Nonlinear Dynamics and Computing in Recurrent Neural Networks

Hideyuki Suzuki

Abstract Nonlinearity is a key concept in the design and implementation of photonic neural networks for computing. This chapter introduces the fundamental models and concepts of recurrent neural networks, with a particular focus on their nonlinear dynamics. We review several types of nonlinear dynamics that emerge in symmetrically connected recurrent neural networks, in which the energy function plays a crucial role. In addition, we introduce the concepts of reservoir computing, covering fundamental models and physical reservoir computing. Overall, this chapter provides a foundation for the theoretical aspects in the subsequent chapters of this book, which explore a variety of photonic neural networks with nonlinear spatiotemporal dynamics.

1 Introduction

For more than half a century, various artificial neural network models have been developed and studied as abstractions of thought processes in the brain and as constructive approaches to thinking machines [1, 2]. Artificial neural networks are currently a fundamental technology in artificial intelligence, applied across various fields and playing crucial roles in our daily lives.

Recurrent neural networks (RNNs) are a type of neural network that can be contrasted with feedforward neural networks, such as multilayer perceptrons (MLPs). Unlike feedforward neural networks, which perform unidirectional information processing from the input layer to the output layer, RNNs allow mutual interactions among the constituent neuron models. These interactions typically induce spatiotemporal dynamics as the network state evolves over time, which performs various computational tasks, such as processing time-sequence data, solving combinatorial optimization problems, and generative statistical modeling.

H. Suzuki (✉)
Graduate School of Information Science and Technology, Osaka University,
Osaka 565–0871, Japan
e-mail: hideyuki@ist.osaka-u.ac.jp

H. Suzuki et al. (eds.), *Photonic Neural Networks with Spatiotemporal Dynamics*,
https://doi.org/10.1007/978-981-99-5072-0_2

Nonlinearity is an indispensable property of neuron models, which naturally leads to the emergence of nonlinear dynamics in RNNs. Thus, understanding and utilizing their nonlinear dynamics is especially important for realizing energy-efficient, high-speed, and large-scale implementations of RNNs using optical technologies.

Based on this motivation, this chapter introduces the fundamental models and concepts of RNNs for computing, with a particular focus on their nonlinear dynamics. In Sect. 2, we introduce the notion of the energy function in two fundamental models of symmetrically connected RNNs: the Amari–Hopfield network and the Boltzmann machine. We explore how these models exhibit computational functions such as associative memory, combinatorial optimization, and statistical learning. Section 3 presents an overview of various types of nonlinear dynamics that arise in RNNs. We discuss how chaotic dynamics contributes to computation in models such as the chaotic neural network and the chaotic Boltzmann machine. We also observe the important roles of nonlinear dynamics in several types of Ising machines, which serve as hardware solvers for combinatorial optimization problems. Moreover, we introduce a sampling algorithm, known as the herding system, which exhibits complex nonlinear dynamics related to the learning process of RNNs. Section 4 provides a brief overview of reservoir computing, which is a lightweight approach that leverages the rich nonlinear dynamics of RNNs for information processing. We introduce the basic models and concepts of reservoir computing, such as the echo state network and echo state property, and further discuss physical reservoir computing. Finally, in Sect. 5, we explain how the concepts introduced in this chapter underlie the studies covered in the subsequent chapters of this book, which explore various aspects of photonic neural networks with nonlinear spatiotemporal dynamics.

2 Fundamental RNN Models and Energy Function

In this section, we introduce the two fundamental models of symmetrically connected RNNs: the Amari–Hopfield network and the Boltzmann machine. We provide a brief overview of their definitions and behavior, highlighting the role of the energy function. These models have computational functions such as associative memory, combinatorial optimization, and statistical learning. Although their dynamics is fundamentally governed by the energy function, they lay the foundation for RNN models with rich nonlinear dynamics as discussed in the following sections.

2.1 Amari–Hopfield Network with Binary States

The Amari–Hopfield network [3, 4] is an RNN model composed of binary McCulloch–Pitts neurons [1] with symmetrical connections. The state of each ith neuron at time t is represented by $s_i(t) \in \{0, 1\}$, with values corresponding to the resting and firing states, respectively. Note that a formulation that employs -1, instead

of 0, as the resting state is also widely used. The input to the ith neuron from the jth neuron is assumed to be $w_{ij}s_j(t)$, where $w_{ij} \in \mathbb{R}$ denotes the synaptic weight. The weights are assumed to be symmetric, $w_{ij} = w_{ji}$, and have no self-connections, $w_{ii} = 0$. The state of the ith neuron takes $s_i = 1$ if the total input, including constant input (or bias) $b_i \in \mathbb{R}$, exceeds zero and $s_i = 0$ if otherwise. Hence, the update rule for the ith neuron can be written as follows:

$$s_i(t+1) = \theta\left(\sum_{j=1}^{N} w_{ij}s_j(t) + b_i\right) , \tag{1}$$

where N denotes the total number of neurons and $\theta(\cdot)$ is the Heaviside unit step function; i.e., $\theta(z) = 1$ for $z \geq 0$ and $\theta(z) = 0$ for $z < 0$. According to this equation, the state of the network, $\mathbf{s}(t) = (s_1(t), s_2(t), \ldots, s_N(t))^\top$, evolves over time in the state space $\{0, 1\}^N$.

The key notion for understanding the behavior of the Amari–Hopfield network is the energy function

$$H(\mathbf{s}) = -\frac{1}{2}\sum_{i,j=1}^{N} w_{ij}s_i s_j - \sum_{i=1}^{N} b_i s_i , \tag{2}$$

which is guaranteed to decrease over time. Specifically, assume that only the ith neuron is updated according to (1), while the states of the other neurons are kept unchanged, i.e., $s_j(t+1) = s_j(t)$ for $j \neq i$. Then, it holds that $H(\mathbf{s}(t+1)) \leq H(\mathbf{s}(t))$ because

$$H(\mathbf{s}(t+1)) - H(\mathbf{s}(t)) = -\left(s_i(t+1) - s_i(t)\right)\left(\sum_{j=1}^{N} w_{ij}s_j(t) + b_i\right) \leq 0 . \tag{3}$$

Consequently, the network state $\mathbf{s}(t)$ evolves until it reaches a local minimum of the energy function, which has no neighboring states with lower energy. These local minima are considered attractors of the network because the network state eventually converges to one of the local minima.

This behavior of Amari–Hopfield network can be interpreted as the process of recalling memory stored within the network, which is referred to as associative memory. Memory patterns can be stored as local minima of the network by designing the weight parameters as follows:

$$w_{ij} = \sum_{k=1}^{K}(2\xi_i^{(k)} - 1)(2\xi_j^{(k)} - 1) , \tag{4}$$

where $\boldsymbol{\xi}^{(k)} = (\xi_1^{(k)}, \ldots, \xi_N^{(k)})^\top \in \{0, 1\}^N$ is the kth memory pattern. This learning rule, known as the Hebbian learning, strengthens the connections between neurons that are simultaneously activated in the memory patterns.

The Amari–Hopfield network is a simple but important foundation of RNN models with energy functions. Furthermore, recent studies have revealed that the generalization of the Amari–Hopfield network, such as the modern Hopfield network [5] and the dense associative memory model [6], share similar attention mechanisms present in modern neural network models, such as transformers and BERT. These generalized models employ continuous state variables, as described below.

2.2 *Amari–Hopfield Network with Continuous States*

The Amari–Hopfield network with continuous variables [7, 8], proposed almost simultaneously as the binary version, is an RNN model composed of symmetrically connected leaky integrators. The continuous-time nonlinear dynamics of state $x_i(t)$ of each ith neuron is expressed by the following ordinary differential equation (ODE):

$$\frac{dx_i}{dt} = -\frac{1}{\tau_{\text{leak}}} x_i + \sum_{j=1}^{N} w_{ij} \phi(x_j) + b_i \ , \tag{5}$$

where $\tau_{\text{leak}} > 0$ is a time constant and $\phi(x)$ is the sigmoid function

$$\phi(x) = \frac{1}{1 + \exp(-x/\varepsilon)} \ . \tag{6}$$

The output from the neurons $\phi(\mathbf{x}(t))$ evolves in the hypercube $[0, 1]^N$, where ϕ operates on each component of vector $\mathbf{x}(t) = (x_1(t), \ldots, x_N(t))^\top$. In the limit $\varepsilon \to 0$, where $\phi(x)$ is the Heaviside unit step function, the energy function (2) for the discrete Amari–Hopfield network also applies to the continuous version with $\mathbf{s} = \phi(\mathbf{x})$. That is, $H(\phi(\mathbf{x}(t)))$ decreases as the system evolves. Therefore, the network state converges to a local minimum of the energy function, which is analogous to that of the discrete model. Note that an energy function exists for $\varepsilon > 0$, while it introduces an extra term to (2).

