Feature extraction of two dimensional Ising model by unsupervised neural networks

Sumito Yokoo

The Graduate University for Advanced Studies , SOKENDAI

Abstract

Deep Neural Network (DNN) has been very successful. Especially, unsupervised DNN can extract the features of complicated dataset without any prior knowledge. However, theoretical understanding of how it extracts the features and what it extracts as the features is not clear. We investigate these questions from the theoretical physics point of view by using the Restricted Boltzmann Machine (RBM) and the Autoencoder (AE) trained by spin configurations of the Ising model.

In order to explore the features RBM and AE extract, we train them by spin configurations of the 2-dimensional Ising model and construct a flow of its parameters (in particular temperature) generated by the trained RBM and AE. This flow is motivated by the renormalization group flow of statistical model. It is thought that this flow emphasizes the "relevant" features the unsupervised networks learn, and eliminate "irrelevant" information from the dataset. We show that the RBM trained by spin configurations at various temperatures generates a flow on which the temperature approaches the critical temperature $T_c \simeq 2.27$ if the number of visible units is less than hidden units. Unlike the RBM case, the temperature of the fixed points of AE trained sufficiently is lower than the critical temperature if AE is sufficiently trained (5000 epochs). On the other hand, if the learning epochs are 1000, the fixed point temperature monotonically increase with the number of hidden units. We also analyze the weight matrices by singular value decomposition in order to read out the information RBM and AE learn.

Contents

Chapter 1

Introduction

Machine learning is a technique which makes computers perform various intellectual tasks such as the discrimination of subtle images, translation of languages, or generation of images. This technique has been very successful and received considerable attention in data science as well as natural science [1, 2, 3, 5, 6, 7]. This success comes from deep learning which is one of the models to achieve machine learning with Deep Neural Network (DNN) calculation motivated by the neural network of biological brain. It was considered that the DNN has too many parameters to calculate, but the recent development of computer makes the calculation possible.

The epoch making example which impressed the effectiveness of DNN is AlexNet [8]. In 2012, AlexNet won with a big difference in past records in ILSVRC (The ImageNet Large Scale Visual Recognition Challenge), which is a worldwide competition of image recognition. Starting with AlexNet, the performance of DNN for image recognition continues to improve and now DNN can discriminate images correctly at error rate 4.8%, which is less than the error rate of humans. In discriminating images, we first provide samples of input images with assigned labels, such as a cat or a dog, and then train the DNN so as to correctly predict the labels of previously unseen input images. This framework is called supervised learning.

On the other hand, another framework in which DNN is trained without assigning labels to the input data is also very important. This framework is called unsupervised learning. An epoch making example of unsupervised learning is Google's cat [9]. In unsupervised DNN trained by ten million images cut from videos on YouTube, the "cat neuron", which is a neuron that reacts only to cats, is found. Conversely, if we find the input image so as to make the "cat neuron" ignite, we can obtain an image very close to a cat. This means that DNN can extract features of complicated input data by unsupervised learning. So, unsupervised learnings are often adopted for pre-training of the supervised DNN or dimensionality reduction of the input data in order to improve the ability of the supervised DNN.

It is true that the DNN is a very useful framework, but why does it work so well? In order that machine learning works successfully, it is crucial to find the suitable representations of input

data for the tasks we want to achieve. In general, however, it is very difficult to design the suitable representations by human hand. One of the ideas for overcoming this problem is to get the representations by machine learning. The machine learning for the suitable representations is called representation learning. It is believed that DNN automatically achieves this representation learning in the process of its training. The representations which DNN gets are called deep expressions.

However, the theoretical understanding of how DNN gets the effective deep expressions is unclear. One of the possible answers to these questions is following. Input data and their specific features usually have a hierarchical structure: an image of a cat can still be identified as an animal in a very low resolution image but you may not be able to distinguish it from a dog. Such hierarchy of features can be efficiently reflected by the deep structure of neural networks. Namely, it is believed that DNN learns low-level (microscopic) representations in the upper stream of the network and gradually extracts higher-level (macroscopic) representations as the input data flows downstream. In other words, the initial data will get coarse-grained towards the output. This view is reminiscent of the renormalization group (RG) in statistical physics and quantum field theories, and various thoughts and studies are given [10, 11, 12, 13, 14, 15, 16] based on this analogy.

The RG is the most important concept and technology to understand the critical phenomena in statistical physics and also plays an essential role to constructively define quantum field theories on the lattice. It is based on the idea (and proved by Kenneth Wilson [17]) that the long-distant macroscopic behavior of a many body system is universally described by relevant operators (*relevant information*) around a fixed point, and not affected by microscopic details in the continuum limit. Through reduction of degrees of freedom in RG, the relevant information is emphasized while other irrelevant information is discarded. Particularly, suppose that the statistical model is described by a set of parameters $\{\lambda_\alpha\}$, and that the parameters are mapped to a different set $\{\tilde{\lambda}_\alpha\}$ by RG transformations[1]. Repeating such RG transformations, we can draw flow diagrams in the parameter space of the statistical model

$$\{\lambda_\alpha\} \to \{\tilde{\lambda}_\alpha\} \to \{\tilde{\tilde{\lambda}}_\alpha\} \to \cdots . \tag{1.1}$$

These RG flows control the behavior of the statistical model near the critical point where a second order phase transition occurs.

In order to compare the RG and the DNN more concretely, let us take a simple example – Ristlicted Boltzmann Machine (RBM). RBM is a network for estimating the probability distribution which generates dataset we focus on. This consists of two layers, a visible layer with variables $\mathbf{v} = (v_i)$ and a hidden layer with variables $\mathbf{h} = (h_a)$, which are coupled to each other through the

[1] In order to describe the RG transformation exactly, infinitely many parameters are necessary to be introduced. However, it can be usually well-approximated by a finite number of parameters.

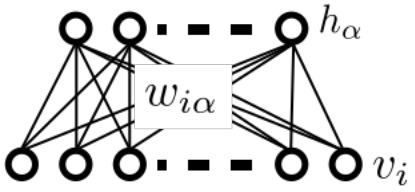


Figure 1.1: Structure of RBM

Hamiltonian

$$\Phi(\mathbf{v},\mathbf{h}) = -(\sum_{i,a} v_i W_{ia} h_a + \sum_i b_i v_i + \sum_a c_a h_a)\,, \tag{1.2}$$

and a joint probability distribution for $\mathbf{v}$ and $\mathbf{h}$ is given by

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z} e^{-\Phi(\mathbf{v},\mathbf{h})} \tag{1.3}$$

where we defined the partition function by $Z = \sum_{\mathbf{v},\mathbf{h}} e^{-\Phi(\mathbf{v},\mathbf{h})}$. No intra-layer couplings are introduced in the RBM. This structure is expressed in Fig. 1.1. Then we get the probability for visible layer $\mathbf{v}$ by marginalizing the joint probability distribution by the hidden variables

$$P(\mathbf{v}) = \sum_{\mathbf{h}} P(\mathbf{v},\mathbf{h})\,. \tag{1.4}$$

In training the RBM, we adjust the parameters so that $P(\mathbf{v})$ close to the distribution which generates the input data. Now suppose that the RBM is already trained and the parameters of the Hamiltonian (1.2), namely $\{W_{ia}, b_i, c_a\}$, are already fixed through a process of training. The joint probability distribution $P(\mathbf{v},\mathbf{h})$ also provides the following conditional probabilities for $\mathbf{h}$ (or $\mathbf{v}$) with the other variables being kept fixed

$$P(\mathbf{h}|\mathbf{v}) = \frac{P(\mathbf{v},\mathbf{h})}{\sum_{\mathbf{h}} P(\mathbf{v},\mathbf{h})}\,, \tag{1.5}$$

$$P(\mathbf{v}|\mathbf{h}) = \frac{P(\mathbf{v},\mathbf{h})}{\sum_{\mathbf{v}} P(\mathbf{v},\mathbf{h})}\,. \tag{1.6}$$

These conditional probabilities generate a flow of distributions, and consequently a flow of parameters $\{\lambda_\alpha\}$ of the corresponding statistical model. Suppose that we have a set of $N(\gg 1)$ initial configurations $\{\mathbf{x}_n = (x_{n,i})\}_{n=1,\cdots,N}$, which are generated by some kind of a statistical model with parameters λ_α, such as the Ising model at temperature T. In the large N limit, the distribution function

$$q_0(\mathbf{v}) = \frac{1}{N}\sum_n \delta_{\mathbf{v},\mathbf{x}_n}\,. \tag{1.7}$$

faithfully characterizes the statistical model with parameters λ_α, where $\delta_{\mathbf{x},\mathbf{y}}$ is Kronecker delta

$$\delta_{\mathbf{x},\mathbf{y}} := \begin{cases} 1 & (\mathbf{x} = \mathbf{y}) \\ 0 & (\mathbf{x} \neq \mathbf{y}) \, . \end{cases} \tag{1.8}$$

Multiplying $q_0(\mathbf{v})$ by the conditional probabilities (1.5) and (1.6) iteratively, we can generate a flow of probability distributions as

$$q_0(\mathbf{v}) \to r_1(\mathbf{h}) = \sum_{\mathbf{v}} P(\mathbf{h}|\mathbf{v}) q_0(\mathbf{v}) \tag{1.9}$$

$$r_1(\mathbf{h}) \to q_1(\mathbf{v}) = \sum_{\mathbf{h}} P(\mathbf{v}|\mathbf{h}) r_1(\mathbf{h}) \tag{1.10}$$

and so on for $q_n(\mathbf{v}) \to r_{n+1}(\mathbf{h})$ and $r_{n+1}(\mathbf{h}) \to q_{n+1}(\mathbf{v})$. Let us focus on (1.9). If the probability distribution $r_1(\mathbf{h})$ is well approximated by the Boltzmann distribution of the same statistical model with different parameters $\tilde{\lambda}_\alpha$, we can say that the RBM generates a transformation[2] from $\{\lambda_\alpha\}$ to $\{\tilde{\lambda}_\alpha\}$. If more than two layers continue to be stacked iteratively, we can obtain a flow of parameters as in (1.1). Or, we can instead look at the transformations $q_0(\mathbf{v}) \to q_1(\mathbf{v}) \to q_2(\mathbf{v}) \to \cdots$ and translate the flow of probability distributions into a flow of parameters $\{\lambda_\alpha\}$.

