이학박사학위논문

Machine learning approach for phase transitions

기계 학습 방법론을 통한 상전이 연구

2021년 2월

서울대학교 대학원

물리·천문학부

최 광 종

이학박사학위논문

Machine learning approach for phase transitions

기계 학습 방법론을 통한 상전이 연구

2021년 2월

서울대학교 대학원

물리·천문학부

최 광 종

Machine learning approach for phase transitions

기계 학습 방법론을 통한 상전이 연구

지도교수 강병남

이 논문을 이학박사 학위논문으로 제출함

2021년 2월

서울대학교 대학원

물리·천문학부

최광종

최광종의 박사 학위논문을 인준함

2020년 12월

위원장: 민홍기 (인)

부위원장: 강병남 (인)

위원: 양범정 (인)

위원: 조정효 (인)

위원: 고광일 (인)

Abstract

Machine learning approach for phase transitions

Kwangjong Choi
Department of Physics and Astronomy
The Graduate School
Seoul National University

This study analyzed the quantum phase transition in a quantum contact process through machine learning approaches based on the artificial neural network and discovered an open quantum system's critical phenomenon different from a classical system. Also, we analyzed the critical phenomena of the synchronization transition of the Kuramoto model with machine learning approaches and predicted the dynamic behavior of the model. It showed that the machine learning approached is an alternative framework for numerical analysis for synchronization phenomena.

Chapter 1 outlines the conventional phase transition theory for critical phenomena and the general concepts of the machine learning approaches. The phase transition theory covers the critical phenomena and their universality near the critical point. Machine learning approaches define machine learning's essential elements and explain the model's type and model optimization as mathematical descriptions. Furthermore, we introduce artificial neural networks, which is a promising machine learning method.

Chapter 2 includes the machine learning approaches for quantum phase transition of the quantum contact process. Using the quantum jump Monte Carlo method, we simulate a one-dimensional spin chain following a quantum contact process. We train the artificial neural networks such as convolutional neural networks and fully-connected neural networks with supervised learning to detect whether the system is in an active state or absorbing states depending on the observed density of active sites. It is hard to estimate the critical point of the quantum phase transition using only the finite-size scaling, but we measure the critical point precisely by extrapolating the well-train the artificial neural networks' results. We employ the finite-size scaling for critical dynamics at the critical point estimated by machine learning and measure the one-dimensional quantum contact process's critical exponents. As a result, we discover that the critical exponent related to the active site density in a homogeneous initial state different from the classical directed percolation model and find that quantum phase transition exhibits new universality.

Chapter 3 includes the machine learning approaches for the synchronization transition of the Kuramoto model. We train the artificial neural networks such as recurrent neural networks and feed-forward networks with supervised learning to estimate the coupling constant, the strength of the interaction between oscillators from the dynamic behavior of oscillators governed by the Kuramoto model. Though the Kuramoto model's conventional order parameter can only estimate the coupling strength only in the synchronized state, the well-trained artificial neural networks measure the coupling strength among the oscillators in the synchronized asynchronous state. This result implicates that the artificial neural networks capture the order parameter for the synchronous state and the latent parameter for the asynchronous state. Also, we train the artificial neural networks such as convolutional neural networks and fully-connected neural networks with supervised learning to detect whether the system is in a synchronous state or asynchronous state according to the configuration of the oscillators' phase. Using extrapolation of the trained artificial neural network's outputs, we could estimate the critical exponent related to a correlation length between oscillators, which was not measured by the data collapse method. The machine learning approach can be an alternative to finite-size scaling methods, including the data collapse method, as a numerical framework for measuring the critical point and critical exponents of synchronization phenomena. Furthermore, as applications, we propose a reservoir computer and a recurrent neural network reproducing the Kuramoto model's dynamics or tracking the network of the interaction between oscillators.

Chapter 4 remarks on the results and the meaning of this study. As the quantum contact process is a typical model of the open quantum system, this study shows that the machine learning approach can be applied to the open quantum system beyond the classical and closed quantum systems. Though this study focuses on the Kuramoto model as a typical nonlinear dynamics model exhibiting synchronization transition, we expect that the artificial neural networks will be a significant breakthrough in follow-up studies to predict the dynamical behavior of the chaotic system and to illuminate synchronization phenomena.

Keywords: Machine learning, Supervised learning, Deep learning, Recurrent neural network, Open quantum system, Quantum contact process, Quantum phase transition, Non-linear dynamics, Kuramoto model, Synchronization transition

Student number: 2014-22378

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last decades, the rapid progress of computer processing devices, such as CPU, GPU, and TPU, and the advanced software for the parallel computation and optimization of linear algebra computing shed new light on machine learning.

Machine learning approaches based on artificial neural networks have recently expanded the scope of the industry application, such as prediction of the stock market in financial, new manufacturing invention of medicines and chemicals, computer vision by recognition of images, automatic driving, cognition of natural language, and forecasting climate change, and factory automation. Furthermore, machine learning covers numerous data called big-data from statistics of social science. For instance, machine learning introduces to classify groups on social network service, predict epidemic spreading, and distinguish the fake news on mass media. Beyond science, in humanity and arts, the machine learning approach is employed to interpret the ancient document, restore old paint, and imitate music and pictures.

Of course, in physics, there are attempts to apply machine learning to analyze data from experiments in particle physics and solid physics and detect phase transitions of classical and quantum models in statistical physics [1]. Here are three main reasons why physics employs machine learning approaches, including artificial neural networks.

First, we employ machine learning approaches to assist in developing and improving physics. Artificial neural networks can refine data and statistics from experiments [2–4] and replace classical fitting methods, such as at the least χ^2 fitting for statistical analysis. For instance, in solid physics, machine learning approaches estimate the solid matter's properties, such as the bandgap energy, the thermal conductivity, and the Curie temperature from experimental data. [5]

On the other hand, we use physical intuition and physics to solve challenging machine learning problems and invent new machine learning models. For example, reinforcement learning and unsupervised learning would need to calculate the expectation values for all possible model parameters. Still, this calculation taking much time is not possible in practice. Therefore, we use the Gipps sampling used in numerical simulation in statistical physics's models for the condensed matter and collective behavior for the calculations. Recently, quantum machine learning [6] has been proposed to overcome this adversity using the well-known quantum system. In this sense, Boltzmann machines [7] and restricted Boltzmann machines [8] were developed based on spin models such as Ising models and spin glass models. Furthermore, we use concepts and terminologies in statistical physics to interpret and to understand their results.

Finally, there are previous studies to compare physics with machine learning approaches. Those studies confirm that the machine learning approach results are consistent with physical theories for the same data and then apply the machine learning approach for frontier areas where the conventional physics methods do not work. Instead, excluding traditional physical methods and results, some studies apply machine learning approaches to analyze data from the experiments and numerical simulations by own-way and find new phenomena and interpretation for the physical system.

In many cases, researches for phase transitions with machine learning approaches follow the above purpose and motivation. Here, pioneer research establishes the machine learning approach by employing artificial neural networks to estimate the critical point and analyze the Ising model's critical phenomena [9]. The Ising model, the typical spin model, exhibits the second-order phase transition as decreasing the temperature without an external magnetic field in a two-dimensional lattice. This study designs artificial neural networks, including fully-connected neural networks and convolutional neural networks, to distinguish the ordered state from the disordered states for a given spin configuration with supervised learning and estimate the Curie temperature and critical exponent related to spin-spin correlation. They confirm that the results obtained by artificial neural networks are consistent with theoretical values and numerical values through the finite-size scaling, which is conventional numerical methods for analyzing critical phenomena. Then, they argue that the machine learning approach is valid in the classical spin models. Then they extend their approach to the square ice model and the Ising gauge model, in which it is hard to use the magnetization to identify the phase transition as the order parameter. Following the above study, the machine learning approaches based on artificial neural networks are used to analyze another model's critical phenomena, such as the long-range Ising model [10], two dimensional XY model, two-dimensional bond/site percolation model [11], and the traverse-field Ising model [12]. Here, the percolation model is the mathematical diffusion model, and the traverse-field Ising model is the typical model for a quantum system.

Another approach to analyzing phase transitions is by using unsupervised learning. Some studies generated the spin configuration of the spin model, such as the two-dimensional Ising model and XY model, then inputted them into the machine learning model, including artificial neural networks such as the auto-encoder and the restricted Boltzmann machine. After the machine learning model is trained with unsupervised learning, they measure the latent parameters from the trained machine learning model and compare the order parameter to find the correlation between latent parameters and the order parameter [13]. Moreover, using the latent parameters, the other studies predict the phase transition or find a new phase for the spin model, such as the long-range Ising model [14] and the Blume-Capel model [15]. For unsupervised learning, although some studies employ artificial neural networks such as the (restricted) Boltzman machine and auto-encoder, most researches still perform with classical models such as the PCA (principal component analysis) and the t-SNE (t-distributed stochastic neighbor embedding) [10, 11, 15, 16].

Our study aims to understand the phase transitions and estimate the critical point and critical exponents using machine learning approaches based on artificial neural networks and supervised learning. Previous studies cover the classical spin model such as the Ising model and XY models and the closed quantum model such as the transverse-field Ising model. Still, there is no research discovering the phase transitions of the open quantum and the nonlinear dynamical system. This study will employ artificial neural networks to detect the open quantum system's phase transition and analyze nonlinear dynamical systems' synchronization transitions. Then, we will determine the limit of the application range of machine learning approaches.

Chapter 2 covers the machine learning approach for the quantum contact process, a typical model for an open quantum system. The classical contact process, the prototype model of a quantum contact process, consists of sites with two states; active and inactive states. Diseases spreading motivates the propagation rule of the contact process. An active site becomes inactivated spontaneously with a rate γ, and an inactivated site is activated by the neighboring activated site with a rate κ. For the theoretical reason, the quantum contact process is developed as an extended contact process, in which the state of a site is determined with probability. We describe the quantum contact process such that the density matrix of the system follows the Lindblad equation [17]. The Lindblad equation consists of a coherent Hamiltonian and an incoherent dissipative term with Lindblad operators, where the omega means the strength of quantum effects from Hamiltonian and the kappa means the strength of classical effects from Lindblad operators. For the geometrical dimension equal to or langer than the upper critical dimension, it is well-known that the type of phase transition follows the result of the mean-field approximation. However, it has been questioned whether the one-dimensional quantum contact process's universality follows the classical directed percolation universal class or exhibits a new quantum universal class. Recently, a quantum contact process has realized with a one-dimensional spin chain made up of the Rydberg atoms [18]. Finding the universality of the one-dimensional quantum contact process satisfies academic and physical needs. However, it is hard to estimate critical exponents using the only quantum jump Monte Carlo simulation. It is a reason why we introduce the machine learning approaches for the quantum contact process.

Chapter 3 covers machine learning approaches for the Kuramoto model, a typical nonlinear dynamical model that exhibits the synchronization transition. Machine learning has been used to analyze and predict brain signals such as EEG [19–22], while there are other studies to predict the dynamics of the chaotic system through machine learning [23–30]. However, it is challenging to cover analyzing data from the real world and predicting the dynamical system using machine learning approaches. Here, the Kuramoto model is a prototype model to study the synchronization transition and describe the power grid [31, 32] and the brain nervous system [33]. Nevertheless, there is no research to analyze the dynamic behavior and synchronization transition of the Kuramoto model using machine learning based on artificial neural networks. It is a reason why we introduce the machine learning approaches for the Kuramoto model.

We will confirm that the machine learning approaches can numerically estimate the critical points and the critical exponent of the quantum phase transition of open quantum systems and synchronization transition of chaotic nonlinear dynamical systems. As our study results, we expect that machine learning based on artificial neural networks will help analyze the critical phenomenon of phase transition, which was difficult to analyze with finite-size scaling methods.

This chapter reminds terminology and the basic theory of phase transitions and introduces the fundamental concepts of machine learning, including artificial neural networks.

1.1 Theory of phase transitions

The kinetic theory for thermodynamics is the cornerstone of statistical mechanics. Since then, statistical physics try to understand macroscopic phenomena as the collective behavior of a microscopic many-particle system consisting of at least Avogadro number ($N_A = 6.02214076 \times 10^{23}$) particles. Statistical physics is a theoretical bridge between the macroscopic and microscopic worlds.

In everyday life, we encounter that the water, a vital substance of our lives, transforms its states depending on the temperature and the pressure. At the pressure of one atm (the standard atmosphere), the liquid water freezes up ice when the temperature is lower than 273.15 K or evaporates up steam (vapor water) when the temperature is higher than 373.13 K. The macrostate of a substance is called phase, for example, ice is a solid-state of the water, and steam is a gas-state of the water. Roughly speaking, the phase can also be changed with an external macroscopic parameter such as temperature and pressure. The change of the phase is called a phase transition. Here, phase transitions theory [34–37] aims to define the phase, describe quantitatively and mathematically the phase and phase transition, and understand their underlying mechanisms.

Phase and phase transition Let us return to the example of water, how we distinguish among the states even though ice, water, and steam are all composed of the same substance, H_2O molecules. To define the system phase, we introduce a state function measured in macroscopic time and space lengths. We can employ the macroscopic physical quantity such as the density, the entropy, and the free energy as the state function depending on external variables.

For example, we can use density ρ, the number of particles per volume, as a state function for the water system. Figure 1.1(b) shows an isobar process where the density of water changes for the temperature with the constant pressure. When the temperature increases to evaporate from the liquid water to the steam, the density decreases smoothly in each liquid-state and gas-state, but the density drastically changes from ρ_g to ρ_l at the particular temperature T_t. Here, we call a domain where the state function is analytical a *phase* and the singular (non-analytic) point of the state function a *transition point*. A phenomenon in which the state function exhibits singularities according to varying the external parameter and is divided into the different analytic regions is

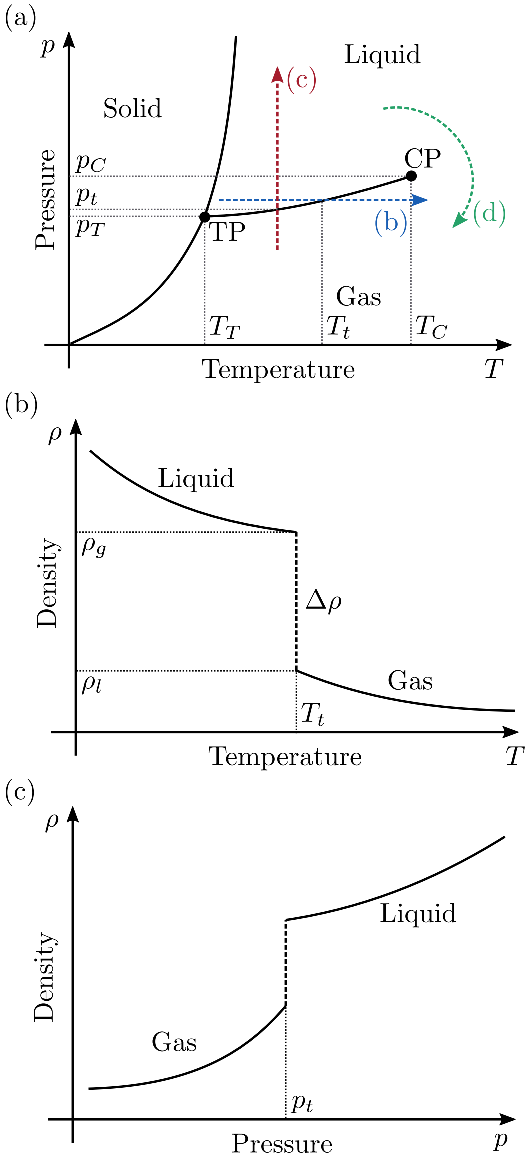

Figure 1.1: Plots of the phase transition of the water. (a) Typical phase diagram with temperature and pressure in an equilibrium state. The solid line denotes the transition curve, which is a set of transition points, TP denotes the triple point where the three transition curves join, and CP denotes the critical point where the transition curve ends. A dashed line (b) indicates an isobar path, the line with constant pressure. A dashed line (c) indicates an isotherm path, the line with a constant temperature. A dashed line (d) indicates a path bypassing the critical point without the phase transition. (b) Plot of the density for the temperature along the dashed line (b) in panel (a). (c) Plot of the density for the pressure along the dashed line (c) in panel (a).

a *phase transition*. Similarly, Figure 1.1(c) shows the phase transition from the gas-state to the liquid-state, increasing the pressure under the constant temperature. In the isotherm process, the phase is determined by pressure, and the density has a singularity at the transition pressure p_T.

We can then obtain discontinuous points in the density by repeating the isotherm process for each pressure or isobar process for each temperature. Figure 1.1(a) shows the phase diagram with the solid line representing a transition curve, a set of singular points of the state function. For example, the transition curve exhibits the transition temperature for each isobar process. Thus, the transition curve divides external parameter space into three phases, solid, liquid, and gas. The transition curve is called a coexistence curve since the system exhibits a mixed state in phases facing the transition curve. Remarkably, three phases can exist simultaneously at the triple point, where the three transition curves join. Refer to Figure 1.1(a), the water's triple point is the temperature (T_T) of 273.16 K and the pressure (p_T) of 0.00604 atm, and we can obtain the mixture of ice, water, and steam at this point.

An anomalous point in the phase diagram is a critical point where the transition curve ends, and the phase transition is terminated. Near the critical point, physically anomalous phenomena called critical phenomena occur. An example of a critical phenomenon is critical opalescence when the correlation length and the scattering cross-section diverge, and it causes the transparent to cloudy. The water's critical point is the critical temperature (T_C) of 647.096 K and the critical pressure (p_C) of 217.75 atm, and it is hard to distinguish whether the water is a liquid-state or a gas-state with the density in high temperature over the critical temperature or high pressure over the critical pressure.

The presence of the critical point raises one question: Following the thermodynamic process along a path that bypasses the critical point, as shown in the dashed line (d) in Figure 1.1(a), it seems like the phase transition occurs from liquid water to vapor water, but there is no transition point in the path (d). However, this irony is the result of misunderstanding the phase transition as everyday words. Strictly speaking, there is no phase transition, and we cannot separate the water state into liquid and vapor following the thermodynamic process along the path (d) because the density is continuous and has no singularity. Thus, to define the phases in the external parameters space, we need the state function and also consider the thermodynamic process relying on the path.

Phase transition theory conventionally uses free energy as a state function for physical systems because we can derive the free energy from a classical system governed by Hamiltonian mechanics to open quantum systems described by the Lindblad equation.

We classify the type of phase transition with the free energy function's analytic behaviors near the transition point. We order the phase transition as the lowest order derivatives of the free energy, showing discontinuity or divergence at the transition point. Therefore, the free energy exhibits a singular behavior such as a discontinuous jump or non-differentiable in the first-order phase transition at the transition point. However, the free energy is continuous in the second-order or the higher-order phase transition at the transition point, and we determine the phases and transition points with the first derivative of free energy in the second-order phase transition.

Thus, free energy is the heart of the phase transition theory to determine the phases and classify the type of phase transition.

On the other hand, it is hard to define free energy of the phase transition in the non-physical system, such as the synchronization transition of fireflies and a flock of birds, a pandemic outbreak resulting from disease spreading, and the global outage in a power grid and mathematical models for these phenomena. As to extend the phase transition theory to a mathematical model without free energy, there are studies to identify mechanisms governing the system's dynamics and find the model's state function corresponding to free energy. For example, Kasteleyn-Fortuin formalism [38] was developed to connect the percolation model [39] with the q-state Potts model [40] to analyze percolation transitions in the percolation model, a mathematical model of propagation and diffusion. See Appendix D. There are also studies in the Kuramoto model to interpret the self-consistent equation as Landau free energy to explain the synchronization transition [41].

Critical phenomena As we mentioned above, the phase transition theory particularly interests in the critical phenomena where the phase transition terminates.

Return the example of water, and we consider the difference in density between liquid water and water vapor at the transition temperature. Increasing the transition temperature, as shown in Figure 1.2(a), the density difference reduces and goes to zero when the transition temperature over the critical temperature. We take the density as the order parameter for water because the density means, roughly speaking, the average distance among water molecules or the occupied volume where a particle freely moves.

In this way, we define the order parameter as the average value of the microscopic quantity characterizing the particle's state. On the other hand, the control parameter is an external macroscopic parameter such as temperature and pressure. Due to a particle's microscopic state relying on the model, we should determine which physical quantity is proper as the order parameter for each model.

For example, in the Ising model, a prototype model of magnetic materials, a particle i has two spin states, up $\sigma_i = 1$ and down $\sigma_i = -1$. The Hamiltonian governing the Ising model follows

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - h \sum_i \sigma_i, \tag{1.1}$$

where J denotes the coupling constant representing interaction strength between two spins, h denotes an external magnetic field, and $\langle i, j \rangle$ denotes an interaction pair of neighboring two spins i and j. When $J > 0$, the Ising model describes the ferromagnetic material. Here, we take the magnetization m, the average value of all spin states,

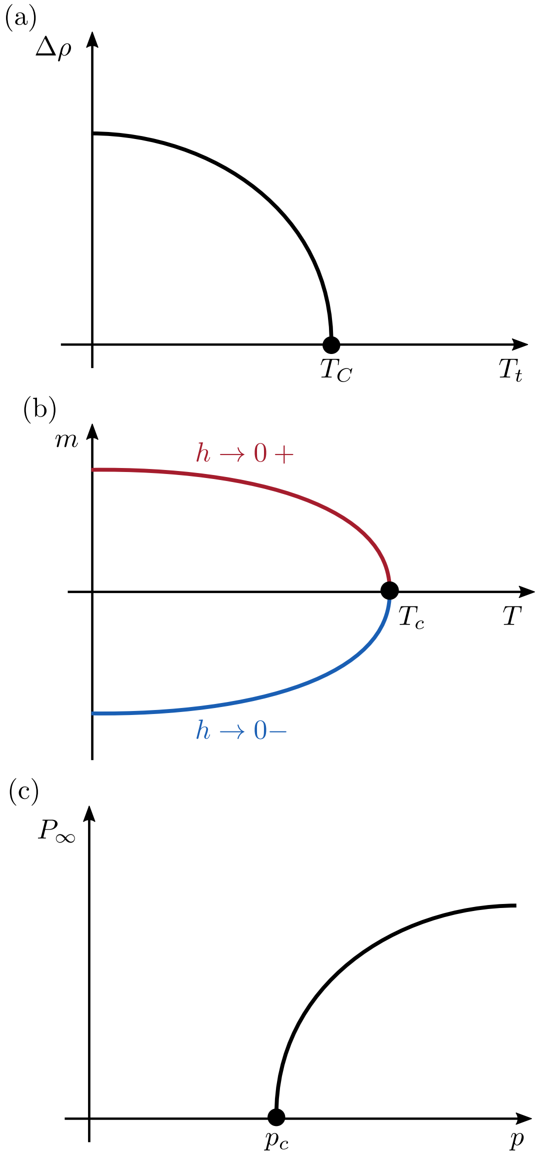

Figure 1.2: Plots of the order parameter for the control parameter (a) in the water system, (b) in the Ising model on the two-dimensional lattice, and (c) in the percolation model on the random graph [42]. (a) In the water system, the order parameter is the difference between the density of liquid water ρ_l and water vapor ρ_g, and the control parameter is the transition temperature T_t. (b) In the Ising model, the order parameter is magnetization m, which is the sum of all spin states, and the control parameter is the temperature T. (c) In the percolation model, the order parameter is the probability that a site belongs to the giant (infinite) cluster P_∞, and the control parameter is the occupation probability p.

as the order parameter as follows,

$$m = \frac{1}{N} \sum_i \sigma_i, \tag{1.2}$$

where N is the system size. Figure 1.2(b) shows that the Ising model's order parameter on the two-dimensional lattice for the temperature T.

When the temperature is over the particular temperature T_c, called the Curie temperature or the critical temperature, the system exhibits paramagnetism. So, with the external magnetic field going to zero, the magnetization also goes to zero. Meanwhile, the temperature is below the critical temperature, and then the system has exhibits ferromagnetism and has a finite magnetization without the external magnetic field. Here, a sign of magnetization depends on the direction of decreasing the external magnetic field. Of course, the positive direction ($h \to 0+$) makes the magnetization positive ($m > 0$), the negative direction ($h \to 0-$) makes the magnetization negative ($m < 0$).

When $J < 0$, the Ising model describes the anti-ferromagnetic material and uses the staggered magnetization instead of the ordinary magnetization as the order parameter. Therefore, even though the model is the same, we may choose a different quantity as the order parameter relying on what phenomenon we want to discover. Sometimes, it is a significant issue to determine the proper order parameter for a complicated model.

Although a mathematical model without the Hamiltonian does not have free energy, the order parameter can be defined if a particle has a microscopic state.

The percolation model is a good example. Let us consider a network consisting of N sites and B bonds, and a bond connects two sites. In the bond percolation, a bond has two states, an occupied state or a vacant state. Bond percolation occupies a bond with the probability of p and empties a bond with the probability of $1 - p$, where p is called an occupation probability. After percolating the network, we call the set of sites linked with occupied bonds a cluster, and the size of the cluster denotes the number of sites belonging to the cluster. Increasing the occupation probability, the giant cluster, which has the size coming near the system size, emerges, and a site has two states, belonging to the giant cluster or not.

Here, we conventionally use the probability that a site belongs to the giant cluster as the order parameter, and Figure 1.2(c) shows the order parameter for the occupation probability. When the occupation probability is lower than the critical point, the giant cluster does not exist, and then the order parameter is zero. However, the occupation probability is over than the critical point, and then the percolation transition occurs.

As shown in Figure 1.2, if we set the physical quantity as the order parameter appropriately, the order parameter is divided into two regions for the control parameter: the disordered state and the ordered state, and the boundary of the two regions is the critical point. The order parameter is zero in the disordered state. On the other hand, the order parameter is finite in the ordered state.

Sometimes, the disordered state area is called a disordered phase, and the area of the ordered state is called an ordered phase because, in many physical models, the region of the disordered and ordered states corresponds to the phase defined by free energy, respectively.

For these empirical reasons, when it is hard to obtain the analytical form of free energy or calculate the singularity of free energy, we consider a desperate measure as the order parameter to distinguish the system's phase even though the Hamiltonian is given. The order parameter is useful for a mathematical model without Hamiltonian in which free energy is not defined. However, distinguishing phases with only the order parameter's value may not be appropriate, strictly speaking, because there is no guarantee that the disordered state defined by the order parameter's value as the system microscopic parameter is the same as the disordered phase defined by the singularity of free energy as the system macroscopic observable. Once again, we note that the textbook models also use the disordered state and the disordered phase based on well-known results, but we should care to distinguish the phase based on the order parameter for a new phase transition.

The order parameter represents the average of the particle's microscopic state. When the system is in a disordered state, the possible particle's states are distributed evenly, and the order parameter goes to zero. When the state distribution is equal, we say that the system has a symmetry of states in the disordered state. However, when the system is in the ordered state, the control parameter as an external variable drives the particles to have a particular state, and the order parameter has non-zero. So, in the ordered state, the system's symmetry is broken through the critical point. This symmetry breaking is a fundamental mechanism underlying critical phenomena.

Let us look at examples of symmetry breaking at the critical point. First, water has isotropic movement in the vapor state, where the interaction such as the van der Waals force between water molecules occurs at a short time in the collisions among the molecules. However, water has anisotropic movement in the liquid state, where the distance between molecules is close to interact with each other, and the particles align along the particular direction. When the transition temperature is below the critical temperature, at the transition point, the water may have two states, isotropic or anisotropic states, and then the symmetry of two states is represented as the difference between the density of the liquid and the vapor states. However, when the temperature is higher than the critical temperature, we cannot distinguish two states, the symmetry of states breaks, and the density difference as the order parameter goes to zero.

The Ising model exhibits a more intuitive example of symmetry breaking. As shown in Figure 1.2(b), when the temperature is higher than the critical temperature without the external magnetic field, a particle has an equal chance of the up or down spins, and then the magnetization as the average spin value goes to zero. Meanwhiles, when the temperature is below the critical temperature, decreasing the external magnetic field, some of the particles remain as an aligned spin cluster depending on the direction of the external field. If up spin clusters remain, the magnetization is positive; otherwise, the magnetization is negative. So, we understand the critical phenomena in the Ising model as the breaking of $\mathbb{Z}_2$ symmetry.

For instance, the percolation transition relates to the breaking of the translation symmetry on the Euclidean lattice, and the synchronization transition relates to the breaking of the rotation symmetry in the frequency phase space.

As the control parameter approaches the critical point in the ordered state, the order parameter exhibits scaling behavior near the critical point. For example, the magnetization as the Ising model's order parameter goes to zero following the power-law scaling such as

$$|m| \sim |t|^{\beta}, \quad (T < T_c). \tag{1.3}$$

Here, $t = (T - T_c)/T_c$ is reduced temperature, the dimensionless quantity, $\sim$ denotes asymptotics, and β is the critical exponent (index) for the order parameter representing the degree of scaling behavior. The phenomenon of order parameters following the power-law near the critical point is one of the significant critical phenomena.

Other physical quantities, including the order parameters, also exhibit divergence or convergence following the power-law scaling. For instance, in the Ising model, interesting physical quantities such as (magnetization) susceptibility ξ, a heat capacity C, a correlation length ξ, and a two-point correlation function $G(r)$ diverge following the power-law near the critical point; Refer to Table 1.1. Critical exponents (α, β, γ, δ, ν, τ, ...) represent the degree of each physical quantity's singularity and scaling behavior near the critical point.

Furthermore, we define the critical exponent differently for the disordered state and the ordered state. For example, the susceptibility diverges along with each direction, and then γ and γ' denote scaling behavior for the disordered state and the ordered state, respectively. Two critical exponents, γ and γ', have the same value in the Ising model. However, in general, there is no guaranty that two critical exponents are always the same. Moreover, for discontinuous transitions, the critical exponent for susceptibility is only defined in the ordered state.