In the continuous Amari–Hopfield network, state $x_i(t)$ of each neuron takes a continuous value that attenuates over time, according to (5). This model, known as a leaky integrator, is the simplest neuron model that describes the behavior of the membrane potentials of real neurons.

The Hopfield–Tank model [9] utilizes the dynamics of the Amari–Hopfield network for finding approximate solutions to combinatorial optimization problems such as the traveling salesman problem. Specifically, it is applicable to combinatorial optimization problems that are formulated as the minimization of the energy function (2), which is often referred to as quadratic unconstrained binary optimization (QUBO).

As the network state evolves, it converges to one of the local minima of the target energy function. This provides an approximate solution to the optimization problem. Once the network state is trapped in a local minimum, the search process terminates as it can no longer escape to explore other solutions. The Hopfield–Tank model can be considered the origin of recent studies on Ising machines (Sect. 3.3).

2.3 Boltzmann Machine

The Boltzmann machine [10] is an RNN model composed of binary stochastic neurons with symmetrical connections. The construction of the model is essentially the same as that of the Amari–Hopfield network with binary neurons, except that the neurons behave stochastically. The update rule is given by the probability that the ith neuron takes $s_i = 1$ in an update as follows:

$$\mathrm{Prob}[s_i(t+1) = 1] = \frac{1}{1 + \exp(-z_i(t)/T)} , \qquad z_i(t) = \sum_{j=1}^{N} w_{ij} s_j(t) + b_i , \quad (7)$$

where $T > 0$ denotes the model temperature and $z_i(t)$ is the total input to the neuron at time t. At the limit of $T \to 0$, the update rule is equivalent to the McCulloch–Pitts model; that is, the network dynamics is equivalent to that of the Amari–Hopfield network. In the limit $T \to \infty$, each neuron takes the states 0 and 1 with the same probability $1/2$, irrespective of the network configuration.

The state of the network $\mathbf{s}(t) = (s_1(t), s_2(t), \ldots, s_N(t))^\top$, evolves over time in the state space $\{0, 1\}^N$. The sequence of states $\{\mathbf{s}(t)\}_t$ eventually follows the Gibbs distribution

$$P(\mathbf{s}) = \frac{1}{Z} \exp\left(-\frac{1}{T} H(\mathbf{s})\right) , \qquad Z = \sum_{\mathbf{s}} \exp\left(-\frac{1}{T} H(\mathbf{s})\right) \quad (8)$$

with respect to the energy function $H(\mathbf{s})$ in (2), where Z is the normalizing constant called the partition function. The Boltzmann machine is more likely to adopt lower-energy states, and this tendency is more intense at lower temperatures. This probabilistic model is essentially equivalent to the Ising model, which is an abstract model of ferromagnetism in statistical mechanics.

Conversely, the Boltzmann machine can be considered to perform sampling from the Gibbs distribution $P(\mathbf{s})$. The Gibbs sampler, one of the Markov chain Monte Carlo (MCMC) methods, yields a sample sequence by updating each variable s_i in each step according to the conditional probability $P(s_i \mid \mathbf{s}_{\setminus i})$ given the values $\mathbf{s}_{\setminus i}$ of all the other variables. If applied to the Gibbs distribution $P(\mathbf{s})$ in (8), the conditional probability of $s_i = 1$ given $\mathbf{s}_{\setminus i}$ is as follows:

$$P(s_i = 1 \mid \mathbf{s}_{\setminus i}) = \frac{P(\mathbf{s}|_{s_i=1})}{P(\mathbf{s}|_{s_i=0}) + P(\mathbf{s}|_{s_i=1})} = \frac{1}{1 + \exp(-(H(\mathbf{s}|_{s_i=0}) - H(\mathbf{s}|_{s_i=1}))/T)} , \tag{9}$$

where $\mathbf{s}|_{s_i=\{0,1\}}$ denotes the vector $\mathbf{s}$ whose ith variable is set to $s_i = \{0, 1\}$. This probability is consistent with the update rule (7). Therefore, the Boltzmann machine is equivalent to the Gibbs sampler applied to the Gibbs distribution $P(\mathbf{s})$.

The Boltzmann machine can be utilized to solve combinatorial optimization problems, following the same approach as the Hopfield–Tank model to minimize the energy function. The stochasticity can help the network state escape the local minima, which is a remarkable difference from the Hopfield–Tank model. This effect is stronger at higher temperatures, whereas low-energy states are preferred at lower temperatures. Therefore, we typically employ simulated annealing to solve combinatorial optimization problems [11, 12], which controls the stochasticity by starting from a high temperature and gradually decreasing it to $T = 0$.

Another remarkable feature of the Boltzmann machine is its learning ability [10]. The learning is performed by tuning the parameters w_{ij} and b_i, such that the model distribution $P(\mathbf{s}) \propto \exp(-H(\mathbf{s}))$ is close to the given data distribution. Here, we omit temperature T by setting $T = 1$ without loss of generality.

The distance from the data distribution is quantified using the log-likelihood as follows:

$$\log L = \langle \log P(\mathbf{s}) \rangle_{\text{data}} = -\langle H(\mathbf{s}) \rangle_{\text{data}} - \log Z , \tag{10}$$

where $\langle \cdot \rangle_{\text{data}}$ denotes the average over the data distribution. We can then derive the learning rule as a gradient ascent on the log-likelihood as follows:

$$w_{ij}(k+1) = w_{ij}(k) + \alpha \frac{\partial}{\partial w_{ij}} \log L , \tag{11}$$

$$b_i(k+1) = b_i(k) + \alpha \frac{\partial}{\partial b_i} \log L , \tag{12}$$

where $\alpha > 0$ is the learning rate. The gradients are given by:

$$\frac{\partial}{\partial w_{ij}} \log L = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}} , \tag{13}$$

$$\frac{\partial}{\partial b_i} \log L = \langle s_i \rangle_{\text{data}} - \langle s_i \rangle_{\text{model}} , \tag{14}$$

where $\langle \cdot \rangle_{\text{model}}$ denotes the average over the model distribution $P(\mathbf{s})$.

The expressive power of the model distribution $P(\mathbf{s})$ can be improved by introducing hidden units into the state variable $\mathbf{s}$ of the Boltzmann machine. Accordingly, the state $\mathbf{s} = (\mathbf{v}, \mathbf{h})$ is composed of the visible part $\mathbf{v}$ and hidden part $\mathbf{h}$. The learning here aims to minimize the distance between the data distribution and the marginal distribution $P(\mathbf{v})$ of the visible part of the Gibbs distribution $P(\mathbf{v}, \mathbf{h}) \propto \exp(-H(\mathbf{v}, \mathbf{h}))$. The hidden units serve as additional latent variables that do not directly correspond to the data, and describe the indirect interactions among the visible units. The intro-

duction of hidden units does not alter the learning rule (11)–(14), whereas only the visible units are clamped to the data distribution in the averaging $\langle\cdot\rangle_{\text{data}}$.

In practice, learning a large-scale Boltzmann machine is challenging. Rigorous computation of the average over $P(\mathbf{s})$ is intractable as the size of the state space $\{0, 1\}^N$ increases exponentially. This expectation can be approximated by averaging over the sample sequence from the Boltzmann machine. However, obtaining an accurate approximation requires massive computation to generate a sufficiently long sample sequence, as the sampling process often gets stuck in local modes of the Gibbs distribution.

The restricted Boltzmann machine (RBM) [13, 14] is an important model of a Boltzmann machine with restricted connections. Specifically, an RBM is a two-layer neural network comprising visible and hidden units, with no connections within each layer. Because of its restricted structure, the RBM can be efficiently trained using the contrastive divergence algorithm to obtain the gradient of log-likelihood. The restricted structure accelerates the Gibbs sampling procedure, because it allows for alternate block sampling of the visible units, given the hidden units, and vice versa. The deep Boltzmann machine (DBM) [15, 16] is a Boltzmann machine with a multilayer structure. It is a type of deep neural network consisting of multiple layers of RBMs. Thus, RBMs and DBMs are important classes of the Boltzmann machine that have led to recent developments in deep learning.