In this thesis, we study the these flows of distributions, (1.9) and (1.10), that the unsupervised RBM generates, in order to investigate "features" which the RBM learns by analogy of RG. Here notice that in defining the flows (1.9) and (1.10) (1.9), we need to specify how we have trained the RBM because the training determines properties of the weights and biases, and accordingly the behavior of the flow. In our simulations, we provide three different types of trainings. One type of RBM (we call type V) is trained by configurations at various temperatures from low to high. Other two types (type H and L) are trained by configurations only at high (and only at low) temperatures. Then we translate these flows of probability distributions into the flows of the temperature of the Ising model;

$$T \to \tilde{T} \to \tilde{\tilde{T}} \to \cdots \, . \tag{1.11}$$

In order to measure the temperature of a distribution of configurations, we prepare another neural network trained by the supervised learning. Our results are following. In type H/L RBM, the temperature approaches higher/lower temperature than T_c, as expected. However, in the type V RBM, the temperature approaches the critical point $T \to T_c$, as opposed to the conventional RG flow of the Ising model. This means that type V RBM extracts critical point as features even though we do not give the information about phase transition. We also analyze these results by singular value decomposition of weight matrix.

[2]The situation is similar to the footnote 1,and infinitely many parameters are necessary to represent the probability distribution $P(\mathbf{h})$ in terms of the statistical model.

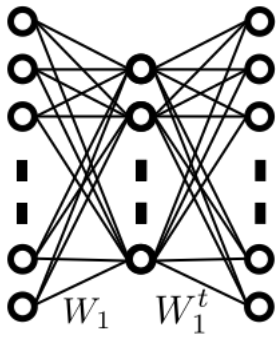


Figure 1.2: Hourgrass type autoencoder

Is it universal that the flow of unsupervised network approaches to T_c? To check this, we investigate the flow by Autoencoder (AE). AE is the unsupervised network trained so as to output the same images as input images. Especially, we focus on the hourglass-type AE which has a symmetrical structure (See Fig. 1.2). In this AE, the input $\mathbf{x}$ is transformed to $\mathbf{h}$ in the internal layer, and $\mathbf{h}$ is transformed to the output $\mathbf{y}$. The first two layers which transform $\mathbf{x}$ to $\mathbf{h}$ are called encoder, and the latter two layers which transform $\mathbf{h}$ to $\mathbf{y}$ are called a decoder. The output of the internal layer $\mathbf{h}$ is called code of $\mathbf{x}$. The code $\mathbf{h}$ contains the features of the input $\mathbf{x}$ because AE is trained so as to reconstruct same vector as $\mathbf{x}$. However, if the number of the internal units is less than the input units, then the code $\mathbf{h}$ cannot contain all information about the input $\mathbf{x}$. So, the code $\mathbf{h}$ is regarded as the coarse-grained representation of the input $\mathbf{x}$ which contains "relevant" information for reconstructing the input $\mathbf{x}$. To check what this "relevant" information is, we construct the temperature flow by trained AE as in the RBM case. Let $\mathbf{y}$ be the output of AE in the case of the input $\mathbf{x}$. We can use this $\mathbf{y}$ as the new input and get its output $\tilde{\mathbf{y}}$. By repeating this process, we get a flow of configuration:

$$\mathbf{x} \to \mathbf{y} \to \tilde{\mathbf{y}} \to \tilde{\tilde{\mathbf{y}}} \cdots . \tag{1.12}$$

Then, we translate this configuration flow to the temperature flow by supervised network. We train AE by spin configurations of the 2D Ising model with various temperatures. We compare the fixed point temperatures of AEs trained by different learning epochs. Unlike the RBM case, the temperature of the fixed points of AE trained sufficiently is lower than the critical temperature if AE is sufficiently trained (5000 epochs). On the other hand, if the learning epochs are 1000, the fixed point temperature monotonically increase with the number of hidden units. We found that the fixed point becomes the critical point when the gapped structure in singular value spectrum of weight matrix vanishes. We also prepare the datasets in narrow range of temperatures: higher temperatures case corresponding to type H ($T = 4.6, 4.7, \cdots, 5.0$), lower temperatures case corresponding to type L ($T = 1.0, 1.1, \cdots, 1.5$) and around the critical temperature case ($T = 2.2, 2.3$).

The fixed point temperatures are higher than the critical temperature, lower than the critical temperature and around the critical temperature, respectively. This result suggests that AE can learn the specific temperature of the dataset and the fixed point of the AE flow corresponds to the features which AE learns.

This thesis is organized as follows. In chapter 2, we explain the basic framework of deep neural network which we use in our simulations. In chapter 3 and 4, we show the results of the numerical simulations of the flows generated by RBM and AE, respectively. We also investigate the properties of the weight matrices by singular value decomposition in each chapter. The final section is devoted to a summary and conclusions.

Chapter 2

Deep Learning

Deep learning is a model of machine learning, which is based on Deep Neural Network (DNN) [18, 19]. In this chapter, we explain the construction of DNN and the framework of its training. In Sec. 2.1, we explain the basic method of DNN with supervised learning. In Sec. 2.2, we explain two models of unsupervised learning – Restricted Boltzmann Machine (RBM) and Autoencoder (AE), which are the main targets of our simulations.

2.1 Deep Neulal Network

In this section, we will explain the structure of the DNN and the framework of supervised learning. First, we introduce the basic feed forward network in Sec. 2.1.1. Then we explain the framework of supervised learning and some techniques in the training of DNN. Finally, we adopt these methods to multi-class classification task.

2.1.1 Structure of DNN

The neural network is constructed from elements called units (See Fig. 2.1). Each unit takes the inputs $\mathbf{x} = (x_i)$, and outputs z as follows

$$z = f\left(\sum_i w_i x_i + b_i\right). \tag{2.1}$$

$\{w_i\}$ is called weight, which reflects the strength of coupling to other units, and $\{b_i\}$ is called bias which reflects how easy the units to ignit. The function $f(\cdot)$ is called activation function which models the ignition of neurons. So, it is often taken as a non-linear function which has the threshold structure such as a step function.

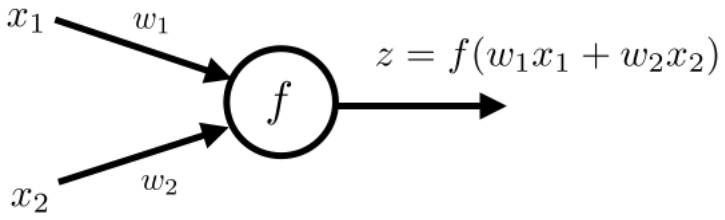


Figure 2.1: The unit of neural network

The deep neural network is the network of the units connected in layers. The schematic picture of the deep neural network is shown in Fig. 2.2. We label each layer by l. In this case, $l = 0$ is

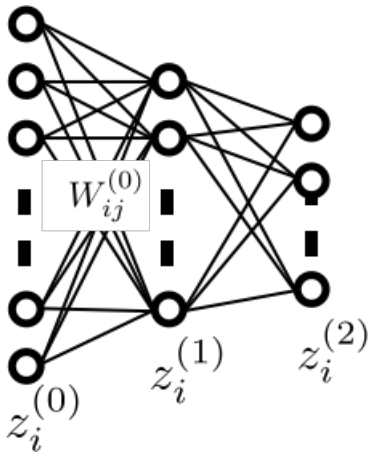


Figure 2.2: 2-layer neural network for supervised learning with an input layer $\{z_i^{(0)}\}$, a hidden layer $\{z_i^{(1)}\}$ and an output layer $\{z_i^{(2)}\}$.

called input layer and $l = 1$ is called hidden (or internal) layer and $l = 2$ is called output layer. This network is called 2-layer feedforward network.[1]

Let us explain the structure of general L-layer feedforward network. We write the output of the i-th unit in the l-th layer as $z_i^{(l)}$.

The input layer ($l = 0$)

The output of the input layer is nothing but the input data $\mathbf{x} = (x_i)$.

$$z_i^{(0)} = x_i \tag{2.2}$$

[1]Usually, the input layer is not included into the number of layers.

The hidden layer ($l = 1, 2, \cdots, L-1$)

The l-th hidden layer receives the output of the $(l-1)$-th layer weighted by $W_{ij}^{(l)}$, and adds the bias $b_i^{(l)}$, and acts activation function [2] $f(\cdot)$. So, the output of the i-th unit in the l-th layer is

$$z_i^{(l)} = f\left(\sum_j W_{ij}^{(l)} z_j^{(l-1)} + b_i^{(l)}\right). \tag{2.3}$$

The output layer ($l = L$)

The output of the L-th layer is the final output of the network. This is a very complicated non-linear function which contains many parameters $\{\mathbf{W}^{(l)} = (W_{ij}^{(l)}), \mathbf{b}^{(l)} = (b_i^{(l)})\}_{l=1,\cdots,L}$.

$$z_i^{(L)} = y_i(\mathbf{x}; \mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \cdots, \mathbf{b}^{(L)}) \tag{2.4}$$

For simplicity, we denote $\theta := \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1,\cdots,L}$ and the output of the network in the case that the input is $\mathbf{x}$ as $\mathbf{y}(\mathbf{x}; \theta)$. Our goal is to fix the parameters so that the network outputs the results we want. This process is the training of the network.

2.1.2 Training of DNN (Supervised Learning)

Next we explain the framework of training. To train the network, we prepare the training dataset $\mathcal{D} := \{(\mathbf{x}_n, \mathbf{y^*}_n)\}_{n=1,\cdots,N}$. This set is composed of pairs of input data $\mathbf{x}_n$ and corresponding label $\mathbf{y}_n^*$. This $\mathbf{y}_n^*$ represents the answer we want. Then we define the function $g(\mathbf{y}(\mathbf{x}_n; \theta), \mathbf{y}_n^*)$ which estimates the difference between the desired answer $\mathbf{y}_n^*$ and the network output $\mathbf{y}(\mathbf{x}_n; \theta)$. By this difference $g(\mathbf{y}(\mathbf{x}_n; \theta), \mathbf{y}_n^*)$, we define a loss function for the data set $\mathcal{D}$ as

$$E(\theta) := \sum_{n=1}^{N} g(\mathbf{y}(\mathbf{x}_n; \theta), \mathbf{y}_n^*), \tag{2.5}$$

which estimates the total error of the prediction of our DNN. Our task is to find the parameters θ^* which minimize the error function $E(\theta)$

$$\theta^* := \underset{\theta}{\operatorname{argmin}}\{E(\theta)\}. \tag{2.6}$$

This framework is called supervised learning because we prepare the labels to train the network.