There are relations called the scaling law among the critical exponents, and then the

Physical quantity	Scaling bebavior and critical exponent
Magnetization, m	$\lvert m\rvert \sim \lvert t\rvert^{\beta}$ $(T < T_c)$, $\lvert m\rvert \sim \lvert h\rvert^{1/\delta}$ $(T = T_c)$
Susceptibilty, χ	$\chi \sim \lvert t\rvert^{-\gamma}$ $(T > T_c)$, $\chi \sim \lvert t\rvert^{-\gamma'}$ $(T < T_c)$
Heat capacity, C	$C \sim \lvert t\rvert^{-\alpha}$ $(T > T_c)$, $C \sim \lvert t\rvert^{-\alpha'}$ $(T < T_c)$
Correlation length, ξ	$\xi \sim \lvert t\rvert^{-\nu}$ $(T > T_c)$, $\xi \sim \lvert t\rvert^{-\nu'}$ $(T < T_c)$
Two point correaltion function, $G(r)$	$G(r) \sim r^{-\tau} e^{-r/\xi}$ $(T \neq T_c)$, $G(r) \sim r^{-d+2+\eta}$ $(T = T_c)$

Table 1.1: Critical exponents and their scaling behavior in the typical spin model. Here, we define a two-point correlation function such as $G(r) = \langle \sigma_i \sigma_j \rangle - \langle \sigma_i \rangle \langle \sigma_j \rangle$, where $r = |r_{ij}|$. d denotes the spatial dimension of the system.

critical exponents are not independent. One of the scaling relations is the Rushbrooke scaling law as follows

$$\alpha' + 2\beta + \gamma' = 2. \tag{1.4}$$

When we consider the real space's dimensionality embedding the system, the critical exponents and the spatial dimension d satisfy the following scaling laws,

$$\begin{aligned} \beta\delta &= \beta + \gamma', & (1.5)\\ \beta &= \frac{1}{2}(d - 2 + \eta)\nu', & (1.6)\\ d\nu' &= 2 - \alpha', & (1.7)\\ \gamma' &= (2 - \eta)\nu'. & (1.8) \end{aligned}$$

We call the relation with the critical exponents and spatial dimension the hyper scaling relation. The Ising model has the same critical exponents for disordered and ordered states, and there are only two degrees of freedom to determine the critical exponents and describe the critical phenomena.

Here, we encounter an essential question: what physical quantities of the Hamiltonian determine the critical exponents. According to the renormalization theory, there are two main factors to dominate the critical exponents, the degrees of freedom related to the symmetry of the particle's states and the spatial length of interaction among the particles.

First, parameters related to the particle's state and the state's symmetry, such as the number of particle's states and the state space's geometry, could change the critical exponents. For example, the Potts model generalized the Ising model describes q-states particles, and the critical exponents in the Potts model depend on the number of possible states, q.

Second, parameters related to the spatial length of the interaction among particles, such as a range of interaction and the dimension of the space on which the system is, determine the critical exponents' value. For instance, there are only interactions between the nearest neighboring two-spin pair in the Ising model. However, in the long-range Ising model, a spin interacts with the other spin with an interaction strength relying on the distance between the two spins. Therefore, the long-range Ising model has different critical exponents' values from the ordinary Ising model; even the interaction range changes the phase transition type. Also, the critical exponents vary on the spatial dimension, determining the distance and the coordinates. For example, the critical exponent's values of the Ising model on the two-dimensional lattice are different from the three-dimensional lattice. We note an upper critical dimension of the Ising model as MATH, and the critical exponents in the dimension higher or equal to the upper critical dimension are the same values in the mean-field approximation.

However, the details of the model do not affect the critical exponents. For example, the Ising model in the two-dimensional lattice has the same critical exponents, regardless of lattice formation, such as triangle lattice and square lattice. The amplitude of spin and the interaction strength unit does not change the critical exponents' value.

There are various phase transitions, and each model different with substance consisting of the system and the interaction strength in detail. Nevertheless, if systems have the same symmetry of the states and the interaction range (length), the different systems exhibit the same phase transition with the same value of critical exponents. That is why the critical phenomena are universal. In this sense, we call a set of critical exponents universal class to determine the critical phenomena' universality.

1.2 Machine learning

Machine learning is a theoretical framework for identifying data.

As shown in Figure 1.3, machine learning involves a series of processes in which we accumulate data like signals, pictures, images, and statistics from nature, society, and experiments, design a model to describe the data, train the model based on the

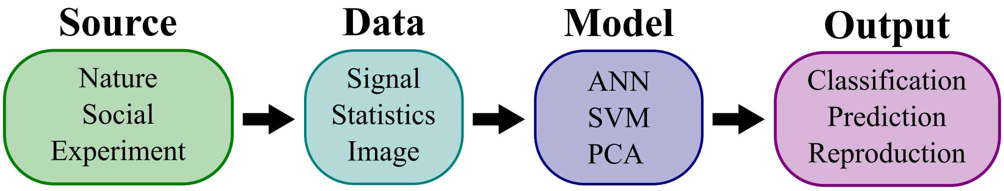

Figure 1.3: A flowchart of the general process in machine learning.

data, and then classify or reproduce new data through the trained model.

This sequence of processes is very similar to the methodology of modern science, especially physics. It is historically correct that scientific procedures, including modeling, experiments, and validation, mathematical approaches in physics motivate machine learning. Machine learning inherits the methodology of physics by extending the subject from nature to general data.

However, machine learning differs from physics in detail; refer to Table 1.2. As mentioned above, physics only focuses on nature, but machine learning does not ask where the data source is and needs the given data to be quantified.

Physics establishes theories and models with physical operators corresponding to measurements in an experiment or a system. Moreover, physics develops theories based on fundamental principles, such as the least action principle, the uncertain principle, and the Lorentz invariant. Meanwhile, a designer is free to build a machine learning model's architecture, and the model should be trained based on data. A physical theory is validated through new experimental results, and then we determine whether the theory is accepted, rejected, or modified. Likewise, machine learning models is also validated through new data, and then we determine whether the model is verified, discarded, or refined.

There is a difference between machine learning and physics in introducing variables. Physics has fundamental physical constants, such as the speed of light in vacuum c, the gravitational constant G, the Planck constant h, the electric constant ϵ_0, and the elementary charge e, and introduces a variable with physical meaning based on physical constants. Physical theory elucidates the connection between physical variables and

	Machine learning	Physics
Data source	Everywhere	Nature
Model	Data-drived	Operator-based
Parameters	Multi-variables	Physical quantities
Principle	Minimization of loss function	Least action principle

Table 1.2: a comparison between machine learning and physics.

physical constants and reduces the degree of freedom of variables. However, machine learning employs multi-variable models and does not require each model parameter to have any physical meaning. Sometimes, the number of model parameters closes to the number of data points. Physics can derive the physical meaning of theoretical results using the physical variables, while machine learning has flexibility in the design of models, but it is hard to infer the meaning of the model's results.

Machine learning is called data science or data mining, depending on which process to focus on [43], and is sometimes considered a branch of artificial intelligence implementation; refer to Figure 1.4. We classify the machine learning approaches into unsupervised learning, supervised learning, and reinforcement learning depending on the problem and requirements, the purpose of a model, and the learning process. Besides, depending on the model's structure and optimization algorithms, we have various machine learning methods, such as classical models including the support vector machine, PCA, and t-SNE, and recently re-illuminated artificial neural networks.

This study employs artificial neural networks for the model structure and supervised learning for model optimization. This section mathematically describes the basic concept and essential elements of machine learning to understand what the above methodology means.

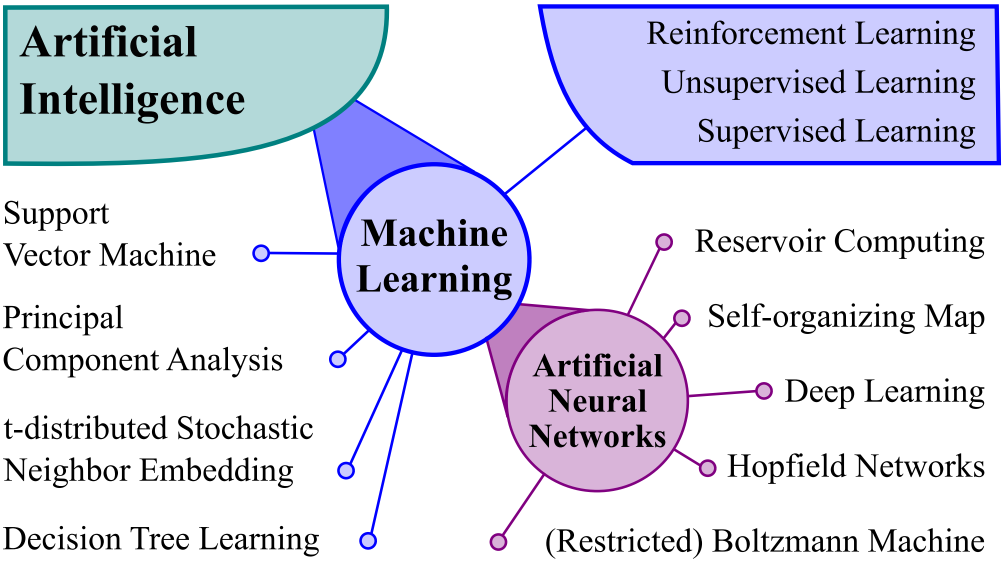

Figure 1.4: A schematic illustration of the relations between the concepts of machine learning.

1.2.1 Data-driven optimization of multivariate functional

Machine learning and physics gather data from the complex system and design a model that analyzes and predicts system behaviors. The physics theory is established on physical operators and quantities, but many user-defined variables, so-called parameters, implement machine learning models. Also, without prior knowledge of the system, a machine learning model goes through a so-called learning (training) process using given data. Therefore, the primary process in machine learning is optimizing the multivariate model based on data. Here, we review the meaning and mathematical expression of data, model, optimization, and other machine learning requisites.

Data

The objectMachine learning is a theoretical framework identifying data, sometimes called patterns. In the dictionary, data is information that is in the form of facts or statistics. More strictly speaking, in machine learning, *data* is a set of the formalized numbers. The formation of data is analog or digital depending on the storage medium, and the data can be expressed as strings, numerals, or bits. Nevertheless, without losing generality, we represent data by multi-dimensional array or mathematically tensor.

$$\boldsymbol{x} \equiv \{x_{i_1 i_2 \dots i_R}\} \in \mathbb{R}^n, \quad i_k \in \mathbb{Z}_{n_k}, \quad n = \prod_{k=1}^{R} n_k, \tag{1.9}$$

where R is the rank of the tensor, n is the total degree of freedom of the tensor, and n is also the dimension of data space.

For an example of data preprocessing, let us recall the Ising model that deals with the two-state spin chain. We can denote a state of spin as an integer; for example, we denote an up spin as 1 and a down spin as 0. Then we represent the spin chain on the one-dimensional lattice as a one-dimensional array in Figure 1.5(a) and the spin chain on the two-dimensional square lattice as a two-dimensional array in Figure 1.5(b). Likewise, we can also represent spin configuration on the Euclidean lattice as a multi-dimensional array with the system's dimension. We can also denote the dynamical change of spin configuration over time by adding a time index to the array. Figure 1.5(c) shows that a one-dimensional spin chain dynamics is denoted as a two-dimensional array.

This process of expressing information as the formatted number is referred to as data preprocessing, data representing, or data encoding. There are different ways for the same information. If the spin state is denoted as a vector, we denote the one-dimensional spin chain as a two-dimensional array in Figure 1.5(d), compared in Figure 1.5(a). Therefore, what is essential in the data preprocessing is consistent representation.

Because there is a one-to-one correspondence between an R rank tensor with total dimension n and an n-dimensional vector, many textbooks for artificial neural net-

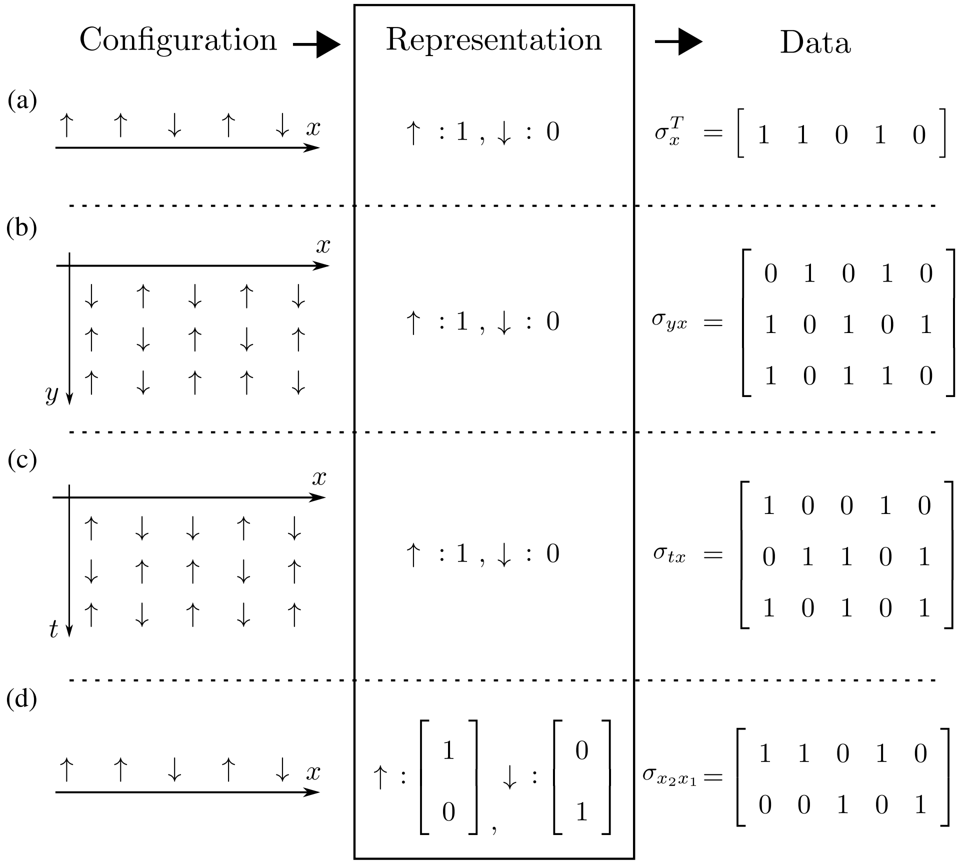
Configuration
Representation
Data
(a)
↑ ↑ ↓ ↑ ↓ x
↑ : 1 , ↓ : 0
$\sigma_x^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \end{bmatrix}$
(b)
x
y
↓ ↑ ↓ ↑ ↓
↑ ↓ ↑ ↓ ↑
↑ ↓ ↑ ↑ ↓
↑ : 1 , ↓ : 0
$\sigma_{yx} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$
(c)
x
t
↑ ↓ ↓ ↑ ↓
↓ ↑ ↑ ↓ ↑
↑ ↓ ↑ ↓ ↑
↑ : 1 , ↓ : 0
$\sigma_{tx} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$
(d)
↑ ↑ ↓ ↑ ↓ x
$\uparrow : \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \downarrow : \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
$\sigma_{x_2x_1} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

Figure 1.5: Data preprocessing for spin configuration obtained from the Ising model. If the spin is denoted as an integer, we represent the spin chain (a) on the one-dimensional lattice as a one-dimensional array (vector) and (b) on the two-dimensional square lattice as a two-dimensional array (matrix). (c) Dynamics of spin configuration on the one-dimensional lattice is denoted as a two-dimensional array. (d) If the spin state is expressed as a vector, we represent the one-dimensional spin chain as a two-dimensional array.

works often simplify data as a vector.

$$\phi : x_{i_1 i_2 \ldots i_R} \mapsto x_j, \quad j \in \mathbb{Z}_n. \tag{1.10}$$

Map ϕ from a tensor to a vector by compressing multi-index to one index is also a sort of data preprocessing, called *flattening*. Flattened data is widely used to introduce artificial neural networks because it helps explain algorithms for the fully-connected neural networks without losing its generality. However, the flattening ignores the difference between indices, and it may cause fatal problems depending on data and methods of machine learning.

For example, we introduce two indices for each spatial axis in two-dimensional spin configurations, and we employ one index to represent the spatial axis and another index to represent the temporal axis in the time series of a one-dimensional spin chain. The index implies temporal-spatial information of the system, the source of data. Therefore, we should use the flattening carefully when temporal-spatial information becomes a significant factor in identifying the data, and the index denotes the spatial axis or temporal axis in the data. We consider the index representing the temporal-spatial information of data because the geometry characterizes the nature of the phase transition. It is important to employ machine learning appropriate for the data structure when introducing machine learning for studying phase transition.

Generally, representing data as a tensor is advantageous to preserve the data's dimension, but it also helps to implement numerical algorithms of machine learning through programming languages. Furthermore, some machine learning methods are designed to reflect the form of the data. For example, the convolutional neural networks [44] learn that the data is in the euclidean space, the graph neural networks learn that the data is a network [45], and the recurrent neural networks learn that the data is sequential such as time series. [46]

A dataset $\boldsymbol{X}$ is a set of data from the same source;

$$\boldsymbol{X} = \{\boldsymbol{x}_0, \cdots, \boldsymbol{x}_{N-1}\}, \quad X_i = \boldsymbol{x}_i, \quad i \in \mathbb{Z}_N \tag{1.11}$$

where N is the number of data and X_i denotes $(i+1)$th data in dataset $\boldsymbol{X}$. While each data may have different dimensions or ranks, using padding that fits the data uniformly based on the data with the largest dimension or rank, we obtain the dataset consisting of the same data form. Sometimes, we represent data, including the dataset to which it belongs:

$$\boldsymbol{X} = \{X_{ji_1i_2\ldots i_p}\} = \{(\boldsymbol{x}_j)_{i_1i_2\ldots i_p}\} \in \mathbb{R}^{Nn} \tag{1.12}$$

where the data $\boldsymbol{x}_j$ follows Eq. (1.9).

According to purpose, we divide the dataset into a training dataset and a testing dataset in a broad sense. Machine learning designs and verifies a model based on the training dataset. In a narrow sense, a training dataset is a dataset for optimizing the model, and a dataset for self-verifying the trained model is a validation dataset. Some machine learning methods do not validate the model, and then the training dataset is not separated into a training dataset and validation dataset. The testing dataset is a dataset for evaluating the completed model, and the model is not modified when dealing with the testing dataset. We obtain the models' results for the testing dataset, evaluate the models' performance, and compare their performance. See Appendix C.1 for know-how to divide and manage datasets for artificial neural networks. This study refers to the dataset as a training dataset unless otherwise noted.

Sometimes a dataset is split into multiple parts for practical computational reasons. For example, in an artificial neural network, the parted dataset called batch is the subset of the entire dataset. The reasons and advantages of introducing the batch are addressed in Appendix C.2.

Model: Multivariate functional

Identifying patterns in machine learning is mapping the data into the other variables that human understands or needs. This mapping is called a *model* in machine learning, and mathematically the model can be represented in the form of functional.

$$\mathcal{F} : \boldsymbol{x} \mapsto \boldsymbol{y} \in \mathbb{R}^m \tag{1.13}$$

Here $\boldsymbol{y}$ is the model's output and a tensor with a total m dimension. The model $\mathcal{F}$ consists of several functions embodied through model parameters in machine learning.

For example, let us look at a linear model as a toy model. To help to understand, we assume that the data and model's output is an n-dimensional and m-dimensional vector, respectively. Here is a mathematical representation of the linear model $\mathcal{F}_0$ as follows

$$\mathcal{F}_0[\boldsymbol{W}](\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}, \quad \boldsymbol{W} \in \mathbb{R}^{mn}, \tag{1.14}$$

where the variable $W \in \boldsymbol{W}$ becomes a model parameter, and the model has mn variables.

If the model is a function when the model parameters are fixed, such as the linear model $\mathcal{F}_0$, the model is called a deterministic model. We generally express the deterministic model as follows

$$\boldsymbol{y} = F[\boldsymbol{\theta}](\boldsymbol{x}), \quad \boldsymbol{\theta} \in \mathbb{R}^k, \tag{1.15}$$

where we denote the model parameter as $\theta \in \boldsymbol{\theta}$. Machine learning usually employs numerous parameters to construct the model compareing the data space. In other words, in many cases, the dimension of the model parameter space k is larger than the the dimension of the data space or the model's output space; $k \gg n, m$. It is different from the classical method of dimension regression, such as the linear regression and the least square fitting, which use the restricted number of parameters for designing the model. In this context, the machine learning model is a *multivariate functional*.

Because the machine learning model has many model parameters, it is hard to find each model parameter's mathematical or physical meaning. Instead, the overall layout of the model parameters becomes essential to understand the model. For example, in the linear model, Uncovering what kind of characteristics the weight matrix $\boldsymbol{W}$ has gives an excellent intuition to understand and design the linear model rather than what each value W of the weight matrix means.

The deterministic model is mainly used for data clustering, data regression, and data compression. As the above works called different words, but we concise them as dimension reduction. In dimension reduction, the model maps the data into a smaller variable space to extract the main feature from the data. A *classifier* is a model with a smaller dimension of the output m than the dimension of data space n: $m \leq n$. Since the deterministic model always the same output for a given input, the deterministic model is suitable for a classifier.

Another machine learning model is a statistical model. The statistical model is a model that uses random variables from the well-known distribution such as uniform distribution, Gaussian distribution, and Poisson distribution. We understand the statistical model as a combined model with the deterministic model and probability density functions. Generally, we express the statistical model as follows

$$\boldsymbol{y} = F[\boldsymbol{\theta}](\boldsymbol{\xi}, \boldsymbol{x}), \quad \boldsymbol{\xi} \in \mathbb{R}^l \tag{1.16}$$

where $\boldsymbol{\xi}$ denotes a tensor consisting of l random variables. Each random variable follows a probability density function independently.

$$\xi_i \sim P_i(\xi_i; \boldsymbol{\alpha}_i), \quad \boldsymbol{\alpha}_i \in \mathbb{R}^{p_i}, \quad i \in \mathbb{Z}_l, \tag{1.17}$$

where $\boldsymbol{\alpha_i}$ denotes the parameters of the probability density function. The random variable vector $\boldsymbol{\xi}$ follows the multi-dimensional probability density function as follows

$$\boldsymbol{\xi} \sim P(\boldsymbol{\xi};\boldsymbol{\alpha}), \quad P(\boldsymbol{\xi};\boldsymbol{\alpha}) = \prod_{i=0}^{l-1} P_i(\xi_i;\boldsymbol{\alpha}_i). \tag{1.18}$$

Here, the parameters $\boldsymbol{\alpha}$ is an tensor with total p dimension.

$$\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_0, \cdots, \boldsymbol{\alpha}_{l-1}\} \in \mathbb{R}^p, \quad p = \prod_{i=0}^{l-1} p_i \tag{1.19}$$

The probability density function $P(\boldsymbol{\xi};\boldsymbol{\alpha})$ is the main factor of the statistical model, and the parameters $\boldsymbol{\alpha}$ of the probability density function are included in the model parameter.

A *generator* is a machine learning model that reproduces outputs similar to given data by imitating the system, which provides the original data. It is the essential point of the generator to produce similar data, not the same data. Because the statistical model's outputs continuously vary depending on the random variables, the statistical model is appropriate for a generator. The deterministic part of the statistical model extracts the main feature of the data and reconstructs the data, and the randomness from the probability distribution of the statistical model adjusts to the similarity of outputs.

So far, we distinguish the machine learning model into the deterministic model and the statistical model based on whether to use random variables and to include probability distribution as a component of the model. According to the model's purpose, we distinguish the machine learning model into the classifier recognizing the data's feature and the generator reproducing similar data. Furthermore, the model's purpose and structure are closely related to each other. Usually, the deterministic model serves as a classifier, and the statistical model serves as a generator. Of course, it is possible to make a stochastic classifier or deterministic generator for a particular purpose.

Optimization: Supervised and Unsupervised learning

We introduced and explained the two requisites of machine learning: the data and the model. For a given data, the model's output varies depending on the model parameters' value, even if the model's structure is not changed. Therefore, another requisite of machine learning is finding a suitable value for the model parameter such that the model outputs what we want. In machine learning, tuning the model parameters to obtain the best model for the purpose is called *optimization*, learning, or training the model.

We distinguish the method of optimizing the model into two types depending on whether the supervisor supports to train the model: supervised learning and unsupervised learning. In supervised learning, we make the model to imitate the supervisor or work better than the supervisor using the supervisor's outputs for the data. On the other hand, in unsupervised learning, we optimize the model using only given data without the supervisor's help.

Supervised learning As briefly mentioned above, supervised learning is a method of optimizing the model with a supervisor's help. In machine learning language, the supervisor $\hat{\mathcal{F}}$ is a prepared model that provides the output for the data. The supervisor's outputs for the data is a target or label $\hat{\boldsymbol{y}}$ in supervised learning.

$$\hat{\mathcal{F}} : \boldsymbol{x} \mapsto \hat{\boldsymbol{y}} \in \mathbb{R}^m, \tag{1.20}$$

where the label $\hat{\boldsymbol{y}}$ is also a tensor with the same rank and dimension as the outputs of the model we want to train.

Because the supervisor is an outsider in supervised learning, who plays a supervisor depends on the problem. For example, if a human being becomes a supervisor, then the human's recognization of the data is a target. We also employ theoretical frameworks such as physics and statistics, and the other pre-trained machine learning model as supervisors. Even if the environment that provides the data becomes a supervisor, we use another signal from the system correlated to a given data as a label.

The ultimate goal of supervised learning is to make a model produce an output similar to the label, which is the supervisor's output for the data. Minimizing a loss function is the most widely used method in supervised learning. The loss function E takes two arguments: the label $\hat{\boldsymbol{y}}$ and the output $\boldsymbol{y}$, and is called an error function, cost function, and energy function, depending on the textbook.

$$E = E(\boldsymbol{y}, \hat{\boldsymbol{y}}). \tag{1.21}$$

where we assume a label and an output as an m-dimensional vector.

Usually, we need the loss function to satisfy two characteristics: local convexity and non-negativity. First, let us define the Hessian matrix of the loss function for the output at the label.

$$\boldsymbol{H} = \mathbb{H}_{\boldsymbol{y}} E\big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}}, \quad \boldsymbol{H}_{ij} = \left.\frac{\partial^2 E}{\partial y_i \partial y_j}\right|_{\boldsymbol{y}=\hat{\boldsymbol{y}}} \tag{1.22}$$

where $\mathbb{H}_{\boldsymbol{y}}$ is the Hessian operator for the domain $\boldsymbol{y}$. If the Hessian matrix $\boldsymbol{H}$ is positive semi-definite, the loss function is a convex function near the fixed label.

$$\boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} \geq 0, \quad \forall \boldsymbol{x} \in \mathbb{R}^m \tag{1.23}$$

Second, the loss function is a non-negative function and has zero value if the label and output are the same.

$$E \geq 0, \quad E\big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}} = 0. \tag{1.24}$$

For example, the mean square loss function E_{MSE} is widely used to satisfy the above

Loss function	Mathematical form	Constraints
Mean absolute	$\frac{1}{m}\lvert\boldsymbol{y}-\hat{\boldsymbol{y}}\rvert = \frac{1}{m}\sum_{i=0}^{m-1}\lvert\hat{y}_i - y_i\rvert$	
Mean square	$\frac{1}{m}\lVert\boldsymbol{y}-\hat{\boldsymbol{y}}\rVert^2 = \frac{1}{m}\sum_{i=0}^{m-1}(\hat{y}_i - y_i)^2$	
Binary cross entropy	$-\left[\hat{y}\ln y + (1-\hat{y})\ln(1-y)\right]$	$\hat{y} \in \{0,1\}$ $y \in (0,1)$
Cross entropy	$-\sum_{i=0}^{m-1}\left[\hat{y}_i \ln y_i\right]$	$y_i, \hat{y}_i \in (0,1)$ $\lVert\boldsymbol{y}\rVert = \lVert\hat{\boldsymbol{y}}\rVert = 1$
Kullback–Leibler divergence	$\sum_{i=0}^{m-1}\left[\hat{y}_i \ln\left(\hat{y}/y_i\right)\right]$	
Huber	$\frac{1}{m}\sum_{i=0}^{m-1} H_d(\hat{y}_i - y_i)$	
Hinge	$\frac{1}{m}\sum_{i=0}^{m-1}\max(1-\hat{y}_i y_i, 0)$	$\hat{y}_i \in \{-1,1\}$

Table 1.3: List of loss functions for optimizing the model in supervised learning. For two vector $\boldsymbol{a}$ and $\boldsymbol{b}$, we denote a Taxicab distance and a Euclidean distance as $\lvert\boldsymbol{a}-\boldsymbol{b}\rvert$ with 1-norm and $\lVert\boldsymbol{a}-\boldsymbol{b}\rVert$ with 2-norm, respectively. If $\lvert x\rvert \leq d$ the Huber loss function is $H_d(x) = \frac{1}{2}x^2$, otherwise $H_d(x) = d\left(\lvert x\rvert - \frac{1}{2}d\right)$ where d is [47]. The cross entropy loss function borrows the cross entropy's mathematical form of two probability distributions in statistics. If the label $\hat{\boldsymbol{y}}$ and output $\boldsymbol{y}$ are a probability mass function, the cross entropy loss function is the cross entropy between the label and output in a statistical sense. However, we should distinguish the cross entropy loss function and the cross entropy because the label and the output do not always mean the probability mass function even though the label and the output satisfy the normalization of the probabilities mass function. Likewise, the binary cross entropy loss function and the Kullback–Leibler divergence loss function copies the mathematical form of the binary cross entropy and the mathematical form of the Kullback–Leibler divergence, respectively.

characteristics.

$$E_{\mathrm{MSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{m} \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2, \tag{1.25}$$

where we denote the Euclidean distance between the label and outputs as $\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|$ with 2-norm. Table 1.3 shows examples of the loss function. In Table 1.3, loss functions such as the cross entropy loss function and the Kullback–Leibler divergence loss function borrows the mathematical functional form used in physics and statistics. We are careful to take the loss function in physical or statistical meaning.

Once the loss function is explicitly determined, we calculate the loss function's value for a given data. If the output is more similar to the label, the loss function' value is smaller. Therefore, we quantify a gap between the model and the supervisor, evaluate the model's performance.

Let us consider that the model is deterministic. An average loss $\bar{E}$ is the expected value of the loss function for a given dataset $\boldsymbol{X}$ as follows

$$\bar{E} = \mathbb{E}_{\boldsymbol{x}}\left[E(\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x}), \hat{\boldsymbol{y}}(\boldsymbol{x}))\right] = \bar{E}(\boldsymbol{\theta}), \quad \boldsymbol{x} \in \boldsymbol{X}, \tag{1.26}$$

where $\mathbb{E}_{\boldsymbol{x}}\left[\cdot\right]$ denotes the mathematical expectation for all data $\boldsymbol{x}$ in dataset. As the dataset and labels are fixed, only the model output depends on the model parameters, but the average loss also depends on the model parameters. Therefore, supervised learning the model is finding the optimal model parameters $\boldsymbol{\theta}^*$ minimizing the average loss:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \bar{E}(\boldsymbol{\theta}). \tag{1.27}$$

If we obtain the optimal model parameters sequentially, we can build the optimal model $\mathcal{F}^*$ such as

$$\mathcal{F}^*(\boldsymbol{x}) = \mathcal{F}[\boldsymbol{\theta}^*](\boldsymbol{x}). \tag{1.28}$$

To specify the above process in supervised learning, we need to determine a model as an explicit form. Moreover, according to the mathematical form of the model, it is possible to solve the equation analytically or not. In many cases, we only use numerical way for solving the minimum value problem. Thus, there are many supervised learning implementations, depending on the model's structure and methodology for solving the minimum value problem. In appendix A, the optimizing process of the feed-forward neural networks is a typical example of supervised learning for a deterministic model.