3 Nonlinear Dynamics in Symmetrically Connected RNNs

This section presents several RNN models with symmetrical connections that exhibit various types of nonlinear dynamics effective for computing. First, we introduce the chaotic neural network model and the chaotic Boltzmann machine, which are variants of the Amari–Hopfield network and Boltzmann machine, respectively, involving nonlinear chaotic dynamics. Then, we review several types of Ising machines that employ more advanced approaches than the Hopfield–Tank model in utilizing their nonlinear dynamics to solve combinatorial optimization problems. We also explore the nonlinear dynamics that arises in the learning process of RNNs. As an example, we introduce the herding system, which is a sampling algorithm with complex nonlinear dynamics that can also be regarded as an extreme case of Boltzmann machine learning.

3.1 Chaotic Neural Network

The chaotic neural network [17] is a variation of the Amari–Hopfield network, which incorporates relative refractoriness and a continuous activation function in the constituent neurons. It exhibits spatiotemporal chaotic dynamics with the ability to perform parallel-distributed processing.

First, we introduce the refractoriness, which is a temporary reduction in excitability after firing, into the Amari–Hopfield network (1). We update the state of the ith neuron in the network as follows:

$$s_i(t+1) = \theta\left(\sum_{j=1}^{N} w_{ij} S_j^{\mathrm{fb}}(t) - \alpha S_i^{\mathrm{ref}}(t) + b_i\right), \tag{15}$$

where $S_i^{\{\mathrm{fb,ref}\}}(t) = \sum_{r=0}^{t} k_{\{\mathrm{fb,ref}\}}^{r} s_i(t-r)$ represents the accumulated past output of the ith neuron with an exponential decay parameterized by $k_{\mathrm{fb}}, k_{\mathrm{ref}} \in (0, 1)$ for the feedback connections and refractoriness, respectively. This model can be considered as a restricted form of Caianiello's neuronic equation [2]. The network dynamics is described by a hybrid dynamical system [18], involving the continuous variables $S_i^{\{\mathrm{fb,ref}\}}(t)$ and a discontinuous function $\theta(\cdot)$. The constituent neuron model with refractoriness is called the Nagumo–Sato model. Its single-neuron dynamics has been investigated by assuming the first term, which represents the input from other neurons, is constant in time, and has been shown to exhibit a complex response with a devil's staircase [18–20].

Next, we introduce a continuous activation function to obtain the chaotic neural network model as follows:

$$s_i(t+1) = \phi\left(\sum_{j=1}^{N} w_{ij} S_j^{\mathrm{fb}}(t) - \alpha S_i^{\mathrm{ref}}(t) + b_i\right), \tag{16}$$

where the Heaviside unit step function $\theta(\cdot)$ is replaced by the sigmoid function $\phi(\cdot)$ in (6).

The chaotic neural network exhibits spatiotemporal chaotic dynamics. Although the energy function in (2) does not necessarily decrease, it helps us to understand its dynamics. Unlike the Amari–Hopfield network, the state of the chaotic neural network continues to move around in the phase space without becoming stuck at a local minimum of the energy function. This is because if the network state remains at a local minimum for a while, the accumulated effects of refractoriness destabilize the local minimum, helping the state escape. Thus, the spatiotemporal chaotic dynamics emerge from a combination of the stabilizing effect, resulting from the mutual interactions in the network, and the destabilizing effect due to the refractoriness.

When applied to associative memory constructed by the Hebbian rule (4), the chaotic neural network continues to visit stored patterns itinerantly [21]. Such associative dynamics, which is characterized by chaotic itinerancy [22], has been demonstrated for a large-scale network in [23].

The itinerant behavior of the chaotic neural network is useful for solving combinatorial optimization problems [24], because the destabilizing effect helps the network state escape from local minima, and the state continues to explore possible solutions.

For hardware implementation of the chaotic neural network, it is crucial to utilize analog computation to simulate chaotic dynamics described by continuous variables.

Large-scale analog IC implementations of the chaotic neural network demonstrate high-dimensional physical chaotic neuro-dynamics and offer efficient applications in parallel-distributed computing, such as solving combinatorial optimization problems [25–27].

3.2 *Chaotic Boltzmann Machine*

The chaotic Boltzmann machine [28, 29] is a continuous-time deterministic system that utilizes nonlinear chaotic dynamics to function as a Boltzmann machine without requiring randomness for the time evolution. This contrasts with the original Boltzmann machine, comprising stochastic neurons updated at discrete-time steps.

Each neuron in the chaotic Boltzmann machine is associated with an internal state $x_i(t) \in [0, 1]$ besides the binary state $s_i(t) \in \{0, 1\}$ of the Boltzmann machine. The internal state x_i evolves according to the differential equation

$$\frac{dx_i}{dt} = (1 - 2s_i)\left(1 + \exp\frac{(1 - 2s_i)z_i}{T}\right), \tag{17}$$

where z_i is the total input as defined in (7). State s_i of the ith neuron flips when x_i reaches 0 or 1 as follows:

$$s_i(t + 0) = 0 \text{ when } x_i(t) = 0 \quad \text{and} \quad s_i(t + 0) = 1 \text{ when } x_i(t) = 1\,. \tag{18}$$

The right-hand side of (17) is positive when $s_i = 0$ and negative when $s_i = 1$. Therefore, the internal state x_i continues to oscillate between 0 and 1. If the states of the other neurons are fixed, the total input z_i becomes constant, and the oscillation continues periodically. Specifically, s_i takes the value 0 for $(1 + \exp(z_i/T))^{-1}$ unit time as x_i increases from 0 to 1, and s_i takes the value 1 for $(1 + \exp(-z_i/T))^{-1}$ unit time as x_i decreases from 1 to 0. Accordingly, the probability of finding $s_i = 1$ at a random instant is $(1 + \exp(-z_i/T))^{-1}$, which is consistent with the update rule (7) of the Boltzmann machine. Note that while this explanation provides intuitive validity to the equation, it does not necessarily imply that the network state $\mathbf{s}(t)$ follows the Gibbs distribution $P(\mathbf{s}) \propto \exp(-H(\mathbf{s})/T)$.

Although the chaotic Boltzmann machine is completely deterministic, it exhibits apparently stochastic behavior because of the chaotic dynamics that emerges from equations (17) and (18), which can be considered a hybrid dynamical system [18] with continuous variables x_i and discrete variables s_i. The entire system can be regarded as a coupled oscillator system because each constituent unit oscillates between $x_i = 0$ and 1, interacting with each other through the binary state s_i. This can also be viewed as a pseudo-billiard [30] in the hypercube $[0, 1]^N$, because the internal state $\mathbf{x}(t) = (x_1(t), \ldots, x_N(t))^\top$ moves linearly inside the hypercube, as shown in (17), and changes its direction only at the boundary, as shown in (18).

It has been numerically demonstrated that the chaotic Boltzmann machine serves as a deterministic alternative to the MCMC sampling from the Gibbs distribution, $P(\mathbf{s}) \propto \exp(-H(\mathbf{s})/T)$. It can be used in simulated annealing to solve combinatorial optimization problems and exhibits computing abilities comparable to those of the conventional Boltzmann machine.

The chaotic Boltzmann machine enables an efficient hardware implementation of the Boltzmann machine, primarily because it eliminates the need for a pseudorandom number generator and also because its mutual interactions are achieved digitally via the binary states. These advantages contribute to large-scale, energy-efficient hardware implementations of the chaotic Boltzmann machine, as demonstrated in analog CMOS VLSI and digital FPGA implementations [31, 32].

3.3 Ising Machines

Ising machines [33] are a class of specialized hardware designed to solve combinatorial optimization problems by finding the (approximate) ground state of the Ising model, which is an abstract model of ferromagnetism in statistical mechanics. They have attracted considerable attention in recent years because of their potential to efficiently solve complex optimization problems.