A common method for finding the parameters θ^* is the gradient descent method. In this method, we choose an initial value of the parameters and calculate the gradients of $E(\theta)$, and

[2] In general, we can choose activation functions independently in each l. In this case we must label activation function by l as $f^{(l)}(\cdot)$

update the values of the parameters to reduce the value of the loss function. If we label renewal steps by t and denote the value of parameters in the t-th step as $W_{ij}^{(l)}(t), b_i^{(l)}(t)$, the updated value of the parameters in the $(t+1)$-th step is given by

$$W_{ij}^{(l)}(t+1) = W_{ij}^{(l)}(t) - \epsilon \left. \frac{\partial E}{\partial W_{ij}^{(l)}} \right|_{W_{ij}^{(l)}=W_{ij}^{(l)}(t)} \tag{2.7}$$

$$b_i^{(l)}(t+1) = b_i^{(l)}(t) - \epsilon \left. \frac{\partial E}{\partial b_i^{(l)}} \right|_{b_i^{(l)}=b_i^{(l)}(t)}, \tag{2.8}$$

where ϵ is the hyper parameter called learning rate [3] which controls the amount of renewal value in one step. We repeat these steps until the value of the loss function $E(\theta)$ decreases sufficiently.

In the above framework all of the input data are used in a renewal step, and this is called batch learning. However, in batch learning there is the risk that the renewal steps are trapped to bad local minima because the same loss function is used in all steps. We can overcome this problem by taking in randomness to the training steps. First, we divide the dataset $\mathcal{D}$ into some subsets called mini-batch. Then, we define the loss function in t-th steps by using different mini-batch $\mathcal{B}(t)$ in each step as follows ;

$$E(\theta; t) := \frac{1}{|\mathcal{B}(t)|} \sum_{n \in \mathcal{B}(t)} g(\mathbf{y}(\mathbf{x}_n; \theta), \mathbf{y}_n^*), \tag{2.9}$$

where $|\mathcal{B}(t)|$ is the number of elements in $\mathcal{B}(t)$. In this case, we can reduce the risk that the renewal steps are trapped to the bad local minima because we use different loss function in each training step. This framework is called mini-batch learning.

2.1.3 Multi-class classification

We explain a choice of the loss function and activation function of the output layer in the case of multi class classification task. The representative example of this task is the classification of handwritten numbers (10-class classification). Later, we will use this task to examine the temperature of the spin configurations of the Ising model.

Let us consider the K-class classification. In this case, we design the network to output a K-dimensional vector $\mathbf{y}(\mathbf{x}_n; \theta)$ whose i-th component $y_i(\mathbf{x}_n; \theta)$ represents a probability that the input data $\mathbf{x}$ is classified into the i-th class. To interpret the output $\mathbf{y}(\mathbf{x}_n; \theta)$ as a probability, it

[3] We must choose the value of the learning rate by hand to converge the renewal steps suitably, but the principle to determine earning rate is not known.

must satisfy

$$y_i(\mathbf{x}_n;\theta) \geq 0 \quad (i = 1, \cdots, K), \quad \sum_{i=1}^{K} y_i(\mathbf{x}_n;\theta) = 1\,. \tag{2.10}$$

We can construct such $\mathbf{y}(\mathbf{x}_n;\theta)$ by

$$y_i(\mathbf{x}_n;\theta) := \frac{e^{u_i(\mathbf{x}_n;\theta)}}{\sum_{k=1}^{K} e^{u_k(\mathbf{x}_n;\theta)}}\,, \quad u_i(\mathbf{x}_n;\theta) := \sum_j W_{ij}^{(L)} z_j^{(L-1)}(\mathbf{x}_n;\theta) + b_i^{(L)} \tag{2.11}$$

where $z_j^{(L-1)}(\mathbf{x}_n;\theta)$ is the output of the $(L-1)$-th layer when the input vector is $\mathbf{x}_n$.

If the input $\mathbf{x}_n$ is classified into the k-th class, the corresponding label $\mathbf{y}_n^*$ is given by

$$y_{n,i}^* = \delta_{ki}. \tag{2.12}$$

The "distance" $g(\mathbf{y}(\mathbf{x}_n;\theta);\mathbf{y^*}_n)$ between ideal probability distribution $\mathbf{y}_n^*$ and the network output $\mathbf{y}(\mathbf{x}_n;\theta)$ is defined by the cross entropy between them

$$g(\mathbf{y}(\mathbf{x}_n;\theta);\mathbf{y^*}_n) := -\sum_{i=1}^{K} y_{n,i}^* \log y_i(\mathbf{x}_n;\theta). \tag{2.13}$$

So, the loss function (in batch learning case) is given by

$$E(\theta) := -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{K} y_{n,i}^* \log y_i(\mathbf{x}_n;\theta). \tag{2.14}$$

Then, we can apply the gradient descent method to minimize the loss function.

2.2 Unsupervised Learning

In this section we explain another important framework of the neural network called unsupervised learning in which we do not prepare the labels in the training. It is believed that unsupervised learning can extract the features of the complicated input data effectively. So, it is used for the pre-training for supervised network (the training for fixing the good initial value of the parameter for gradient descent method) and dimensionality reduction of the input data. We introduce two representative models — Restricted Boltzmann Machine and Autoencoder. These two models are the main targets of this thesis.

2.2.1 Restricted Boltzmann Machine

Let us assume that the input data $\{\mathbf{x}_n\}_{n=1,\cdots,N}$ which we focus on are the realized values of random variables $\mathbf{v}$ that obey a probability distribution $P_{data}(\mathbf{v})$. Usually, we cannot know $P_{data}(\mathbf{v})$. So, we must assume the model distribution $P_{model}(\mathbf{v};\theta)$ and estimate $P_{data}(\mathbf{v})$. Such model is called a generative model. A well known generative model is Restricted Boltzmann Machine (RBM).

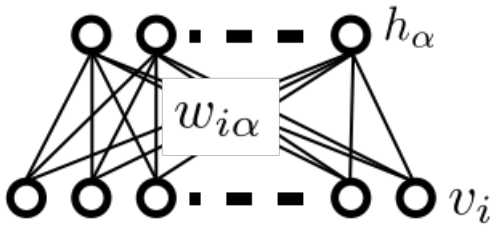


Figure 2.3: 1-layer RBM with a visible layer $\{v_i\}$ and a hidden layer $\{h_a\}$. These two layers are coupled but there are no intra-layer couplings.

In RBM, we construct the model distribution $P_{RBM}(\mathbf{v};\theta)$ as follows. First, we define an energy function (or Hamiltonian) by

$$\Phi(\mathbf{v},\mathbf{h};\theta) := -\sum_{i,a} v_i W_{ia} h_a - \sum_i b_i v_i - \sum_a c_a h_a. \tag{2.15}$$

This structure is described by a graph in Fig. 2.3. $\mathbf{v} = (v_i)$ is the visible layer corresponding to the input data and $\mathbf{h} = (h_a)$ is the hidden layer. We assume $v_i, h_a = \pm 1$. The weight matrix W_{ia} is the coupling between the visible layer and the hidden layer. b_i and c_a are biases for the visible layer and the hidden layer, respectively. We define a joint probability distribution by

$$P(\mathbf{v},\mathbf{h};\theta) := \frac{e^{-\Phi(\mathbf{v},\mathbf{h};\theta)}}{Z(\theta)}, \tag{2.16}$$

where Z is the partition function

$$Z(\theta) := \sum_{\mathbf{v},\mathbf{h}} e^{-\Phi(\mathbf{v},\mathbf{h};\theta)}. \tag{2.17}$$

Then, we define a probability distribution for the visible layer by marginalizing $P(\mathbf{v},\mathbf{h};\theta)$

$$P_{RBM}(\mathbf{v};\theta) := \frac{1}{Z(\theta)} \sum_{\mathbf{h}} e^{-\Phi(\mathbf{v},\mathbf{h};\theta)}. \tag{2.18}$$

In RBM, we deduce $P_{data}(\mathbf{v})$ by using $P_{RBM}(\mathbf{v};\theta)$.

The characteristic of RBM is that there are no intra-layer couplings. We can improve the expression ability by introducing a hidden layer and make some calculations easier because of the absence of the intra-layer couplings. For example, the conditional probabilities are written as products of probability distributions of each variable;

$$P(\mathbf{h}|\mathbf{v};\theta) = \prod_a P(h_a|\mathbf{v};\theta) = \prod_a \frac{1}{1+\exp\left[-2h_a\left(\sum_i v_i W_{ia} + c_a\right)\right]} \tag{2.19}$$

$$P(\mathbf{v}|\mathbf{h};\theta) = \prod_i P(v_i|\mathbf{h};\theta) = \prod_i \frac{1}{1+\exp\left[-2v_i\left(\sum_a W_{ia} h_a + b_i\right)\right]} \,. \tag{2.20}$$

This means that the units of the visible (hidden) layer are independent of each other when the values of the hidden (visible) layer are fixed. So, this feature is called conditional independence. Then, the expectation values of variables in the hidden or visible layer in the case that the values of the other layer are given are calculated as

$$\langle h_a \rangle_{P(\mathbf{h}|\mathbf{v};\theta)} = \tanh\left(\sum_i W_{ia} v_i + c_a\right) \tag{2.21}$$

$$\langle v_i \rangle_{P(\mathbf{v}|\mathbf{h};\theta)} = \tanh\left(\sum_a W_{ia} h_a + b_i\right) \,. \tag{2.22}$$

Next we explain the process of training RBM. The loss function is Kullback-Leibler (KL) divergence D_{KL} between the true distribution $P_{data}(\mathbf{v})$ and the model distribution $P_{RBM}(\mathbf{v};\theta)$

$$\begin{aligned} E(\theta) &= D_{KL}(P_{data}||P_{RBM}) && (2.23)\\ &:= \sum_{\mathbf{v}} P_{data}(\mathbf{v}) \log \frac{P_{data}(\mathbf{v})}{P_{RBM}(\mathbf{v};\theta)} && (2.24)\\ &= \text{const.} - \sum_{\mathbf{v}} P_{data}(\mathbf{v}) \log P_{RBM}(\mathbf{v};\theta) \,. && (2.25) \end{aligned}$$

The KL divergence is the indication of the difference between two probability distributions. The KL divergence is always non-negative and it becomes zero if and only if two distributions are equal. However, because we cannot know $P_{data}(\mathbf{v})$, we approximate $P_{data}(\mathbf{v})$ by the empirical distribution of the input data

$$q(\mathbf{v}) := \frac{1}{N} \sum_{n=1}^{N} \delta_{\mathbf{v},\mathbf{x}_n} \,. \tag{2.26}$$

, where $\delta_{\mathbf{x},\mathbf{y}}$ is Kronecker delta

$$\delta_{\mathbf{x},\mathbf{y}} := \begin{cases} 1 & (\mathbf{x} = \mathbf{y}) \\ 0 & (\mathbf{x} \neq \mathbf{y}) \,. \end{cases} \tag{2.27}$$