The statistical model can also be trained through supervised learning. However, the statistical model takes random variables as arguments, and the outputs are distributed for given data and the label. To define the average loss for a statistical model, we need to consider how the model follows the supervisor from a statistical perspective.

We first suppose that the average of the statistical model's outputs will be the same as the supervisor's label for the given, then we introduce the average loss as follows

$$\bar{E} = \mathbb{E}_{\boldsymbol{x}}\left[E\left(\mathbb{E}_{\boldsymbol{\xi}}\left[\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{\xi}, \boldsymbol{x})\right], \hat{\boldsymbol{y}}(\boldsymbol{x})\right)\right], \quad \boldsymbol{x} \in \boldsymbol{X}, \quad \boldsymbol{\xi} \sim P(\boldsymbol{\xi}; \boldsymbol{\alpha}). \tag{1.29}$$

Theoretically, we can obtain the expectation values of the output for the random variables because we already know the probability distribution we set.

$$\mathbb{E}_{\boldsymbol{\xi}}\left[\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{\xi}, \boldsymbol{x})\right] = \int \left[\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{\xi}, \boldsymbol{x}) P(\boldsymbol{\xi}; \boldsymbol{\alpha})\right] d\boldsymbol{\xi}. \tag{1.30}$$

However, in most cases, it is hard to calculate the expectation values such as Eq. (1.30) analytically. Instead, we employ the numerical methods for statistical inference, such as simulated annealing, Monte Carlo method, and Gibbs sampling.

Another way to define the average loss for the statistical model is to sum the value of the loss function for each output.

$$\begin{aligned}\bar{E} &= \mathbb{E}_{\boldsymbol{\xi},\boldsymbol{x}}\left[E(\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{\xi},\boldsymbol{x}),\hat{\boldsymbol{y}}(\boldsymbol{x}))\right] \\ &= \mathbb{E}_{\boldsymbol{\xi}}\left[\mathbb{E}_{\boldsymbol{x}}\left[E(\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{\xi},\boldsymbol{x}),\hat{\boldsymbol{y}}(\boldsymbol{x}))\right]\right] \\ &= \mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{\xi}}\left[E(\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{\xi},\boldsymbol{x}),\hat{\boldsymbol{y}}(\boldsymbol{x}))\right]\right].\end{aligned} \tag{1.31}$$

The above two types of average losses are typical examples, but we can invent various forms, such as mixing the two average losses. Nevertheless, once an average loss is defined in a statistical model, as in a deterministic model, the heart of supervised learning is to find the optimal model parameters minimizing the average loss, such as Eq. (1.27).

For some statistical models, the probability distribution parameters $\boldsymbol{\alpha}$ are not constant and are included as model parameters, so we also need to find the optimal values for the probability distribution parameters minimizing the average loss.

$$(\boldsymbol{\theta}^*,\boldsymbol{\alpha}^*) = \underset{\boldsymbol{\theta},\boldsymbol{\alpha}}{\operatorname{argmin}}\,\bar{E}(\boldsymbol{\theta},\boldsymbol{\alpha}). \tag{1.32}$$

Then, we can build an optimal statistical model as follows in

$$\mathcal{F}^*(\boldsymbol{x}) = \mathcal{F}[\boldsymbol{\theta}^*](\boldsymbol{\xi},\boldsymbol{x}), \quad \boldsymbol{\xi} \sim P(\boldsymbol{\xi},\boldsymbol{\alpha}^*). \tag{1.33}$$

For example, the variational auto-encoder [48] is a statistical model with probability distribution called a sampler and is optimized by supervised learning.

Unsupervised learning In supervised learning, the supervisor only gives the guidelines on the forms and the meaning of the outputs but provides the label as the correct answer for each training data. To opposite supervised learning, another method of machine learning is unsupervised learning in which the model is trained with the only dataset without an external model such as the supervisor. Thus, in unsupervised learning, we have no choice but to focus on the dataset's statistics, clustering and similarity between data, and the physical meaning of data. Thus, unsupervised learning aims to build a model that can store a given dataset or regenerate outputs similar to the given data.

In a statistical sense, if we know the probability of data on the data space, then we restore the given dataset. Following this view, unsupervised learning mainly employs the methods of statistical inference. Here, we introduce the example of unsupervised learning based on the Bayesian inference.

Let us consider a dataset consisting of N data, as shown in Eq. (1.11). The dataset is evidence of Bayesian statistics for establishing a hypothesis, and the data (point) is sampled from an unknown population. We propose the Bayesian hypothesis that the machine learning model works as a likelihood (function) of Bayesian, then we develop the Bayesian inference as the method of unsupervised learning.

Here, we have to determine the type of our model is whether a deterministic model or a statistical model. The reason is that the likelihood's parameter is a random variable following the probability distribution called a prior distribution in Bayesian statistics. We intuitively correspond the statistical model's probability distribution to Bayesian's prior distribution, but the deterministic model has no random variables to match the prior distribution's parameters.

First, let us look at unsupervised learning for a deterministic model. The model should be a real-valued function and a non-negative function to use the model as a likelihood function.

$$\mathcal{F} : \boldsymbol{x} \mapsto y \in \mathbb{R}, \quad \mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x}) \geq 0, \quad \forall \boldsymbol{x}. \tag{1.34}$$

The model also satisfies the normalization condition as follows

$$\boldsymbol{Z} = \int [\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x})] \, d\boldsymbol{x} < \infty, \tag{1.35}$$

where $\boldsymbol{Z}$ denotes a normalization factor. Note that the normalization factor depends on the model parameters; $\boldsymbol{Z} = \boldsymbol{Z}(\boldsymbol{\theta})$.

Back again, to establish the hypothesis, we suggest that the probability of choosing the data point from the population is proportional to the machine learning model's output for the data. In the language of Bayesian statistics, the sampling distribution for a single data denotes with the model's output and the normalization factor as follows

$$\mathbb{P}(\boldsymbol{x}|\boldsymbol{\theta}) = \frac{1}{\boldsymbol{Z}}\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x}), \quad \boldsymbol{x} \in \boldsymbol{X} \tag{1.36}$$

Moreover, we define the likelihood of a given dataset as

$$\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}\right) \equiv \prod_{\boldsymbol{x}\in\boldsymbol{X}} \mathbb{P}(\boldsymbol{x}|\boldsymbol{\theta}) = \frac{1}{\boldsymbol{Z}^N}\prod_{i=0}^{N-1}\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x}_i). \tag{1.37}$$

Here, we attempt a little logical leap. In standard Bayesian statistics, the likelihood parameter is a random variable following a prior distribution, but in machine learning, especially for the deterministic model, the model parameter is a variable tuned by the optimization but not a random variable. Nevertheless, as above, we consider the model parameter as the likelihood parameter. So we *assume* that the model parameter follows a probability distribution, which refers to a prior distribution.

$$\boldsymbol{\theta} \sim \mathbb{P}(\boldsymbol{\theta}) \tag{1.38}$$

We can use a uniform distribution, Gaussian distribution, and even Dirac delta function as an example of the prior distribution. As will be mentioned later, according to which probability distribution is used as the prior distribution, the analytical theory for optimization for the model differs. The prior distribution is closely related to the probability distribution used in the initialization of the model parameters, and sometimes two distributions have the same meaning in numerical approaches.

According to the Bayesian rule, which is the core of Bayesian inference, we obtain a posterior distribution, which gives the probability of the likelihood parameters for a given dataset as follows

$$\mathbb{P}\left(\boldsymbol{\theta}|\boldsymbol{X}\right) = \frac{\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}\right)\mathbb{P}\left(\boldsymbol{\theta}\right)}{\mathbb{P}\left(\boldsymbol{X}\right)}. \tag{1.39}$$

Here, $\mathbb{P}\left(\boldsymbol{X}\right)$ denotes a marginal likelihood such as

$$\mathbb{P}\left(\boldsymbol{X}\right) = \int \left[\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}\right)\mathbb{P}\left(\boldsymbol{\theta}\right)\right] d\boldsymbol{\theta}. \tag{1.40}$$

In Bayesian inference, we estimate the relevant likelihood parameters as the mode of the posterior distribution. This estimation is called the maximum posterior estimation. Following the mind of maximum a posterior estimation, unsupervised learning aims to find the model parameters that maximize the posterior distribution.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{P}\left(\boldsymbol{\theta}|\boldsymbol{X}\right) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}\right)\mathbb{P}\left(\boldsymbol{\theta}\right)\right]. \tag{1.41}$$

If a prior distribution is a uniform distribution, then the maximum posterior estimation coincides with the maximum likelihood estimation.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}\right), \quad \mathbb{P}\left(\boldsymbol{\theta}\right) = 1. \tag{1.42}$$

Moreover, we introduce the log-likelihood to solve the maximum likelihood estimation as follows

$$\mathbb{L}\left(\boldsymbol{X}|\boldsymbol{\theta}\right) \equiv \frac{1}{N}\sum_{\boldsymbol{x}\in\boldsymbol{X}} \log\left[\mathbb{P}(\boldsymbol{x}|\boldsymbol{\theta})\right] = \mathbb{E}_{\boldsymbol{x}}\left[\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x})\right] - \boldsymbol{Z}. \tag{1.43}$$

Theoretically, the points at which the likelihood is maximum are the same as the points at which the log-likelihood is maximum. Then, we again obtain the optimal parameter such that

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\mathbb{E}_{\boldsymbol{x}} \left[\mathcal{F}[\boldsymbol{\theta}](\boldsymbol{x}) \right] - \boldsymbol{Z} \right]. \tag{1.44}$$

However, when we take the uniform prior distribution, the prior distribution may do not satisfy the normalization condition of the probability distribution required by standard Bayesian statistics because there is no regularization for the model parameters in machine learning.

Once we obtain the optimal model parameter, regardless of using the maximum posterior estimation or the maximum likelihood estimation, we also find the optimal deterministic model $\mathcal{F}^*$ such as Eq. (1.28).

By applying the optimized model to sampling distribution, we obtain a probability distribution to get new data $\boldsymbol{x}'$

$$\mathbb{P}(\boldsymbol{x}'|\boldsymbol{\theta}^*) = \frac{1}{\boldsymbol{Z}(\boldsymbol{\theta}^*)} \mathcal{F}[\boldsymbol{\theta}^*](\boldsymbol{x}') \tag{1.45}$$

This distribution allows us to reconstruct a new dataset similar to the given dataset.

For example, the Hopfield model [49] is deterministic and is optimized by unsupervised learning. The Hopfield model is a prototype model for storing data in artificial neural networks.

Second, we develop unsupervised learning based on Bayesian inference for a statistical model. The statistical model already has a probability distribution, and we use the probability distribution as a prior distribution. Unlike the deterministic model, we take the random variables $\boldsymbol{\xi}$ following the statistical model's probability distribution as the Bayesian parameters, not the model parameters $\boldsymbol{\theta}$.

$$\boldsymbol{\xi} \sim P(\boldsymbol{\xi}; \boldsymbol{\alpha}) = \mathbb{P}(\boldsymbol{\xi}|\boldsymbol{\alpha}). \tag{1.46}$$

Then, the parameters $\boldsymbol{\alpha}$ of the probability distribution become the hyperparameter of the prior distribution. This approach is reasonable to follow the Bayesian statistics' theoretical framework because the Bayesian parameter is a random variable following a prior distribution.

Of course, likewise the deterministic model, the statistical model satisfies conditions of the probability distribution. For observed random variables, the model is a real-valued function and non-negative function.

$$\mathcal{F} : \boldsymbol{x} \mapsto y \in \mathbb{R}, \quad \mathcal{F}[\boldsymbol{\theta}, \boldsymbol{\xi}](\boldsymbol{x}) \geq 0, \quad \forall \boldsymbol{x} \tag{1.47}$$

Furthermore, the model follows the normalization condition.

$$\boldsymbol{Z} = \int \left[\mathcal{F}[\boldsymbol{\theta}, \boldsymbol{\xi}](\boldsymbol{x})\right] d\boldsymbol{x} < \infty \tag{1.48}$$

Next, we suggest that the probability of sampling the data point from the population for the given model parameters and the given random variables is proportional to the model's outputs. In other words, the sampling distribution of single data for given Bayesian parameters is

$$\mathbb{P}(\boldsymbol{x}|\boldsymbol{\xi}, \boldsymbol{\theta}) = \frac{1}{\boldsymbol{Z}} \mathcal{F}[\boldsymbol{\theta}, \boldsymbol{\xi}](\boldsymbol{x}), \quad \boldsymbol{x} \in \boldsymbol{X} \tag{1.49}$$

Moreover, we define the likelihood of a given dataset as

$$\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\xi}, \boldsymbol{\theta}\right) \equiv \prod_{\boldsymbol{x} \in \boldsymbol{X}} \mathbb{P}(\boldsymbol{x}|\boldsymbol{\xi}, \boldsymbol{\theta}) = \frac{1}{\boldsymbol{Z}^N} \prod_{i=0}^{N-1} \mathcal{F}[\boldsymbol{\theta}, \boldsymbol{\xi}](\boldsymbol{x}_i). \tag{1.50}$$

Subsequently, we obtain the marginal likelihood as follows

$$\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}, \boldsymbol{\alpha}\right) = \int \left[\mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\xi}, \boldsymbol{\theta}\right) \mathbb{P}\left(\boldsymbol{\xi}|\boldsymbol{\alpha}\right)\right] d\boldsymbol{\xi}. \tag{1.51}$$

The marginal likelihood depends on only the Bayesian hyperparameters, which are model parameters $\boldsymbol{\theta}$ and probability distribution parameters $\boldsymbol{\alpha}$.

For two sets of Bayesian hyperparameters $M_1 = \{\boldsymbol{\theta}_1, \boldsymbol{\alpha}_1\}$ and $M_2 = \{\boldsymbol{\theta}_2, \boldsymbol{\alpha}_2\}$, the Bayes factor K_{12} is defined as the ratio of two marginal likelihoods of each hyperparameter.

$$K_{12} = \frac{\mathbb{P}\left(\boldsymbol{X}|M_1\right)}{\mathbb{P}\left(\boldsymbol{X}|M_2\right)} \tag{1.52}$$

The Bayes factor indicates how much the given dataset supports the Bayesian hyperparameters. If the Bayer factor greater than 1 ($K_{12} > 1$), M_1 is more robust supported by the dataset than M_2. It means that the relevant Bayesian hyperparameters are the arguments in which the marginal likelihood is maximum. So, the goal of unsupervised learning for the statistical model is to find the optimal model parameters and probability distribution parameters.

$$\left(\boldsymbol{\theta}^*, \boldsymbol{\alpha}^*\right) = \underset{\boldsymbol{\theta}, \boldsymbol{\alpha}}{\operatorname{argmax}} \, \mathbb{P}\left(\boldsymbol{X}|\boldsymbol{\theta}, \boldsymbol{\alpha}\right). \tag{1.53}$$

Finally, using the above optimal hyperparameters, we can build an optimal statistical model $\mathcal{F}^*$ such as Eq. (1.33).

For example, Boltzmann machines [7], including the restricted Boltzmann machine [8], are the statistical model and are optimized by unsupervised learning.

There are two ways to optimize the model: supervised learning and unsupervised learning. Depending on whether there is a supervisor, we distinguish supervised learning and unsupervised learning, and then we obtain two different optimized model. As a result of supervised learning, we obtain the optimal model to imitate the supervisor. On the other hand, due to unsupervised learning, we obtain the optimal model to be the probability distribution of data from the population. Furthermore, whether the model is deterministic or statistical, each learning methods are different in detail.

Another machine learning algorithm is reinforcement learning [50–52]. Reinforcement learning considers more particular situations than supervised and unsupervised learning. In reinforcement learning, the model called an agent determines the action as output for the given data called a state. Here, the probability distribution of the action for a given state is called a policy. An environment renews the state from the agent's action and returns a new state to the agent. The environment feedback on the action as a reward shows how the action is suitable for the previous state. Therefore, reinforcement learning aims to find the best policy to maximize the rewards for any state. Reinforcement learning may seem to supervised learning in that the environment, similar to the supervisor, outside the agent gives feedback and the reward, similar to the average loss, quantifies the agent's fitness. On the other hand, optimizing the policy, which is the probability distribution of the action for a given state, seems similar to optimizing the likelihood function in unsupervised learning. It is not easy to compare reinforcement learning to supervised and unsupervised learning in the same criteria. Because supervised and unsupervised learning focuses on optimizing a fixed model with the given data, the beginning of problem-solving is to determine a proper learning method for the above problem situations in reinforcement learning. Therefore, according to internal implementation, reinforcement learning employs supervised learning or unsupervised learning.

So far, we mathematically described three fundamental factors of machine learning: data, models, and optimization. To sum up the discussions, the heart of machine learning is the data-driven optimization of multivariate functional.

1.2.2 Artificial neural networks

The root of machine learning is to gather the amount of data, and then the stem of machine learning is the optimization algorithm. Nevertheless, what a tree needs to bear fruit are branches and leaves, which let us know the tree's identity. So, we need a model architecture, like branches and leaves, to realize and embody the machine learning approach.

The artificial neural network is a generic term for a machine learning model composed of artificial neurons. It motivated the artificial neural network approach that a biological neural network, such as the brain, works as a mathematical function in a cognition process. We benchmark biological neural networks to construct artificial neural networks and describe artificial neural networks mathematically.

Neuron The nerve system consists of nerve cells called biological neurons. We consider an artificial neuron or a perceptron [53] as a basic unit of an artificial neural network. From now on, we call an artificial neuron just a neuron unless otherwise mentioned.

A biological neuron has two states; an excited state and a resting state. The biological resting neuron becomes the excited state when stimulated by an electrical impulse that exceeds an absolute value called the threshold voltage from the neuron outside. To emulate the biological neuron's excitation process, we introduce a stimulus s and a neuron's state a. For the given stimulus s, the neuron is activated following an activation function f such as

$$f : s \mapsto a, \quad s, a \in \mathbb{R}, \tag{1.54}$$

where the stimulus s is called an inactivated neuron's state.

In the early artificial neural network, the neuron's state was expressed as 0 and 1, corresponding to the biological neuron's resting state and excited state, respectively. Here, we denote the neuron's state as a real number. For example, using the Heaviside step function $H(x;\theta)$ as an activation function, we retrieve traditional expression and can describe a biological neuron with a threshold voltage θ as follows

$$a = H(s,\theta) = \begin{cases} \frac{d}{ds}\max(s;\theta) & (s \neq \theta) \\ 1 & (s = \theta) \end{cases}. \tag{1.55}$$

In artificial neural networks, we use various activation functions, such as Table 1.4. Some artificial neural network approaches need the activation function to satisfy the

Activation function	Mathematical form
Sigmoid	$f(x) = 1/(1 + e^{-x})$
Rectified linear unit (ReLU)	$f(x) = \max(x, 0)$
Linear	$f(x) = x$
Heaviside step	$f(x) = H(x; \theta)$
Hyperbolic tangent	$f(x) = \tanh(x)$
Softmax	$\boldsymbol{x} = f(\boldsymbol{x}), \quad x_i = e^{x_i} / \sum_j e^{x_j}$

Table 1.4: Activation functions for a neuron. Here, the Heaviside step function $f(x) = H(x; \theta)$ is the same as Eq. (1.55), and θ is a constant. A softmax function is an activation function for a layer $\boldsymbol{z}$, a group of neurons, not one neuron.

function condition, such as continuity, differentiability, or injection.

We generally express the inactivated neuron's state $\boldsymbol{s}$ and the neuron's state $\boldsymbol{a}$ as an R rank tensor in the total D dimensions.

$$\boldsymbol{a} \equiv \{a_{i_1 i_2 \dots i_R}\} \in \mathbb{R}^D, \quad \boldsymbol{s} \equiv \{s_{i_1 i_2 \dots i_R}\} \in \mathbb{R}^D, \quad i_k \in \mathbb{Z}_{D_k}, \quad D = \prod_{k=1}^{R} D_k. \tag{1.56}$$

If an activation function takes an inactivated neuron's state as an argument, it means that the activation function is applied to each element of the inactivated neuron's state as follows,

$$\boldsymbol{a} = f(\boldsymbol{s}) \to a_{i_1 i_2 \dots i_R} = f(s_{i_1 i_2 \dots i_R}). \tag{1.57}$$

This notation is not mathematically strict but is useful in the theory of artificial neuron networks. If a real-valued function takes a tensor, it is applied to each element unless otherwise mentioned.

The propagation of signals in the biological neural network consists of continuous transmission of neurons' excited state. A biological neuron has dendrites accepting a stimulus from excited neurons and axon terminals, which is send out its state to other neurons. The connection between a neuron's dendrite and the other neuron's axon terminal is called a synapse. In the biological neural network, signals propagate through the synapses by a chemical reaction.

Imitating the directed transmission of a signal in the synapse, we link two neurons with a directed edge, as shown in Figure 1.6. We call a graph consisting of the edges and neurons an artificial neural network. A neuron can receive inputs from multiple neurons simultaneously, and it can also output multiple neurons simultaneously. Now let us define what input and output for neurons are.

First, we define how neurons update their state for given inputs through directed edges in the artificial neural network. Figure 1.7 shows that a neuron i receives input

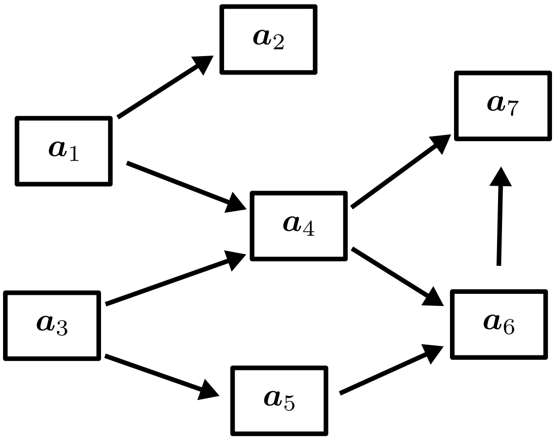

Figure 1.6: A schematic illustration of the connections between neurons. A neuron receives input and outputs to others. A squared box denotes a neuron, and an arrow denotes a direction of input.

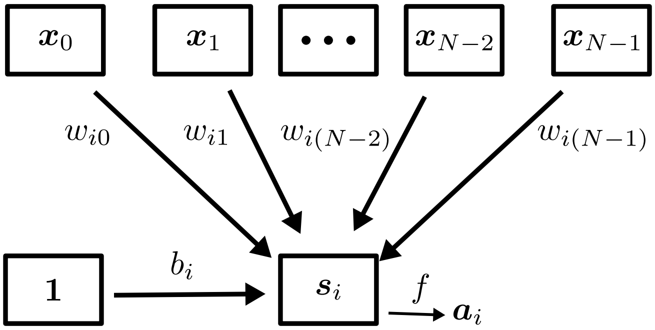

Figure 1.7: A schematic illustration of the inputs of a neuron. A neuron i receives input from N neurons and the bias neuron in the identical state and renews its states. A squared box denotes a neuron, and an arrow represents a weight for input and input direction.

from N neurons and the bias neuron in the identical state $\mathbf{1}$ and renews its states. $\boldsymbol{x}_j$ denotes the input given by neuron j, and w_{ij} denotes its weight.

We need each input to have the same tensor form of the neuron i's stimulus $\boldsymbol{s}_i$ satisfying Eq. (1.56), and the weight for each input is a real number.

$$\boldsymbol{x}_j \equiv \{x_{j;i_1 i_2 \ldots i_R}\} \in \mathbb{R}^D, \quad w_{ij} \in \mathbb{R} \quad \forall j \in \mathbb{Z}_m. \tag{1.58}$$

The input of the bias neuron, $\mathbf{1}$, is also the same tensor form of the other inputs but is an identity tensor with all values equal to one. The weight for the bias neuron, b_i, is called a bias.

We define the neuron's stimulus as the linear combination of the inputs and their weights of all neurons, including the bias neuron, such as

$$\boldsymbol{s}_i = \sum_{j=0}^{N-1} w_{ij} \boldsymbol{x}_j + b_i \mathbf{1}. \tag{1.59}$$

Once we obtain the neuron's stimulus for given inputs, we can renew the neuron i's state following Eq. (1.57).

In many cases, the neuron's output is defined as its state. For example, the neuron i's output, input $\boldsymbol{x}_i$ from the neuron i to other neurons, is the state of the neuron i;

$$\boldsymbol{x}_i = \boldsymbol{a}_i. \tag{1.60}$$

Layer Structurally similar biological neurons in the brain, such as the cerebral cortex and cerebellum cortex, form a cell layer. Biological neurons exhibit a collective behavior in which neurons in the cell layer activate simultaneously rather than separately.

Following the nerve system's layer structure, artificial neural network approaches gather neurons in a group called a layer and update their state concurrently. Neurons in a layer have the same state's tensor form and share an activation function. Therefore, an artificial neural network's conceptual unit is a neuron, but the functional unit of an artificial neural network for calculation is a layer.

Let us consider a layer consisting of M neurons. We define a layer's inactivated state $\boldsymbol{z}$ as a set of the neurons' inactivated state, such as

$$\boldsymbol{z} \equiv \{\boldsymbol{s}_0, \cdots, \boldsymbol{s}_{M-1}\} \in \mathbb{R}^{MD}, \quad z_j = \boldsymbol{s}_j, \quad j \in \mathbb{Z}_M, \tag{1.61}$$

where z_j is the $(j+1)$th neuron's inactivated state. Similarly, we define a layer's state $\boldsymbol{y}$ as a set of the neurons' state, such as

$$\boldsymbol{y} \equiv \{\boldsymbol{a}_0, \cdots, \boldsymbol{a}_{M-1}\} \in \mathbb{R}^{MD}, \quad y_j = \boldsymbol{a}_j, \quad j \in \mathbb{Z}_M, \tag{1.62}$$

where y_j is the $(j+1)$th neuron's state.

Because neurons in the layer share an activation function, the layer is updated for the given layer's inactivated state as follows

$$\boldsymbol{y} = f(\boldsymbol{z}) \to y_i = f(z_i), \quad i \in \mathbb{Z}_N \tag{1.63}$$

As in Eq. (1.57), if an activation function takes the layer's inactivated state, it means to apply the activation function for each neuron in the layer.

We will repeat the process of a neuron getting inputs and renewing its states about a layer. To update the neurons' state in a layer simultaneously, we gather all inputs received in the layer and call a set of inputs a layers' input. When there are N inputs, and each input follows Eq. (1.58), the layer's input denotes

$$\boldsymbol{x} \equiv \{\boldsymbol{x}_0, \cdots, \boldsymbol{x}_{N-1}\} \in \mathbb{R}^{ND}. \tag{1.64}$$

Here, we can design various layers depending on how to determine the layer's inactivated state for a given layer's input.

As in Figure 1.8, we consider that each neuron in the layer receives all the given

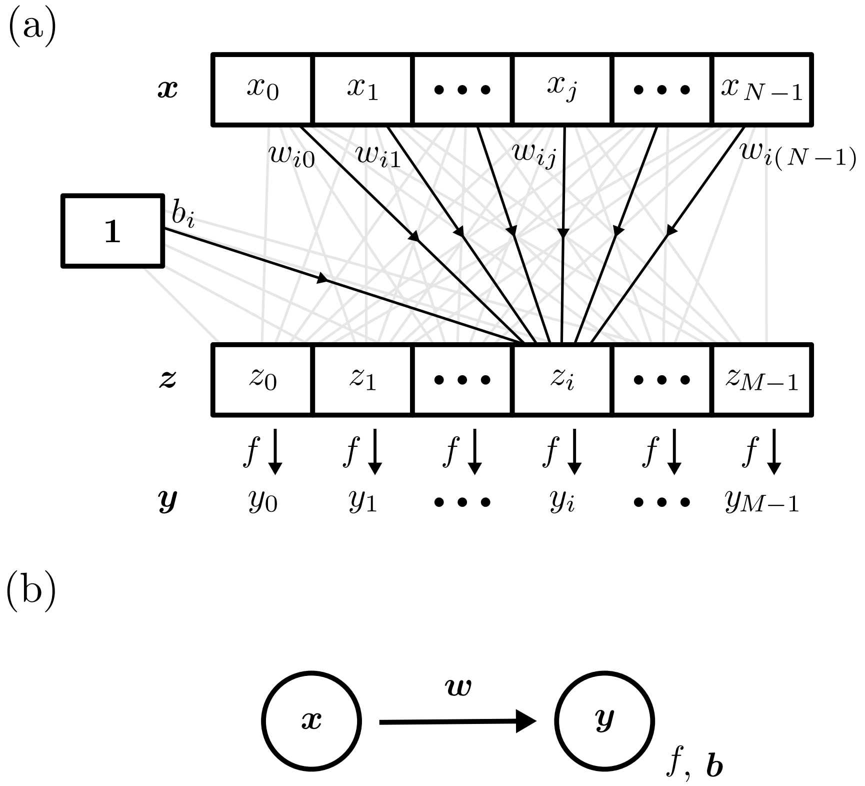

Figure 1.8: Panel (a) shows a fully-connected layer based on a neuron unit, and panel (b) shows the equal layer based on a layer unit. (a) A schematic illustration for a fully-connected layer. A square box is a neuron, and an arrow connects inputs to the neuron with the weight. Each neuron receives all the given inputs and additional input from the bias neuron, the identity tensor. We activate each neuron's state using the shared activation function. (b) A graphical representation for a fully-connected layer. A circle denotes a layer, and an arrow connects the input to the layer with weights $\boldsymbol{w}$ and the biases $\boldsymbol{b}$.

inputs and additional input from the bias neuron, the identity tensor. Such a layer is called a full-connected layer, both the primary and widely used layer in an artificial neural network.

For neuron i, w_{ij} denotes the weight of the input $\boldsymbol{x}_j$, and b_i is the bias.

$$\boldsymbol{w} = \{w_{ij}\} \in \mathbb{R}^{MN}, \quad \boldsymbol{b} = \{b_i\} \in \mathbb{R}^{M}, \quad i \in \mathbb{Z}_M, \quad j \in \mathbb{Z}_N \tag{1.65}$$

According to Eq. (1.59), neurons in the fully-connected layer have an inactivated state, the total stimulus, as follows

$$z_i = \sum_{j=0}^{N-1} w_{ij}\boldsymbol{x}_j + b_i\mathbf{1}, \quad i \in \mathbb{Z}_M. \tag{1.66}$$

Note that the z_i indicates the inactivated states of the $(i+1)$th neuron in the layer and has the same tensor form of the input $\boldsymbol{x}_j$.