The energy function of the Ising model is given by:

$$H(\boldsymbol{\sigma}) = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij}\sigma_i\sigma_j \ , \tag{19}$$

where $\sigma_i \in \{-1, +1\}$ denotes the ith Ising spin. As is evident from the energy function, the Ising model is almost equivalent to the Boltzmann machine. For simplicity, we omit the linear bias term, which can be represented by introducing an additional spin fixed at $+1$. Coefficient J_{ij} represents the coupling strength between spins i and j, which is assumed to be symmetric $J_{ij} = J_{ji}$.

Ising machines are designed to find a spin configuration $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_N)^\top$ that approximately minimizes $H(\boldsymbol{\sigma})$. To solve a combinatorial optimization problem using an Ising machine, we need to formulate it as an Ising problem, in a way analogous to the Hopfield–Tank model and the Boltzmann machine.

We provide a brief overview of three types of Ising machines: the coherent Ising machine (CIM) [34], the simulated bifurcation machine (SBM) [35], and the oscillator Ising machine (OIM) [36]. We also introduce a continuous-time solver for boolean satisfiability (SAT) problems [37].

3.3.1 CIM: Coherent Ising Machine

The coherent Ising machine (CIM) [34] is a network of optical parametric oscillators (OPOs) designed to solve Ising problems.

The fundamental, noiseless dynamics of CIM can be described by the following ordinary differential equation:

$$\frac{dx_i}{dt} = (-1 + p - x_i^2)x_i + \sum_{j=1}^{N} J_{ij}x_j \ , \tag{20}$$

where x_i is the amplitude of the ith OPO mode and p represents the pump rate. Intuitively, the dynamics of CIM can be viewed as a variant of the Hopfield–Tank model with bistability introduced into each neuron. Basic dynamics of each OPO, without the coupling term, undergoes a pitchfork bifurcation at $p = 1$. That is, for $p < 1$, the equilibrium at $x_i = 0$ is stable; however, for $p > 1$, $x_i = 0$ becomes unstable and two symmetric stable equilibria $x_i = \pm\sqrt{p-1}$ emerge. These two equilibria in each OPO correspond to the binary state of spin $\sigma_i \in \{-1, +1\}$. Therefore, by gradually increasing the pump rate p, we expect the state of the OPO network to converge to a low-energy spin state, as each OPO is forced to choose one of the binary states by the inherent bistability. Thus, the obtained low-energy state corresponds to an approximate solution to the Ising problem.

3.3.2 SBM: Simulated Bifurcation Machine

The simulated bifurcation machine (SBM) [35] is an Ising machine described as an Hamiltonian system, which is given by the following ordinary differential equations:

$$\frac{dx_i}{dt} = \Delta y_i \ , \tag{21}$$

$$\frac{dy_i}{dt} = -(Kx_i^2 - p + \Delta)x_i + \xi_0 \sum_{j=1}^{N} J_{ij}x_j \ , \tag{22}$$

where x_i and y_i denote the position and momentum of the ith unit, respectively, and Δ, K, and ξ_0 are constants. Parameter p controls the bistability of each unit. This Hamiltonian system conserves the Hamiltonian

$$H_{\mathrm{SB}}(\mathbf{x}, \mathbf{y}) = \frac{\Delta}{2} \sum_{i=1}^{N} y_i^2 + V(\mathbf{x}) \ , \tag{23}$$

where the potential function $V(\mathbf{x})$ is given by

$$V(\mathbf{x}) = \sum_i \left(\frac{\Delta - p}{2} x_i^2 + \frac{K}{4} x_i^4 \right) - \frac{\xi_0}{2} \sum_{i,j=1}^{N} J_{ij} x_i x_j \ . \tag{24}$$

The SBM employs the symplectic Euler method, a structure-preserving time-discretization method, to conserve the Hamiltonian in simulating the Hamiltonian dynamics. Unlike the CIM, the SBM wanders the state space according to the complex Hamiltonian dynamics to explore low-energy states.

The SBM has been implemented on FPGA and GPUs, making it an efficient hardware solver of the Ising problems.

3.3.3 OIM: Oscillator Ising Machine

The oscillator Ising machine (OIM) [36] is a coupled nonlinear oscillator system that utilizes subharmonic injection locking (SHIL) to solve Ising problems. The dynamics of the OIM is given by:

$$\frac{d\phi_i}{dt} = -\sum_{j=1}^{N} J_{ij} \sin(\phi_i - \phi_j) - K \sin(2\phi_i) \ , \tag{25}$$

where ϕ_i denotes the phase of the ith oscillator. It has a global Lyapunov function,

$$E(\boldsymbol{\phi}) = -\sum_{i,j=1}^{N} J_{ij} \cos(\phi_i - \phi_j) - K \sum_i \cos(2\phi_i) \ , \tag{26}$$

which is guaranteed to never increase in time. The first term of the Lyapunov function corresponds to the Ising Hamiltonian, where the phase $\phi_i \in \{0, \pi\}$ modulo 2π represents the Ising spin $\sigma_i \in \{+1, -1\}$. The second term enforces the phase ϕ_i to be either 0 or π, where $\cos(2\phi_i) = 1$. As a result, the oscillator phases evolve to minimize the Ising Hamiltonian, converging toward a low-energy state that represents an approximate solution to the Ising problem. Further details regarding the OIM can be found in Chap. 9.

3.3.4 Continuous-time Boolean Satisfiability Solver

A continuous-time dynamical system (CTDS) for solving Boolean satisfiability (SAT) problems was proposed in [37]. This aims to find an assignment that satisfies the given Boolean formula. Although the system is not an Ising machine designed specifically for solving (quadratic) Ising problems, it seeks a set of binary states that minimizes a given objective function, which can be understood as an Ising Hamiltonian with high-order terms.

The CTDS solver explores the assignment of Boolean variables $X_1, \ldots, X_N$ satisfying the Boolean formula given in the conjunctive normal form (CNF). CNF is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals, which can be a Boolean variable X_i or its negation $\neg X_i$.

Essentially, the CTDS solver is a gradient system of an objective function $V(\mathbf{x})$, defined on the search space $\mathbf{x} = (x_1, \ldots, x_N)^\top \in [-1, +1]^N$, where the ith component $x_i \in \{-1, +1\}$ corresponds to the Boolean variable $X_i \in \{\text{False}, \text{True}\}$. To define the objective function $V(\mathbf{x})$, the CNF is represented as a matrix $[c_{mi}]$; each component c_{mi} takes $+1$ or -1 if the mth clause includes X_i or $\neg X_i$, respectively, and $c_{mi} = 0$ if neither is included. The objective function is defined as

$$V(\mathbf{x}) = \sum_{m=1}^{M} a_m K_m(\mathbf{x})^2 \,, \qquad K_m(\mathbf{x}) = \prod_{i=1}^{N} \frac{1 - c_{mi} x_i}{2} \,, \tag{27}$$

where $a_m > 0$ is the weight coefficient of the unsatisfiedness $K_m(\mathbf{x})$ to the current assignment $\mathbf{x}$ in the mth clause of the CNF. The objective function takes $V(\mathbf{x}) = 0$ if the CNF is satisfied, and takes a positive value otherwise. Therefore, the states with $V(\mathbf{x}) = 0$ constitute global minima of the objective function, regardless of the weight values $a_m > 0$. The CTDS solver is a gradient system of $V(\mathbf{x})$ with time-varying coefficients a_m defined as follows:

$$\frac{d\mathbf{x}}{dt} = -\nabla V(\mathbf{x}) \,, \qquad \frac{da_m}{dt} = a_m K_m(\mathbf{x}) \,. \tag{28}$$

If the mth clause is not satisfied, the weight a_m increases because of the positive unsatisfiedness $K_m(\mathbf{x}) > 0$, which modifies the objective function $V(\mathbf{x})$. This effect helps the dynamics to escape from local minima, which is similar to the refractoriness of chaotic neural networks. The CTDS solver exhibits a transient chaotic behavior until it converges to a global minimum. The interaction dynamics of $\mathbf{x}$ and a_m was investigated in [38].

Although the original CTDS solver is described as a gradient system of a time-varying objective function, its variant is represented by a recurrent neural network [39]. For efficient numerical simulation of the CTDS solver, structure-preserving time discretization using the discrete gradient is effective in the gradient part of the solver [40].

3.4 Herding System

The RNNs discussed in this section thus far have fixed connection weights and exhibit nonlinear dynamics in their network states. In contrast, the learning process of neural networks, which modifies the connection weights through a learning rule, as in (11) and (12), introduces nonlinear dynamics into the parameter space.