Then, the loss function becomes

$$E(\theta) = \text{const.} - \sum_{\mathbf{v}} q(\mathbf{v}) \log P_{RBM}(\mathbf{v};\theta) \tag{2.28}$$

$$= \text{const.} - \frac{1}{N} \sum_{n=1}^{N} \log P_{RBM}(\mathbf{x}_n;\theta) \,. \tag{2.29}$$

The derivatives of $E(\theta)$ with respect to the weight W_{ia} and the biases b_i, c_a are given by

$$\begin{aligned} \frac{\partial E}{\partial W_{ia}} &= \langle v_i h_a \rangle_{model} - \langle v_i h_a \rangle_{data} \\ \frac{\partial E}{\partial b_i} &= \langle v_i \rangle_{model} - \langle v_i \rangle_{data} \\ \frac{\partial E}{\partial c_a} &= \langle h_a \rangle_{model} - \langle h_a \rangle_{data} \,, \end{aligned} \tag{2.30}$$

where the expectation values in Eq. (2.30) are defined by

$$\langle A(\mathbf{v}) \rangle_{data} = \sum_{\mathbf{v}} q(\mathbf{v}) A(\mathbf{v}) = \frac{1}{N} \sum_{n=1}^{N} A(\mathbf{x}_n) \tag{2.31}$$

$$\langle A(\mathbf{v},\mathbf{h}) \rangle_{model} = \sum_{\mathbf{v},\mathbf{h}} P_{RBM}(\mathbf{v},\mathbf{h}) A(\mathbf{v},\mathbf{h}) \,, \tag{2.32}$$

and h_a in $\langle \cdots \rangle_{data}$ is replaced by $\langle h_a \rangle_{P(\mathbf{h}|\mathbf{v};\theta)}$ of Eq. (2.21). Then, we update the values of the weights and biases as

$$\begin{aligned} W_{ia} &\to W_{ia}^{\text{new}} = W_{ia} - \delta W_{ia} \\ b_i &\to b_i^{\text{new}} = b_i - \delta b_i \\ c_a &\to c_a^{\text{new}} = c_a - \delta c_a \,, \end{aligned} \tag{2.33}$$

where

$$\begin{aligned}
\delta W_{ia} &= \epsilon \left(\langle v_i h_a \rangle_{model} - \langle v_i h_a \rangle_{data} \right) \\
\delta b_i &= \epsilon \left(\langle v_i \rangle_{model} - \langle v_i \rangle_{data} \right) \\
\delta c_a &= \epsilon \left(\langle h_a \rangle_{model} - \langle h_a \rangle_{data} \right) .
\end{aligned} \tag{2.34}$$

Here ϵ denotes the learning rate.

The second terms $\langle \cdots \rangle_{data}$ in Eq.(2.34) are easy to calculate, but the first terms $\langle \cdots \rangle_{model}$ are difficult to evaluate since it requires the knowledge of the full partition function Z. To avoid this problem, we need to use the method of Gibbs sampling to approximately evaluate these expectation values $\langle \cdots \rangle_{model}$. However, this method takes very high calculational cost because we need sufficiently long time every time in order that the Markov chain for Gibbs sampling burns in. So, practically, we employ a more simplified method, which is called the contrastive divergence (CD) method. The idea is very simple, and reminiscent of the mean field approximation in statistical physics. Given the input data, the expectation value of the hidden variable h_a can be easily calculated as Eq. (2.21). We write the input data $v(0)_{n,i} = x_{n,i}$. Then, the expectation value of h_a given $v_i = v(0)_{n,i}$ is

$$h(0)_{n,a} := \langle h_a \rangle_{P(\mathbf{h}|\mathbf{v}(0)_n;\theta)} = \tanh \left(\sum_i W_{ia} v(0)_{n,i} + c_a \right) . \tag{2.35}$$

If we fix the values of hidden layer to this $\mathbf{h}(0)_n = (h_{n,a}(0))$, then the expectation value of v_i can be again easily calculated by using Eq. (2.22). We write it as

$$v(1)_{n,i} := \langle v_i \rangle_{P(\mathbf{v}|\mathbf{h}(0)_n)} = \tanh \left(\sum_a W_{ia} h(0)_{n,a} + b_i \right) . \tag{2.36}$$

Then we obtain $\mathbf{h}(1)_n := \langle \mathbf{h} \rangle_{P(\mathbf{h}|\mathbf{v}(1)_n;\theta)}$, and so on. We iterate this procedure T times and replace the first terms in Eq. (2.34) by $\mathbf{v}(T)_n$ and $\mathbf{h}(T)_n$. Especially, this method is T-step CD method (CD_T). Empirically, it is known that we can take a much smaller number of steps in CD_T than the number of steps in Gibbs sampling method, for example $T = 1$. In $T = 1$ case (CD_1), the formulas for calculating the expectation values in Eq. (2.34) are given by

$$\langle v_i \rangle_{data} = \frac{1}{N} \sum_{n=1}^{N} v(0)_{n,i}, \quad \langle h_a \rangle_{data} = \frac{1}{N} \sum_{n=1}^{N} h(0)_{n,a}, \quad \langle v_i h_a \rangle_{data} = \frac{1}{N} \sum_{n=1}^{N} v(0)_{n,i} h(0)_{n,a} , \tag{2.37}$$

and

$$\langle v_i \rangle_{model} = \frac{1}{N}\sum_{n=1}^{N} v(1)_{n,i}, \quad \langle h_a \rangle_{model} = \frac{1}{N}\sum_{n=1}^{N} h(1)_{n,a}, \quad \langle v_i h_a \rangle_{model} = \frac{1}{N}\sum_{n=1}^{N} v(1)_{n,i} h(1)_{n,a}. \tag{2.38}$$

2.2.2 Autoencoder

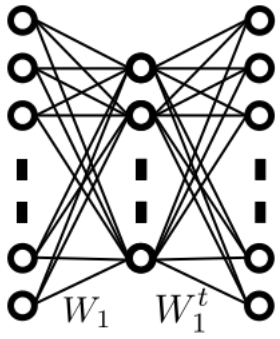


Figure 2.4: Hourglass-type autoencoder
paipai

Another unsupervised model is the Autoencoder. This is the neural network which is trained so as to output the same configurations as the input data. In other words, this is the neural network which is trained by the training data set $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{x}_n)\}_{n=1,\cdots,N}$ through the framework of Sec. 2.1.2.

In the autoencoder, we evaluate the difference between the network output $\mathbf{y}(\mathbf{x};\theta)$ and the input vector $\mathbf{x}$ by the square norm

$$g(\mathbf{y}(\mathbf{x};\theta);\mathbf{x}) := \sum_i (y_i(\mathbf{x};\theta) - x_i)^2 \,. \tag{2.39}$$

The error function for the data set $\mathcal{D} = \{\mathbf{x}_n\}_{n=1,\cdots,N}$ is

$$E(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_i (y_i(\mathbf{x}_n;\theta) - x_{n,i})^2 \,. \tag{2.40}$$

Then, we can apply the gradient descent method to minimize this loss function.

We use hourglass-type autoencoder which has a symmetric structure like Fig. 2.4. The output of internal layer is called the code regarded as deep expression which contains important features for reconstruction of input data. Then, the former part from the input layer to internal layer is called encoder, and the latter part from internal layer to output layer is called a decoder. In this

case, we can use the transpose of the first weight matrix as the second one to reduce the number of parameters to be adjusted.

2.3 Singular Value Decomposition (SVD)

Next, we introduce Singular Value Decomposision (SVD) to analyze weight matrices of RBM and AE.

2.3.1 Principle of SVD

Consider $M \times N$ real matrix A with rank r ($r \leq \min\{M, N\}$). We introduce AA^t and A^tA, which are $M \times M$ and $N \times N$ real symmetric matrices, respectively. We denote eigenvector of A^tA with eigenvalue λ_i as $|v_i\rangle$

$$A^tA\,|v_i\rangle = \lambda_i\,|v_i\rangle\,, i = 1, 2, \cdots, N\,, \tag{2.41}$$

satisfing $\langle v_i|v_j\rangle = \delta_{ij}$. For all i, $\lambda_i \geq 0$, because AA^t is a positive semi-definite matrix. The number of eigenvectors with non-zero eigenvalue is r. Then, we can arrange eigenvalues to satisfy

$$\begin{aligned} \lambda_1 > \lambda_2 > \cdots > \lambda_r > 0\,, \\ \lambda_{r+1} = \lambda_{r+2} = \cdots = \lambda_N = 0\,. \end{aligned} \tag{2.42}$$

On the other hand, we denote eigenvectors of A^tA as $\{|u_i\rangle\}_{i=1,\cdots,M}$. For $i = 1, \cdots, r$,

$$|u_i\rangle = \frac{1}{\sqrt{\lambda_i}} A\,|v_i\rangle\ , \tag{2.43}$$

and its eigenvalue is λ_i

$$AA^t\,|u_i\rangle = \frac{1}{\sqrt{\lambda_i}} A(A^tA)\,|v_i\rangle = \lambda_i\,|u_i\rangle\ . \tag{2.44}$$

These vectors satisfy orthonormal condition

$$\langle u_i|u_j\rangle = \frac{1}{\sqrt{\lambda_i\lambda_j}}\,\langle v_i|A^tA|v_j\rangle = \delta_{ij} \quad (i, j = 1, \cdots, r)\,. \tag{2.45}$$

For $i = r + 1, \cdots, M$, eigenvalue of $|u_i\rangle$ is zero and we can make these vectors satisfy orthonormal condition by Gram-Schmidt orthonormalization. Eigenvectors of a symmetric real matrix with different eigenvalues are orthogonal. So, all eigenvectors $\{|u_i\rangle\}_{i=1,\cdots,M}$ satisfy orthonormal

condition

$$\langle u_i|u_j\rangle = \delta_{ij} \quad (i,j = 1, \cdots, M)\,. \tag{2.46}$$

By using $\{|u_i\rangle\}_{i=1,\cdots,M}$ and $\{|v_i\rangle\}_{i=1,\cdots,N}$, we define orthogonal matrices $U = (U_{ij})$ and $V = (V_{ij})$ as follows

$$U_{ij} := (|u_j\rangle)_i, \quad V_{ij} := (|v_j\rangle)_i\,, \tag{2.47}$$

where $(|u_j\rangle)_i$ means the i-th component of vector $|u_j\rangle$. Because $\{|u_i\rangle\}_{i=1,\cdots,M}$ and $\{|v_i\rangle\}_{i=1,\cdots,N}$ are eigenvectors of AA^t and A^tA respectively, U and V satisfy

$$\begin{aligned} \sum_k (AA^t)_{ik} U_{kj} &= \lambda_j U_{ij}\,, \\ \sum_k (A^tA)_{ik} V_{kj} &= \lambda_j V_{ij}\,. \end{aligned} \tag{2.48}$$