When we can obtain a neuron's state as a real number ($D = 1$), we represent the layer's state as an M-dimensional vector and the layer's input as an N-dimensional vector. Furthermore, the weights are an matrix with MN dimensions, and the biases are an M-dimensional vector. So, we rewrite Eq. (1.66) for the real-valued neuron's state as matrix form such as

$$\boldsymbol{z} = \boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}. \tag{1.67}$$

A sparsely-connected layer is another type of layer. Figure 1.9 shows the sparsely-connected layer, in which Each neuron receives the input from the bias neuron and only the linked inputs. We generally represent the connection between a neuron and an input by an adjacency matrix $\boldsymbol{A}$ of the graph theory. The adjacency matrix is $A_{ij} = 1$ if neuron i links to an input $\boldsymbol{x}_j$, otherwise the adjacency matrix $A_{ij} = 0$. We define the inactivated neuron's state in the sparsely-connected layer for the given adjacency

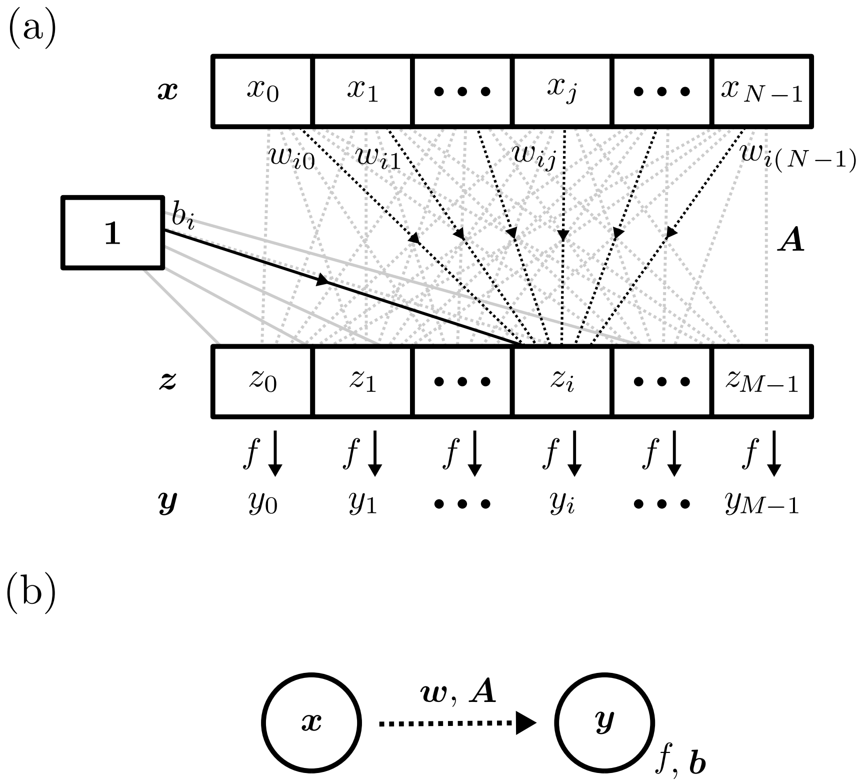

Figure 1.9: Panel (a) shows a sparsely-connected layer based on a neuron unit, and panel (b) shows the equal layer based on a layer unit. (a) A schematic illustration for a sparsely-connected layer. A square box is a neuron, and an arrow connects inputs to the neuron with the weight. Each neuron receives the input from the bias neuron and only the linked inputs. The dotted line links an input to a neuron following the adjacency matrix $\boldsymbol{A}$. We activate each neuron's state using the shared activation function. (b) A graphical representation for a sparsely-connected layer. A circle denotes a layer, and an arrow connects the input to the layer with weights $\boldsymbol{w}$, the biases $\boldsymbol{b}$, and the adjacency matrix $\boldsymbol{A}$.

matrix as follows

$$z_i = \sum_{j=0}^{N-1} A_{ij} w_{ij} \boldsymbol{x}_j + b_i \mathbf{1}, \quad i \in \mathbb{Z}_M. \tag{1.68}$$

This equation equivalent to Eq. (1.66) with constrain holding the weight w_{ij} to zero for non-connection between neuron i and input $\boldsymbol{x}_j$.

We can adjust the connection between the input and the neuron by the adjacency matrix, and multiplying the adjacency matrix to weights in the fully-connected layer is called masking. We employ masking for a dropout [54], which is a method to prevent overfitting the artificial neuron network.

Similar to the sparsely-connected layer, a neuron in a locally connected layer receives inputs from neighboring neurons. A convolutional layer [44, 55] is the most famous locally connected layer in which each neuron has the same number of inputs and shares weights with other neurons. However, the convolutional layer's weight called feature map is a tensor with the rank of the layer's input, not scalar, and the convolutional layer employs multiple weights for the same input. Thus, we can not represent the inactivated state of neurons in the convolutional layer as a linear combination of inputs and weights but instead express a complicated form, including a convolution operator of input tensor and weight tensor.

Above layers do not have intra-connections between neurons in the same layer but only have interconnections between neurons in the different layers. However, a recurrent layer in Appendix B has intra-connections to let an artificial neuron network have a memory for sequential data.

In addition to the above layers, various layers have been developed, and new layers are still proposed according to how to define an inactivated state for given inputs.

Once we obtain the inactivated state of the layer, we also calculate the layer's state following Eq. (1.63). Of course, like neurons, a layer can output its state. For example, a layer l outputs its state to give input to other layers.

$$\boldsymbol{x}^{(l)} = \boldsymbol{y}^{(l)} \tag{1.69}$$

Modeling So far, we have looked at neurons and layers for constructing artificial neural networks. Here, we employ artificial neural networks to implement a machine learning model defined in subsection 1.2.1. It is possible to configure both the deterministic model and the statistical model with an artificial neural network.

First, let us design an artificial neural network for the deterministic model, satisfying Eq. (1.15). We introduce an input layer that has its state as the given data following Eq. (1.9). The input layer's state becomes a model's input. Moreover, we introduce an output layer with a state tensor with the same form of the model's output following Eq. (1.13). The output layer's state becomes a model's output.

The next to complete the artificial neural networks is connecting the input layer and the output layer. For example, as shown in Figure 1.10(a), we can directly link the input layer to the output layer. We assume that the flattened data is an n-dimensional vector and the input layer's neuron has a real-valued state. ($D = 1$, $N = n$) And the model's output is an m-dimensional vector and and the output layer's neuron has a real-valued state. ($D = 1$, $M = m$) If the output layer is the fully-connected layer, we obtain the artificial neural network $\mathcal{M}_0$ as a machine learning model as follows

$$\mathcal{M}_0[\boldsymbol{w}, \boldsymbol{b}, f](\boldsymbol{x}) = f(\boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}). \tag{1.70}$$

The artificial neural network $\mathcal{M}_0$ consisting of a single fully-connected layer is called a perceptron model. Here, the perceptron model's weights and biases are the model variables defined in the machine learning approach.

The perceptron model has limited performance but is essential in the artificial neural network approach as a null or baseline model. The perceptron model is the base for designing artificial neural networks, and we estimate models' performance by compar-

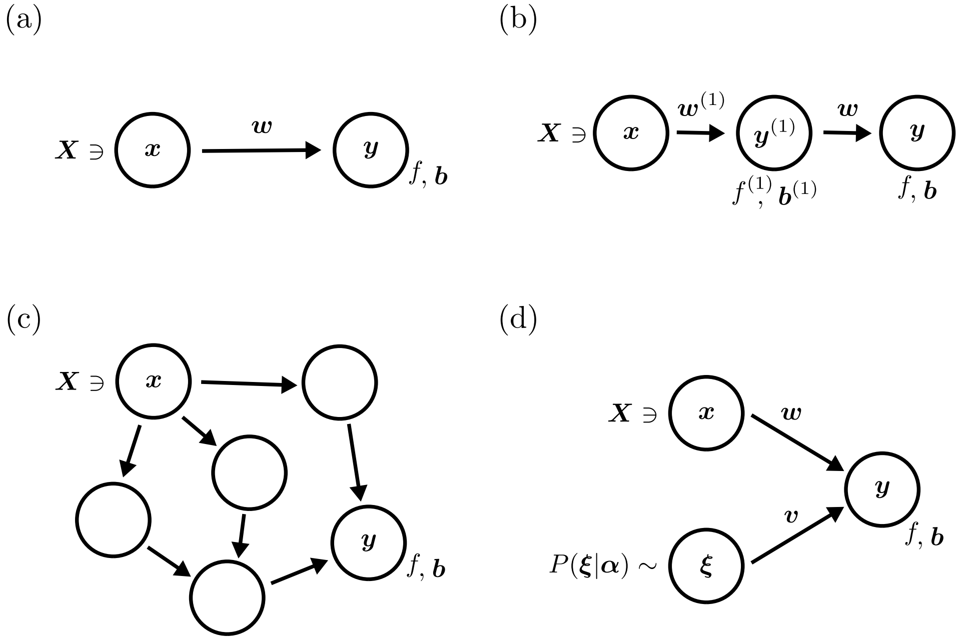

Figure 1.10: Examples of artificial neural networks. A circle denotes a layer with the state, an activation function, and biases, and an arrow denotes a fully-connection with weights. (a) The perceptron model consisting of an input layer and a fully-connected output layer. (b) A simple model of deep learning model with a single hidden layer. (c) A example of a feed-forward neuron network with multiple hidden layers. (d) The modified perceptron model for a statistical model.

ing the perceptron model's performance.

We can consider an artificial neural network $\mathcal{M}_1$ with a layer between the input and output layers, as shown in Figure 1.10(b). A layer between the input layer and the output layer is called a hidden layer, and a model with at least one hidden layer is called a deep learning model. If the hidden layer and the output layer in Figure 1.10(b) is the fully-connected layer, we obtain the model $\mathcal{M}_1$ as follows

$$\mathcal{M}_1[\boldsymbol{w}^{(1)}, \boldsymbol{b}^{(1)}, f^{(1)}, \boldsymbol{w}, \boldsymbol{b}, f](\boldsymbol{x}) = f\left(\boldsymbol{w} f^{(1)}\left(\boldsymbol{w}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}\right). \tag{1.71}$$

One of the useful properties of artificial neural networks is that they have modularity. Thus, we rewrite the model $\mathcal{M}_1$ as a combination of two perceptron models as follows

$$\mathcal{M}_1[\boldsymbol{w}^{(1)}, \boldsymbol{b}^{(1)}, f^{(1)}, \boldsymbol{w}, \boldsymbol{b}, f](\boldsymbol{x}) = \mathcal{M}_0[\boldsymbol{w}, \boldsymbol{b}, f]\left(\mathcal{M}_0[\boldsymbol{x}^{(1)}, \boldsymbol{b}^{(1)}, f^{(1)}](\boldsymbol{x}))\right). \tag{1.72}$$

It is possible to design a complex model by placing several hidden layers between the input and output layers, as shown in Figure 1.10(c).

Second, let us design an artificial neural network for the statistical model, satisfying Eq. (1.16). We introduce that a sampler is a layer such that its state $\boldsymbol{\xi}$ following the probability distribution $P(\boldsymbol{\xi}|\boldsymbol{\alpha})$.

Figure 1.10(d) shows a perceptron model modified for the statistical model. The output layer receives the given data as the input layer's state and the random variables as the sampler's state. Because the fully-connected layer's inactivated state is a linear combination of all the given inputs and their weight, for the multiple inputs, the fully-connected layer accepts each layer's state with its weights. Since the output layer in the perceptron model is a fully-connected layer, and we obtain the modified perceptron model $\mathcal{M}_s$ as follows

$$\mathcal{M}_s[\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{b}, f](\boldsymbol{\xi}, \boldsymbol{x}) = f(\boldsymbol{w}\boldsymbol{x} + \boldsymbol{v}\boldsymbol{\xi} + \boldsymbol{b}), \quad \boldsymbol{\xi} \sim P(\boldsymbol{\xi}|\boldsymbol{\alpha}). \tag{1.73}$$

We can design various artificial neural networks with input layers, output layers, hidden layers, and samplers for the machine learning model.

Artificial neural network approaches are welcomed and widely used in many fields because the principles of developing a model are clear and concise, but the model exhibits powerful performance. Furthermore, an artificial neural network has modularity, which let us develop from a simple model to a complicated model combined with basic models. In practically, an artificial neural network's flexibility helps us design the model considering technical environments such as computing capacity and speed.

Chapter 2

Machine learning approach for open quantum systems

Quantum critical phenomena in nonequilibrium systems have drawn considerable attention recently from physics community [56–75] as experimental techniques are developed in cold atomic physics such as trapped ions [59] and lattices of ultracold ions [60–62]; driven circuit quantum electrodynamics systems [63]; and semiconductor microcavities [64]. The quantum criticality in equilibrium state may be perturbed by external environment, and thus the combined system is left in a non-equilibrium state. One of the associated phenomena is the quantum phase transition arising in a Josephson junction, from a normal to a superconductor state, depending on the value of an external shunt resistor [76]. We are interested in dissipative phase transitions arising from the competition between coherent Hamiltonian dynamics and incoherent dissipation process [74, 76–86, 86, 87]. For these systems, the questions arise whether the competition between quantum and classical fluctuations leads to novel universal behavior [74, 79], and under which conditions they exhibit classical critical behavior in terms of the loss rates by the environment [76, 81, 83].

Here, we aim to answer these questions by considering the quantum contact process [88–94] in one dimension lattice. A recent numerical study of the quantum contact process with setting $\kappa = 0$ in one dimension [92–94] revealed that only a continuous transition occurs and the discontinuous transition disappears. Moreover, when the quantum contact process starts from a homogeneous state. In other words, all sites

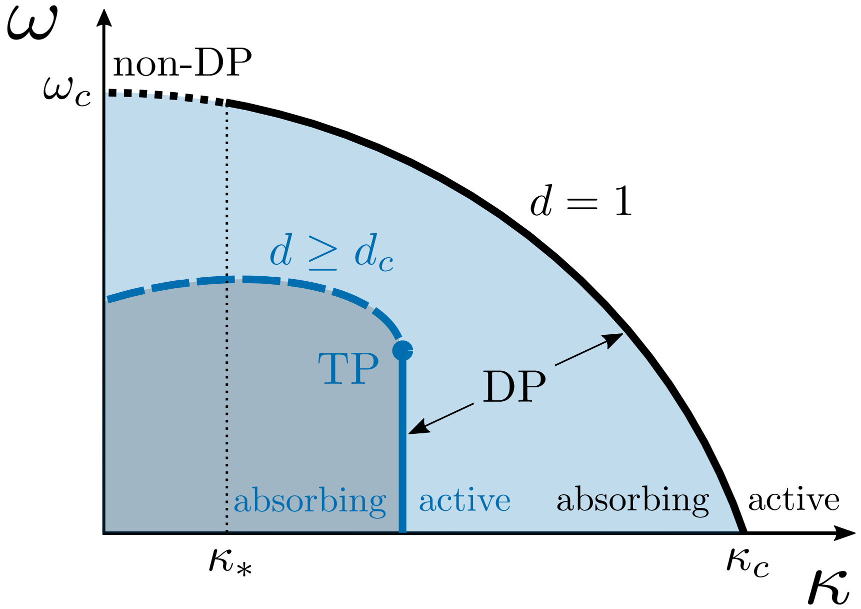

Figure 2.1: A schematic phase diagram for the quantum contact process model in the parameter space (κ, ω) in the mean-field limit (inside) and one dimension (outside). For the former, a discontinuous (dashed curve) and a continuous transition (solid line) occur and they meet at a tricritical point. For the latter, a continuous transition occurs over the entire region $[0, \kappa_c]$, however, in the interval $[0, \kappa_*]$, the exponent α of the density of active sites $n(t) \sim t^{-\alpha}$ continuously decreases as κ is increased with non-directed percolation values. In the region $\kappa \in [\kappa_*, \kappa_c]$, it has the directed percolation value.

are in active state, the density of active sites at time t, denoted as $n(t)$, decays as $n(t) \sim t^{-\alpha}$ at the transition point ω_c for $\kappa = 0$. The exponent α is estimated as 0.36 different from the directed percolation value 0.16. It was argued that one dimensional quantum contact process creates a novel critical behavior. This result was obtained using the tensor network algorithm.

In this study, we will confirm the above result and further show that there exists an interval $[0, \kappa_*]$, in which the exponent α decreases continuously as κ is increased, and for $\kappa \geq \kappa_*$, it has the directed percolation value. The phase diagram for the one dimensional quantum contact process is shown in Figure 2.1. We obtain this result by performing the quantum jump Monte Carlo method [95–101]. Moreover, using the neural network machine learning algorithm, we determine the transition point $\omega_c(\kappa)$ for each κ. Applying the finite-size scaling analysis, we determine the exponent $\nu_\perp$ associated with the correlation length. This exponent value is also deviated from that obtained using the tensor network approach [94]. However, it is consistent with the directed percolation value within the error bar. We find that when the one dimensional quantum contact process starts from a single active site, all the critical exponents are consistent with the directed percolation values. Based on these results, we conclude that when the one dimensional quantum contact process starts from the homogeneous state, the quantum coherence is long-ranged and it plays a similar role to the Lévy-flight long-range interaction in the tricritical contact process. When an active site interacts to an inactive site at a distance r with rate $\kappa P(r) \sim \kappa/r^{d+\sigma}$ in one dimension [90, 102, 103], the exponent α depends on σ in an appropriate range of σ.

2.1 Quantum contact process

The contact process is a prototype model exhibiting a nonequilibrium phase transition. Each element of the system is in active or inactive state, and it changes its state according to the rule of contact process model [102, 104–109]. When all elements are in inactive state, then the system becomes trapped in a frozen configuration. Examples include the catalytic reactions arising in the oxidation of carbon monoxide on platinum surface [107]. Recently this dynamics was realized using the spin orientations of Rydberg atoms in one dimension [18]. The classical contact process problem extends to the contact process in dissipative quantum systems in one dimension, denoted as one dimensional quantum contact process. The dynamics of the one dimensional quantum contact process model is described by the Lindblad equation, which consists of coherent Hamiltonian and incoherent dissipative terms. Their contributions to overall dynamics are adjusted by model parameters ω for the coherent quantum effect and κ for the incoherent classical dynamics.

The system would exhibit a quantum or classical phase transition in the extreme cases. A previous result based on the mean-field solution [88] showed that the quantum contact process exhibits a continuous (discontinuous) phase transition when κ is large (small). There exists a tricritical point as shown in Figure 2.1. This result is similar to the phase diagram generated by the so-called tricritical contact process explored in classical systems [108]. We remark that the absorbing phase transition of the classical contact process belongs to the directed percolation universality class.

We consider a one-dimensional quantum spin chain with a periodic boundary condition, where each state of a site, either active or inactive, represents the up or down spin state, denoted as $|\uparrow\rangle$ or $|\downarrow\rangle$. The time evolution of the density matrix $\hat{\rho}$ is described by the Lindblad equation, which consists of the Hamiltonian and dissipative terms [17]:

$$\partial_t \hat{\rho} = -i \left[\hat{H}_S, \hat{\rho} \right] + \sum_{a=b,c,d} \sum_{\ell=1}^{N} \left[\hat{L}_\ell^{(a)} \hat{\rho} \hat{L}_\ell^{(a)\dagger} - \frac{1}{2} \left\{ \hat{L}_\ell^{(a)\dagger} \hat{L}_\ell^{(a)}, \hat{\rho} \right\} \right] . \tag{2.1}$$

The Hamiltonian $\hat{H}_S$, which governs the branching and coagulation processes and represents coherent interactions, is expressed as

$$\hat{H}_S = \omega \sum_{\ell=1}^{N} \left[\left(\hat{n}_{\ell-1} + \hat{n}_{\ell+1} \right) \hat{\sigma}_\ell^x \right] . \tag{2.2}$$

The Lindblad operators of decay, branching, and coagulation are given by

$$\hat{L}_\ell^{(d)} = \sqrt{\gamma} \hat{\sigma}_\ell^- , \tag{2.3}$$

$$\hat{L}_\ell^{(b)} = \sqrt{\kappa} (\hat{n}_{\ell-1} + \hat{n}_{\ell+1}) \hat{\sigma}_\ell^+ , \tag{2.4}$$

$$\hat{L}_\ell^{(c)} = \sqrt{\kappa} (\hat{n}_{\ell-1} + \hat{n}_{\ell+1}) \hat{\sigma}_\ell^- , \tag{2.5}$$

respectively. $\hat{\sigma}_\ell^+$ and $\hat{\sigma}_\ell^-$ are the raising and lowering operators of the spin at site ℓ, respectively. They are defined in terms of the spin basis as $\hat{\sigma}^+ = |\uparrow\rangle\langle\downarrow|$ and $\hat{\sigma}^- = |\downarrow\rangle\langle\uparrow|$. In addition, $\hat{n} = \hat{\sigma}^+ \hat{\sigma}^-$ and $\hat{\sigma}^x = \hat{\sigma}^+ + \hat{\sigma}^-$ are the number operator and spin

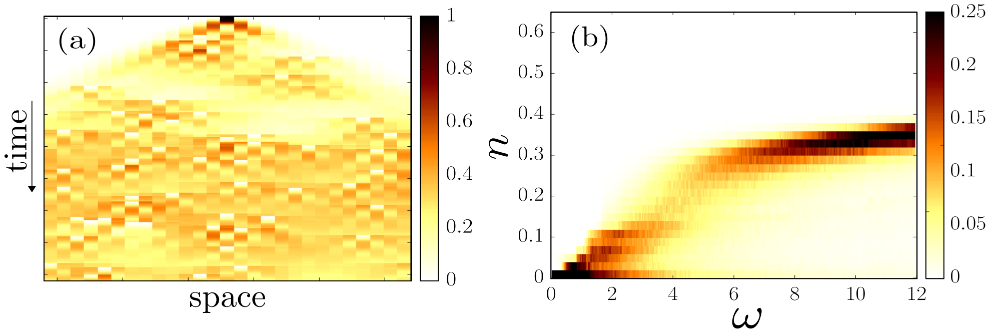

Figure 2.2: (a) Trajectory of the one dimensional quantum contact process with $\kappa = 0$ and $\omega > \omega_c$ from a single active site at the center. Branching and coagulation processes are coherently driven by the Hamiltonian. (b) Histogram of the densities of active sites in steady states as a function of ω for system size $N = 20$. The data are obtained using quantum jump Monte Carlo simulations. Time t and the control parameter ω are given in units of $1/\gamma$ and γ, respectively.

flip operator, respectively.

Quantum branching and coagulation processes occur at a rate ω, and the corresponding classical processes occur at a rate κ. We first consider the pure quantum limit $\kappa \to 0$ but ω is finite. In addition, we rescale time and the quantum control parameter ω in units of γ; therefore, we set $\gamma = 1$.

When ω is small, inactive particles become more abundant with time, and eventually the system is fully occupied by inactive particles. The system is no longer dynamic and falls into an absorbing state, which is represented by $\hat{\rho}_{\text{ab}} = |\downarrow \cdots \downarrow\rangle\langle\downarrow \cdots \downarrow|$.

When ω is large, the system remains in an active state with a finite density of active particles in Figure 2.2(a). Thus, the quantum contact process exhibits a phase transition from an active to an absorbing state as the control parameter ω is decreased. In Figure 2.2(b), the phase transition seems to be continuous. In fact, it was conjectured that the one dimensional quantum contact process exhibits a continuous transition [91].

The transition point and spatial correlation length exponent were obtained numerically using the tensor network approach as $\omega_c = 6.0 \pm 0.05$ and $\nu_\perp = 0.5 \pm 0.2$ [92]. However, we obtain them using the following artificial neural network approach as

$\omega_c \approx 6.04$ and $\nu_\perp = 1.06 \pm 0.04$.

2.2 Finding the quantum phase transition

The artificial neural network approach has recently served as a powerful tool [1,110] for classifying the phases in classical systems [9]. In this case, occupation of each element is represented by a binary value. However, in quantum systems, it is represented by real value, thus the collective pattern would be more complex. Nevertheless, the supervised learning for quantum systems has reportedly been successfully used to identify the transition point of closed quantum systems on the basis of simulation data [10,12,111,112] and experimental images [2,3].

The unsupervised learning has recently been applied to generate the configurations of open quantum systems in steady state using the restricted Boltzmann machine [113–117]. It is challenging to investigate the critical behaviors using unsupervised learning techniques such as the restricted Boltzmann machine.

However, the artificial neural network approach has never been attempted to critical phenomena of dissipative phase transitions. For the first time, we try the supervised learning to identify the transition point $\omega_c(\kappa)$.

Here is the overview of artificial neural network approach to detect quantum phase transition. We first take labeled snapshots of the one dimensional quantum contact process generated by quantum jump Monte Carlo simulations of a finite system far from a transition point in both directions, and then organized in datasets. The well-optimized artificial neural networks then respond sensitively to the transition point. This supervised learning method correctly identifies the position of the transition point. Second, using the obtained transition points $\omega_c(N)$ for given system sizes, we perform finite-size scaling analysis and identify the transition point in the thermodynamic limit ω_c. We also determine the correlation length exponent $\nu_\perp$. Next, we determine other critical exponents by performing extensive quantum jump Monte Carlo simulations at ω_c for a large system.

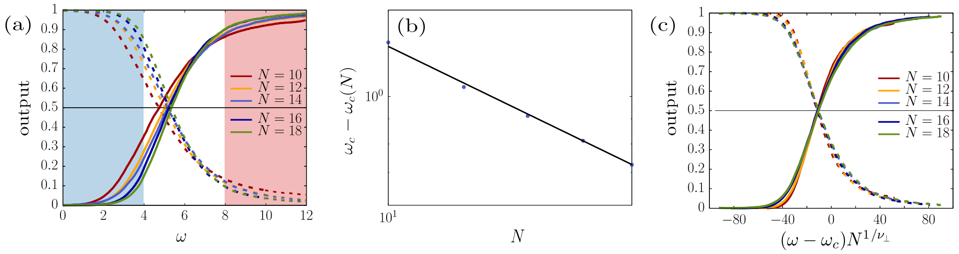

Figure 2.3: Plots of the artificial neural network's outputs. (a) Plot of the output averaged over a test set as a function of ω for different system sizes. The value of the first (second) output neuron is represented as solid (dashed) line. From this plot, we estimate the crossing point of the two outputs and regard it as the transition point $\omega_c(N)$ for a given system size N. The shaded regions $[0, 4]$ and $[8, 12]$ indicate the training sets used in the convolutional artificial neural network analysis. (b) Plot of $\omega_c - \omega_c(N)$ versus N, where ω_c is chosen so as to yield power-law behavior, and regarded as the transition point in the thermodynamic limit. The slope represents the value of the critical exponent $-1/\nu_\perp$. (c) Scaling plot of the output versus $(\omega - \omega_c)N^{1/\nu_\perp}$. For the obtained numerical values of $\nu_\perp$ and ω_c, the data well collapse for system sizes $N = 10, 12, 14, 16,$ and 18. From (b) and (c), we obtain $\omega_c \approx 6.04$ and $\nu_\perp = 1.06 \pm 0.04$.

To implement the artificial neural network approach, we first organize a dataset of the occupation probability of site ℓ, which is denoted as $p_\ell(t) = \mathrm{Tr}[\hat{\rho}(t)\hat{n}_\ell]$. Using the quantum jump Monte Carlo method, we generate a steady-state configuration and obtain the occupation probabilities of each site, $\{p_\ell\}$. We collect 5000 configurations in $\omega \in [0, 12]$ at $\Delta\omega = 0.04$ intervals. To prepare the training dataset for supervised learning, we label the configurations using one-hot encoding [118] of the absorbing state ($\omega \in [0, 4]$) as $(0, 1)$ and of the active state ($\omega \in [8, 12]$) as $(1, 0)$. See shaded regions in Figure 2.3(a).

Let us introduce the details of the structures of our neural network. We construct the hidden layers of the artificial neural network, including one-dimensional convolutional layers, batch normalization layers, and fully connected layers, as shown in Fig. 2.4. We employ the framework of TENSORFLOW [119] and use ReLU and $\tanh$ for the activation function in the hidden layer. Two neurons in the output layer are used, and a softmax function is used as the activation function in the output layer. We employ

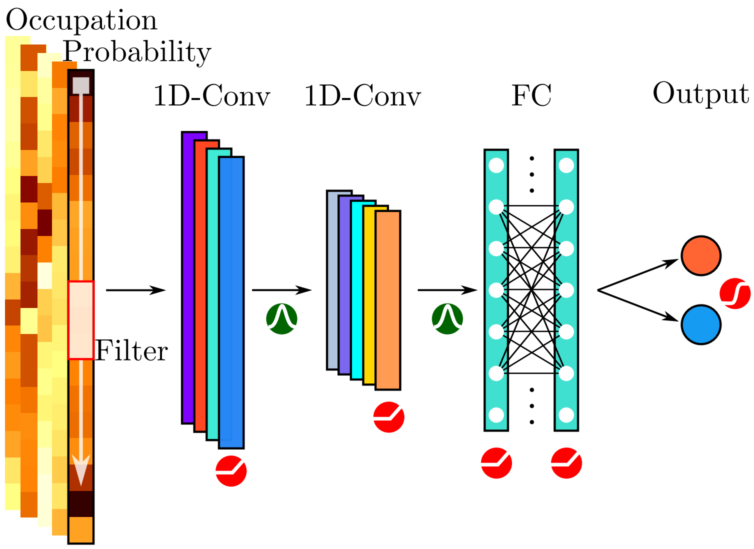

Figure 2.4: A schematic illustration of the convolutional neural network built in combination of a one-dimensional convolutional layer and a fully connected layer. The red circles represent the activation function of each layer. The green circles below the arrows represent the batch normalization.

the cross-entropy or the mean-square loss function as the loss function of the artificial neural network, which is then optimized using Adam [120] or RMSProp [121]. We change the architecture and optimization algorithms in various ways. Regardless of these changes, the well-trained machines produce consistent results.

Once the artificial neural network is well-trained with the labeled training dataset in the two regions, we obtain the outputs for the entire ω region. In Figure 2.3(a), we plot the output averaged over 5×10^3 configurations for system sizes $N = 10$, 12, 14, 16, and 18. The two outputs provide the probabilities that the system will fall into the absorbing state and remain in the active state, respectively. The crossing point of the two outputs indicates a transition point $\omega_c(N)$ for a given system size N in Figure 2.3(a).

Using the obtained $\omega_c(N)$ for different system sizes, we determine ω_c in the thermodynamic limit by plotting $\omega_c - \omega_c(N)$ versus N, which is expected to behave as

$\omega_c - \omega_c(N) \sim N^{-1/\nu_\perp}$, as shown in Figure 2.3(b). Indeed, the plot exhibits power-law behavior when an appropriate value of ω_c is chosen, and the critical exponent $\nu_\perp$ is obtained from the slope of the power-law behavior. We obtain $\omega_c \approx 6.04$ and $\nu_\perp = 1.06 \pm 0.04$. Those results are in agreement with $\nu_\perp \approx 1.096$ for the directed percolation class in one dimension, but differs from the value $\nu_\perp \approx 0.5 \pm 0.2$ obtained using the tensor network approach. Finally, scaling plot is drawn in the form of the output versus $(\omega - \omega_c)N^{1/\nu_\perp}$ for different N values. In Figure 2.3(c), the data seem to collapse for different system sizes.