Herding is a deterministic sampling algorithm that can be viewed as an extreme case of parameter learning in statistical models [41, 42]. It exhibits complex dynamics, yielding sample sequences guaranteed to satisfy the predefined statistics asymptotically, which are useful for estimating other statistics of interest. Thus, statistical learning and inference are combined in the single algorithm of herding.

In this section, we introduce the herding algorithm as the zero-temperature limit of the Boltzmann machine learning. A more general and detailed description of the herding algorithm is provided in Chap. 10. The learning rule of the Boltzmann machine $P(\mathbf{s}) \propto \exp(-H(\mathbf{s})/T)$ including the temperature parameter T is given as follows:

$$w_{ij}(t+1) = w_{ij}(t) + \frac{\alpha}{T}\left(\langle s_i s_j\rangle_{\text{data}} - \langle s_i s_j\rangle_{\text{model}}\right) , \tag{29}$$

$$b_i(t+1) = b_i(t) + \frac{\alpha}{T}\left(\langle s_i\rangle_{\text{data}} - \langle s_i\rangle_{\text{model}}\right) . \tag{30}$$

Let us consider the low-temperature limit, $T \to 0$, which corresponds to the Amari–Hopfield network. That is, the model distribution $P(\mathbf{s})$ reduces to a point distribution on the minimizer of the Hamiltonian $\arg\min_{\mathbf{s}} H(\mathbf{s})$. Because the minimizer is invariant under the positive scalar multiplication of parameters, we can omit the scaling factor α/T, without loss of generality, to obtain the following update rule:

$$w_{ij}(t+1) = w_{ij}(t) + \langle s_i s_j\rangle_{\text{data}} - s_i(t)s_j(t) , \tag{31}$$

$$b_i(t+1) = b_i(t) + \langle s_i\rangle_{\text{data}} - s_i(t) , \tag{32}$$

where $\mathbf{s}(t) = (s_1(t), \ldots, s_N(t))^\top$ is the minimizer of the energy function with parameters at the tth iteration, that is,

$$\mathbf{s}(t) = \arg\min_{\mathbf{s}} \left(-\frac{1}{2}\sum_{i,j=1}^{N} w_{ij}(t)s_i s_j - \sum_{i=1}^{N} b_i(t)s_i \right) . \tag{33}$$

Equations (31)–(33) describe the herding system applied to the Boltzmann machine, which is a nonlinear dynamical system on the parameter space of w_{ij}'s and b_i's. In each update of the parameter values, we obtain a sample $\mathbf{s}(t)$. The sequence of network states, $\{\mathbf{s}(t)\}_t$, can be considered as a sample sequence from the neural network.

Interestingly, the idea of updating the parameters of the Amari–Hopfield network away from the equilibrium state $\mathbf{s}(t)$ was proposed as "unlearning" by Hopfield et al. [43]. Weakly updating the parameters suppresses spurious memories, which are undesirable local minima that do not correspond to any of the memory patterns stored through the Hebbian rule. Thus, the herding algorithm can be viewed as performing strong unlearning within the Amari–Hopfield network.

The herding algorithm is described as a discrete-time nonlinear dynamical system that belongs to the class of piecewise isometries. As with many piecewise isometries

[18, 44–46], the herding system typically exhibits complex dynamics with a fractal attracting set [41, 42]. As the Lyapunov exponents of the dynamics are strictly zero, the complexity originates only from the discontinuities of the piecewise isometry. This non-chaotic dynamics of the herding system is closely related to chaotic billiard dynamics [47].

As a sampling method, the herding algorithm exhibits a prominent convergence rate $O(1/\tau)$, which is significantly faster than $O(1/\sqrt{\tau})$ of random sampling algorithms, such as MCMC. Specifically, for a sample sequence of length τ, the deviation of the sample average of $s_i(t)s_j(t)$ from the target $\langle s_i s_j \rangle_{\text{data}}$ is given by:

$$\frac{1}{\tau}\sum_{t=1}^{\tau} s_i(t)s_j(t) - \langle s_i s_j \rangle_{\text{data}} = -\frac{1}{\tau}(w_{ij}(\tau) - w_{ij}(0)) \ , \tag{34}$$

which converges to zero as τ goes to infinity, at rate $O(1/\tau)$, because $w_{ij}(t)$ is assured to be bounded if the minimizer is obtained in each step (33).

The herded Gibbs sampling [48] is a deterministic sampling algorithm that incorporates the herding algorithm into the Gibbs sampling. It can be used as an alternative to the Gibbs sampling to promote efficient sampling from probabilistic models in general situations. The convergence behavior of the herded Gibbs sampling has been analyzed in detail [49].

4 Reservoir Computing

In the previous sections, we focused on the role of the energy function, which is crucial for understanding the dynamics of symmetrically connected RNNs. However, this approach is not applicable to RNNs with asymmetrical connections, which are more likely to exhibit complex nonlinear dynamics, making training more challenging. Reservoir computing is a lightweight approach that leverages the rich dynamics of an RNN for information processing without training the RNN itself, which is referred to as a reservoir. In this section, we present an overview of the fundamental models and concepts of reservoir computing, illustrating how the nonlinear dynamics of RNNs can be utilized in various computing applications.

4.1 Training Input–Output Relation of RNNs

In the RNNs presented in the previous sections, neurons do not receive explicit input from outside the networks, whereas in some cases, inputs are implicitly provided as the initial state for the autonomous dynamics of neural networks.

In this section, time-varying inputs are explicitly incorporated into neurons of an RNN, and we consider the input–output relation of the network. While feedforward

neural networks, such as multilayer perceptrons (MLPs), learn input–output relations for static information, RNNs can handle sequential information because the effects of past inputs remain within the network and subsequently influence its current state and output.

Let us consider an RNN model with time-varying inputs. The state $x_i(t+1)$ of the ith neuron at time $t+1$ is given by

$$x_i(t+1) = f\left(\sum_{j=1}^{N} w_{ij} x_j(t) + \sum_{k=1}^{K} w_{ik}^{\text{in}} u_k(t+1)\right), \tag{35}$$

where $u_k(t+1)$ denotes the kth input at time $t+1$, and w_{ij} and w_{ik}^{in} are the synaptic weights of recurrent and input connections, respectively. The activation function f is assumed to be tanh throughout this section. For the sake of simplicity, the bias term is omitted. The output from the lth output neuron is determined by

$$y_l(t+1) = f\left(\sum_{i=1}^{N} w_{li}^{\text{out}} x_i(t+1)\right), \tag{36}$$

where w_{li}^{out} is the weight of the output (readout) connection from the ith neuron.

Next, we consider the training of the neural network using input–output relation data. Specifically, given a pair of an input sequence

$$\mathbf{u}(t) = (u_1(t), \ldots, u_K(t))^\top, \qquad t = 1, \ldots, \tau, \tag{37}$$

and the corresponding output sequence

$$\mathbf{d}(t) = (d_1(t), \ldots, d_L(t))^\top, \qquad t = 1, \ldots, \tau, \tag{38}$$

we adjust the connection weights in the network to minimize or decrease the output error

$$E = \sum_{t=1}^{\tau} \|\mathbf{d}(t) - \mathbf{y}(t)\|^2, \tag{39}$$

where $\{\mathbf{y}(t)\}_t$ is the output sequence from the RNN given the input sequence $\{\mathbf{u}(t)\}_t$.

The backpropagation through time (BPTT) [50] and real-time recurrent learning (RTRL) [51] are well-known gradient-based algorithms for training RNNs. In computing the gradients of the output error E in (39), the gradients are recursively multiplied at each time step due to the influence of past inputs on the outputs. This often leads to the gradients either vanishing or exploding, which makes the gradient-based learning of connection weights in RNNs challenging. Various techniques have been proposed to address this problem. Consequently, recent RNN models, such as long short-term memory (LSTM) [52], effectively handle sequential data, whereas the computation of the gradients for training remains computationally expensive.