Then, we can calculate U^tAV as follows

$$(U^tAV)_{ij} = \sum_{k,l} U_{ki} A_{kl} V_{lj} = \sum_{k,l,m} \frac{1}{\sqrt{\lambda_i}} A_{km} V_{mi} A_{kl} V_{lj} = \frac{1}{\sqrt{\lambda_i}} \langle v_i|A^tA|v_j\rangle = \sqrt{\lambda_i}\delta_{ij}\,. \tag{2.49}$$

This form is called singular value normal form and $\{\sqrt{\lambda_i}\}_{i=1,\cdots,r}$ is called singular value of A. We get singular value decomposition of matrix A

$$A_{ij} = \sum_k \sqrt{\lambda_k} U_{ik} V_{jk}\,. \tag{2.50}$$

2.3.2 Function of SVD

SVD enables us to decompose a image into different scales. Let us consider a 2-dimensional image as $M \times N$ matrix A. A_{ij} represents the color at pixel (i,j). We write SVD of A as

$$A_{ij} = \sum_{k=1}^{r} \sqrt{\lambda_k} U_{ik} V_{jk} = \sum_{k=1}^{r} A_{ij}^{(k)} \tag{2.51}$$

where $A_{ij}^{(k)} := \sqrt{\lambda_k} U_{ik} V_{jk}$ and $r := rank A$. Positive eigenvalues are arranged in descending order

$$\lambda_1 > \lambda_2 > \cdots > \lambda_r > 0\,. \tag{2.52}$$

$A_{ij}^{(k)}$ with large λ_k corresponds to large scale in image A and $A_{ij}^{(k)}$ with small λ_k corresponds to small scale in image A. Eq. (2.51) is scale decomposition of image A. The leading components in decomposition (2.51) are $A_{ij}^{(k)}$ with large λ_k. Then, we define χ-th order approximation of image A by cutting off the sum in (2.51) at χ-th component $A^{(\chi)}$;

$$A_\chi := \sum_{k=1}^{\chi} A^{(k)} \,. \tag{2.53}$$

We define the snapshot entropy of A which estimates information about scales encoded in A. Because of positivity of singular values, we can interpret singular values as a probability distribution by normalizing as follows

$$p_i := \frac{\lambda_i}{\sum_k \lambda_k} \,. \tag{2.54}$$

The snapshot entropy is defined as Shanon entropy of this probability distribution

$$S := -\sum_{i=1}^{r} p_i \log p_i \,. \tag{2.55}$$

For coarse-grained image A_χ, we also define the "coarse-grained" snapshot entropy $S(\chi)$

$$S(\chi) := -\sum_{i=1}^{\chi} p_i \log p_i \,. \tag{2.56}$$

It is known that $S(\chi)$ obeys scaling relation which reflects coarse-graining of original image A.[20, 21, 22, 23, 24, 25]

Chapter 3

Restricted Boltzmann Machine

In this chapter, we explain the method of numerical simulations of RBM and show various results and analysis of weight matrix by singular value decomposition. This chapter is mainly based on [26].

3.1 Setup

3.1.1 Monte-Carlo simulations of Ising model

We first construct samples of configurations of the two-dimensional (2d) Ising model by using Monte-Carlo simulations. The spin variables $\sigma_{x,y} = \pm 1$ are defined on a two dimensional lattice of size $L \times L$. The index (x, y) represents each lattice point and takes $x, y = 1, 2, \ldots, L$. We impose the periodic boundary conditions

$$\sigma_{L+1,y} := \sigma_{1,y}\,, \quad \sigma_{0,y} := \sigma_{L,y}\,, \quad \sigma_{x,L+1} := \sigma_{x,1}\,, \quad \sigma_{x,0} := \sigma_{x,L}\,. \tag{3.1}$$

for the spin variables. The Ising model Hamiltonian is given by

$$H = -J \sum_{x,y=1}^{L} \sigma_{x,y} \left(\sigma_{x+1,y} + \sigma_{x-1,y} + \sigma_{x,y+1} + \sigma_{x,y-1}\right). \tag{3.2}$$

It describes a ferromagnetic model for $J > 0$ and an anti-ferromagnetic model for $J < 0$.

Sampling of spin configurations at temperature T are performed by Metropolis Monte Carlo (MMC) method. In the method, we first generate a random configuration $\{\sigma_{x,y}\}$. We then choose

one of the spins $\sigma_{x,y}$ and flip its spin with the probability

$$p_{x,y} = \begin{cases} 1 & (\text{when } dE_{x,y} < 0) \\ e^{-dE_{x,y}/k_B T} & (\text{when } dE_{x,y} > 0)\,, \end{cases} \tag{3.3}$$

where $dE_{x,y}$ is the change of energy of this system

$$dE_{x,y} = 2J\sigma_{x,y}\left(\sigma_{x+1,y} + \sigma_{x-1,y} + \sigma_{x,y+1} + \sigma_{x,y-1}\right). \tag{3.4}$$

The probability of flipping the spin (3.3) satisfies the detailed balance condition $P_{s\to s'}\rho_s = P_{s'\to s}\rho_{s'}$, where $\rho_s \propto e^{-E_s/k_B T}$ is the canonical distribution of the spin configuration $s = \{\sigma_{x,y}\}$ at temperature T. Thus, after many iterations of flipping all the spins, the configuration approaches the equilibrium distribution at T. We can set the Boltzmann constant k_B and the interaction parameter J to be equal to 1 without loss of generality, because all physical quantities are written in terms of a combination of $J/(k_B T)$.

In the following analysis, we set the lattice size $L^2 = 10 \times 10$ and repeat the procedure of MMC simulations $100L^2 = 10000$ times to construct spin configurations. In our simulations,we generated spin configurations at various temperatures $T = 0, 0.25, 0.5, \ldots, 6$.[1] Some of typical spin configurations are shown in Fig. 3.1.

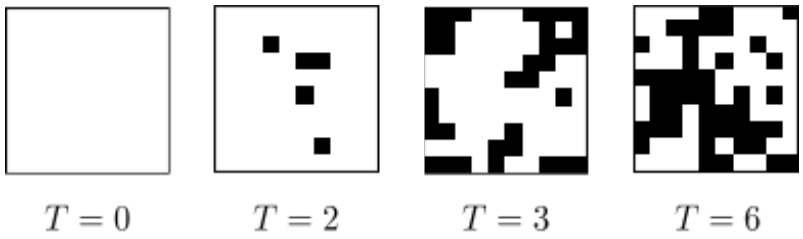


Figure 3.1: Examples of spin configurations at temperatures $T = 0, 2, 3, 6$

3.1.2 Setup of RBM

Next we train RBM by using the algorithm explained in Sec.2.2.1. In the following, we prepare 7 different sizes of the hidden spin variables; $N_h = 16, 36, 64, 81, 100, 225$ and 400, and train these models through various data sets:

Type V RBM

This is trained by configurations at temperatures ranging widely from low to high, $T = 0, 0.25, \ldots, 6$.

[1] For $T = 0$, we practically set $T = 10^{-12}$ for numerical calculations.

Type H RBM

This is trained by configurations at higher temperatures $T = 4, 4.25, \ldots, 6$ than $T_c \simeq 2.27$.

Type L RBM

This is trained by configurations only at the lowest temperature $T = 0$.

3.1.3 Generation of RBM flows

As discussed in the Introduction, once the RBM is trained and the weights and biases are fixed, the RBM generates a sequence of probability distributions. Then we translate the sequence into a flow of parameters. In generating the sequence, the initial set of configurations should be prepared differently from the configurations that are used to train the RBM.

We can also generate a flow of parameters in a slightly different method. For a specific configuration $v_i = v_i^{(0)}$, we can define a sequence of configurations following Eqs. (2.35) and (2.36) as

$$\{v_i^{(0)}\} \to \{h_a^{(0)}\} \to \{v_i^{(1)}\} \to \{h_a^{(1)}\} \to \{v_i^{(2)}\} \to \{h_a^{(2)}\} \to \cdots . \tag{3.5}$$

Since each value of $v_i^{(n)}$ (for $n > 0$) and $h_a^{(n)}$ (for $n \geq 0$) is defined by an expectation value as in Eqs. (2.36) and (2.35), it does not take an integer ± 1 but a fractional value between them. In order to get a flow of spin configurations, we need to replace these fractional values by ± 1 with a probability $\frac{1}{2}(1 \pm \langle v \rangle)$ for the expectation value $\langle v \rangle$. It turns out that the replacement is usually a good approximation since the expectation values mostly take values close to ± 1 owing to the property of the trained weights $|W_{ia}| \gg 1$. In this way, we obtain a flow of spin configurations

$$\{v_i^{(0)}\} \to \{v_i^{(1)}\} \to \{v_i^{(2)}\} \to \cdots \to \{v_i^{(n)}\} \tag{3.6}$$

starting from the initial configuration $\{v_i^{(0)}\}$. In order to obtain a flow of temperature, we provide a set of configurations. The set should be different from the set of configurations that are used for training the RBM.[2] Then we get a flow of the set of configurations. Finally, by using the NN of supervised learning explained in Sec. 3.1.4, we obtain the temperature distribution of the configurations. In this way, we can translate the above flow of configurations into a flow of temperature distributions.

[2]Therefore, we also generate additional 25000 spin configurations other than the configuration used for training the RBM.

3.1.4 Temperature measurement by a supervised-learning NN

Next we design a neural network (NN) to measure temperature of spin configurations by using the algorithm explained in Sec. 2.1.3.

We note how we measure temperature of a configuration. If size of a single configuration generated at temperature T is large enough, say $L = 10^{100}$, the trained NN will reproduce the temperature of the configuration faithfully. However, our configurations are small sized with only $L = 10$. Thus we instead need an ensemble of many spin configurations and measure a temperature distribution of the configurations. The supervised learning gives us this probability distribution of temperature.