2.3 Finite-size scaling on quantum jump Monte Carlo

We measure the values of the other critical exponents using the numerical data obtained by the quantum jump Monte Carlo method in the critical region around ω_c.

2.3.1 The pure quantum limit

To do this, firstly, an initial state is taken as that a single active seed is present at $\ell = 0$, whereas the remaining sites are inactive. This configuration is expressed as $\hat{\rho}(0) = \hat{\sigma}_0^+ \rho_{\text{ab}} \hat{\sigma}_0^-$.

We measure the following quantities:

1. The survival probability, the probability that the system does not fall into an absorbing state, $P(t) = 1 - \text{Tr}[\hat{\rho}(t)\hat{\rho}_{\text{ab}}]$.
2. The number of active sites, $N_a(t) = \sum_\ell \text{Tr}[\hat{\rho}(t)\hat{n}_\ell]$.
3. The mean square distance of the active sites from the origin, $R^2(t) = \sum_\ell \text{Tr}[\ell^2\, \hat{\rho}(t)\hat{n}_\ell]/N_a(t)$.
4. The density of active sites over all runs, $\rho_d(t) = \text{Tr}[\hat{\rho}(t)\hat{n}_{\ell=0}] = N_a(t)/R(t)$.
5. The density of active sites over surviving runs, $\rho_{d,s}(t) = \rho_d/P(t)$.

	1D QCP	1D QCP	1D DP
	CNN+QJMC	Tensor network [92, 93]	FSS
ω_c	6.04	6.0 ± 0.05	—
δ'	0.16 ± 0.05	0.26 ± 0.04	0.159
z	1.55 ± 0.06	1.61 ± 0.16	1.581
η	0.30 ± 0.05	0.26 ± 0.05	0.313
$\delta + \delta'$	0.32 ± 0.01	0.36 ± 0.12	0.318
α	0.32 ± 0.01	0.36 ± 0.08	0.159
$\nu_\perp$	1.06 ± 0.04	0.5 ± 0.2	1.096

Table 2.1: Critical point and critical exponents for the one dimensional quantum contact process. The first column shows the results of artificial neural network approach, and the second column shows the results of the tensor network approach for the one dimensional quantum contact process. The third column shows that the results of the ordinary finite-size scaling method for one dimensional directed percolation.

At the transition point, these quantities exhibit the following power-law behaviors: $P(t) \propto t^{-\delta'}$, $N_a(t) \propto t^{\eta}$, $R^2(t) \propto t^{2/z}$, $\rho_d(t) \propto t^{\eta-1/z}$, and $\rho_{d,s}(t) \propto t^{-\delta}$. Using the relation $\rho_d(t) \sim \rho_{d,s}(t)P(t) \sim t^{-\delta-\delta'}$, the scaling relation $\eta - 1/z = -(\delta + \delta')$ holds [122].

We estimate the exponents $\delta + \delta'$, η, δ', z, and δ by measuring the slopes directly in the double-logarithmic plots as shown in Figs. 2.5. We estimate the exponent z using the data-collapse technique. For instance, for the survival probability $P(t)$, we plot $P(t)t^{\delta'}$ versus tN^{-z} for different system sizes N. We determine z as the value by which the data for different system sizes collapse onto a single curve. The critical exponents are obtained as $\delta + \delta' = 0.32 \pm 0.01$, $\eta = 0.30 \pm 0.05$, $\delta' = 0.16 \pm 0.05$, $z = 1.55 \pm 0.06$, and $\delta = 0.16 \pm 0.06$. These exponent values are in good agreement with the directed percolation values within the error bars. The exponent values are listed in Table2.1.

Secondly, we take a homogeneous initial state. That is, the entire system is occupied by active sites at $t = 0$, which is represented by $\hat{\rho}(0) = |\uparrow \cdots \uparrow\rangle\langle\uparrow \cdots \uparrow|$. From

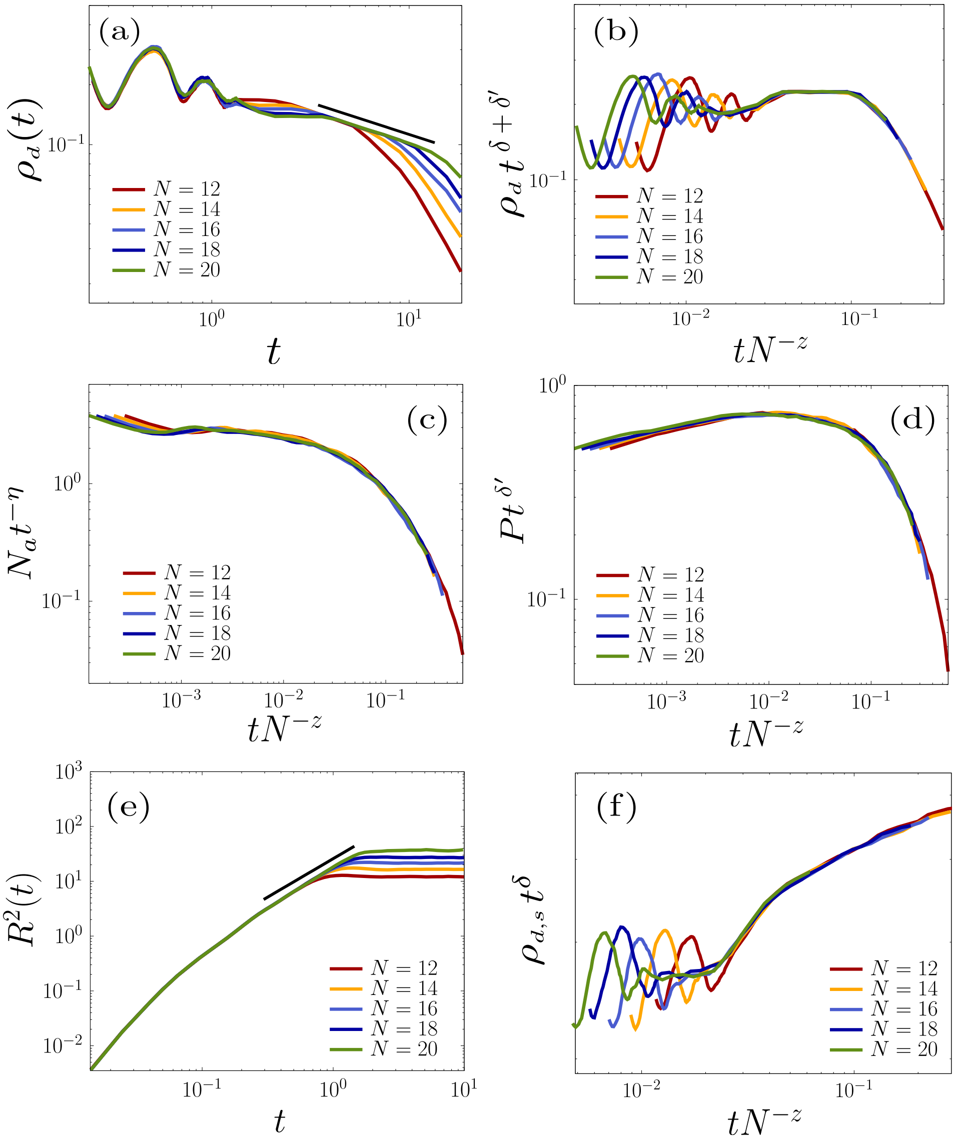

Figure 2.5: Estimates of the critical exponents in the pure quantum limit from a single active site. (a) Plot of $\rho_d(t)$ versus t, which behaves as $\rho_d(t) \sim t^{-\delta-\delta'}$. The solid line is a guideline with slope -0.32. (b) Scaling plot of $\rho_d(t)t^{\delta+\delta'}$ versus tN^{-z} for $\delta+\delta' = 0.32$ and $z = 1.55$. (c) Scaling plot of $N_a(t)t^{-\eta}$ versus tN^{-z} for $\eta = 0.30$ and $z = 1.55$. (d) Scaling plot of $P(t)t^{\delta'}$ versus tN^{-z} for $\delta' = 0.16$ and $z = 1.55$. (e) Plot of $R^2(t)$ as a function of t. The solid line is a guideline with slope $2/z$ for $z = 1.55$. (f) Scaling plot of $\rho_{d,s}(t)t^{\delta}$ versus tN^{-z} for $\delta = 0.16$ and $z = 1.55$.

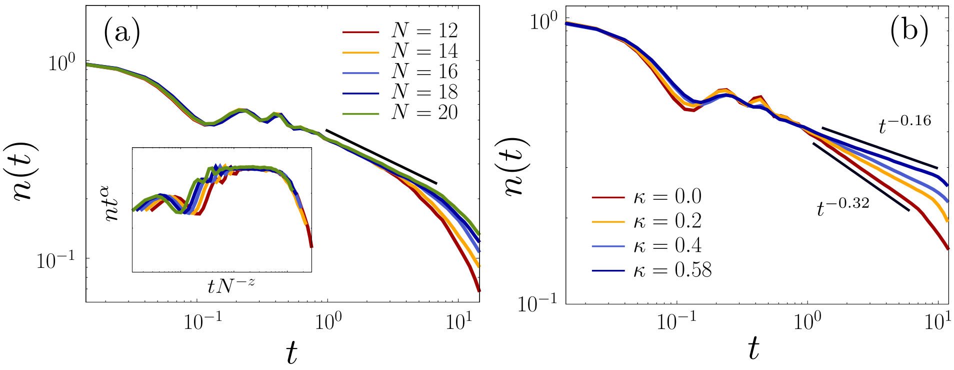

Figure 2.6: Estimates of the critical exponent α in the pure quantum limit from the homogeneous state. (a) Plot of $n(t)$ as a function of t for different system sizes, which shows $n(t) \sim t^{-\alpha}$. The solid line is a guideline with slope -0.32. The inset represents the scaling plot of $n(t)t^{\alpha}$ versus tN^{-z} for $\alpha = 0.32$ and $z = 1.55$. (b) Plot of $n(t)$ as a function of t for different κ in the range $\kappa \in [0, 0.58]$ in steps of 0.2. The lower (upper) solid line is a guideline with slope -0.32 (-0.16).

this initial state, we measure the density $n(t)$ of active sites at time t averaged over all runs. This quantity is formulated as $n(t) = (\sum_{\ell} \mathrm{Tr}[\hat{\rho}(t)\hat{n}_{\ell}])/N$. We find that $n(t)$ decays in power-law way as $n(t) \sim t^{-\alpha}$ with the exponent $\alpha = 0.32 \pm 0.01$ as shown in Figure 2.6. This value is consistent with the result obtained by applying the tensor network approach. However, it is not consistent with the corresponding directed percolation value, which was estimated as $\alpha_{\mathrm{DP}} = 0.16$. Therefore, the one dimensional quantum contact process with $\kappa = 0$ creates a novel universality class.

2.3.2 The classcial limit

When $\omega \to 0$, the model is reduced to the classical contact process, which belongs to the directed percolation class. To check the consistency of the finite-size scaling with the small system size, we obtain the critical exponents of one dimensional classical contact process using the finite-size scaling from the data of quantum jump Monte Carlo method. At the critical point, we perform the finite-size scaling to one dimension classical contact process using the quantum jump Monte Carlo method. The observables correspond to the above definitions.

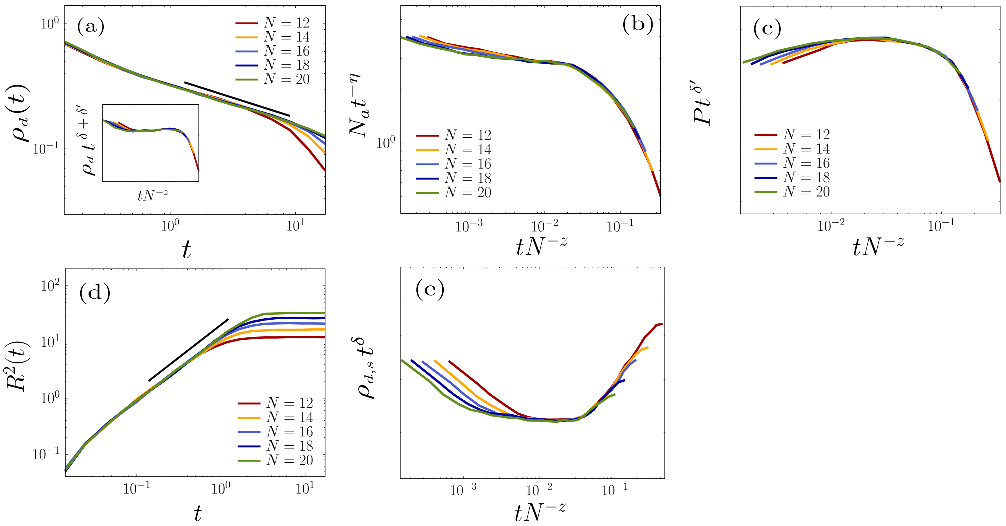

Figure 2.7: Estimates of the critical exponents in classical limit from the single active initial state. (a) Plot of $\rho_d(t)$ versus t, which behaves as $\rho_d(t) \sim t^{-\delta-\delta'}$. The solid line is a guideline with slope -0.32. Inset: scaling plot of $\rho_d(t)t^{\delta+\delta'}$ versus tN^{-z} for $\delta + \delta' = 0.32$ and $z = 1.58$. (b) Scaling plot of $N_a(t)t^{-\eta}$ versus tN^{-z} for $\eta = 0.30$ and $z = 1.58$. (c) Scaling plot of $P(t)t^{\delta'}$ versus tN^{-z} for $\delta' = 0.16$ and $z = 1.58$. (d) Plot of $R^2(t)$ as a function of t. The solid line is a guideline with slope $2/z$ for $z = 1.58$. (e) Scaling plot of $\rho_{d,s}(t)t^{\delta}$ versus tN^{-z} for $\delta = 0.16$ and $z = 1.58$. The units of t is given as $1/\gamma$.

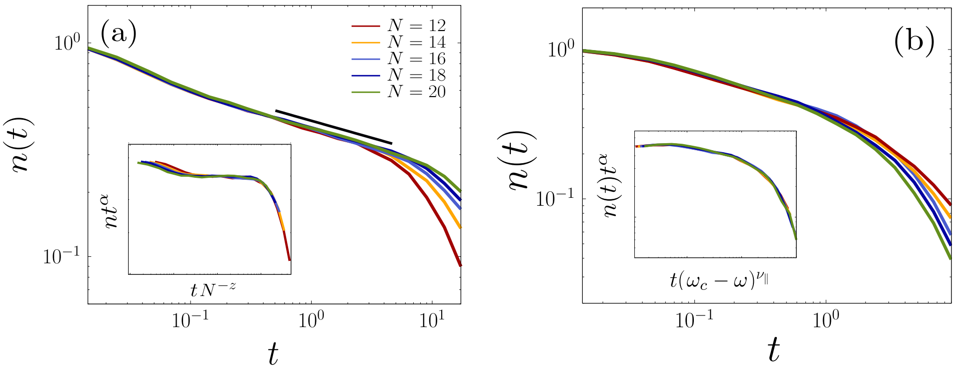

Figure 2.8: Estimates of the critical exponents in classical limit from the fully active initial state. (a) Plot of $n(t)$ as a function of t, which shows $n(t) \sim t^{-\alpha}$. The solid line is a guideline with slope -0.16. The inset represents the scaling plot of $n(t)t^{\alpha}$ versus tN^{-z} for $\alpha = 0.16$ and $z = 1.58$. (b) Plot of $n(t)$ as a function of t for different values of $\omega < \omega_c$. Inset: Data points collapse well onto a single curve for $\alpha = 0.16$, and $\nu_{\parallel} = 1.73$. The units of t is given as $1/\gamma$.

First, we obtain the exponents $\delta + \delta'$, η, δ', z, δ, and α directly by measuring the slopes in the double-logarithmic plots shown in Figs. 2.7 and 2.8. Then, we collapse the data by using the obtained exponents to compute the dynamic exponent z. Specifically, we plot $n_{\text{seed}}t^{\delta+\delta'}$ versus tN^{-z} in Figure 2.7(a), $N_a t^{-\eta}$ versus tN^{-z} in Figure 2.7(b), and $P(t)t^{-\delta'}$ versus tN^{-z} in Figure 2.7(c) for different system sizes N. We measure the exponent z directly using the plot of $R^2(t)$ versus t in Figure 2.7(d). In classical limit, we can classify the surviving runs and thus we measure the exponent $-\delta$ directly using the plot of $n_{\text{seed,sur}}(t)$ versus t in Figure 2.7(e). Next, we plot $n(t)t^{-\alpha}$ versus tN^{-z} in Figure 2.8(a) for different system sizes N. The exponent $\nu_{||}$ is obtained from the rescaling plot of $n(t)t^{\alpha}$ versus $t(\omega_c - \omega)^{\nu_{||}}$ for different ω values in Figure 2.8(b).

The critical exponents are thus obtained as $\delta + \delta' = 0.32 \pm 0.01$, $\eta = 0.31 \pm 0.02$, $\delta' = 0.16 \pm 0.01$, $\delta = 0.16 \pm 0.02$, $z = 1.58 \pm 0.03$, and $\alpha = 0.16 \pm 0.01$. Note that $\delta = \alpha$. In addition, $\alpha = \delta'$ implying that rapidity-reversal symmetry holds. All the critical exponents are in good agreement with the directed percolation values within the error bars. Thus we verified that the critical exponents on classical contact process can be successfully obtained using quantum jump Monte Carlo method with the same system size.

2.4 Discussion and Summary

We note that $\rho_d(t)$ and $n(t)$ are actually the same quantities even though they start from different initial states. They exhibit the same critical behaviors in the classical contact process. Nevertheless, for the one dimensional quantum contact process, they exhibit different critical behaviors. This is unconventional, because universality class is independent of initial states according to the theory of critical phenomena. To understand the underlying mechanism, we increase the control parameter κ from zero to $\kappa = 0.58$ in steps of 0.2, and explore the behavior of $n(t)$ at each $\omega_c(\kappa)$.

To confirm the continuously varying critical exponent α, we perform the finite-size scaling for for $\kappa \in [0, 0.6]$ in steps of 0.1. Finite size scaling for each value of κ is

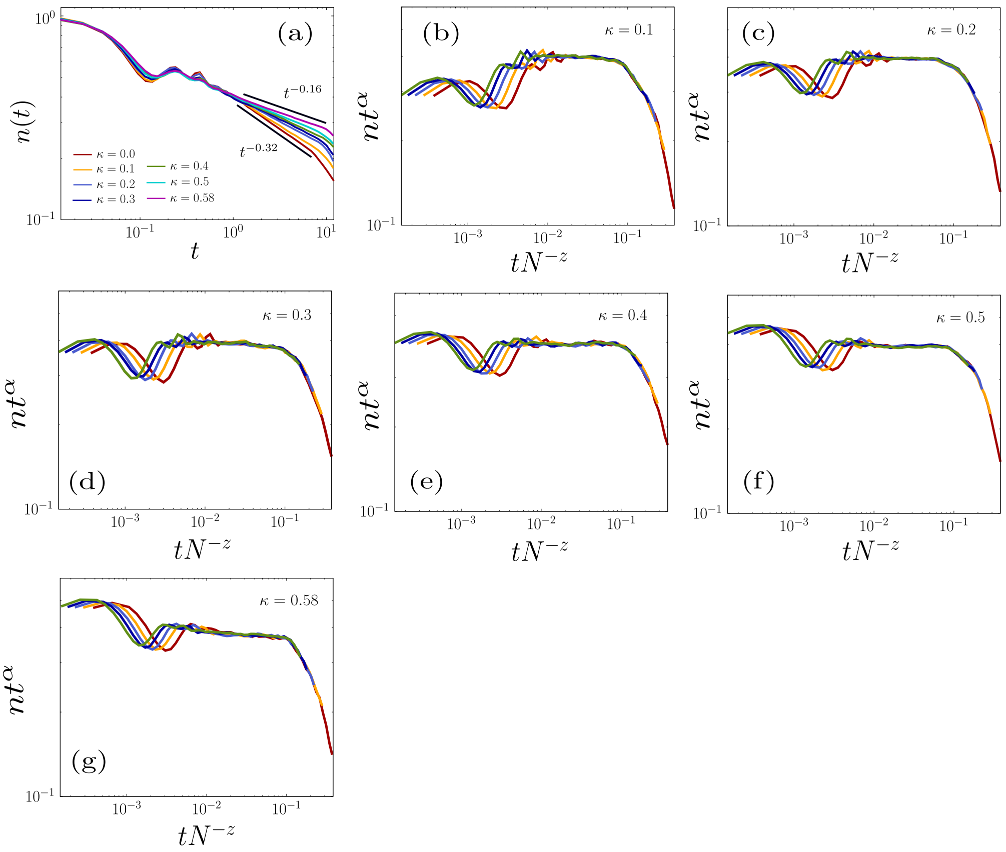

Figure 2.9: Estimates of the critical exponents of the one dimension quantum contact process from the single active initial state. (a) Plot of $n(t)$ as a function of t for different κ in the range $\kappa \in [0, 0.58]$ in steps of 0.1. The lower (upper) solid line is a guideline with slope -0.32 (-0.16). Scaling plot of $n(t)t^\alpha$ versus tN^{-z}. Incoherent control parameter are taken as $\kappa = 0.1$ for (b), $\kappa = 0.2$ for (c), $\kappa = 0.3$ for (d), $\kappa = 0.4$ for (e), $\kappa = 0.5$ for (f), and $\kappa = 0.58$ for (g). The units of t is given as $1/\gamma$.

κ	α	δ'	z	η	δ
0.0	0.32 ± 0.01	DP values			
0.1	0.28 ± 0.01				
0.2	0.24 ± 0.01				
0.3	0.22 ± 0.01				
0.4	0.20 ± 0.01				
0.5	0.18 ± 0.01				
≥ 0.58	DP values				

Table 2.2: Critical exponent α for the one dimensional quantum contact process for finite value of κ. DP values denotes the critical exponent values of the directed percolation universality.

shown in Fig. 2.9. We remark that the critical exponents starting from the single-seed initial state have the directed percolation universal class in this region.

We find that the exponent value of α decreases continuously from 0.32 for $\kappa = 0$ to $\alpha = 0.16$ for $\kappa = 0.58$. These results suggest that α is continuously varying as κ is increased and reaches the directed percolation value at $\kappa_* \approx 0.58$. When $\kappa = 0$ and all sites are active at $t = 0$, the wave functions of each site are overlapped and thus the coherence spreads over the entire system. This quantum effect is extremely dominant when $\kappa = 0$. As the parameter κ is increased, this long-range coherence is gradually reduced. This may be related to the behavior that occurs in the tricritical contact process with power-law interactions. Each active particle activates an inactive particle at distance r with rate $\kappa/r^{d+\sigma}$, where d is the spatial dimension and σ is a parameter. Then, the critical exponent value of α depends on the power σ in an appropriate range of σ. As σ is increased, the effect of long-range interactions becomes weaker and the exponent α becomes smaller [103]. This behavior is similar to the one of the exponent α.

We investigated the phase transitions arising in the one dimensional quantum contact process, as a prototypical example of dissipative phase transitions. The phase diagram was obtained as shown in Figure 2.1, in the parameter space spanned by κ

and ω, representing the degrees of the classical and quantum effects, respectively. The transition curve between absorbing and active phases is composed of two parts: the non-directed percolation region $[0, \kappa_*]$ and the directed percolation region $[\kappa_*, \kappa_c]$. In the non-directed percolation region, the critical exponent α, associated with the density of active sites $n(t)$ under the homogeneous initial state, decreases continuously as κ is increased. It is also interesting to note that the discontinuous transition near $\kappa = 0$ in the mean-field limit changes to the continuous transition with the continuously varying exponent in one dimension.

Chapter 3

Machine learning approach for non-linear dynamics systems

Incorporating machine learning approaches, recent progressive advances have been achieved in diverse fields of science and engineering. In particular, studies in recognizing phases and phase transition through machine learning methods became an central issue in physics. Applying to various systems including classical Ising model, XY model, and quantum systems, a well-trained machine used for classifying phases and finding the critical point exhibits successful performance [3, 9, 11, 12, 111, 123, 124]. In the meanwhile, as one of the machine learning model, reservoir computing approach [125–127] has demonstrated progresses in model-free prediction of the dynamical behavior and inferring unmeasured variables of chaotic systems [23–30]. As chaotic behaviors are observed in a variety of systems in nature such as cardiac cycle, neuroscience, climate and stock market, it is a subject of the utmost interest to explore the chaotic signals for predicting dynamical evolution of such systems.

Since such chaotic behaviors or signals of individuals represent the state of the system, it is also crucial to figure out underlying correlations and connections between them [128–130]. For example, in neurophysiology, studies for classifying and capturing the physiological events such as seizure, stroke or headache has been progressed by identifying the correlations between patterns of EEG signals, and machine learning approaches has obviously contributed in such works as well [19–22].

In this chapter, we focus on the Kuramoto model which has been widely dealt with as a mathematical model for the collective synchronization behavior [131, 132]. As the system exhibits the synchronization transition at the critical point for appropriate choice of natural frequencies, analytical and numerical analysis for its critical phenomena and dynamical behavior has been explored [133–139]. Despite the progress in describing the collective behavior of the Kuramoto oscillators, the machine learning study for this system has not yet been worked out thoroughly.

We here examine whether underlying parameters of the system can be deduced from the limited amount of information by introducing machine learning techniques. In the first part of this work, exploiting the machine learning to predict the coupling strength from the dynamics of the order parameter, we demonstrate that accurate value of coupling is obtained from the well-trained machine. Next, the snapshot of phases for all oscillators are also adopted to discriminate synchronized and asynchronized states and to determine the critical value of the Kuramoto system. As the learning the neural network with parameters of the Kuramoto model is achieved, we applied it to other model-free tasks. We predict the future evolution of the system by considering a situation where a detailed description for the dynamics of the system is unavailable or insufficient, but observational data of time evolution is given. Finally, underlying connections between oscillators are reconstructed from phase dynamics of all oscillators using the machine learning techniques.

3.1 The Kuramoto model

As one of the model for descibing the synchronization phenomena, the Kuramoto model consists of N globally coupled oscillators interacting with each other via nonlinear coupling, which is written as

$$\dot{\theta}_i = \omega_i + \frac{K}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i) \tag{3.1}$$

where K denotes the coupling strength and ω_i is the natural frequency of oscillator i that follows the distribution $g(\omega)$. The collective behavior of the system is quantified by the complex order parameter Z, which is defined as

$$Z = re^{i\psi} = \frac{1}{N}\sum_{j=1}^{N} e^{i\theta_j} \tag{3.2}$$

where ψ is the average phase and r indicates the phase coherence of oscillators and plays a role of the order parameter. As K increases, r becomes a nonzero value in the thermodynamic limit of $N \to \infty$ at the critical strength K_c, which implies the occurence of the phase synchronization. In this study, we consider the Kuramoto model with natural frequencies following the normal distribution,

$$g(\omega) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\omega^2}{2}}\,, \tag{3.3}$$

and for this set of natural frequencies, the second-order synchronization transition arises at the critical point $K_c = 2/[\pi g(0)] = \sqrt{8/\pi}$ in the limit $N \to \infty$ [131, 132]. For simplicity, we here used the normalized coupling strength $J \equiv K/K_c$ in the following, hence the normalized critical value is $J_c = K_c/K_c = 1$.

For a finite population of oscillators with frequencies randomly drawn from the Gaussian distribution, the result of critical exponents with

$$\beta = 1/2 \quad \text{and} \quad \bar{\nu} = \bar{\nu}' = 5/2 \tag{3.4}$$

is obtained using the finite size scaling analysis

$$r = N^{-\beta/\bar{\nu}} f(\epsilon N^{1/\bar{\nu}}) \tag{3.5}$$

where $\epsilon = |J - J_c|$ [140–143].

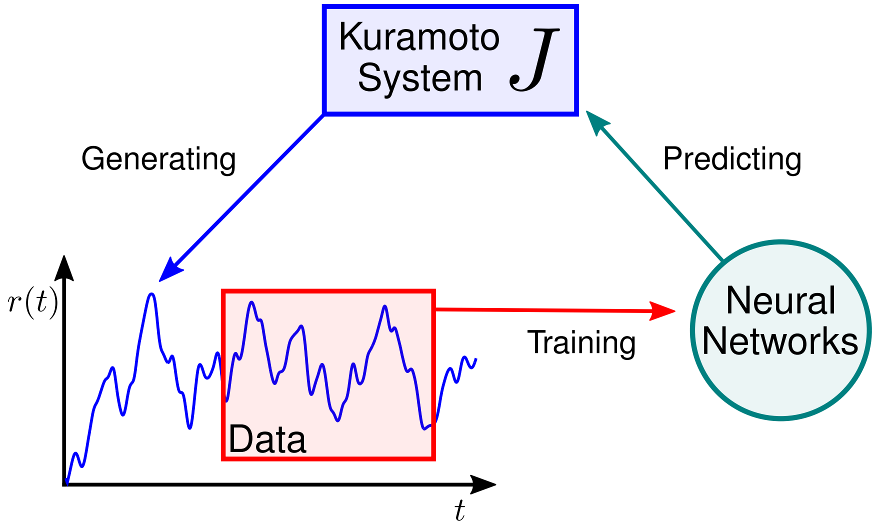

Figure 3.1: Schematic illustration of the process carried out for finding the coupling strength.

3.2 Finding the coupling strength

Assuming a situation where any concrete form of dynamics is unavailable, we investigate that how accurate values of underlying coupling strengths can be obtained by the neural network from the data for the dynamics of the order parameter, $r(t)$, only. As phases of oscillators and the corresponding order parameter proceed according to Eq. (3.1), $r(t)$ is obtained for given value of J. Once the generated time series of the order parameter for given J goes into the neural network as an input, learning process is carried out by a model as shown in Figure 3.1. Taking the trained neural network, we tested for other generated 10^4 sets of $r(t)$ with randomly chosen J in the range of $[0, 2]$.