4.2 Echo State Network

The echo state network (ESN) [53, 54] is a type of reservoir computing that employs a different approach to training the input–output relations of RNNs. In ESNs, input weights and recurrent weights are typically generated randomly and remain fixed, while only the output connections are trained using a simple linear regression algorithm. This makes ESNs computationally efficient and easy to train. Instead of the nonlinear activation function in (36), the lth output from the ESN is obtained linearly as

$$y_l(t+1) = \sum_{i=1}^{N} w_{li}^{\text{out}} x_i(t+1) \ . \tag{40}$$

The underlying principle is that the state of a random RNN, or a reservoir, reflects the input sequence through nonlinear transformations. If the nonlinear dynamics of the reservoir is sufficiently rich, inferences can be performed effectively using only linear regression methods, such as ridge regression and FORCE (first-order reduced and controlled error) learning.

Ridge regression is a batch algorithm that can be used to train the readout connection weights, which introduces an L_2 regularization term, parameterized by $\alpha > 0$, into the error function as follows:

$$E_{\text{ridge}} = \sum_{t=1}^{\tau} \|\mathbf{d}(t) - W^{\text{out}}\mathbf{x}(t)\|^2 + \frac{\alpha}{2} \sum_{l=1}^{L} \sum_{i=1}^{N} |w_{li}^{\text{out}}|^2 \ . \tag{41}$$

The readout connection weight $W^{\text{out}} = [w_{li}^{\text{out}}]$ that minimizes the error function is given by:

$$W^{\text{out}} = DX^{\top}(XX^{\top} + \alpha I)^{-1} \ , \tag{42}$$

where I is the identity matrix, and $D = [\mathbf{d}(1), \ldots, \mathbf{d}(\tau)]$ and $X = [\mathbf{x}(1), \ldots, \mathbf{x}(\tau)]$.

The FORCE learning [55] is an online regression algorithm that updates W^{out} iteratively as follows:

$$P(t+1) = P(t) - \frac{P(t)\mathbf{x}(t)\mathbf{x}(t)^{\top}P(t)}{1 + \mathbf{x}(t)^{\top}P(t)\mathbf{x}(t)} \ , \tag{43}$$

$$W^{\text{out}}(t+1) = W^{\text{out}}(t) - \frac{\mathbf{e}(t)\mathbf{x}(t)^{\top}P(t)}{1 + \mathbf{x}(t)^{\top}P(t)\mathbf{x}(t)} \ , \tag{44}$$

$$\mathbf{e}(t) = W^{\text{out}}(t)\mathbf{x}(t) - \mathbf{d}(t) \ , \tag{45}$$

where $P(0) = I/\alpha$.

These linear regression algorithms require significantly less computation time compared to conventional gradient-based learning methods for RNNs. However, as these algorithms still involve the manipulation of large matrices, more lightweight

and biologically plausible learning algorithms [56, 57] can be employed for efficient reservoir computing.

4.3 Echo State Property and Reservoir Design

The primary principle of reservoir computing is that only the readout weights are trained. This implies that the performance of reservoir computing is largely dependent on the design of the reservoir.

The echo state property (ESP) is a concept that ensures a reservoir adequately reflects the input sequence, which is crucial for further information processing. When the inputs are transformed into reservoir states nonlinearly, it is important that the reservoir state becomes independent of its initial state after a sufficient amount of time has passed. This is critical because, without this property, the same input could yield different outputs, which is undesirable for the reproducibility of information processing. To prevent such inconsistencies, reservoirs are typically designed to satisfy the ESP.

Let $\mathbf{x}(t)$ and $\mathbf{x}'(t)$ represent the reservoir states with different initial states $\mathbf{x}(0)$ and $\mathbf{x}'(0)$, after receiving the same input sequence $\{\mathbf{u}(t)\}_t$. The ESP of a reservoir is defined as satisfying $\lim_{t\to\infty} \|\mathbf{x}(t) - \mathbf{x}'(t)\| = 0$ for any pair of different initial states $\mathbf{x}(0)$ and $\mathbf{x}'(0)$, and any input $\{\mathbf{u}(t)\}_t$.

Assuming tanh as the activation function, a sufficient condition for the ESP is that the largest singular value of $W = [w_{ij}]$ is less than 1. However, this condition is known to be empirically overly restrictive. Instead, we often require that W has the spectral radius of less than 1, which is expected to satisfy the ESP, though not necessarily in all cases [53].

However, using connection weights W with an excessively small spectral radius is undesirable, even though it indeed satisfies the ESP. If the spectral radius is small, the past input information is rapidly lost from the reservoir, making it difficult to effectively utilize a long input sequence for information processing. The memory to retain past input information can be measured using the memory capacity [58]. Empirically, the spectral radius is set slightly below 1 to promote both richer memory capacity and reservoir dynamics, expectedly without violating the ESP. Memory effects can also be enhanced by introducing leaky integrators as the constituent neurons.

The diversity of neuronal behavior is essential for the performance of reservoir computing, as the output is generated through a linear combination of neuronal activities in the reservoir. The diversity can be enhanced by introducing sparsity into the connection weight matrix W, because a dense W makes the inputs to the neurons become more similar. Increasing the number of neurons is also effective; however, this in turn can lead to an increased computation time and a higher risk of overfitting.

4.4 Neural Network Reservoirs

We have described basic concepts of reservoir computing using ESNs as a representative model for implementing a reservoir. However, it is important to note that reservoirs are not limited to this specific model.

Liquid state machines (LSMs) [59] are another important model of reservoir computing, proposed almost simultaneously and independently of ESNs. In contrast to ESNs, LSMs employ spiking neural network models, which are more biologically plausible and adequate for the modeling of information processing in the brain. LSMs also facilitate energy-efficient hardware implementations, such as FPGAs [60].

More generally, various RNN models, ranging from artificial neural network models to biologically plausible models, can serve as reservoirs. As described in Sect. 3, RNNs often exhibit chaotic dynamics. However, chaotic behavior is considered undesirable in reservoir computing, as its sensitive dependence on initial conditions contradicts the ESP. Therefore, attenuation mechanisms need to be introduced to ensure consistency. For instance, in a study on reservoir computing based on the chaotic neural networks [61], attenuation is achieved through connection weights W with a small spectral radius. Similarly, the chaotic Boltzmann machine serves as a reservoir by incorporating a reference clock to which each component is attenuated [62]. Moreover, an analog CMOS VLSI implementation of the chaotic Boltzmann machine has been utilized for energy-efficient reservoir computing [31].

As demonstrated in these examples, the fundamental concept of reservoir computing, which does not require manipulation of the reservoir itself, increases flexibility and enables efficient hardware implementations.

4.5 Physical Reservoir Computing

Furthermore, the concept of reservoir computing is not limited to utilizing RNNs as reservoirs. As there is no need to train the reservoir itself, any physical system exhibiting rich nonlinear dynamics can potentially be utilized as reservoirs.

The approach that leverages physical devices and phenomena as reservoirs, instead of using simulated models, is referred to as physical reservoir computing [63]. The nonlinear dynamics of the reservoir implemented physically is expected to be used for high-speed and energy-efficient computation.

A physical reservoir transforms the input to the reservoir state using its nonlinear dynamics. The output is obtained through a readout linear transformation of the measurements from the reservoir. As only the readout weights are trained, we do not have to manipulate the reservoir itself, which enables us to utilize various physical devices for computing. However, as in the case of ESNs, the performance of physical reservoir computing largely depends on the characteristics of the reservoir such as nonlinearity and memory effects. Building and tuning the physical reservoir, which may be sensitive to various environmental factors such as noise, can be challenging.

Therefore, it is crucial to select appropriate physical phenomena depending on the tasks performed by the reservoir.

Various types of physical devices and phenomena have been applied to reservoir computing [63]. Even if limited to photonic devices [64], there have been various studies utilizing optical node arrays [65, 66], optoelectric oscillators with delayed feedback [67–69], etc., as well as quantum dot networks [70] and optoelectric iterative-function systems [71] as presented in Chaps. 4 and 11 of this book.

5 Towards Photonic Neural Network Computing

We have seen how the nonlinear dynamics of RNNs can be utilized for computing. These basic models and concepts should serve as an important foundation for the implementation of neural networks using optical computing technologies.

Chapters in Part II of this book discuss fluorescence energy transfer (FRET) computing based on nanoscale networks of fluorescent particles, referred to as FRET networks. This can be considered as a type of physical reservoir computing in which FRET networks are employed as reservoirs.