Let us see the results of the NN for measuring temperatures. On the left side of Fig. 3.2, we plot the behavior of the loss function (2.14) as we iterate the renewal of the weights and biases. The

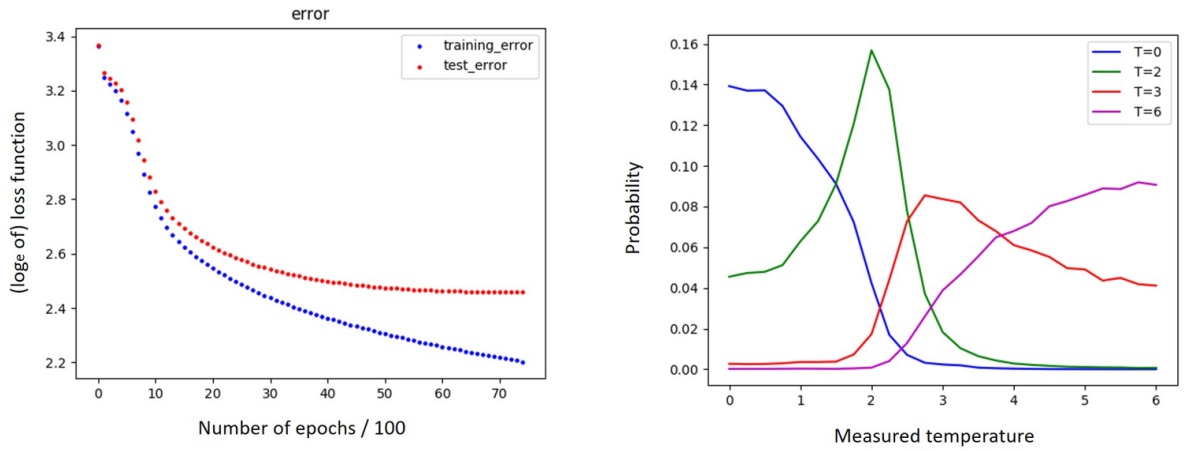


Figure 3.2: (Left) Training error and test error up to 7500 epochs. (Right) The probability distribution of measured temperature when the configurations at $T = 0, 2, 3, 6$ are input.

blue line shows the training error, namely values of the loss function after iterations of training using the 25000 configurations. It is continuously decreasing, even after 7500 epochs. On the other hand, the red line shows the test error, namely values of the loss function for additional 25000 configurations which are not used for the training. This is also decreasing at first, but after 6000 epochs it becomes almost constant. After 7500 epochs, in fact, it turns to increase. This means the machine becomes over-trained, therefore we stopped the learning at 7500 epochs before it occurs.

Then on the right side of Fig. 3.2 we show the probability distribution of temperature this NN measures. Here we use the configurations at $T = 0, 2, 3, 6$ which are not used for the training. Thus we can confirm the NN can correctly measure the temperature of configurations, even if it does not learn those configurations.

3.2 Numerical results

3.2.1 RBM flows

Now we show our main results for flows generated by the unsupervised RBM. Let us call them the RBM flows. We provide three different types of unsupervised RBMs explained in Sec. 3.1.2. Each of them is trained by a different set of spin configurations generated at a different set of temperatures. We then generate flows of temperature distributions by using these trained RBMs, following the methods of Sec. 3.1.3 and Sec. 3.1.4.

TypeV RBM: Trained by configurations at $T = \{0,\ 0.25, 0.5, \cdots, 6\}$

First, we study type V RBM. The temperature range includes the critical temperature $T_c \simeq 2.27$. The unsupervised RBM will have learned the features of spin configurations at various temperatures.

Using the methods in Sec. 3.1.3, we generate a sequence of reconstructed configurations as in Eq. (3.6). We prepare two different sets of initial configurations. One is a set of configurations at $T = 0$, and another at $T = 6$. These configurations are not used for the trainings of the RBM. Then by using the supervised NN in Sec. 3.1.4, we translate the flow of configurations to a flow of temperature distributions.

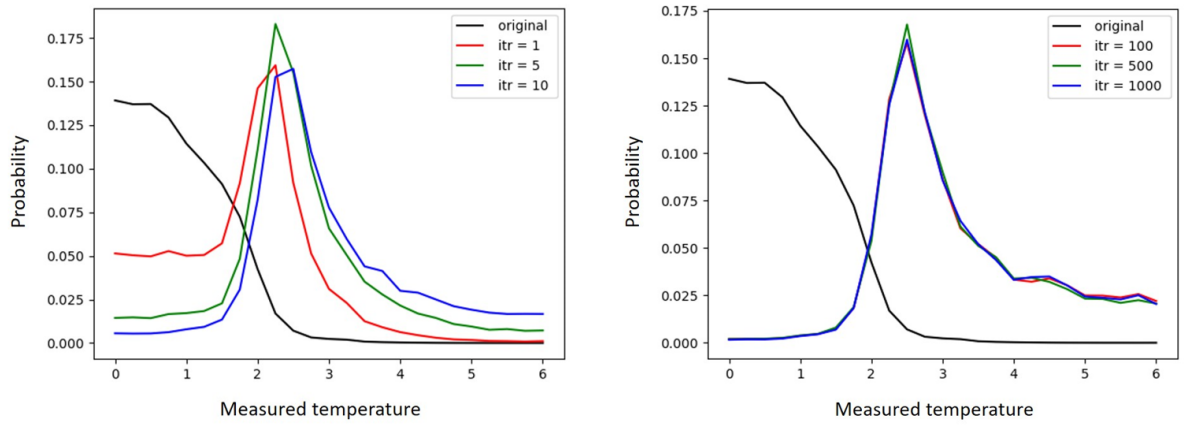


Figure 3.3: Temperature distributions after various numbers of iterations of type V RBM, which is trained by the configurations at $T = 0, 0.25, \ldots, 6$. The original configurations are generated at $T = 0$.

In Figs. 3.3 and 3.4, we plot temperature distributions of configurations generated by iterating RBM reconstruction Sec. 3.1.3 . Fig. 3.3 shows a flow of temperature distributions starting from $T = 0$. Fig. 3.4 starts from $T = 6$. In all the figures, the black lines are the measured temperature

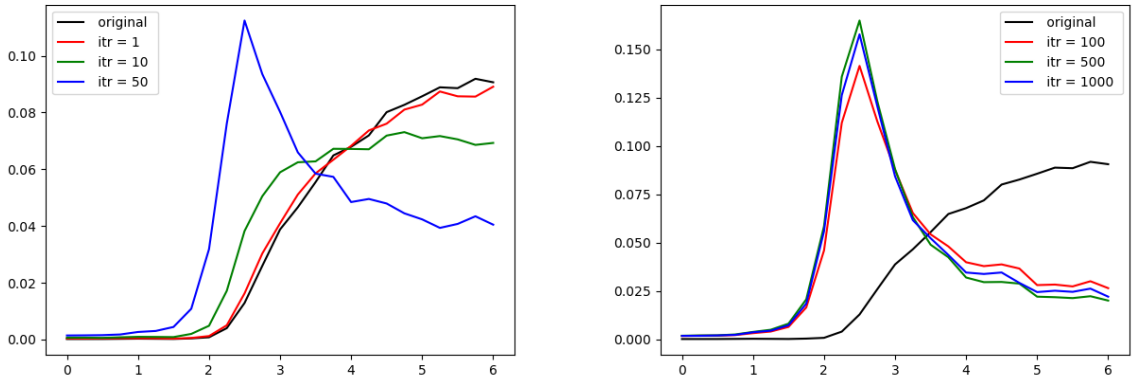


Figure 3.4: Temperature distributions after various numbers of iterations of the same RBM as Fig. 3.3.The original configurations are generated at $T = 6$.

distributions of the original configurations[3]. The colored lines show temperature distributions of the reconstructed configurations $\{v_i^{(n)}\}$ after various numbers of iterations. The "itr" in the legends means the numbers of iterations n by the unsupervised RBM. The left panels show the temperature distributions at small iterations (up to 10 in Fig. 3.3 and 50 in Fig. 3.4), while the right panels are at larger iterations up to 1000. These results indicate that the critical temperature T_c is a stable fixed point of the flows of the type V RBM. Apparently, the results are different from a naive expectation that the RBM may show similar behaviors to the conventional RG flow. Indeed, from whichever temperature $T = 0$ or $T = 6$ we start, the peak of the temperature distributions approaches the critical point ($T_c \simeq 2.27$) as we repeat the iterations by the unsupervised RBM.

In order to confirm the above behavior, we provide another set of configurations at $T = 2.25$ as initial configurations and generate the flow of probability distributions by the same trained RBM. The flow of temperature distributions is shown in Fig. 3.5. We can see that the temperature distribution of the reconstructed configurations remains near the critical point, and never flows away from there[4]. If the process of the unsupervised RBM corresponds to coarse-graining of spin configurations, the temperature distributions of the reconstructed configurations must flow away

[3]Though the original configurations are generated by the MMC method in Sec. 3.1.1at $T = 0$ and $T = 6$, the measured temperature distributions are not sharply peaked at $T = 0$ or $T = 6$.There are two reasons for this broadening of the distributions.One is due to the finiteness of the size of a configuration $N = L \times L = 10 \times 10$. Another is due to the limit of measuring temperature by the NN. If the size of a configuration was infinite and if the ability of discriminating subtle differences of different temperature configurations was limitless, we would have obtained a very sharp peak at the labelled temperature.

[4]We also trained the RBM using configurations of a wider range of temperatures; $T = 0, 0.25, \ldots, 10, \infty$. The results are very similar, and the temperature distributions of reconstructed configurations always approach the critical point $T_c \simeq 2.27$.

from T_c.

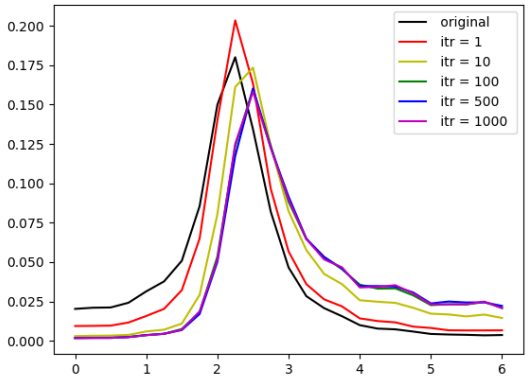


Figure 3.5: Temperature distributions after various numbers of iterations of the same RBM as Figs. 3.3 and 3.4. The original configurations are generated at $T = 2.25$.

So far, in obtaining the above results of Fig. 3.3, Fig. 3.4 and Fig. 3.5, we used an unsupervised RBM with 64 neurons in the hidden layer. We also trained other RBMs with different sizes of the hidden layer, but by the same set of spin configurations. When the size of the hidden layer is smaller than (or equal to) the number of the visible layer, namely $N_h = 100, 81, 64, 36$ or 16, we find that the temperature distribution again approaches the critical point. The difference is that the speed of the flow to approach T_c becomes more rapid (i.e., arrive at T_c by fewer iterations).

In contrast, in the case of the RBM which has more than 100 neurons in the hidden layer, we have different results. Fig. 3.6 shows the case of $N_h = 225$ neurons. Until about ten iterations, the measured temperature distribution behaves similarly to the case of $N_h < 100$, i.e., it approaches the critical temperature. However, afterward it passes the critical point and flows away to higher temperature. In the case of 400 neurons, it moves towards high temperature at faster speed. This behavior suggests that, if the hidden layer has more than a necessary size, the NN tends to learn a lot of noisy fluctuations. We come back to this point later.