Here, we produced configurations of $r(t)$ for the system of $N = 1000$ using the fourth-order Runge-Kutta method with a discrete time step $\delta t = 0.05$ up to a total of 2×10^3 time steps. The sets of natural frequencies, $\{\omega_i\}$, are selected randomly from the normal distribution given in Eq. (3.3) and initial phases, $\{\theta_i(t = 0)\}$, are chosen

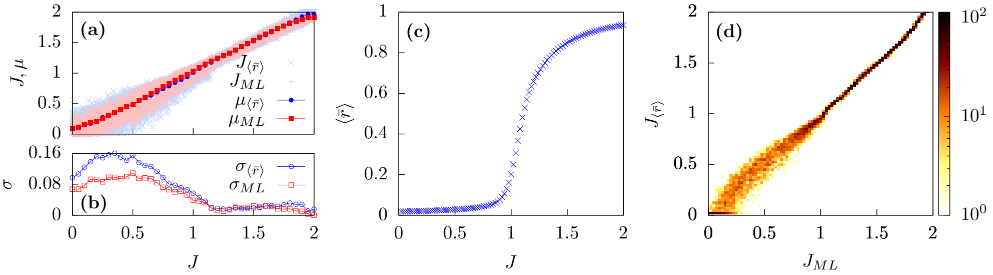

Figure 3.2: (a) The scatter plot of prediction for coupling strength using a recurrent neural network (light red) and using $\langle \bar{r} \rangle$ (light blue). The line of $y = x$ indicates the correct value of J. For both methods of the recurrent neural network (red) and $\langle \bar{r} \rangle$ (blue), the mean μ and the standard deviation σ are depicted in (a) and (b), respectively. (c) Ensemble and time averaged value of the order parameter, $\langle \bar{r} \rangle$, as a function of coupling J for generated sample data of $r(t)$. The system exhibits the synchronization transition near the critical value $J_c = 1$. (d) 2D histogram of J_{ML} and $J_{\langle \bar{r} \rangle}$ obtained for each given test datasets.

randomly from the range of $[0, 2\pi]$. We generate 2×10^4 training datasets of $r(t)$ for each value of $J \in [0.02, 2]$ with an interval of $\delta J = 0.02$.

As shown in Figure 3.2(a), output values, J_{ML}, using recurrent neural networks fits well with exact values of J in supercritical $(J > 1)$ and subcrtical $(J < 1)$ regions both. To evaluate how accurate values of J_{ML} are obtained, we calculated the Pearson correlation coefficient and the root mean square error, and obtained 0.9921 and 0.0727, respectively. By measuring the mean μ_{ML} and the standard deviation σ_{ML} with varying actual coupling strength J, one can confirm that μ_{ML} is close to J in both super and subcritical regions, while the difference in standard deviations between two regions appears as depicted in Figure 3.2(b). Although outputs are scattered broader in the region $J < 1$ than $J > 1$, the tendency following the correct value of J provides the fact that the neural network can distinguish configurations of $r(t)$ and identify corresponding coupling strengths.

As the result obtained can lead one to wonder what information is extracted from the data by the neural network during training, we also compare the result with the estimation of J through the calculation for values of the order parameter to support the quality of the result and investigate the learning of the neural network. Here, we used time averaged value of the order parameter, $\bar{r}$, over final 1000 steps of each training datasets and calculated mean, $\langle\bar{r}\rangle$, over all configurations with same value of J. As $\langle\bar{r}\rangle$ increases monotonically with J as shown in Figure 3.2(c), one can estimate the coupling strength for each test datasets by obtaining $\bar{r}$ and comparing with Figure 3.2(c). From results of this estimation, denoted as $J_{\langle\bar{r}\rangle}$, their mean $\mu_{\langle\bar{r}\rangle}$ and standard deviation $\sigma_{\langle\bar{r}\rangle}$ are plotted in Figure 3.2(a) and (b) for comparison with μ_{ML} and σ_{ML}. One can find that $J_{\langle\bar{r}\rangle}$ is distributed in broader range than J_{ML} in subcritical region as the Pearson correlation and the root mean square error with 0.9870 and 0.0945 are obtained respectively for the case of $J_{\langle\bar{r}\rangle}$. Then, the larger value of $\sigma_{\langle\bar{r}\rangle}$ than σ_{ML} in $J < 1$ region demonstrates the better quality of the result achieved by the machine learning.

In addition, narrow deviation in supercritical region is observed for the correlation between J_{ML} and $J_{\langle\bar{r}\rangle}$ as depicted in Figure 3.2(d) while the deviation in subcritical region is broader. This implies that the information of the time-averaged value is dominant for the neural network to evaluate the coupling strength in supercritical region, while it can evaluate through acquisition of more information than the time-averaged value from the data of $r(t)$ in subcritical region, as smaller deviations for the machine learning case are shown in Figure 3.2(b).

3.3 Finding the synchronized state

As the machine learning of thermodynamical phase transitions for diverse systems have been paid much attention in previous studies [3, 9, 11, 12, 123, 124], we also performed another study of machine learning model for the phase transition of the Kuramoto system. Here, taking the neural network, we classify phase snapshots of all oscillators in the steady state into subcritical and supercritical regimes and eventually find the critical point of the system.

Again, the fourth-order Runge-Kutta method with a time step of $\delta t = 0.01$ is

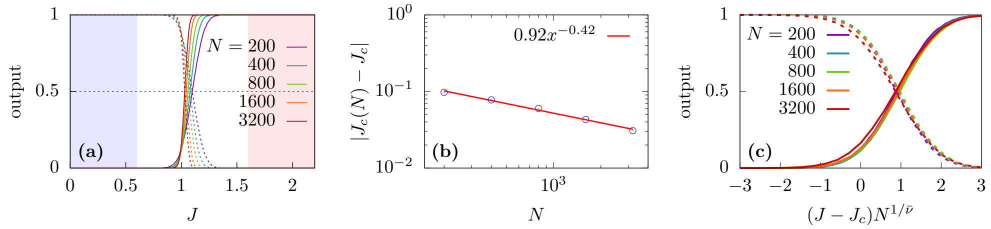

Figure 3.3: (a) Outputs of the neural network trained using data of phase snapshots as a function of J with system size of $N = 200, 400, 800, 1600$ and 3200. The crossing point of two output lines indicates $J_c(N)$ for given N. (b) Behavior of $|J_c(N) - J_c|$ with increasing N. The straught line is the fitting line with slope of -0.42. (c) Scaling plot for output lines against $(J - J_c)N^{1/\bar{\nu}}$ with $1/\bar{\nu} = 0.42$.

adopted for generating 2×10^4 datasets of $\{\theta_i\}$ for each values of system size N and $J \in [0.01, 2.20]$ with $\delta J = 0.01$. As in the previous section, for each configuration, natural frequencies and initial phases are selected randomly from the normal distribution and the uniform distribution, respectively. To avoid any transient behavior, we collect snapshots of phases, $\{\theta_i\}$, after the first 10^6 steps. For training datasets, each snapshots are labelled through one-hot encoding where configurations obtained in supercritical region of $J \in [0.01, 0.6]$ are encoded as $(0, 1)$ and ones in supercritical region of $J \in [1.6, 2.2]$ are encoded as $(1, 0)$. Moreover, we used data preprocessing by taking $\cos\theta_i$ and $\sin\theta_i$ for each phases as an input, due to the cyclic feature of θ_i which contains the 2π periodicity.

Constructing the fully-connected neural network, we train it with labelled snapshots of N phases, $\{\theta_i\}$. When the network is optimized after the training, generated snapshots in whole region of J goes in as an input at the test stage.

The trained neural network produces two outputs representing predictabilities for the system to be in subcritical and supercritical region, respectively, as shown in Figure 3.3(a). And the crossing point of two output lines indicates a transition point $J_c(N)$ for given system size N. Since $J_c(N)$ approaches to the critical point $J_c = 1$ as N is increased, we can determine the value of $\bar{\nu}$ using the behavior $|J_c(N) - J_c| \sim N^{-1/\bar{\nu}}$. Figure 3.3(b) exhibits the finite size scaling of the $J_c(N)$ with the exponent of $1/\bar{\nu} =$

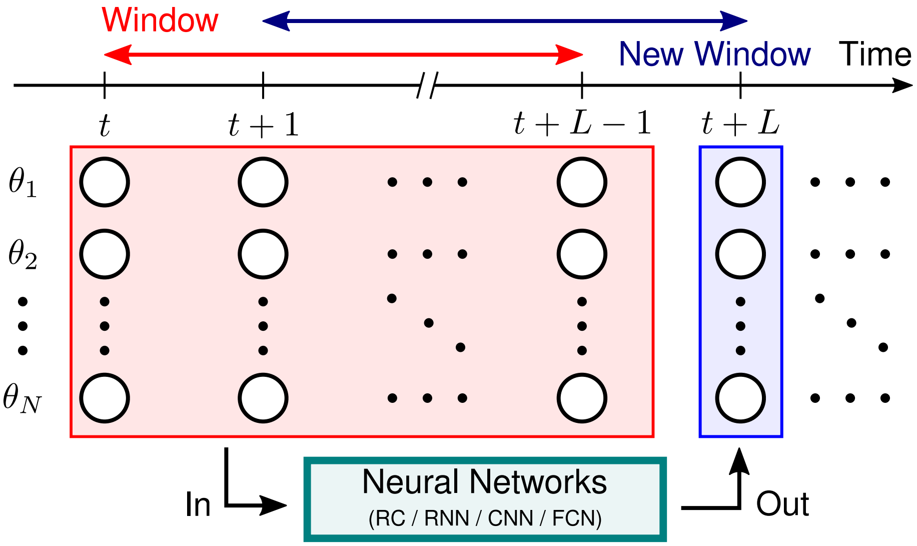

Figure 3.4: Schematic illustration of learning processes by artificial neural networks for prediction of phase dynamics. Using feedback process for given length of the time window, L, inputs for all models are determined from the data of phase dynamics $\{\theta_i(t)\}$ for all oscillators.

0.39 which is consistent with the exponent obtained in Ref. [140–143]. Using Eq. (3.5) and the exponent $1/\bar{\nu} = 0.39$ calculated, output lines for different sizes are collapsed as depicted in Figure 3.3(c).

In terms of phase transition and critical phenomena, machine learning approaches for classifying phases and finding the critical point have been reported in classical spin models, closed and open quantum models so far [9,11,111]. We here observe that such studies on the nonlinear dynamical system are also achieved where the synchronization transition is exhibited.

3.4 Prediction of the phase dynamics

Since the sine function in the coupling term of Eq. (3.1) induces nonlinear/chaotic behavior of oscillators, the system has a sensitive dependence on initial conditions. With the outstanding progress on the model-free prediction of chaotic dynamical systems [26–30], widespread application of the Kuramoto model provide sufficient motivation for the study of predicting the dynamics of each oscillators in the Kuramoto model. In particular, reservoir computer approach has recently been adopted for various studies of low-dimensional nonlinear systems due to its simplicity and efficiency. In addition to the reservoir computers, we here used artificial neural networks, including fully-connected neural networks, convolutional neural networks, and recurrent neural networks for prediction of the Kuramoto system.

As an input for this study, time series of phase, $\theta_i(t)$, for each oscillators are generated. We implemented the fourth-order Runge-Kutta method with a discrete time step $\delta t = 0.005$ up to a total of 1.95×10^5 time steps for this generation of datasets. To examine predictability for behaviors of all oscillators, for reservoir computer, we take 9×10^4 steps of $\{\cos\theta_i(t), \sin\theta_i(t)\}$ as washout period and subsequent 10^5 steps as an training data to produce phases for last 5×10^3 time steps by feeding the output data back to the reservoir in turn. For other models, taking 1.9×10^5 steps for training datasets, subsequent 5×10^3 time steps of phases are produced as an output by the neural network to compare with the exact dynamics of $\theta_i(t)$. The detailed description for these methods is illustrated in Figure 3.4. For reservoir computer, we set time length $L = 1$ to predict the henceforth phase dynamics, while $L = 200$ is given for inputs of the other models.

Figure 3.5 shows the prediction for phase dynamics using various machine learning models. As shown in Figure 3.5(b), reservoir computer produces accurate predicted data for observed time steps. And one can predict the phase dynamics for about 2000 and 3000 steps using a recurrent neural network, a convolutional neural networks, and a fully-connected neural network, as depicted in Figure 3.5(c), (d), and (e). Although the Kuramoto model with Eq. (3.1) contains nonlinearity, machine learning approaches can be applied to learn the behavior of the phase dynamics and predict future behaviors.

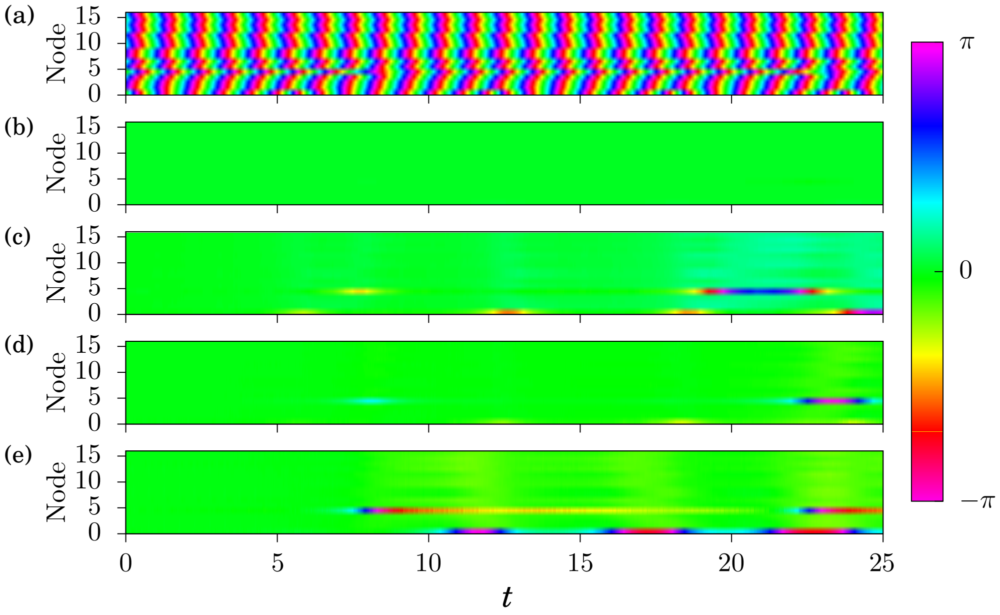

Figure 3.5: Prediction for the phase dynamics of the Kuramoto oscillators using several machine learning methods. (a) The actual evolution of $\{\theta_i(t)\}$. The difference between the actual data and the predicted solution obtained using (b) a reservoir computer, (c) a recurrent neural network, (d) a convolutional neural network, and (e) a fully-connected neural network.

3.5 Reconstructing the network structure

Identifying the network topology can be one of the main problem for predicting the behavior of the system and understanding properties of individuals or implicit mechanisms in various systems such as neuronal connections in the brain and epidemics in social networks. Since it is impossible to directly determine the neuronal structure, for example, developments for measuring the time evolution of nodes has been studied to indirectly recover the structure. In the Kuramoto model, it has been studied for the relationship between the modular structure and the synchronization dynamics by observing the correlation between pairs of oscillators which is ordered in a hierarchical way [144]. Applying to more general networks than the modular network, we here assume the situation where the connections of network are not provided but only individual patterns which is produced through inherent interactions between them. As it is of great potential application, machine learning approach is performed to detect whole network topology by comprehending interactions between signals.

For this purpose, we produce 10^6 training datasets for the coupled oscillators on random networks and calculate dynamics of each oscillators following the equation

$$\dot{\theta}_i = \omega_i + K \sum_{j=1}^{N} A_{ij} \sin(\theta_j - \theta_i) \tag{3.6}$$

where A_{ij} is the adjacency matrix of the given network. Here we adopt $N = 29$ and the natural frequency set, $\{\omega_i\}$, selected regularly from the normal distribution, given in Eq. (3.3). And the given set of $\{\omega_i\}$ are shuffled and assigned to the 29 nodes on the visual cortex network [145], and calcuation for Eq. (3.6) is progressed to generate test datasets. As in the previous section, implementing the fourth-order Runge-Kutta method with a time step $\delta t = 0.05$ up to a total of 200 steps, sets of time series of phases, $\{\theta_i(t)\}$, are generated for input. And we set the initial phase as $\theta_i(0) = 0$ for all i.

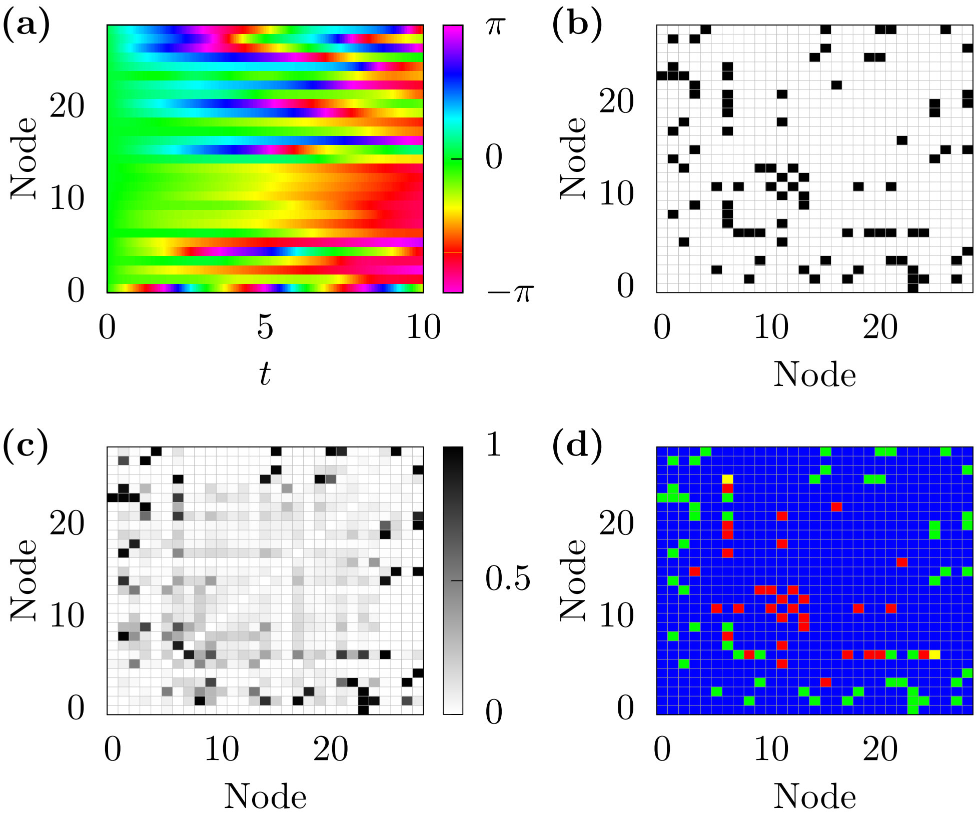

Figure 3.6: Reconstruction of the visual cortex network with $N = 29$. (a) An input dataset of the actual phase evolution, $\{\theta_i(t)\}$, for one of the sample set of shuffled natural frequencies. (b) The actual adjacency matrix of the visual cortex network. Black square indicates the link existing on the network. (c) Outputs obtained through recurrent neural network. Output elements are in a range of $[0, 1]$. (d) Comparison between the actual adjacency matrix and the rounded values of obtained output elements in Panel (c). Blue (green) square denotes the case when the corresponding element in the actual network is zero (one) and the one in the rounded output element is also zero (one). Red (yellow) square denotes the case when the corresponding element in the actual network is zero (one) and the one in the rounded output element is one (zero).

Figure 3.6(a) illustrates one of the pattern samples used for testing the neural network. The actual given network topology for this pattern is depicted as the adjacency matrix shown in Figure 3.6(b). As the trained neural network produces real numbers in the range of $[0, 1]$ (see Figure 3.6(c)) which implies the probabilities for each links or elements in the adjacency matrix to exist, these outputs are rounded off to 0 or 1 as in Figure 3.6(d), to compare with the actual matrix. Although phase dynamics of all osillators exhibit irregular and complex behaviors, the prediction for the underlying network topology is successfully achieved by the neural network as shown in Figure 3.6(b) and (d). We obtained the accuracy of 93.0% for all elements in adjacency matrices of total 10^3 samples.

3.6 Summary

To summarize, we have demonstrated several machine learning tasks for the Kuramoto model using various machine learning approaches. In spite of abundant properties of the system such as phase transition and chaotic behaviors, little attention has been paid to machine learning approaches for the Kuramoto system. As the strength of interaction affects the dynamics of the system, the trained machine that observes the behavior of the order parameter to evaluate the coupling strength is achieved. In subcritical region, especially, the neural network displays noteworthy performance by overcoming the limit of the distinguishability using statistical figures. In addition, we observe that a set of phases in the steady state can play a role for discriminating states and determining the critical behavior of the system.

Despite of the nonlinearity of the system, successful prediction for the future behavior of the Kuramoto system is achieved by employing machine learning approaches. Furthermore, training the patterns of individuals on real brain network, underlying connections between them can be identified by the well-trained machine.

As great performance is demonstrated for the Kuramoto model, which is a simple and basic model for the synchronization phenomena, such machine learning methods have extensive applicability to other nonlinear models or systems in nature. Consequently, we believe that our work may shed light on the further studies of machine learning on nonlinear and chaotic systems.

Chapter 4

Conclusion

With growing interest in the machine learning, recent works on physical systems has demonstrated successful progresses by adopting the machine learning approaches for tasks of classification and generation.

This study analyzed the quantum phase transition in a quantum contact process through machine learning approaches based on the artificial neural network and discovered an open quantum system's critical phenomenon different from a classical system. Also, we analyzed the critical phenomena of the synchronization transition of the Kuramoto model with machine learning approaches and predicted the dynamic behavior of the model. It showed that the machine learning approached is an alternative framework for numerical analysis for synchronization phenomena.

Phase transitions in dissipative quantum systems are intriguing, because they are induced by the competition between coherent quantum and incoherent classical fluctuations. We investigated the interplay of quantum and classical absorbing phase transitions arising in quantum contact process in one dimension. The Lindblad equation contains two model parameters ω and κ that adjust the degrees of quantum and classical effects, respectively. We found a characteristic value κ_* that separate a novel class of the quantum contact process from the directed percolation class. In the region $[0, \kappa_*]$, the exponent α associated with the density of active sites is continuously varying, whereas for $\kappa > \kappa_*$. α has the directed percolation value. We used the neural network machine learning technique to identify a transition point $\omega_c(\kappa)$ and determine the correlation length exponent. By performing extensive quantum jump Monte Carlo simulations at $\omega_c(\kappa)$, we successfully determined all the other critical exponents of the one dimensional quantum contact process. Finally, we found that the continuously varying exponent behaves similarly to the one in the tricritical contact process with long-range interactions.

We performed various machine learning approaches for the Kuramoto system. As the system displays rich properties such as synchronization transition and nonlinearity with varying parameters, we applied machine learning for finding the value of the coupling strength and the critical value. Considering the finite size scaling, we confirmed that results follow the critical behavior of the Kuramoto system. By focusing on the phase dynamics of all oscillators, we applied the performance of the neural network for predicting future behaviors of all oscillators and detecting underlying network topology. As the Kuramoto model offers support for the application on real-world systems exhibiting synchronization phenomena or nonlinear behaviors, our work has potential for utilizing the machine learning approaches for such systems.

As the quantum contact process is a typical model of the open quantum system, this study shows that the machine learning approach can be applied to the open quantum system beyond the classical and closed quantum systems. Though this study focuses on the Kuramoto model as a typical nonlinear dynamics model exhibiting synchronization transition, we expect that the artificial neural networks will be a significant breakthrough in follow-up studies to predict the dynamical behavior of the chaotic system and to illuminate synchronization phenomena.

Appendices

Appendix A

Feed-forward neural networks

This appendix introduces a feed-forward neural network (FNN), the basic model of artificial neural networks, but is widely used in supervised learning. The main feature of FNN is forwarding propagation and backpropagation. In forwarding propagation, the neurons' state is updated sequentially from the input layer to the output layer. Oppositely, in backpropagation, the model parameters between the layers are updated sequentially from the output layer to the input layer [146].

A.1 Forwarding propagation

A fully-connected neural network (FCN) is a basic FNN model that consisted of only fully-connected layers. Figure A.1 shows a graphical representation of the FCN consisting of k fully-connected layers. We assume that the input and the neurons' state of the ith layer are m_i-dimensional vector $\boldsymbol{x}^{(i)}$ and n_i-dimensional vector $\boldsymbol{y}^{(i)}$. Also, the weight matrix and the bias vector are denoted as $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$, respectively.

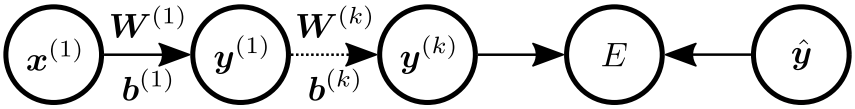

Figure A.1: Graphical representation of the forwarding propagation of FCN. The circle represents a fully-connected layer, including the neurons' state. The arrow represents the weight matrix and the bias vector between layers. Forwarding propagation progresses following the arrow's direction, and the layer passes its neurons' state to the next layer.

An input layer is the first layer that does not have weight matrix and bias vector and its states $\boldsymbol{x}^{(1)}$ is an input of the FNN: $\boldsymbol{x}^{(1)} = \boldsymbol{x}$. Here, we regard the data as vectors $\boldsymbol{x}$ without losing generality because the input layer fully connects with the first hidden layer. An output layer is the last layer and its states $\boldsymbol{y}^{(k)}$ is an output of the FNN. The rest of the layers except the input layer and the output layer are called the hidden layers.

The main structure of the FNN is the sequential connections from the input layer to the output layer through several hidden layers. The FNN may have multiple branches, but branches should merge in one output layer. Additionally, a loop is not allowed.

The prototype that has no hidden layer is the brain model, so-called *perceptrons* [147, 148]. On the other hand, a deep neural network has multiple hidden layers ($k \geq 2$), and the method for the deep neural network are called *deep learning* [110].

Generally, The FNN employs another layer type instead of a fully-connected layer. For example, a convolutional neural network is the FNN built of convolutional layers, pooling layers, and fully-connected layers [44].

In forwarding propagation, The current ith layer's input is the previous $(i-1)$th layer's output as follows

$$\boldsymbol{x}^{(i)} = \boldsymbol{y}^{(i-1)}. \tag{A.1}$$

We obtain the output for a given input, updating all layers' states in the FNN sequentially as above.

A.2 Backpropagation

Training data $\boldsymbol{x}$ of supervised learning has a label by a correct answer $\hat{\boldsymbol{y}}$. Supervised learning aims to train the model and obtain a result close to the correct answer. In Section 1.2.1, the loss function E is set by the explicit analytic form with the correct answer and the model's output: $E = E(\boldsymbol{y}^{(k)}, \hat{\boldsymbol{y}})$. Again, the learning is to adjust model

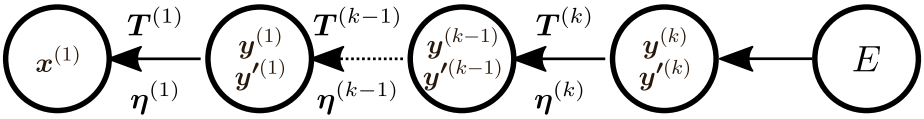

Figure A.2: Graphical representation of the backpropagation of FCN. The structure of FNN is the same as it of the FNN in Figure A.1. However, the backpropagation following the direction of the arrow is reversed in Figure A.1.

parameters, including weight matrix and bias vector, so that the average value of the loss function is minimized for the training dataset.

Since the loss function is a convex function near the correct answers for a given input, tuning the model parameter to the opposite direction of the gradient vector minimizes the loss function's value. This method is called the gradient descent method (GDM), or the steepest descent method:

$$\begin{aligned} \theta_{new} - \theta &= -\alpha \partial_\theta E \quad (\theta \in \boldsymbol{\theta}), \\ \boldsymbol{\theta}_{new} - \boldsymbol{\theta} &= -\alpha \nabla_{\boldsymbol{\theta}} E \quad (\boldsymbol{\theta} \in \boldsymbol{W}, \boldsymbol{b}), \end{aligned} \tag{A.2}$$

where θ is a model parameter, $\boldsymbol{W} = \{\boldsymbol{W}^{(1)}, \cdots, \boldsymbol{W}^{(k)}\}$, and $\boldsymbol{b} = \{\boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(k)}\}$, and α is a learning parameter. The learning rate is a hyperparameter in the model tuning the training speed and is lower than one, for example, 10^{-3} and 10^{-4}.

As changing data, adjusting the model parameter by GDM is repeated until the average value of the loss function reaches the minimum point (threshold). As above, tuning the model parameter for each data is called the method of stochastic gradient descent [149, 150], but tuning the model parameter for grouped data is the mini-batch algorithm: Refer Appendix C.2.

The core of GDM is calculating the gradient of the loss function for each model parameter. Although each model parameter's gradient vector is calculated in parallel, it is too expensive to compute them. In the mind of dynamic programming, the gradient is calculated sequentially from the output layer to the input layer, such as in Figure A.2, to avoid repeated calculations. This algorithm is *backpropagation*.

Here, we derive the backpropagation algorithm for the FCN in Figure A.2. Note that the inactivated neurons' state of the ith fully-connected layers is

$$\boldsymbol{z}^{(i)} = \boldsymbol{W}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{b}^{(i)}, \tag{A.3}$$

and

$$y_j^{\prime(i)} = f^{\prime(i)}\left(z_j^{(i)}\right), \quad j \in \mathbb{Z}_{n_i}, \tag{A.4}$$

where $f^{\prime(i)}(x)$ is the derivative function of the ith layer's activation function $f^{(i)}(x)$ and $z_j^{(i)}$ is the $(j+1)$th element of the $\boldsymbol{z}^{(i)}$. We denote ith layer's an *error* vector as $\boldsymbol{\eta}^{(i)}$, which is the differential change of the loss function for the bias vector on the ith layer. The output layer's error is

$$\boldsymbol{\eta}^{(k)} \equiv \nabla_{\boldsymbol{b}^{(k)}} E = \boldsymbol{y}^{\prime(k)} \circ \nabla_{\boldsymbol{y}^{(k)}} E. \tag{A.5}$$

where and $\circ$ is a coefficient-wise product (the Hadamard product).

The differential chain rule induces the relation of the errors between two neighboring layers:

$$\boldsymbol{\eta}^{(i)} = \boldsymbol{y}^{\prime(i)} \circ \boldsymbol{T}^{(i+1)}\boldsymbol{\eta}^{(i+1)} \quad (1 \leq i < k), \tag{A.6}$$

where $\boldsymbol{T}^{(i)}$ is the transpose of $\boldsymbol{W}^{(i)}$; $\boldsymbol{T}^{(i)} = \left(\boldsymbol{W}^{(i)}\right)^T$. For each layer, the differential change of the loss function for the weight matrix is obtained by the error vector as follows

$$\nabla_{\boldsymbol{W}^{(i)}} E = \boldsymbol{\eta}^{(i)} \otimes \boldsymbol{x}^{(i)} \quad (1 \leq i \leq k), \tag{A.7}$$

where $\otimes$ is a tensor product.

