Inf Theory 33: 461–482.
87. Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equations of state calculations by fast computing machines. J Chem Phys 21: 1087–1092.
88. Michel AN, Si J, Yen G (1991) Analysis and synthesis of a class of discrete-time neural networks described on hypercubes. IEEE Trans Neural Netw 2: 32–46.
89. Minsky ML, Papert S (1969) Perceptrons. MIT Press Cambridge MA.

90. Morita M (1993) Associative memory with nonmonotonicity dynamics. Neural Netw 6: 115–126.

91. Muezzinoglu MK, Guzelis C, Zurada JM (2003) A new design method for the complex-valued multistate Hopfield associative memory. IEEE Trans Neural Netw 14: 891–899.

92. Nemoto I, Kubono M (1996) Complex associative memory. Neural Netw 9: 253–261.

93. Oja E (1982) Simplified neuron model as a principal component analyzer. J Math & Biolo 15: 267–273.

94. Personnaz L, Guyon I, Dreyfus G (1986) Collective computational properties of neural networks: New learning mechanisms Phys Rev A 34: 4217–4228.

95. Peterson C, Anderson JR (1987) A mean field learning algorithm for neural networks. Complex Syst 1: 995–1019.

96. Peterson C (1990) Applications of mean field theory neural networks. In: Lima R, Streit R, Mendes RV (eds) Dynamic and stochastic processes: Theory and applications Springer Notes in Physics 355: 141–173.

97. Pineda FJ (1987) Generalization of back-propagation to recurrent neural networks. Phys Rev Lett 59: 2229–2232.

98. Richardt J, Karl F, Muller C (1998) Connections between fuzzy theory, simulated annealing, and convex duality. Fuzzy Sets Syst 96: 307–334.

99. Rojas R (1996) Neural networks: A systematic introduction. Springer Berlin.

100. Rose K (1998) Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. Proc IEEE 86: 2210–2239.

101. Rose K, Gurewitz E, Fox GC (1990) A deterministic annealing approach to clustering. Pattern Recogn Lett 11: 589–594.

102. Rosenblatt R (1958) The Perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65: 386–408.

103. Rosenblatt R (1962) Principles of neurodynamics. Spartan Books New York.

104. Roska T, Chua LO (1993) The CNN universal machine: An analogic array computer. IEEE Trans Circ Syst II 40: 163–173.

105. Rutenbar RA (1989) Simulated annealling algorithms: An overview. IEEE Circ Dev Mag 5: 19–26.

106. Schneider RS, Card HC (1998) Analog hardware implementation issues in deterministic Boltzmann machines. IEEE Trans Circ Syst II 45: 352–360.

107. Si J, Michel AN (1995) Analysis and synthesis of a class of discrete-time neural networks with multilevel threshold neurons. IEEE Trans Neural Netw 6: 105–116.

108. Storkey AJ (1997) Increasing the capacity of the Hopfield network without sacrificing functionality. In Gerstner W, Germond A, Hastler M, Nicoud J (eds.) ICANN97, LNCS Springer 1327, 451–456.

109. Storkey AJ, Valabregue R (1997) Hopfield learning rule with high capacity storage of time-correlated patterns. Electron Lett 33: 1803–1804.

110. Swamy MNS, Thulasiraman K (1981) Graphs, networks, and algorithms. Wiley New York.

111. Szu HH, Hartley RL (1987) Nonconvex optimization by fast simulated annealing. Proc. IEEE 75:1538–1540 Also published: Szu H Fast simulated annealing Phys Lett A 122: 152–162.

112. Tagliarini GA, Christ JF, Page EW (1991) Optimization using neural networks. IEEE Trans Comput 40: 1347–1358.

113. Tank DW, Hopfield JJ (1986) Simple “neural” optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. IEEE Trans Circ Syst 33: 533–541.

114. Tsallis C, Stariolo DA (1996) Generalized simulated annealing. Physica A 233: 395–406.

115. Venkatesh SS, Psaltis D (1989) Linear and logarithmic capacities in associative memory. IEEE Trans Inf Theory 35: 558–568.

116. Ventura D, Martinez T (2000) Quantum associative memory. Inf Sci 124: 273–296.

117. Wang J, Li H (1994) Solving simultaneous linear equations using recurrent neural networks. Inf Sci 76: 255–277.

118. Wang L, Smith K (1998) On chaotic simulated annealing. IEEE Trans Neural Netw 9: 716–718.

119. Wang RL, Tang Z, Cao QP (2002) A learning method in Hopfield neural network for combinatorial optimization problem. Neurocomput 48: 1021–1024.

120. Wang L, Li S,Tian F, Fu X (2004) A noisy chaotic neural network for solving combinatorial optimization problems: stochastic chaotic simulated annealing. IEEE Trans Syst Man Cybern B 34: 2119–2125.

121. Williams RJ, Zipser D (1989) A learning algorithm for continually running fully recurrent neural networks. Neural Comput 1: 270–280.

122. Wu Y, Batalama SN (2000) An efficient learning algorithm for associative memories. IEEE Trans. Neural Netw 11: 1058–1066.

123. Wu JM (2004) Annealing by two sets of interactive dynamics. IEEE Trans Syst Man Cybern B 34: 1519–1525.

124. Yan H (1991) Stability and relaxation time of Tank and Hopfield’s neural network for solving LSE problems. IEEE Trans Circ Syst 38: 1108–1110.

125. Yang WH, Chan KK, Chang PR (1994) Complex-valued neural network for direction of arrival estimation. Electron Lett 30: 574–575.

126. Younes L (1996) Synchronous Boltzmann machines can be universal approximators. Appl Math Lett 9: 109–113.

127. Yoshizawa S, Morita M, Amari SI (1993) Capacity of associative memory using a nonmonotonic neuron model. Neural Netw 6: 167–176.

128. Yuh JD, Newcomb RW (1993) A multilevel neural network for A/D conversion. IEEE Trans Neural Netw 4: 470–483.

129. Zurada JM, Cloete I, van der Poel E (1996) Generalized Hopfield networks for associative memories with multi-valued stable states. Neurocomput 13: 135–149.