Type H/L RBM: Trained by configurations at Higher/Lower temperatures

Next we study type H RBM, which is trained by configurations at higher temperatures $T = 4, 4.25, \ldots, 6$ than $T_c \simeq 2.27$. The results of the flows of temperature distributions are drawn in Fig. 3.7. In this case, even when the hidden layer has less than 100 neurons, the measured temperature passes the critical point and goes away towards higher temperature. This behavior is intuitively understandable. The probability distribution flows towards high temperature because the RBM has learned the features of high temperature configurations. We also find that, if the

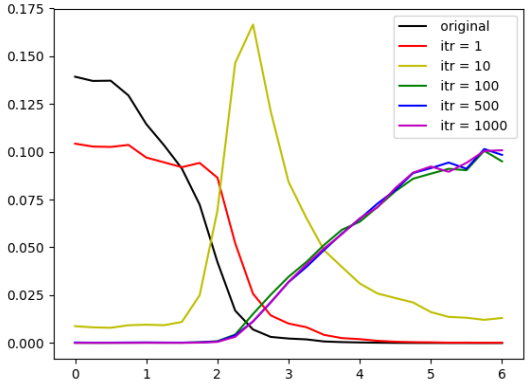


Figure 3.6: Temperature distribution after various numbers of iterations of type V RBM with 225 neurons in the hidden layer. The original configurations are generated at $T = 0$.

number of neurons in the hidden layer is increased, the temperature distributions changes more slowly.

Finally, we study type L RBM, which is trained by configurations only at the lowest temperature $T = 0$. Fig. 3.8 shows the numerical results of flows of type L RBM. Similar to the type H RBM, the measured temperature passes the critical point, but flows towards lower temperature instead of higher temperature. In type L RBM, as far as we have studied, the flow never goes back to higher temperature even for large N_h. This suggests that the RBM does not learn features that are not included in the training data set.

Summary and conjectures

We first summarize the numerical results:

For type V RBM

- When $N_h < 100 = N_v$, the measured temperature T approaches T_c.
- However, for $N_h > 100 = N_v$, the flow eventually goes away towards $T = 6$.
- Speed of flows is slower for a larger N_h.

For type H/L RBM

- For type H/L RBM, T flows towards $T = 6/T = 0$ respectively.
- Speed of flows is slower for a larger N_h.

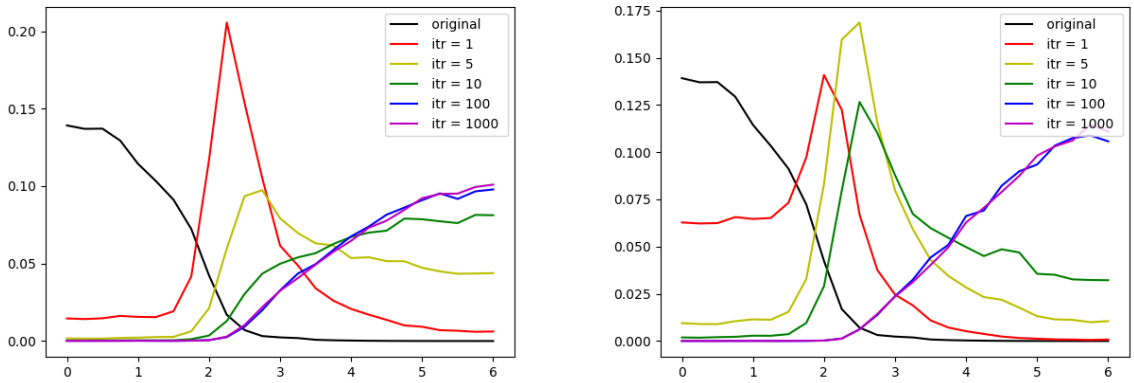


Figure 3.7: Flow of temperature distributions starting from $T = 0$ in type H RBM. RBM is trained by configurations at only $T = 4, 4.25, \ldots, 6$, and the NN has 64 neurons (left) and 225 neurons (right) in the hidden layer. The speed of the flow is slower for the larger sized hidden layer.

Here N_h and N_v are numbers of hidden and visible neurons in the RBM. These behaviors must be reflections of the weights and biases that the unsupervised RBMs have learned in the process of training.

Understanding the above behaviors is equivalent to answering what the unsupervised RBMs have learned in the process of trainings. The most important question will be why the temperature approaches T_c in the type V RBM with $N_h < N_v$, instead of, e.g., broadening over the whole regions of temperature from $T = 0$ to $T = 6$. Note that we did not teach the NN either about the critical temperature or the presence of the phase transition. We just have trained the NN by configurations at various temperatures, from $T = 0$ to $T = 6$. Nevertheless the numerical simulations show that the temperature distributions are peaked at T_c after some iterations of the RBM reconstruction. Thus we are forced to conclude that the RBM automatically learns features specific to the critical temperature T_c .

An important feature at T_c is the scale invariance. We have generated spin configurations at various temperatures by the Monte Carlo method, and each configuration has typical fluctuations specific to each temperature. At very high temperature, fluctuations are almost random at each lattice site and there are no correlations between spins at distant positions. At lower temperature, they become correlated: the correlation length becomes larger as $T \to T_c$ and diverges at T_c. On the other hand, at $T \ll T_c$, spins are clustered and in each domain all spins take $\sigma_{x,y} = +1$ or $\sigma_{x,y} = -1$. At low temperature configurations have only big clusters, and as temperature increases small sized clusters appear. At T_c, spin configurations become to have clusters of various sizes in a scale-invariant way. Now let us come back to the question why the type V RBM generates a flow

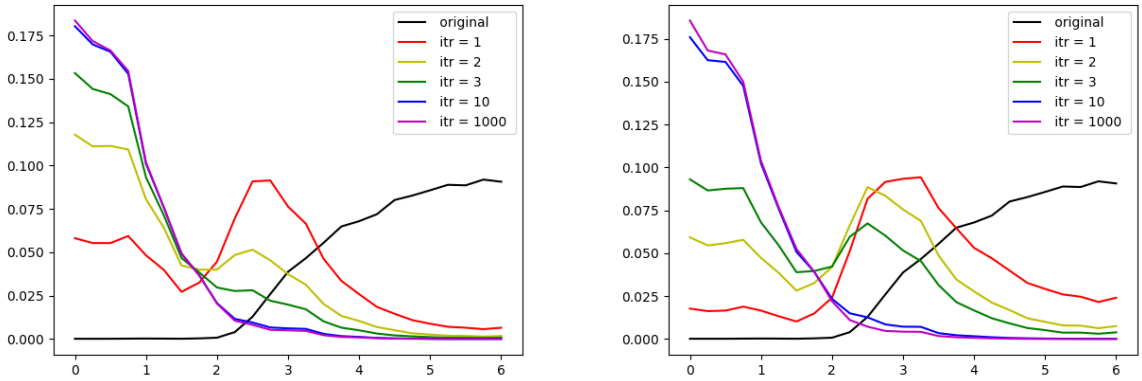


Figure 3.8: Flow of temperature distributions starting from $T = 6$ in type L RBM. RBM is trained by configurations at only $T = 0$, and the NN has 64 neurons (left) and 225 neurons (right) in the hidden layer.

approaching T_c and does not randomize to broaden the temperature distribution over the whole regions. We have trained the type V RBM by using configurations at various temperatures with different sized clusters, and in the process the machine must have simultaneously acquired features at various temperatures. Consequently the process of the RBM reconstruction adds various features that the machine has learned to a reconstructed configuration. If only a single feature at a specific temperature was added to the reconstructed configuration, the distribution would become to have a peak at this temperature. But it cannot happen because various features of different temperatures will be added to a single configuration by iterations of reconstruction processes. Then one may ask if there is a configuration that is stable under additions of features at various different T.

Our first conjecture about this question is that a set of configurations at T_c is a stabilizer (and even more an attractor) of the type V RBM with $N_h \geq N_v$. It must be due to the scale invariant properties of the configurations at T_c. Namely, since these configurations are scale invariant, they have all the features of various temperatures simultaneously, and consequently they can be the stabilizer of this RBM. This sounds plausible since the scale invariance means that the configurations have various different characteristic length scales. However, we notice that this doesn't mean that the RBM has forgotten the features of configurations away from the critical point. Rather, it means that the RBM has learned features of all temperatures simultaneously. This doesn't mean either that the configurations at $T = T_c$ have especially affected strong influence on the machine in the process of training. It can be confirmed as follows. Suppose we have trained a RBM by configurations at temperatures excluding $T = T_c$, namely trained by configurations at all temperatures except $T = 2.25$ and 2.5 . We found in the numerical simulations that the RBM generates a flow towards the critical point, though we did not provide configurations at $T = T_c$.

Therefore, we can say that the type V RBM has learned the features at all the temperatures and that configurations at T_c are special because they contain all the features of various temperatures in the configurations.

Our second conjecture, which is related to the behavior of the type V RBM with $N_h > N_v$, is that RBMs with unnecessarily large sized hidden layer tend to learn lots of irrelevant features. In the present case, they are noisy fluctuations of configurations at high temperatures. High temperature configurations have only short distance correlations, whose behavior is similar to the typical behavior of noise. The conjecture will be partially supported by the similarity of the RBM flows between the type V RBM with $N_h > N_v$ and the type H RBM. Namely, both RBM flows flow away to $T = 6$. The similarity indicates that the NN with a larger number of N_h may have learned too much noise-like features of configurations at higher temperatures. The above considerations will teach us that the moderate size of the hidden layer, $N_h < N_v$, is the most efficient to properly extract the features.

3.3 Analysis of the weight matrix

In the previous section, we showed the numerical results of flows generated by unsupervised RBMs, and proposed a conjecture that the $T = T_c$ configuration is a stabilizer of type V RBM. In this section, to further understand the theoretical basis of feature extractions, we analyze the weight matrices and biases of the trained RBMs.

3.3.1 Singular value decomposition (SVD) of weights

All the information that the machine has learned is contained in the weights W_{ia} and the biases b_i, c_a. Since the biases have typically smaller values than the weights (at least in the present situations), we will concentrate on the weight matrix W_{ia} ($i = 1, \ldots, N_v (= L^2)$; $a = 1, \ldots, N_h$) in the following.

Let us first note that the weight matrix W_{ia} transforms as

$$W_{ia} \;\rightarrow\; \tilde{W}_{ia} = \sum_{j,b} U_{ij} W_{jb} (V^t)_{ba} \tag{3.7}$$

under transformations[5] of exchanging the basis of neurons in the visible layer (U_{ij}) and in the hidden layer (V_{ab}). Since the choice of basis in the hidden layer is arbitrary, relevant information in the visible layer is stored in a combination of W_{ia} that is invariant under transformations of V_{ab}.