Part III is devoted to spatial-photonic spin systems and is primarily related to Sect. 3 of this chapter. The spatial-photonic Ising machine (SPIM) introduced in Chap. 8 is an optical system capable of efficiently computing the energy function of the Ising model (19) using spatial light modulation. Recently, a new computing model for the SPIM has been proposed that improves its applicability to a variety of Ising problems and enables statistical learning as a Boltzmann machine [72]. Chapters 9 and 10 discuss the details of the herding system and OIM, which have been briefly introduced in this chapter.

Part IV consists of chapters discussing recent topics related to reservoir computing and its photonic implementation. In Chap. 11, a reservoir-computing system utilizing an electronic-optical implementation of iterated function systems (IFSs) as a reservoir is introduced. Chapter 12 introduces the hidden-fold network, which achieves high parameter efficiency by, in a sense, introducing the idea of reservoir computing into deep MLPs. Chapter 13 discusses brain-inspired reservoir computing, in which multiple reservoirs are hierarchically structured to model predictive coding for multimodal information processing.

Acknowledgements The author appreciates valuable comments from Professor Yuichi Katori and Dr. Hiroshi Yamashita.

References

1. W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 115–133 (1943). https://doi.org/10.1007/bf02478259
2. E.R. Caianiello, Outline of a theory of thought-processes and thinking machines. J. Theor. Biol. **1**, 204–235 (1961). https://doi.org/10.1016/0022-5193(61)90046-7
3. S. Amari, Learning patterns and pattern sequences by self-organizing nets of threshold elements. IEEE Trans. Comput. **C-21**, 1197–1206 (1972). https://doi.org/10.1109/T-C.1972.223477
4. J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. **79**, 2554–2558 (1982). https://doi.org/10.1073/pnas.79.8.2554
5. H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, T. Adler, D. Kreil, M.K. Kopp, G. Klambauer, J. Brandstetter, S. Hochreiter, Hopfield networks is all you need, in *International Conference on Learning Representations* (2021). https://openreview.net/forum?id=tL89RnzIiCd
6. D. Krotov, J.J. Hopfield, Large associative memory problem in neurobiology and machine learning, in *International Conference on Learning Representations* (2021). https://openreview.net/forum?id=X4y_10OX-hX
7. S. Amari, Characteristics of random nets of analog neuron-like elements. IEEE Trans. Syst. Man Cybern. **SMC-2**, 643–657 (1972). https://doi.org/10.1109/tsmc.1972.4309193
8. J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Natl. Acad. Sci. **81**, 3088–3092 (1984). https://doi.org/10.1073/pnas.81.10.3088
9. J.J. Hopfield, D.W. Tank, "Neural" computation of decisions in optimization problems. Biol. Cybern. **52**, 141–152 (1985). https://doi.org/10.1007/bf00339943
10. D.H. Ackley, G.E. Hinton, T.J. Sejnowski, A learning algorithm for Boltzmann machines. Cogn. Sci. **9**, 147–169 (1985). https://doi.org/10.1016/S0364-0213(85)80012-4
11. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing. Science **220**, 671–680 (1983). https://doi.org/10.1126/science.220.4598.671
12. J.H.M. Korst, E.H.L. Aarts, Combinatorial optimization on a Boltzmann machine. J. Parallel Distrib. Comput. **6**, 331–357 (1989). https://doi.org/10.1016/0743-7315(89)90064-6
13. P. Smolensky, Information processing in dynamical systems: foundations of harmony theory, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. by D.E. Rumelhart, J.L. McClelland (MIT Press, Cambridge, 1986)
14. G.E. Hinton, Training products of experts by minimizing contrastive divergence. Neural Comput. **14**, 1771–1800 (2002). https://doi.org/10.1162/089976602760128018
15. R. Salakhutdinov, G. Hinton, Deep Boltzmann machines, in *The 12th International Conference on Artificial Intelligence and Statistics (AISTATS 2009), vol. 5 of Proceedings of Machine Learning Research*, eds. by D. van Dyk, M. Welling (2009), pp. 448–455. https://proceedings.mlr.press/v5/salakhutdinov09a.html
16. R. Salakhutdinov, G. Hinton, An efficient learning procedure for deep Boltzmann machines. Neural Comput. **24**, 1967–2006 (2012). https://doi.org/10.1162/NECO_a_00311
17. K. Aihara, T. Takabe, M. Toyoda, Chaotic neural networks. Phys. Lett. A **144**, 333–340 (1990). https://doi.org/10.1016/0375-9601(90)90136-C
18. K. Aihara, H. Suzuki, Theory of hybrid dynamical systems and its applications to biological and medical systems. Philos. Trans. R. Soc. A **368**(1930), 4893–4914 (2010). https://doi.org/10.1098/rsta.2010.0237
19. J. Nagumo, S. Sato, On a response characteristic of a mathematical neuron model. Kybernetik **10**, 155–164 (1972). https://doi.org/10.1007/BF00290514
20. M. Hata, Dynamics of Caianiello's equation. J. Math. Kyoto Univ. **22**, 155–173 (1982). https://doi.org/10.1215/kjm/1250521865
21. M. Adachi, K. Aihara, Associative dynamics in a chaotic neural network. Neural Netw. **10**, 83–98 (1997). https://doi.org/10.1016/s0893-6080(96)00061-5

22. K. Kaneko, I. Tsuda, Chaotic itinerancy. Chaos **13**, 926–936 (2003). https://doi.org/10.1063/1.1607783
23. M. Oku, K. Aihara, Associative dynamics of color images in a large-scale chaotic neural network. Nonlinear Theory Appl. IEICE **2**, 508–521 (2011). https://doi.org/10.1587/nolta.2.508
24. M. Hasegawa, T. Ikeguchi, K. Aihara, Combination of chaotic neurodynamics with the 2-opt algorithm to solve traveling salesman problems. Phys. Rev. Lett. **79**, 2344–2347 (1997). https://doi.org/10.1103/PhysRevLett.79.2344
25. Y. Horio, K. Aihara, O. Yamamoto, Neuron-synapse IC chip-set for large-scale chaotic neural networks. IEEE Trans. Neural Netw. **14**, 1393–1404 (2003). https://doi.org/10.1109/tnn.2003.816349
26. Y. Horio, T. Ikeguchi, K. Aihara, A mixed analog/digital chaotic neuro-computer system for quadratic assignment problems. Neural Netw. **18**, 505–513 (2005). https://doi.org/10.1016/j.neunet.2005.06.022
27. Y. Horio, K. Aihara, Analog computation through high-dimensional physical chaotic neurodynamics. Physica D **237**, 1215–1225 (2008). https://doi.org/10.1016/j.physd.2008.01.030
28. H. Suzuki, J. Imura, Y. Horio, K. Aihara, Chaotic Boltzmann machines. Sci. Rep. **3**, 1610 (2013). https://doi.org/10.1038/srep01610
29. H. Suzuki, Monte Carlo simulation of classical spin models with chaotic billiards. Phys. Rev. E **88**, 052144 (2013). https://doi.org/10.1103/PhysRevE.88.052144
30. M. Blank, L. Bunimovich, Switched flow systems: pseudo billiard dynamics. Dyn. Syst. **19**, 359–370 (2004). https://doi.org/10.1080/14689360412331304309
31. M. Yamaguchi, Y. Katori, D. Kamimura, H. Tamukoh, T. Morie, A chaotic Boltzmann machine working as a reservoir and its analog VLSI implementation, in *2019 International Joint Conference on Neural Networks (IJCNN)* (2019). https://doi.org/10.1109/ijcnn.2019.8852325
32. I. Kawashima, T. Morie, H. Tamukoh, FPGA implementation of hardware-oriented chaotic Boltzmann machines. IEEE Access **8**, 204360–204377 (2020). https://doi.org/10.1109/access.2020.3036882
33. N. Mohseni, P.L. McMahon, T. Byrnes, Ising machines as hardware solvers of combinatorial optimization problems. Nat. Rev. Phys. **4**, 363–379 (2022). https://doi.org/10.1038/s42254-022-00440-8
34. T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P.L. McMahon, T. Umeki, K. Enbutsu, O. Tadanaga, H. Takenouchi, K. Aihara, K.-I. Kawarabayashi, K. Inoue, S. Utsunomiya, H. Takesue, A coherent Ising machine for 2000-node optimization problems. Science **354**, 603–606 (2016). https://doi.org/10.1126/science.aah4243
35. H. Goto, K. Tatsumura, A.R. Dixon, Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems. Sci. Adv. **5**, eaav2372 (2019). https://doi.org/10.1126/sciadv.aav2372
36. T. Wang, J. Roychowdhury, OIM: Oscillator-based Ising machines for solving combinatorial optimisation problems, in *Unconventional Computation and Natural Computation (UCNC 2019)*, eds. by I. McQuillan, S. Seki (2019), pp. 232–256. https://doi.org/10.1007/978-3-030-19311-9_19
37. M. Ercsey-Ravasz, Z. Toroczkai, Optimization hardness as transient chaos in an analog approach to constraint satisfaction. Nat. Phys. **7**, 966–970 (2011). https://doi.org/10.1038/nphys2105
38. H. Yamashita, K. Aihara, H. Suzuki, Timescales of Boolean satisfiability solver using continuous-time dynamical system. Commun. Nonlinear Sci. Numer. Simul. **84**, 105183 (2020). https://doi.org/10.1016/j.cnsns.2020.105183
39. B. Molnar, Z. Toroczkai, M. Ercsey-Ravasz, Continuous-time neural networks without local traps for solving boolean satisfiability, in *13th International Workshop on Cellular Nanoscale Networks and their Applications*. (IEEE, 2012). https://doi.org/10.1109/cnna.2012.6331411
40. H. Yamashita, K. Aihara, H. Suzuki, Accelerating numerical simulation of continuous-time Boolean satisfiability solver using discrete gradient. Commun. Nonlinear Sci. Numer. Simul. **102**, 105908 (2021). https://doi.org/10.1016/j.cnsns.2021.105908