Finally, we calculate the differential change of all layers' model parameters sequentially from the output layer to the input layer, such as in Figure A.2. Furthermore, we update the model parameters following the GDM.

It is practically useful to keep the inactivated neurons' state $\boldsymbol{z}^{(i)}$ for each layer when implementing a backpropagation algorithm in a programming language. Otherwise, keeping only the neurons' state $\boldsymbol{y}^{(i)}$, the forwarding propagation is performed in backpropagation to obtain $\boldsymbol{y'}^{(i)}$ is required to calculate errors $\boldsymbol{\eta}^{(i)}$. To avoid redundant calculations, we store the inactivated neurons' state calculated in forwarding propagation, and we reuse $\boldsymbol{z}^{(i)}$ to calculate errors in backpropagation. This technique considerably improves calculation speed for deep neural networks, which has many hidden layers.

Appendix B

Recurrent neural networks

As deep learning model for dealing with the sequential data, recurrent neural network has recurrent layers with cyclic connection topology, which leads to distinction from the fully-connected neural networks. This structure of recurrent neural network resembles biological brain modules which also exhibits recurrent connection pathways.

B.1 Reservoir computer

Reservoir computing [126, 127], a class of recurrent neural networks, is composed of an input layer, an output layer and a reservoir layer which connects between input and output layers. And neurons in the reservoir layer has internal links including the self-loop as in the recurrent layer.

In this section, as the input vector $\boldsymbol{u}(t)$ goes in, the state vector $\boldsymbol{r}$ is updated ac-

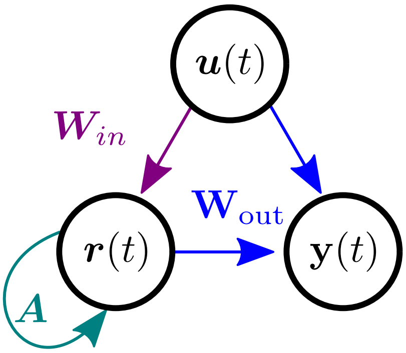

Figure B.1: A graphical representation of a reservoir computer.

cording to the equation

$$\boldsymbol{r}(t+1) = (1-\lambda)\boldsymbol{r}(t) + \lambda\tanh(\boldsymbol{A}\boldsymbol{r}(t) + \boldsymbol{W}_{in}\boldsymbol{u}(t) + \boldsymbol{b}_{in}) \tag{B.1}$$

where $\boldsymbol{A}$ is the weighted adjacency matrix of the reservoir network, $\boldsymbol{W}_{\text{in}}$ is the random matrix which maps an input vector $\boldsymbol{u}(t)$ into a state vector $\boldsymbol{r}(t)$, λ is the leaking rate, and $\boldsymbol{b}_{in}$ is the bias. And the output vector $\boldsymbol{y}(t)$ is determined by a linear function

$$\boldsymbol{y}(t) = \mathbf{W}_{\text{out}}\boldsymbol{x}(t), \quad \boldsymbol{x}^T(t) = \left[1|\boldsymbol{u}^T(t)|\boldsymbol{r}^T(t)\right] \tag{B.2}$$

where $\boldsymbol{W}_{\text{out}}$ denotes the output matrix which maps a reservoir state into a output vector.

In the case of reservoir cumputering, when the updated state vector $\boldsymbol{r}$ of the reservoir is given through Eq. (B.1), the output weights $\boldsymbol{W}_{\text{out}}$ are determined by the equation [28, 127]

$$\boldsymbol{W}_{\text{out}} = \hat{\boldsymbol{Y}}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top + \gamma\boldsymbol{I})^{-1} \tag{B.3}$$

where λ is the ridge regularization parameter and $\mathbb{I}$ is an identity matrix. And $\boldsymbol{X}$ and $\hat{\boldsymbol{Y}}$ are the collecting matrix of state vector $\boldsymbol{x}$ and the desired output vector $\hat{\boldsymbol{y}}(t)$ in the training process, respectively.

$$\boldsymbol{X} = [\boldsymbol{x}(1)|\cdots|\boldsymbol{x}(t_{max})], \quad \hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}(1)|\cdots|\hat{\boldsymbol{y}}(t_{max})] \tag{B.4}$$

While the classical recurrent neural network adopts the back-propagation through time which is based on a gradient descent method for recurrent layers, for reservoir computer, $\boldsymbol{A}$, $\boldsymbol{W}_{\text{in}}$, and $\boldsymbol{b}_{\text{in}}$ are randomly created and unchanged during training, and only the output weights $\boldsymbol{W}_{\text{out}}$ are computed using the Moore–Penrose pseudo inverse.

Appendix C

Techniques for deep learning

In deep learning, the mains are which layer to employ to build an artificial neural network and connect between layers to implement the structure. Whereas artificial neural networks' architecture depends on the problem, algorithms for optimizing artificial neural networks continue to be suggested and developed. This appendix includes the know-how to manage datasets for preparing optimization, advanced algorithms based on the gradient descent method, and other technical remarks.

C.1 Data management

Splitting dataset

Artificial neural networks have two types of parameters. One is a model parameter that is components of the model, such as weight matrix, bias vector. The other is a hyperparameter that is a parameter related to the structure of the model such as the number of layer and neuron, type of layer, and connection between layers and is a parameter related to the learning algorithm such as loss function, learning rate, regularization rate, momentum rate. In many cases, the designer tunes hyperparameters outside the model, and the model parameter is internally determined by the learning method after the hyperparameters are fixed.

Because the artificial neural network parameters are divided into model parameters and hyperparameters, we need to separate the total dataset into two d for each parameter. So, we split the dataset into the training dataset for adjusting the model parameters and the validation dataset for tuning the hyperparameters.

The process of consuming the training dataset and the validation dataset is as follows. First, we make a baseline model for arbitrary hyperparameters, and we optimize a baseline model using a training dataset, then we obtain the trained baseline model. We measure the performance of a trained baseline model for the validation dataset. The model performance is usually quantified with the average loss or average accuracy for a given dataset. Of course, the trained baseline model's model parameter is not changed when the trained baseline model deals with the validation dataset. Then, we make a new model perturbing a baseline model's hyperparameters, the model learns the training dataset, and we evaluate the model performance for the validation dataset. We compare the model performance with the baseline model and the previous models. We repeat the above process until we find the model showing the best performance for the validation dataset.

What ratio should we divide a given dataset into a training dataset and validation dataset? In various tutorials, it is recommended to divide the training dataset and validation dataset among the total datasets given in the design into a specific ratio: 10:1, 20:1. However, the above ratios of the two datasets are just a guideline, not an unconditional one. Instead, a ratio between the two datasets depends on the data's complexity and the entire dataset's size. The main factor in dividing the dataset is the size of the validation dataset. The larger the validation dataset, we obtain the model's average values with the smaller standard errors. Furthermore, we estimate precisely the performance of the model when the model is verified. Besides, there is a limit to enhancing a fixed model's performance by increasing the training dataset's size. Therefore, the designer retains the minimum number of validation datasets to ensure confidence intervals for the model's performance among the entire dataset, then allocates the remaining dataset as the training dataset.

C.2 Advanced optimization

Mini-batch algorithm

The gradient descent method is a numerical algorithm to find a global minimum on the parameter space, adjusting model parameters to reduce the loss function's value infinitesimally. Appendix A.2 establishes the model parameter change for single data using the gradient descent method following Eq. (A.2). How we perform the gradient descent method for a given dataset $\boldsymbol{X}$ consisting of N data?

We first suggest calculating the model parameter change for each data and updating the model parameters as an average of them. We revise Eq.(A.2) and obtain the model parameter change for the dataset as follows

$$\theta_{new} - \theta = -\alpha \mathbb{E}_{\boldsymbol{x}} \left[\partial_\theta E \right], \quad \boldsymbol{x} \in \boldsymbol{X}, \tag{C.1}$$

where θ is a model parameter, α is a learning rate, and E is the loss function. This method is an ordinary gradient descent method.

A model optimized with the ordinary gradient descent method is affected by the model parameters' initial condition. The crucial problem is that the model parameter is trapped in the local minimum on the parameter space, leading to a failure to optimize it.

To solve the problem of being trapped in the local minimum, we introduce the stochastic process into the ordinary gradient descent method. We select one data from the dataset randomly, calculate the model parameter change, and update it. This method is the method of stochastic gradient descent (SGD) or online learning.

The number of data used to update model parameters divided by the number of total data in the dataset indicates how much the model parameters are updated where the unit is an epoch. When the model is optimized with the SGD for one epoch, the model parameters are updated the number of the total data N times.

Here, we encounter a technical problem when implementing artificial neural networks on the Turing machine. The artificial neural network's model parameters are maintained through computer storage devices such as memory or disk drives, so it is necessary to physically update model parameters. Moreover, the physical time increase with the number of model parameters. Since artificial neural networks consist of many model parameters, it is very time-consuming to update the model parameters for data by data.

To resolve this problem, we expand the SGD to select multiple data from datasets and update model parameters all at once. This algorithm is called a mini-batch algorithm. We choose data B times randomly from the dataset, where the group of selected data is called a *batch* $\boldsymbol{X}_{\text{Batch}}$.

$$\boldsymbol{X}_{\text{Batch}} = \{\boldsymbol{x}_0, \cdots, \boldsymbol{x}_{B-1}\} \subset \boldsymbol{X}. \tag{C.2}$$

We calculate the model parameter change for each data in a batch and update the model parameters as an average of them. We revise Eq. (C.1) for a batch as follows

$$\theta_{new} - \theta = -\alpha \mathbb{E}_{\boldsymbol{x}}\left[\partial_\theta E\right], \quad \boldsymbol{x} \in \boldsymbol{X}_{\text{Batch}}. \tag{C.3}$$

If the batch size is one, the mini-batch learning reverts to online learning. If the batch size is the dataset's size, the mini-batch learning reverts to the ordinary gradient descent method, sometimes called full-batch learning.

When the model is optimized with mini-batch learning for one epoch, the model parameter is updated N/B times, Then computational time to update the model parameter is reduced by B times compared to online learning.

Another advantage of the mini-batch algorithm is calculating the model parameter change for each data in parallel. In other words, we employ efficient methods of parallel computing, such as multiprocessing and distributed computing, to implement the mini-batch algorithm.

The weakness of the mini-batch algorithm is difficult to find a suitable batch size. Many tutorials often set the batch size as a power of two, such as 16, 32, and 64, which is the numbers usually related to the method of parallel computing. Increasing the batch size decreases the calculation time, but it needs a better computer specification to calculate the model parameter change for the batch, and it may induce the cumulative error. Even if the model and optimization methods are the same, the appropriate batch size may differ depending on the computer which performs the calculation.

Accelerating the gradient descent method

The momentum method [151, 152] is an algorithm modifying the gradient descent method to accelerate convergence and reduce iterations. Let us introduce a momentum m_i for each model parameter θ_i. Before updating model parameters, we first renew momentums with the loss function's gradient for the model parameters, as follows

$$\boldsymbol{m}_{new} = \mu \boldsymbol{m} + \nabla_{\boldsymbol{\theta}} E, \tag{C.4}$$

where μ denotes the momentum rate determining how long the momentum is maintained and usually is less than one, and E denotes the loss function. We set the initial value of the momentums zero. Then, we modify the updating rule of the gradient descent method as above Eq. (A.2) to

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \alpha \boldsymbol{m}_{new}, \tag{C.5}$$

where α is the learning rate. If the momentum rate is zero, the momentum method retrieves the original gradient descent method. Nesterov's method [153] is the variation of the momentum method.

Another algorithm for accelerating the gradient descent method is the adaptive method. The adaptive method normalizes the change of model parameters for each updating step. There are various adaptive methods such as AdaGrad [154], RMSprop [121], and Adam [120,155], depending on how to normalize the changes. Here, we introduce RMSprop. Similar to the momentum method, we first consider history n_i for each model parameter θ_i. Before updating the model parameters, we renew the history with the square of the loss function's gradient for the model parameter, as follows

$$\boldsymbol{n}_{new} = \nu \boldsymbol{n} + (1-\nu)\boldsymbol{g}^2, \quad \boldsymbol{g} = \nabla_{\boldsymbol{\theta}} E, \quad \left(\boldsymbol{g}^2\right)_i = (\boldsymbol{g}_i)^2 \tag{C.6}$$

where ν denotes the discounting factor ($0 < \nu < 1$). We set the initial value of the histories zero. Then, we modify the updating rule of the gradient descent method to

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \alpha \frac{\boldsymbol{g}}{\sqrt{\boldsymbol{n}_{new}} + \varepsilon}, \quad \sqrt{\boldsymbol{n}_{new}} = \sqrt{\sum_i n_{i;new}} \tag{C.7}$$

where ε is a small constant for numerical stability to prevent the denominator be zero ($\varepsilon \ll 1$). Histories suppress the model parameters to be updated slowly along the previous direction but promote the model parameter updated along the new direction.

We usually employ both the momentum and adaptive methods and update the model parameters with the momentum and history.

Overfitting

When we train the model with the gradient descent method repeatedly, the average loss for the given training dataset decreases gradually for the epochs.

Figure C.1(a) shows that the baseline model's average losses for a training dataset and a validation dataset decrease and converge to each value. The baseline model, such as the linear fitting model or the perceptron model, has a relatively small number of model parameters than the deep learning model. The baseline model's average loss for even the training dataset converges to a value greater than 0.

On the other hand, deep learning models' average loss for the training dataset decreases continuously over the epoch. However, the average loss for the validation dataset decreases initially, then increases after the point in which the average loss for

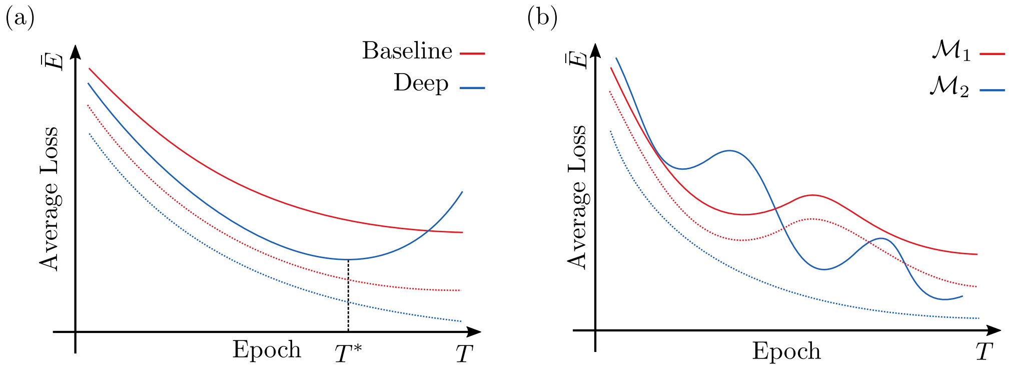

Figure C.1: Schematic plots of the baseline and deep learning models' average loss for a training dataset and a validation dataset. (a) Plots of the average loss of the baseline and deep learning models over epoch. The baseline model's average loss for the training dataset and the validation dataset denotes a red dotted line and a solid red line. The deep learning model's average loss for the training dataset and the validation dataset denotes a blue dotted line and a solid blue line. We denote the point of the minimum average loss of the deep learning model for the validation dataset as T^*. (b) Plots of the average loss of the several deep learning models over epoch. We depict the average loss for the training dataset as a dotted line and the validation dataset as a solid line.

the validation dataset is minimum. Since we evaluate the model's performance by the average loss for the validation dataset, the optimal model is in the epoch T^*. The model, which has a bigger average loss than the optimal model after the epoch T^*, is called an overfitted model.

Theoretically, the model is overfitting when the model parameter space is larger than the training dataset space depending on the data space and the number of training data points.

Classical fitting methods appropriate to the baseline model have a restricted number of model parameters and do not face the overfitting problem. However, deep learning encounters the overfitting problem frequently because the deep learning model has numerous model parameters even close to the dataset space's dimension.

Although an optimizer may overfit a deep learning model, the optimal deep learning model performs better than the baseline model. So, we stop training the model before the model is overfitted and take the optimal model.

As shown in Figure C.1(a), the training and validation datasets' average losses temporarily increase and decrease again to deviate from the local minimum. Meanwhile, only the average loss for the validation dataset decreases with up and down repeatedly. When the model is at the point of a minimum average loss, it is hard to know whether the model is in a local minimum or a global minimum of the average loss for the validation dataset, practically. Therefore, the methods are developed as possible to prevent and restrain overfitting in the gradient descent method.

We first consider increasing the number of training dast. Sampling additional data makes the training dataset space larger and solves the overfitting problem. However, it is often difficult to obtain additional training data in practice, and enlarging the training dataset with noise and bootstrapping brings another problem. Keeping in this mind, controversially, we reduce the model parameters such as removing the layer and decreasing the number of neurons to prevent overfitting. However, reducing the model affects the optimal model performance to decrease: No free lunch.

Regularization Another way to prevent overfitting is regularization. When the gradient descent method is overfitting a model, the particular model parameters' value grows extremely. Therefore, the overfitted model only memorizes the training dataset but does not learn about the data population.

Therefore, to prevent an inhomogeneous change of model parameters, we add a penalty for increasing the weight's value to the average loss. This penalty is called regularization.

For example, L1 regularization adds an absolute value of the weights to the average loss $\bar{E}$, such as

$$\bar{E}_1 = \bar{E} + \lambda \sum_i |\theta_i|, \tag{C.8}$$

and L2 regularization adds a squared value of the weights to the average loss, such as

$$\bar{E}_2 = \bar{E} + \frac{\lambda}{2} \sum_i \theta_i^2, \tag{C.9}$$

where the λ denotes the regularization rate and the θ_i denotes the model paramter, including the weights and biases.

We apply the gradient descent method for the average loss with L1 regularization and revise the updating rule as above Eq. (A.2) to

$$\theta_{new} - \theta = -\alpha \partial_\theta \bar{E} - \alpha\lambda \operatorname{sgn}(\theta), \quad \theta \in \boldsymbol{\theta}, \tag{C.10}$$

where α denotes a learning rate and $\operatorname{sgn}(x)$ denote a sign of a real number x. Likewise, we obtain the updating rule for L2 regularization as follows

$$\theta_{new} - \theta = -\alpha \partial_\theta \bar{E} - \alpha\lambda\theta, \quad \theta \in \boldsymbol{\theta}. \tag{C.11}$$

We remark that increasing the regularization rate to restrain the overfitting, but a large regularization rate may disturb training the model. Thus, we tune the regularization rate comparing a learning rate.

Dropout The heart of regularization is to prevent the inhomogeneous rapid growth of some weights' value. In this mind, the dropout [54] is developed to select updating weights stochastically, comparing the original gradient descent method updates all weights concurrently.

Here, we introduce an adjacency factor for weights called a mask $\boldsymbol{A}$, such as Eq. (1.68) in a sparsely-connected layer. If weight θ_i participates in the updating rule, then the mask for weight, A_i is 1. Otherwise, the mask A_i is zero. The probability of the mask is 0 is p, the probability of the mask is 1 is $1 - p$, and p is called a dropout rate.

Each updating, we first sampling the mask for the weight from the probability distribution. We perform the gradient descent method for modified weights θ'_i, masked weights, and update the original weights with changes in the masked weights.

$$\theta_{i;new} - \theta_i = -\alpha \partial_{\theta'_i} \bar{E}(\theta'_i), \quad \theta'_i = A_i \theta_i \tag{C.12}$$

Like the regularization rate, we note that increasing the dropout rate affects regularization but interrupts training a model.

Appendix D

Kasteleyn-Fortuin formalism

Owing to Kasteleyn-Fortuin formalism [38] that connects the percolation model [39] to the Potts model [40], though there is no free energy governing the percolation transition, we can analyze the percolation transition following the phase transition theory. Nevertheless, there is no review including calculations of the Kasteleyn-Fortuin formalism on complex networks, so we contain the details here.

Percolation on networks

Let us consider a network consisting of N sites and B bonds. A bond links between two sites, and we represent the connectivity of the network as the adjacency matrix A or the graph $\mathcal{G}(N, B)$ in the graph theory. If the adjacency matrix is a constant for the time, the network is quenched.

Here, we introduce the bond percolation model on the quenched networks. In the bond percolation model, the bond has two states, occupied state or vacant state; Meanwhile, the site percolation has two-state sites. Under the bond percolation, each bond is occupied with the occupation probability p or becomes a vacant state with the probability $(1-p)$, independently. Once determining all bond's state, we obtain a graph $\mathcal{G}'$ consisting of N sites and the B' occupied bonds, which is the subgraph of the graph $\mathcal{G}$. So, the probability that we take the subgraph $\mathcal{G}'$ by percolating the graph $\mathcal{G}$ is as follows

$$\boldsymbol{P}(\mathcal{G}') = p^{B'}(1-p)^{B-B'}. \tag{D.1}$$

We can measure the physical or mathematical quantity $A(\mathcal{G}')$ on the subgraph $\mathcal{G}'$, and then the average value of the $\langle A\rangle$ for percolating the graph $\mathcal{G}$ follows as

$$\langle A\rangle = \sum_{\mathcal{G}'\in\mathcal{G}} \boldsymbol{P}(\mathcal{G}')A(\mathcal{G}'). \tag{D.2}$$

We obtain the average of the $\langle A\rangle$ in the thermodynamic limit ($N\to\infty$) such that

$$\langle A\rangle_\infty = \lim_{N\to\infty} N^{-1}\langle A\rangle \tag{D.3}$$

We are interested in the emergence of the giant (infinite) cluster after percolating the network, depending on the occupation probability p in the percolation model. Here, a cluster is a set of sites connected through occupied bonds, and the cluster's size is the number of sites belonging to the cluster. We first look up the order parameter P_∞, the probability that a site belongs to the giant clusters such that

$$P_\infty = 1 - \left\langle \sum_c{}' s_c \right\rangle_\infty, \tag{D.4}$$

where $\sum_c{}'$ denotes the summation for only finite clusters, and s_c denotes the cluster c's size.

Next, the mean cluster size S is as follows,

$$S = \left\langle \sum_c{}' s_c^2 \right\rangle_\infty. \tag{D.5}$$

Potts model on networks

The Potts model is a generalized model of the Ising model for extending the spin's state. In the Potts model on the network, a spin i is put on each site and can have q states; $\sigma_i = 1,\cdots,q$. A spin interacts with the other spin through the (occupied) bond, and J denotes the coupling constant, determining the strength of an interaction between two spins. The Hamiltonian is defined as follows,

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \delta_{\sigma_i \sigma_j} - h \sum_i \delta_{\sigma_i \sigma_0}, \tag{D.6}$$

where $\sum_{\langle i,j \rangle}$ is the summation for pairs of neighboring two spins i and j, h is the external field along the direction of σ_0, and $\delta_{\sigma_i \sigma_j}$ is the Kronecker delta such that

$$\delta_{\sigma_i \sigma_j} = \begin{cases} 1 & (\sigma_i = \sigma_j) \\ 0 & (\sigma_i \neq \sigma_j) \end{cases} . \tag{D.7}$$

When $q = 2$, the Potts model retrieves the Ising model.

We calculate the partition function of the Potts model as follows

$$Z = \sum_{\{\sigma\}} \exp(-\beta \mathcal{H}) = \sum_{\{\sigma\}} \prod_{\langle i,j \rangle} \exp(K \delta_{\sigma_i \sigma j}) \prod_i \exp(H \delta_{\sigma_i \sigma_0}), \tag{D.8}$$

where $K = \beta J$, $H = \beta h$, $\beta = 1/(k_B T)$, T denotes temperature, and k_B is the Boltzmann constant.

To compute the partition function for the Potts model on a network consisting of N spins, we first focus on two spins to understand the problem. We introduce a mathematical trick such that

$$\begin{aligned} e^{K \delta_{\sigma_i \sigma j}} &= 1 + (e^K - 1) \delta_{\sigma_i \sigma j} = 1 + u \delta_{\sigma_i \sigma j}, \\ e^{H \delta_{\sigma_i \sigma_0}} &= 1 + (e^H - 1) \delta_{\sigma_i \sigma_0} = 1 + v \delta_{\sigma_i \sigma_0}, \end{aligned} \tag{D.9}$$

where $u = e^K - 1$, and $v = e^H - 1$. Then, we obtain partition function for two spins, such as

$$
\begin{aligned}
Z_2 &= \sum_{\{\sigma_1,\sigma_2\}} \exp(K\delta_{\sigma_1\sigma 2} + H\delta_{\sigma_1\sigma_0} + H\delta_{\sigma_2\sigma_0}) \\
&= \sum_{\{\sigma_1,\sigma_2\}} (1 + u\delta_{\sigma_1\sigma 2})(1 + v\delta_{\sigma_1\sigma_0})(1 + v\delta_{\sigma_2\sigma_0}) \\
&= q^2 + q(u + 2v) + 2uv + uv^2 + v^2 \\
&= u^0(q + (1 + v)^1 - 1)^2 + u(q + (1 + v)^2 - 1) \qquad \text{(D.10)}
\end{aligned}
$$

Here, we use the following calculations,

$$
\sum_{\{\sigma\}} (1) = q^2, \qquad \text{(D.11)}
$$

$$
\sum_{\{\sigma\}} \delta_{\sigma_1\sigma_2} = q, \qquad \text{(D.12)}
$$

$$
\sum_{\{\sigma\}} \delta_{\sigma_1\sigma_0} = \sum_{\{\sigma\}} \delta_{\sigma_2\sigma_0} = q, \qquad \text{(D.13)}
$$

$$
\begin{aligned}
\sum_{\{\sigma\}} \delta_{\sigma_1\sigma 2}\delta_{\sigma_1\sigma_0} &= \sum_{\{\sigma\}} \delta_{\sigma_1\sigma 2}\delta_{\sigma_2\sigma_0} = \sum_{\{\sigma\}} \delta_{\sigma_1\sigma_0}\delta_{\sigma_2\sigma_0} \\
&= \sum_{\{\sigma\}} \delta_{\sigma_1\sigma 2}\delta_{\sigma_1\sigma_0}\delta_{\sigma_2\sigma_0} = 1. \qquad \text{(D.14)}
\end{aligned}
$$

Let us consider the meaning of Eq. (D.10). The first term of the equation means the number of cases where two spins can have a state independently; q^2. However, the correction $(1 + v) - 1$ comes from the interaction between the external field σ_0 and the single spin. The second term of the equation means the number of cases where two spins have the same state; q. The correction $(1 + v)^2 - 1$ and also from the interaction between the external field σ_0 and the spin cluster consisting of two spins, and the weight u comes from the interaction between the external field and the bond connecting two spins.

Following this interpretation, we can obtain the partition function for the graph $\mathcal{G}$

as follows

$$Z = \sum_{\mathcal{G}' \in \mathcal{G}} u^{B'} \prod_{c} \left(q + (1+v)^{s_c} - 1 \right). \tag{D.15}$$

Kasteleyn-Fortuin mapping

To connect the Potts model with the Bond percolation model, let us parameterize the Potts model in terms of the percolation model. So, we assume the relation between the two models, such as

$$u = \frac{p}{1-p}, \quad p = 1 - e^{-K}. \tag{D.16}$$

We can rewrite the partition function of Eq. (D.15) as the following expectation value in the percolation model,

$$\begin{aligned} Z &= e^{BK} \sum_{\mathcal{G}' \in \mathcal{G}} p^{B'} (1-p)^{B-B'} \prod_{c} \left(q + e^{H s_c} - 1 \right) \\ &= e^{BK} \left\langle \prod_{c} \left(q + e^{H s_c} - 1 \right) \right\rangle. \end{aligned} \tag{D.17}$$

Furthermore, in the thermodynamic limit, free energy per site f is as follows,

$$f(q, K, H) = \lim_{N \to \infty} N^{-1} \ln Z. \tag{D.18}$$

Using free energy f, we analyze the phase transition for the Potts model, not the percolation model.

Here, we introduce a new free energy g, the first derivative of the free energy f for

q in the limit $q \to 1$.

$$\begin{aligned} g(K, H) &\equiv \frac{\partial}{\partial q} f(q, K, H)\bigg|_{q \to 1} \qquad \text{(D.19)} \\ &= \lim_{q \to 1} \lim_{N \to \infty} \frac{1}{NZ} \frac{\partial Z}{\partial q} \\ &= \lim_{N \to \infty} \frac{1}{N} \left\langle \sum_c e^{-Hs_c} \right\rangle \\ &= \begin{cases} \left\langle \sum_c e^{-Hs_c} \right\rangle_\infty, & (H = 0) \\ \left\langle {\sum_c}' e^{-Hs_c} \right\rangle_\infty, & (H > 0) \end{cases} \qquad \text{(D.20)} \end{aligned}$$

The Potts model's order parameter is defined as the first derivative of the free energy f for the effective external field H as follows

$$m \equiv \frac{\partial f}{\partial H}\bigg|_{H \to 0+}. \qquad \text{(D.21)}$$

Following the above formalism, we can consider the order parameter for a spin model governed by the free energy g, such as

$$m_p \equiv \frac{\partial g}{\partial H}\bigg|_{H \to 0+}. \qquad \text{(D.22)}$$

Here, we find the order parameter m_p corresponding to the order parameter of the percolation model;

$$P_\infty = 1 - m_p. \qquad \text{(D.23)}$$

Furthermore, the magnetic susceptibility χ is defined as the second derivative of the free energy f;

$$\chi \equiv \frac{\partial^2 f}{\partial H^2}\bigg|_{H \to 0+}. \qquad \text{(D.24)}$$

Then, susceptibility for the free energy g, χ_p is the same as the mean cluster size of the percolation model,

$$\chi_p \equiv \left.\frac{\partial^2 g}{\partial H^2}\right|_{H\to 0+} = S. \tag{D.25}$$

Therefore, even though the percolation model does not have the Hamiltonian, Kasteleyn-Fortuin formalism maps the percolation model to a spin model governed by free energy g and analyzes the percolation transition by following phase transition theory. Moreover, the formalism provides physical quantities such as heat capacity, hardly to be imaged mathematically for the percolation model, thereby enriches the understanding of percolation transition.

Bibliography

[1] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, “Machine learning and the physical sciences,” *Rev. Mod. Phys.*, vol. 91, p. 045002, Dec 2019.

[2] B. S. Rem, N. Käming, M. Tarnowski, L. Asteria, N. Fläschner, C. Becker, K. Sengstock, and C. Weitenberg, “Identifying quantum phase transitions using artificial neural networks on experimental data,” *Nature Physics*, vol. 15, no. 9, pp. 917–920, 2019.

[3] A. Bohrdt, C. S. Chiu, G. Ji, M. Xu, D. Greif, M. Greiner, E. Demler, F. Grusdt, and M. Knap, “Classifying snapshots of the doped hubbard model with machine learning,” *Nature Physics*, vol. 15, no. 9, pp. 921–924, 2019.