41. M. Welling, Herding dynamical weights to learn, in *Proceedings of the 26th Annual International Conference on Machine Learning* (ACM, 2009). https://doi.org/10.1145/1553374.1553517
42. M. Welling, Y. Chen, Statistical inference using weak chaos and infinite memory. J. Phys. Conf. Ser. **233**, 012005 (2010). https://doi.org/10.1088/1742-6596/233/1/012005
43. J.J. Hopfield, D.I. Feinstein, R.G. Palmer, 'Unlearning' has a stabilizing effect in collective memories. Nature **304**(5922), 158–159 (1983). https://doi.org/10.1038/304158a0
44. A. Goetz, Dynamics of piecewise isometries. Illinois J. Math. **44**, 465–478 (2000). https://doi.org/10.1215/ijm/1256060408
45. H. Suzuki, K. Aihara, T. Okamoto, Complex behaviour of a simple partial-discharge model. Europhys. Lett. **66**, 28–34 (2004). https://doi.org/10.1209/epl/i2003-10151-x
46. H. Suzuki, S. Ito, K. Aihara, Double rotations. Discrete Contin. Dyn. Syst. A **13**, 515–532 (2005). https://doi.org/10.3934/dcds.2005.13.515
47. H. Suzuki, Chaotic billiard dynamics for herding. Nonlinear Theory Appl. IEICE **6**, 466–474 (2015). https://doi.org/10.1587/nolta.6.466
48. Y. Chen, L. Bornn, N. de Freitas, M. Eskelin, J. Fang, M. Welling, Herded Gibbs sampling. J. Mach. Learn. Res. **17**(10), 1–29 (2016). http://jmlr.org/papers/v17/chen16a.html
49. H. Yamashita, H. Suzuki, Convergence analysis of herded-Gibbs-type sampling algorithms: effects of weight sharing. Stat. Comput. **29**, 1035–1053 (2019). https://doi.org/10.1007/s11222-019-09852-6
50. P.J. Werbos, Backpropagation through time: what it does and how to do it. Proc. IEEE **78**, 1550–1560 (1990). https://doi.org/10.1109/5.58337
51. R.J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks. Neural Comput. **1**, 270–280 (1989). https://doi.org/10.1162/neco.1989.1.2.270
52. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**, 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
53. H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks, GMD Report 148, GMD Forschungszentrum Informationstechnik (2001). https://doi.org/10.24406/publica-fhg-291111
54. H. Jaeger, H. Haas, Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004). https://doi.org/10.1126/science.1091277
55. D. Sussillo, L.F. Abbott, Generating coherent patterns of activity from chaotic neural networks. Neuron **63**, 544–557 (2009). https://doi.org/10.1016/j.neuron.2009.07.018
56. G.M. Hoerzer, R. Legenstein, W. Maass, Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. Cereb. Cortex **24**, 677–690 (2012). https://doi.org/10.1093/cercor/bhs348
57. M. Nakajima, K. Inoue, K. Tanaka, Y. Kuniyoshi, T. Hashimoto, K. Nakajima, Physical deep learning with biologically inspired training method: gradient-free approach for physical hardware. Nat. Commun. **13**, 7847 (2022). https://doi.org/10.1038/s41467-022-35216-2
58. H. Jaeger, Short term memory in echo state networks, GMD Report 152, GMD Forschungszentrum Informationstechnik (2001). https://doi.org/10.24406/publica-fhg-291107
59. W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput. **14**, 2531–2560 (2002). https://doi.org/10.1162/089976602760407955
60. B. Schrauwen, M. D'Haene, D. Verstraeten, J.V. Campenhout, Compact hardware liquid state machines on FPGA for real-time speech recognition. Neural Netw. **21**, 511–523 (2008). https://doi.org/10.1016/j.neunet.2007.12.009
61. Y. Horio, Chaotic neural network reservoir, in *2019 International Joint Conference on Neural Networks (IJCNN)* (2019). https://doi.org/10.1109/IJCNN.2019.8852265
62. Y. Katori, H. Tamukoh, T. Morie, Reservoir computing based on dynamics of pseudo-billiard system in hypercube, in *2019 International Joint Conference on Neural Networks (IJCNN)* (2019). https://doi.org/10.1109/IJCNN.2019.8852329

63. G. Tanaka, T. Yamane, J.B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, A. Hirose, Recent advances in physical reservoir computing: a review. Neural Netw. **115**, 100–123 (2019). https://doi.org/10.1016/j.neunet.2019.03.005
64. G. Van der Sande, D. Brunner, M.C. Soriano, Advances in photonic reservoir computing. Nanophotonics **6**, 561–576 (2017). https://doi.org/10.1515/nanoph-2016-0132
65. K. Vandoorne, W. Dierckx, B. Schrauwen, D. Verstraeten, R. Baets, P. Bienstman, J.V. Campenhout, Toward optical signal processing using photonic reservoir computing. Opt. Express **16**, 11182 (2008). https://doi.org/10.1364/oe.16.011182
66. K. Vandoorne, P. Mechet, T.V. Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, P. Bienstman, Experimental demonstration of reservoir computing on a silicon photonics chip. Nat. Commun. **5**, 3541 (2014). https://doi.org/10.1038/ncomms4541
67. L. Appeltant, M.C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C.R. Mirasso, I. Fischer, Information processing using a single dynamical node as complex system. Nat. Commun. **2**, 468 (2011). https://doi.org/10.1038/ncomms1476
68. L. Larger, M.C. Soriano, D. Brunner, L. Appeltant, J.M. Gutierrez, L. Pesquera, C.R. Mirasso, I. Fischer, Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing. Opt. Express **20**, 3241 (2012). https://doi.org/10.1364/oe.20.003241
69. Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, S. Massar, Optoelectronic reservoir computing. Sci. Rep. **2**, 287 (2012). https://doi.org/10.1038/srep00287
70. N. Tate, Y. Miyata, S. Sakai, A. Nakamura, S. Shimomura, T. Nishimura, J. Kozuka, Y. Ogura, J. Tanida, Quantitative analysis of nonlinear optical input/output of a quantum-dot network based on the echo state property. Opt. Express **30**, 14669–14676 (2022). https://doi.org/10.1364/OE.450132
71. N. Segawa, S. Shimomura, Y. Ogura, J. Tanida, Tunable reservoir computing based on iterative function systems. Opt. Express **29**, 43164 (2021). https://doi.org/10.1364/oe.441236
72. H. Yamashita, K. Okubo, S. Shimomura, Y. Ogura, J. Tanida, H. Suzuki, Low-rank combinatorial optimization and statistical learning by spatial photonic Ising machine. Phys. Rev. Lett. **131**, 063801 (2023). https://doi.org/10.1103/PhysRevLett.131.063801