[4] W. Yang, X. Zhang, Y. Tian, W. Wang, J. Xue, and Q. Liao, “Deep learning for single image super-resolution: A brief review,” *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3106–3121, 2019.

[5] J. Schmidt, M. R. G. Marques, S. Botti, and M. A. L. Marques, “Recent advances and applications of machine learning in solid-state materials science,” *npj Computational Materials*, vol. 5, p. 83, dec 2019.

[6] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning,” *Nature*, vol. 549, pp. 195–202, sep 2017.

[7] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive Science*, vol. 9, no. 1, pp. 147 – 169, 1985.

[8] G. E. Hinton and R. R. Salakhutdinov, “A better way to pretrain deep boltzmann machines,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, pp. 2447–2455, Curran Associates, Inc., 2012.

[9] J. Carrasquilla and R. G. Melko, “Machine learning phases of matter,” *Nature Physics*, vol. 13, no. 5, pp. 431–434, 2017.

[10] A. Canabarro, F. F. Fanchini, A. L. Malvezzi, R. Pereira, and R. Chaves, "Unveiling phase transitions with machine learning," *Phys. Rev. B*, vol. 100, p. 045129, Jul 2019.

[11] W. Zhang, J. Liu, and T. C. Wei, "Machine learning of phase transitions in the percolation and XY models," *Physical Review E*, vol. 99, no. 3, pp. 1–14, 2019.

[12] J. Venderley, V. Khemani, and E. A. Kim, "Machine learning out-of-equilibrium phases of matter," *Physical Review Letters*, vol. 120, no. 25, p. 257204, 2018.

[13] S. J. Wetzel, "Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders," *Physical Review E*, 2017.

[14] K.-I. Aoki and T. Kobayashi, "Restricted Boltzmann machines for the long range Ising models," *Modern Physics Letters B*, vol. 30, no. 34, p. 1650401, 2016.

[15] W. Hu, R. R. Singh, and R. T. Scalettar, "Discovering phases, phase transitions, and crossovers through unsupervised machine learning: A critical examination," *Physical Review E*, vol. 95, no. 6, pp. 1–14, 2017.

[16] L. Wang, "Discovering phase transitions with unsupervised learning," *Physical Review B*, vol. 94, no. 19, pp. 2–6, 2016.

[17] H.-P. Breuer, *The Theory of Open Quantum Systems*. Oxford University Press, 2007.

[18] R. Gutiérrez, C. Simonelli, M. Archimi, F. Castellucci, E. Arimondo, D. Ciampini, M. Marcuzzi, I. Lesanovsky, and O. Morsch, "Experimental signatures of an absorbing-state phase transition in an open driven many-body quantum system," *Phys. Rev. A*, vol. 96, p. 041602(R), Oct 2017.

[19] F. Mormann, T. Kreuz, C. Rieke, R. G. Andrzejak, A. Kraskov, P. David, C. E. Elger, and K. Lehnertz, "On the predictability of epileptic seizures," *Clinical neurophysiology*, vol. 116, no. 3, pp. 569–587, 2005.

[20] P. Mirowski, D. Madhavan, Y. LeCun, and R. Kuzniecky, "Classification of patterns of eeg synchronization for seizure prediction," *Clinical neurophysiology*, vol. 120, no. 11, pp. 1927–1940, 2009.

[21] S. Chandaka, A. Chatterjee, and S. Munshi, “Cross-correlation aided support vector machine classifier for classification of eeg signals,” *Expert Systems with Applications*, vol. 36, no. 2, pp. 1329–1336, 2009.

[22] J. R. Williamson, D. W. Bliss, D. W. Browne, and J. T. Narayanan, “Seizure prediction using eeg spatiotemporal correlation structure,” *Epilepsy & behavior*, vol. 25, no. 2, pp. 230–238, 2012.

[23] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, “Reservoir observers: Model-free inference of unmeasured variables in chaotic systems,” *Chaos*, vol. 27, no. 4, 2017.

[24] T. L. Carroll, “Using reservoir computers to distinguish chaotic signals,” *Physical Review E*, vol. 98, no. 5, p. 52209, 2018.

[25] Z. Lu, B. R. Hunt, and E. Ott, “Attractor reconstruction by machine learning,” *Chaos*, vol. 28, no. 6, 2018.

[26] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, “Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data,” *Chaos*, vol. 27, no. 12, 2017.

[27] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, “Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach,” *Physical Review Letters*, vol. 120, no. 2, p. 24102, 2018.

[28] T. Weng, H. Yang, C. Gu, J. Zhang, and M. Small, “Synchronization of chaotic systems and their machine-learning models,” *Physical Review E*, vol. 99, no. 4, pp. 1–7, 2019.

[29] J. Jiang and Y.-C. Lai, “Model-free prediction of spatiotemporal dynamical systems with recurrent neural networks: Role of network spectral radius,” *Physical Review Research*, vol. 1, no. 3, p. 33056, 2019.

[30] H. Fan, J. Jiang, C. Zhang, X. Wang, and Y.-C. Lai, “Long-term prediction of chaotic systems with machine learning,” *Physical Review Research*, vol. 2, p. 012080, mar 2020.

[31] G. Filatrella, A. H. Nielsen, and N. F. Pedersen, “Analysis of a power grid using a Kuramoto-like model,” *The European Physical Journal B*, vol. 61, pp. 485–491, feb 2008.

[32] J. M. V. Grzybowski, E. E. N. Macau, and T. Yoneyama, “On synchronization in power-grids modelled as networks of second-order Kuramoto oscillators,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 26, p. 113113, nov 2016.

[33] M. Breakspear, S. Heitmann, and A. Daffertshofer, “Generative Models of Cortical Oscillations: Neurobiological Implications of the Kuramoto Model,” *Frontiers in Human Neuroscience*, vol. 4, p. 190, nov 2010.

[34] H. Nishimori and G. Ortiz, *Elements of Phase Transitions and Critical Phenomena*. Oxford Graduate Texts, OUP Oxford, 2011.

[35] 김두철, *相轉移와 臨界現象*. 민음사, 1983.

[36] K. Christensen and N. Moloney, *Complexity and Criticality*. Advanced physics texts, Imperial College Press, 2005.

[37] N. Goldenfeld, *Lectures On Phase Transitions And The Renormalization Group*. CRC Press, 2018.

[38] P. W. Kasteleyn and C. M. Fortuin, “Phase Transitions in Lattice Systems with Random Local Properties,” *Physical Society of Japan Journal Supplement*, vol. 26, p. 11, jan 1969.

[39] A. Aharony and D. Stauffer, *Introduction To Percolation Theory*. Taylor & Francis, 2003.

[40] F. Y. Wu, “The potts model,” *Rev. Mod. Phys.*, vol. 54, pp. 235–268, Jan 1982.

[41] J. U. Song, J. Um, J. Park, and B. Kahng, “Effective-potential approach to hybrid synchronization transitions,” *Phys. Rev. E*, vol. 101, p. 052313, May 2020.

[42] P. Erdös and A. Rényi, “On random graphs i,” *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[43] J. D. Kelleher and B. Tierney, *Data science*. MIT Press essential knowledge series, 2018.

[44] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[45] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," 2019.

[46] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[47] P. J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, pp. 73–101, 03 1964.

[48] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[49] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[50] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), vol. 12, pp. 1057–1063, MIT Press, 2000.

[51] C. Szepesvári, *Algorithms for Reinforcement Learning*, vol. 4. 01 2010.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[53] M. A. Aizerman, E. A. Braverman, and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning.," in *Automation and Remote Control,*, no. 25 in Automation and Remote Control,, pp. 821–837, 1964.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[55] S. Kiranyaz, T. Ince, R. Hamila, and M. Gabbouj, "Convolutional neural networks for patient-specific ecg classification," in *2015 37th Annual International*

Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 2608–2611, 8 2015.

[56] I. Carusotto and C. Ciuti, "Quantum fluids of light," *Rev. Mod. Phys.*, vol. 85, pp. 299–366, Feb 2013.

[57] C. Noh and D. G. Angelakis, "Quantum simulations and many-body physics with light," *Reports on Progress in Physics*, vol. 80, p. 016401, nov 2016.

[58] H. J. Carmichael, "Breakdown of photon blockade: A dissipative quantum phase transition in zero dimensions," *Phys. Rev. X*, vol. 5, p. 031028, Sep 2015.

[59] M. Müller, S. Diehl, G. Pupillo, and P. Zoller, "Engineered open systems and quantum simulations with atoms and ions," in *Advances in Atomic, Molecular, and Optical Physics*, vol. 61, pp. 1–80, Elsevier, 2012.

[60] K. Baumann, C. Guerlin, F. Brennecke, and T. Esslinger, "Dicke quantum phase transition with a superfluid gas in an optical cavity," *Nature*, vol. 464, no. 7293, pp. 1301–1306, 2010.

[61] K. Baumann, R. Mottl, F. Brennecke, and T. Esslinger, "Exploring symmetry breaking at the dicke quantum phase transition," *Phys. Rev. Lett.*, vol. 107, p. 140402, Sep 2011.

[62] I. Bloch, "Ultracold quantum gases in optical lattices," *Nature physics*, vol. 1, no. 1, pp. 23–30, 2005.

[63] J. M. Fink, A. Dombi, A. Vukics, A. Wallraff, and P. Domokos, "Observation of the photon-blockade breakdown phase transition," *Phys. Rev. X*, vol. 7, p. 011012, Jan 2017.

[64] T. Fink, A. Schade, S. Höfling, C. Schneider, and A. Imamoglu, "Signatures of a dissipative phase transition in photon correlation measurements," *Nature Physics*, vol. 14, no. 4, pp. 365–369, 2018.

[65] M. Fitzpatrick, N. M. Sundaresan, A. C. Y. Li, J. Koch, and A. A. Houck, "Observation of a dissipative phase transition in a one-dimensional circuit qed lattice," *Phys. Rev. X*, vol. 7, p. 011016, Feb 2017.

[66] S. Helmrich, A. Arias, G. Lochead, T. Wintermantel, M. Buchhold, S. Diehl, and S. Whitlock, "Signatures of self-organized criticality in an ultracold atomic gas," *Nature*, vol. 577, no. 7791, pp. 481–486, 2020.

[67] T. E. Lee, S. Gopalakrishnan, and M. D. Lukin, "Unconventional magnetism via optical pumping of interacting spin systems," *Phys. Rev. Lett.*, vol. 110, p. 257204, Jun 2013.

[68] J. Jin, A. Biella, O. Viyuela, L. Mazza, J. Keeling, R. Fazio, and D. Rossini, "Cluster mean-field approach to the steady-state phase diagram of dissipative spin systems," *Phys. Rev. X*, vol. 6, p. 031011, Jul 2016.

[69] A. Le Boité, G. Orso, and C. Ciuti, "Steady-state phases and tunneling-induced instabilities in the driven dissipative bose-hubbard model," *Phys. Rev. Lett.*, vol. 110, p. 233601, Jun 2013.

[70] J. Klinder, H. Keßler, M. Wolke, L. Mathey, and A. Hemmerich, "Dynamical phase transition in the open dicke model," *Proceedings of the National Academy of Sciences*, vol. 112, no. 11, pp. 3290–3295, 2015.

[71] L. J. Zou, D. Marcos, S. Diehl, S. Putz, J. Schmiedmayer, J. Majer, and P. Rabl, "Implementation of the dicke lattice model in hybrid quantum system arrays," *Phys. Rev. Lett.*, vol. 113, p. 023603, Jul 2014.

[72] D. Nagy and P. Domokos, "Nonequilibrium quantum criticality and non-markovian environment: Critical exponent of a quantum phase transition," *Phys. Rev. Lett.*, vol. 115, p. 043601, Jul 2015.

[73] A. A. Houck, H. E. Türeci, and J. Koch, "On-chip quantum simulation with superconducting circuits," *Nature Physics*, vol. 8, no. 4, pp. 292–299, 2012.

[74] J. Marino and S. Diehl, "Driven markovian quantum criticality," *Phys. Rev. Lett.*, vol. 116, p. 070407, Feb 2016.

[75] C. Pérez-Espigares, M. Marcuzzi, R. Gutiérrez, and I. Lesanovsky, "Epidemic dynamics in open quantum spin systems," *Phys. Rev. Lett.*, vol. 119, p. 140401, Oct 2017.

[76] E. G. Dalla Torre, E. Demler, T. Giamarchi, and E. Altman, “Quantum critical states and phase transitions in the presence of non-equilibrium noise,” *Nature Physics*, vol. 6, no. 10, pp. 806–810, 2010.

[77] L. M. Sieberer, S. D. Huber, E. Altman, and S. Diehl, “Dynamical critical phenomena in driven-dissipative systems,” *Phys. Rev. Lett.*, vol. 110, p. 195301, May 2013.

[78] L. M. Sieberer, S. D. Huber, E. Altman, and S. Diehl, “Nonequilibrium functional renormalization for driven-dissipative bose-einstein condensation,” *Phys. Rev. B*, vol. 89, p. 134310, Apr 2014.

[79] J. Lang and F. Piazza, “Critical relaxation with overdamped quasiparticles in open quantum systems,” *Phys. Rev. A*, vol. 94, p. 033628, Sep 2016.

[80] S. Diehl, A. Tomadin, A. Micheli, R. Fazio, and P. Zoller, “Dynamical phase transitions and instabilities in open atomic many-body systems,” *Phys. Rev. Lett.*, vol. 105, p. 015702, Jul 2010.

[81] E. G. Dalla Torre, E. Demler, T. Giamarchi, and E. Altman, “Dynamics and universality in noise-driven dissipative systems,” *Phys. Rev. B*, vol. 85, p. 184302, May 2012.

[82] U. C. Täuber and S. Diehl, “Perturbative field-theoretical renormalization group approach to driven-dissipative bose-einstein criticality,” *Phys. Rev. X*, vol. 4, p. 021010, Apr 2014.

[83] A. Mitra, S. Takei, Y. B. Kim, and A. J. Millis, “Nonequilibrium quantum criticality in open electronic systems,” *Phys. Rev. Lett.*, vol. 97, p. 236808, Dec 2006.

[84] E. G. D. Torre, S. Diehl, M. D. Lukin, S. Sachdev, and P. Strack, “Keldysh approach for nonequilibrium phase transitions in quantum optics: Beyond the dicke model in optical cavities,” *Phys. Rev. A*, vol. 87, p. 023831, Feb 2013.

[85] R. Rota, F. Storme, N. Bartolo, R. Fazio, and C. Ciuti, “Critical behavior of dissipative two-dimensional spin lattices,” *Phys. Rev. B*, vol. 95, p. 134431, Apr 2017.

[86] R. Rota, F. Minganti, C. Ciuti, and V. Savona, “Quantum critical regime in a quadratically driven nonlinear photonic lattice,” *Phys. Rev. Lett.*, vol. 122, p. 110405, Mar 2019.

[87] W. Verstraelen, R. Rota, V. Savona, and M. Wouters, “Gaussian trajectory approach to dissipative phase transitions: The case of quadratically driven photonic lattices,” *Phys. Rev. Research*, vol. 2, p. 022037, May 2020.

[88] M. Marcuzzi, M. Buchhold, S. Diehl, and I. Lesanovsky, “Absorbing state phase transition with competing quantum and classical fluctuations,” *Phys. Rev. Lett.*, vol. 116, p. 245701, Jun 2016.

[89] M. Buchhold, B. Everest, M. Marcuzzi, I. Lesanovsky, and S. Diehl, “Nonequilibrium effective field theory for absorbing state phase transitions in driven open quantum spin systems,” *Phys. Rev. B*, vol. 95, p. 014308, Jan 2017.

[90] M. Jo, J. Um, and B. Kahng, “Nonequilibrium phase transition in an open quantum spin system with long-range interaction,” *Phys. Rev. E*, vol. 99, p. 032131, Mar 2019.

[91] D. Roscher, S. Diehl, and M. Buchhold, “Phenomenology of first-order dark-state phase transitions,” *Phys. Rev. A*, vol. 98, p. 062117, Dec 2018.

[92] F. Carollo, E. Gillman, H. Weimer, and I. Lesanovsky, “Critical behavior of the quantum contact process in one dimension,” *Phys. Rev. Lett.*, vol. 123, p. 100604, Sep 2019.

[93] E. Gillman, F. Carollo, and I. Lesanovsky, “Numerical simulation of critical dissipative non-equilibrium quantum systems with an absorbing state,” *New Journal of Physics*, vol. 21, p. 093064, sep 2019.

[94] E. Gillman, F. Carollo, and I. Lesanovsky, “Numerical simulation of critical quantum dynamics without finite size effects,” *arXiv:2010.10954*, 2020.

[95] R. Dum, P. Zoller, and H. Ritsch, “Monte carlo simulation of the atomic master equation for spontaneous emission,” *Phys. Rev. A*, vol. 45, pp. 4879–4887, Apr 1992.

[96] N. Gisin and I. C. Percival, “The quantum-state diffusion model applied to open systems,” *Journal of Physics A: Mathematical and General*, vol. 25, pp. 5677–5691, nov 1992.

[97] J. Dalibard, Y. Castin, and K. Mølmer, “Wave-function approach to dissipative processes in quantum optics,” *Phys. Rev. Lett.*, vol. 68, pp. 580–583, Feb 1992.

[98] K. Mølmer, Y. Castin, and J. Dalibard, “Monte carlo wave-function method in quantum optics,” *J. Opt. Soc. Am. B*, vol. 10, pp. 524–538, Mar 1993.

[99] H.-P. Breuer and F. Petruccione, “Dissipative quantum systems in strong laser fields: Stochastic wave-function method and floquet theory,” *Phys. Rev. A*, vol. 55, pp. 3101–3116, Apr 1997.

[100] M. B. Plenio and P. L. Knight, “The quantum-jump approach to dissipative dynamics in quantum optics,” *Rev. Mod. Phys.*, vol. 70, pp. 101–144, Jan 1998.

[101] A. J. Daley, “Quantum trajectories and open many-body quantum systems,” *Advances in Physics*, vol. 63, no. 2, pp. 77–149, 2014.

[102] H. Hinrichsen, “Non-equilibrium critical phenomena and phase transitions into absorbing states,” *Advances in Physics*, vol. 49, no. 7, pp. 815–958, 2000.

[103] M. Jo and B. Kahng, “Tricritical directed percolation with long-range interaction in one and two dimensions,” *Phys. Rev. E*, vol. 101, p. 022121, Feb 2020.

[104] J. Marro and R. Dickman, *Nonequilibrium phase transitions in lattice models*. Cambridge University Press, 2005.

[105] T. E. Harris, “Contact interactions on a lattice,” *Ann. Probab.*, vol. 2, pp. 969–988, 12 1974.

[106] G. Ódor, “Universality classes in nonequilibrium lattice systems,” *Rev. Mod. Phys.*, vol. 76, pp. 663–724, Aug 2004.

[107] R. M. Ziff, E. Gulari, and Y. Barshad, “Kinetic phase transitions in an irreversible surface-reaction model,” *Phys. Rev. Lett.*, vol. 56, pp. 2553–2556, Jun 1986.

[108] R. Dickman and I. Jensen, “Time-dependent perturbation theory for nonequilibrium lattice models,” *Phys. Rev. Lett.*, vol. 67, pp. 2391–2394, Oct 1991.

[109] M. Henkel, H. Hinrichsen, S. Lübeck, and M. Pleimling, *Non-equilibrium phase transitions*, vol. 1. Springer, 2008.

[110] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[111] P. Broecker, J. Carrasquilla, R. G. Melko, and S. Trebst, "Machine learning quantum phases of matter beyond the fermion sign problem," *Scientific Reports*, vol. 7, no. 1, pp. 1–10, 2017.

[112] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, "Machine learning phases of strongly correlated fermions," *Phys. Rev. X*, vol. 7, p. 031038, Aug 2017.

[113] M. Schuld, I. Sinayskiy, and F. Petruccione, "Neural networks take on open quantum systems," *Physics*, vol. 12, p. 74, 2019.

[114] N. Yoshioka and R. Hamazaki, "Constructing neural stationary states for open quantum many-body systems," *Phys. Rev. B*, vol. 99, p. 214306, Jun 2019.

[115] M. J. Hartmann and G. Carleo, "Neural-network approach to dissipative quantum many-body dynamics," *Phys. Rev. Lett.*, vol. 122, p. 250502, Jun 2019.

[116] A. Nagy and V. Savona, "Variational quantum monte carlo method with a neural-network ansatz for open quantum systems," *Phys. Rev. Lett.*, vol. 122, p. 250501, Jun 2019.

[117] F. Vicentini, A. Biella, N. Regnault, and C. Ciuti, "Variational neural-network ansatz for steady states in open quantum systems," *Phys. Rev. Lett.*, vol. 122, p. 250503, Jun 2019.

[118] D. Harris and S. Harris, *Digital Design and Computer Architecture, Second Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2nd ed., 2012.

[119] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas,

O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[120] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[121] G. Hinton, "'neural networks for machine learning - lecture 6a - overview of mini-batch gradient descentn," 2012.

[122] J. F. F. Mendes, R. Dickman, M. Henkel, and M. C. Marques, "Generalized scaling for models with multiple absorbing states," *Journal of Physics A: Mathematical and General*, vol. 27, pp. 3019–3028, may 1994.

[123] M. J. Beach, A. Golubeva, and R. G. Melko, "Machine learning vortices at the Kosterlitz-Thouless transition," *Physical Review B*, vol. 97, no. 4, pp. 1–8, 2018.

[124] Q. Ni, M. Tang, Y. Liu, and Y. C. Lai, "Machine learning dynamical phase transitions in complex networks," *Physical Review E*, vol. 100, no. 5, pp. 1–11, 2019.

[125] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[126] H. Jaeger and H. Haas, "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[127] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[128] M. Nitzan, J. Casadiego, and M. Timme, "Revealing physical interaction networks from statistics of collective dynamics," *Science Advances*, vol. 3, no. 2, 2017.

[129] W.-X. Wang, Y.-C. Lai, and C. Grebogi, "Data based identification and prediction of nonlinear and complex dynamical systems," *Physics Reports*, vol. 644, pp. 1–76, 2016.

[130] D. Eroglu, M. Tanzi, S. van Strien, and T. Pereira, “Revealing dynamics, communities, and criticality from data,” *Physical Review X*, vol. 10, no. 2, p. 021047, 2020.

[131] Y. Kuramoto, *Self-entrainment of a population of coupled non-linear oscillators*. Berlin/Heidelberg: Springer-Verlag.

[132] Y. Kuramoto, *Chemical Oscillations, Waves, and Turbulence*, vol. 19 of *Springer Series in Synergetics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984.

[133] D. Pazó, “Thermodynamic limit of the first-order phase transition in the Kuramoto model,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 72, no. 4, pp. 1–6, 2005.

[134] L. Basnarkov and V. Urumov, “Phase transitions in the Kuramoto model,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, no. 5, pp. 1–4, 2007.

[135] L. H. Tang, “To synchronize or not to synchronize, that is the question: Finite-size scaling and fluctuation effects in the Kuramoto model,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 1, 2011.

[136] B. C. Coutinho, A. V. Goltsev, S. N. Dorogovtsev, and J. F. Mendes, “Kuramoto model with frequency-degree correlations on complex networks,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 87, no. 3, pp. 1–11, 2013.

[137] C. Choi, M. Ha, and B. Kahng, “Extended finite-size scaling of synchronized coupled oscillators,” vol. 032126, pp. 1–7, 2013.

[138] S. Yoon, M. Sorbaro Sindaci, A. V. Goltsev, and J. F. Mendes, “Critical behavior of the relaxation rate, the susceptibility, and a pair correlation function in the Kuramoto model on scale-free networks,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 91, no. 3, pp. 1–10, 2015.

[139] F. A. Rodrigues, T. K. Peron, P. Ji, and J. Kurths, “The Kuramoto model in complex networks,” *Physics Reports*, vol. 610, pp. 1–98, 2016.

[140] H. Hong, H. Park, and M. Y. Choi, “Collective synchronization in spatially extended systems of coupled oscillators with random frequencies,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 72, no. 3, pp. 1–18, 2005.

[141] H. Hong, H. Chaté, H. Park, and L. H. Tang, “Entrainment transition in populations of random frequency oscillators,” *Physical Review Letters*, vol. 99, no. 18, pp. 1–4, 2007.

[142] J. Um, H. Hong, and H. Park, “Nature of synchronization transitions in random networks of coupled oscillators,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 89, no. 1, pp. 1–8, 2014.

[143] H. Hong, H. Chat, L.-h. Tang, and H. Park, “Finite-size scaling , dynamic fluctuations , and hyperscaling relation in the Kuramoto model,” vol. 022122, pp. 1–8, 2015.

[144] A. Arenas, A. Díaz-Guilera, and C. J. Pérez-Vicente *Phys. Rev. Lett.*, vol. 96, p. 114102, Mar 2006.

[145] R. A. Rossi and N. K. Ahmed, “The network data repository with interactive graph analytics and visualization,” in *AAAI*, 2015.

[146] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[147] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” 1958.

[148] A. A. Mullin and F. Rosenblatt, “Principles of Neurodynamics.,” *The American Mathematical Monthly*, vol. 70, p. 586, may 1963.

[149] H. Robbins and S. Monro, “A stochastic approximation method,” *Ann. Math. Statist.*, vol. 22, pp. 400–407, 09 1951.

[150] F. Bach and E. Moulines, “Non-strongly-convex smooth stochastic approximation with convergence rate o(1/n),” in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 773–781, Curran Associates, Inc., 2013.

[151] B. Polyak, “Some methods of speeding up the convergence of iteration methods,” *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1 – 17, 1964.

[152] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1139–1147, PMLR, 17–19 Jun 2013.

[153] Y. Nesterov, “A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$,” *Proceedings of the USSR Academy of Sciences*, vol. 269, pp. 543–547, 1983.

[154] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.

[155] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” in *International Conference on Learning Representations*, 2018.

초 록

본 연구는 양자 접촉 모형의 양자 상전이를 인공신경망에 대한 기반 기계 학습 방법론을 통해 분석하였고 열린 양자계의 새로운 보편성을 가지는 임계현상이 있음을 밝혀내었다. 또한 기계 학습 방법론으로 구라모토 모형의 동기화 상전이의 임계현상을 분석하고 동역학 거동을 예측하므로써 동기화 현상에 대한 기계 학습을 대안적인 수치 분석의 틀거리로 활용할 수 있음을 보였다.

제1장은 임계 현상에 대한 상전이 이론과 기계 학습 방법론에 대한 일반적인 개념들을 개괄한다. 전통적인 상전이 이론은 통계물리학을 기반으로 전개되며 임계점 근방에서 발현하는 임계 현상과 그 보편성을 다룬다. 기계 학습 방법론은 기계 학습의 주요 요소들을 정의하고 모형의 형태와 모형의 최적화 과정에 대한 수학적인 기술을 제공한다. 더불어서 기계 학습 방법론 가운데서 유망한 인공신경망을 구현하는 방법을 소개하였다.

제2장은 양자 접촉 과정의 양자 상전이에 대한 기계 학습 방법론을 다룬다. 우리는 양자 도약 몬테 카를로를 이용하여 양자 접촉 과정을 따르는 1차원 스핀 사슬을 시뮬레이션하였고 관측된 활성 밀도에 따라 시스템이 활성 상태인지 흡수 상태인지를 판단할 수 있도록 합성곱 신경망과 완전결합 신경망과 같은 인공신경망을 지도 학습시켰다. 유한 크기 축적법만을 이용해서는 찾기 어려웠던 양자 상전이의 임계점을 인공신경망의 학습 결과에 외삽법을 적용하여 정교하게 측정할 수 있었다. 기계 학습을 통해서 얻어진 임계점에서 관측되는 임계 동역학을 유한 크기 축척법을 적용하여 1차원에서 양자 접촉 과정의 임계지수들을 구할 수 있었다. 1차원 양자 접촉 과정의 임계지수들 가운데서 스핀 체인의 활성 밀도에 관한 임계지수가 고전적인 단방향 스미기 모형에서 얻어지는 임계지수와 다르다는 것을 확인하였고 양자 상전이가 새로운 보편성을 보임을 밝혀내었다.

제3장은 쿠라모토 모형의 동기화 상전이에 대한 기계 학습 방법론을 다룬다. 구라모토 모형을 따라 움직이는 진동자들의 위상 거동을 관측하고 진동자들 간의

상호작용의 세기인 결합 상수를 예측할 수 있도록 순환 신경망과 전방향 신경망과 같은 인공신경망에 지도 학습시켰다. 학습된 인공신경망은 동기화된 상태의 진동자들의 상호작용의 세기를 측정할 수 있었을 뿐만 아니라 기존의 구라모토의 질서맺음 매개변수로는 추정할 수 없었던 비동기 상태에 놓여진 진동자들의 상호작용의 세기도 측정할 수 있었다. 이 결과는 인공신경망이 동기 상태에 대한 순서 매개변수와 비동기 상태에 대한 잠복 매개변수를 포착한다는 것을 나타낸다. 또한 우리는 진동자들의 위상 조합을 보고 시스템이 동기화 상태에 있는지 비동기 상태에 있는지를 판단할 수 있도록 합성곱 신경망과 완전결합 신경망과 같은 인공신경망을 지도 학습시켰다. 데이터 중첩법으로 측정되지 않았던 진동자 간의 상호작용의 거리에 관한 임계지수를 인공신경망으로 측정한 결과를 외삽법으로 얻어냈다. 기계 학습 방법론이 동기화 상전이의 임계점과 임계지수를 분석하기 위한 수치적인 틀거리로 데이터 중첩법을 포함한 유한 크기 축척법에 대한 대안이 될 수 있다. 더 나아가 진동자들의 시간에 따른 위상 변화를 저수지 컴퓨터와 순환 신경망에 학습시켜 구라모토 모형의 동역학을 재현하거나 진동자들 간의 상호작용하는 연결망을 추적하기도 하였다.

제4장에서 본 연구의 결과에 의의를 정리해보았다. 양자 접촉 모형은 열린 양자계의 대표적인 모형으로 본 연구는 기계 학습 방법론이 고전적인 물리계와 닫힌 양자계를 넘어서 열린 양자계에도 적용될 수 있음을 보여준다. 또한 본 연구는 동기화 상전이를 보이는 대표적인 비선형 동역학 모형으로 쿠라모토 모형을 다루었지만 혼돈계의 거동을 예측하고 동기화 현상을 규명하기 위한 후속 연구에서 인공신경망 기법이 중요한 역할을 할 것으로 기대된다.

주요어: 기계 학습, 지도 학습, 딥러닝, 인공 신경망, 순환 신경망, 열린 양자계, 양자 접촉 과정, 양자 상전이, 비선형 동역학, 구라모토 모형, 동기화 상전이

학번: 2014-22378