[5]Since spin variables on each lattice site are restricted to take values ± 1, the matrices, U_{ij} and V_{ab}, are elements of the symmetric group, not the orthogonal Lie group.

The simplest combination is a product,

$$(WW^t)_{ij} = \sum_a W_{ia} W_{ja} \,. \tag{3.8}$$

Its size is $N_v \times N_v = 100 \times 100$, and independent of the size of N_h. But its property depends on N_h because the rank of WW^t must be always smaller than $\min(N_v, N_h)$. Thus, if $N_h < N_v$, the weight matrix is strongly constrained; e.g. a unit matrix $WW^t = 1$ is not allowed.

Though it is not the only relevant combination, this simplest product (3.8) will play an important role in the dynamics of the flow generated by the RBM. It can be shown as follows. If the biases are ignored, the conditional probability (2.19) and the expectation value (2.21) for h_a in the background of v_i become

$$P(h_a|\mathbf{v}) = \frac{e^{\sum_i v_i W_{ia} h_a}}{2\cosh\left(\sum_i v_i W_{ia}\right)} \,, \qquad \langle h_a \rangle = \tanh\left(\sum_i v_i W_{ia}\right) \,. \tag{3.9}$$

In $P(h_a|\mathbf{v})$, a combination $\sum_i v_i W_{ia} =: B_a$ can be regarded as an external magnetic field for h_a. Thus, these two variables, B_a and h_a, tend to correlate with each other. Namely, the probability $P(h_a|\mathbf{v})$ becomes larger when they have the same signs. Moreover, for $|B_a| < 1$, $\langle h_a \rangle$ is approximated by B_a and we can roughly identify these two variables,

$$h_a \sim B_a := \sum_i v_i W_{ia} \,. \tag{3.10}$$

It is usually not a good approximation since weights can have larger values. For a large value of $|B_a|$, h_a is saturated at $h_a = B_a/|B_a|$.

Suppose that the input configuration is given by $\{v_i^{(0)} = \sigma_i\}$. If Eq. (3.10) is a good approximation, we have $h_a^{(0)} = B_a^{(0)} = \sum_i v_i^{(0)} W_{ia}$. Then the conditional probability in the background of $\{h_a^{(0)}\}$ with $c_a = 0$

$$P(\mathbf{v}|\mathbf{h}^{(1)}) = \prod_i \frac{e^{\sum_a v_i W_{ia} h_a^{(1)}}}{2\cosh\left(\sum_a W_{ia} h_a^{(1)}\right)} \tag{3.11}$$

is approximated as

$$P(\mathbf{v}|\mathbf{h}^{(0)}) \sim \prod_i \frac{e^{\sum_a v_i W_{ia} \sum_j W_{ja} v_j^{(0)}}}{2\cosh\left(\sum_a W_{ia} \sum_j W_{ja} v_j^{(0)}\right)} = \prod_i \frac{e^{\sum_j v_i (WW^t)_{ij} v_j^{(0)}}}{2\cosh\left(\sum_j (WW^t)_{ij} v_j^{(0)}\right)} \,. \tag{3.12}$$

The RBM learns the input data $\mathbf{v}^{(0)}$ so that the probability distribution reproduces the probability

distribution of the initial data, $q(\{v_i\}) = \frac{1}{N}\sum_{n=1}^{N}\delta(v_i - \sigma_{n,i})$. Therefore, the training of the RBM tends to tune the weight matrix W so as to enhance $v_i^{(0)}(WW^t)_{ij}v_j^{(0)}$. This means that W will reflect correlations between spins on the two dimensional lattice.

In this simplified discussion, it is expected that all of the information that the RBM learns is included in the combination WW^t. Of course, most importantly, we neglected the nonlinear property of tanh function and it cannot be justified. Nevertheless, we will find below that the analysis on WW^t is useful to understand how the RBM works.

3.3.2 Spin correlations in WW^t

In Fig. 3.9, we plot magnitudes of matrix elements of the 100×100 matrices WW^t. The variables in the visible layer are spin configuration $\sigma_{x,y}$ with $x, y = 1, \ldots, L = 10$, and we lined them up as $(\sigma_{1,1}, \sigma_{1,2}, \ldots, \sigma_{1,L}, \sigma_{2,1}, \ldots, \sigma_{2,L}, \sigma_{3,1}, \ldots, \sigma_{L,L})$, which we call σ_i with $i = 1, \ldots, L^2$. In the following, we mostly discuss type V RBM unless otherwise stated.

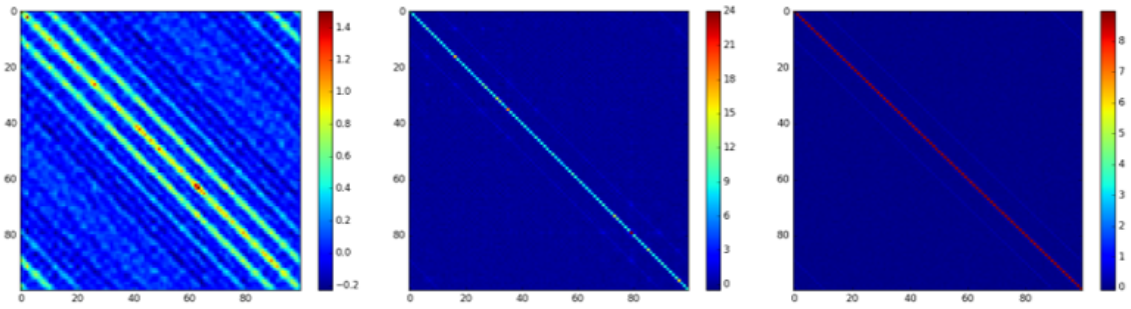


Figure 3.9: Elements of WW^t when the hidden layer has 16 (left), 100 (center), 400 (right) neurons.

As discussed above, the product of weight matrices WW^t reflect correlations between spin variables of the input configurations used for the training of the RBM. The strongest correlation in $v_i^{(0)}(WW^t)_{ij}v_j^{(0)}$ is of course the diagonal part, $i = j$. Thus we expect that the matrix WW^t will have large diagonal components. Indeed, such behavior can be seen in Fig. 3.9. In particular, for $N_h = 400 > N_v = 100$ (the rightmost figure), WW^t is clearly close to a diagonal matrix. It is also true for the case of $N_h = 100 = N_v$. However, for $N_h = 16 < N_v = 100$, off-diagonal components of $(WW^t)_{ij}$ have also relatively larger values, in particular, at $j = i+1$ and $j = i+2$.[6] It is related to the fact that the rank of WW^t is smaller than N_v and then WW^t cannot be a unit matrix. Thus, only much less information can be stored in the weight matrix for a smaller number of hidden

[6]Off-diagonal components of $j = i + L$ or $j = i + 2L$ are also large, which corresponds to correlations along y-direction. Large off-diagonal components at $j = i + 1$ and $j = i + 2$ mean correlations along x-direction.

neurons. But it is interesting and a bit surprising that the RBM with fewer hidden neurons seems to learn more efficiently the off-diagonal correlations between spins.

In order to see how this correlation of WW^t is related to the expected correlation of spins, we study other types of the RBMs trained by configurations at a specific (single) temperature. In Fig. 3.10, we plot behaviors of the off-diagonal components of WW^t for various RBMs. Each RBM is trained by the configurations at each temperature $T = 0$ (type L), $T = 2$, $T = 3$ and $T = 6$. The size of the hidden layer is $N_h = 16$. For comparison, we also plot the case of type V RBM.

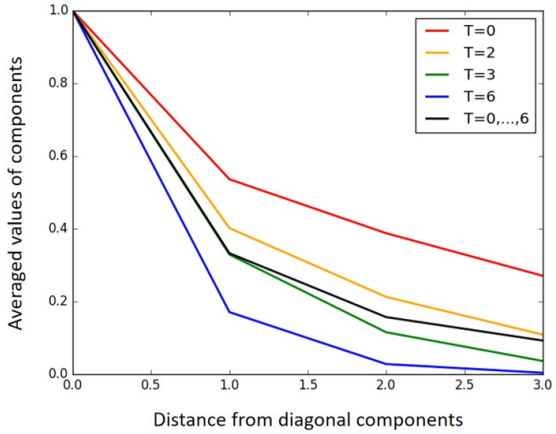


Figure 3.10: The averaged values of off-diagonal components of WW^t (divided by diagonal components). We use RBMs learning the configurations at a specific temperature $T = 0, 2, 3, 6$, and also type V RBM learning all the temperatures $T = 0, \ldots, 6$.

The figure shows that the correlation of WW^t decays more rapidly at higher temperature, which is consistent with the expected behavior of spin correlations. Therefore, the RBM seems to learn correctly about the correlation length, or the size of clusters, which becomes smaller at higher temperature.

In addition, we find that, for type V RBM which learns all temperatures $T = 0, \ldots, 6$, the off-diagonal elements are damped between $T = 2$ case and $T = 3$ case. This should mean that type V RBM learns well the features of configurations at around the critical temperature $T_c = 2.27$. Therefore, even in this simple analysis, we can get a supporting staff for our conjecture in Sec. 3.2.

3.3.3 Magnetization and SVD

Information of the weight matrix W can be inferred by using the method of the singular value decomposition (See, e.g., [21, 25]). Suppose that the matrix WW^t has eigenvalues λ_a $(a = 1, \ldots, N_v)$

with corresponding eigenvectors $\mathbf{u}_a$;

$$WW^t\mathbf{u}_a = \lambda_a \mathbf{u}_a \,. \tag{3.13}$$

Decomposing an input configuration vector $\mathbf{v}^{(0)}$ in terms of the eigenvectors $\mathbf{u}_a$ as $\mathbf{v}^{(0)} = \sum_a c_a \mathbf{u}_a$ with a normalization condition $\sum_a (c_a)^2 = 1$, we can rewrite $\mathbf{v}^{(0)T} WW^t \mathbf{v}^{(0)}$ as

$$\mathbf{v}^{(0)T} WW^t \mathbf{v}^{(0)} = \sum_a c_a^2 \lambda_a \,. \tag{3.14}$$

Thus, if a vector $\mathbf{v}^{(0)}$ contains more components with larger eigenvalues of WW^t, the quantity $\mathbf{v}^{(0)t} WW^t \mathbf{v}^{(0)}$ becomes larger. Fig. 3.11 shows averaged values of $\mathbf{v}^{(0)t} WW^t \mathbf{v}^{(0)}$ over the 1000 configurations $\{\mathbf{v}_n^{(0)}\}_{n=1,\cdots,1000}$ at each temperature.

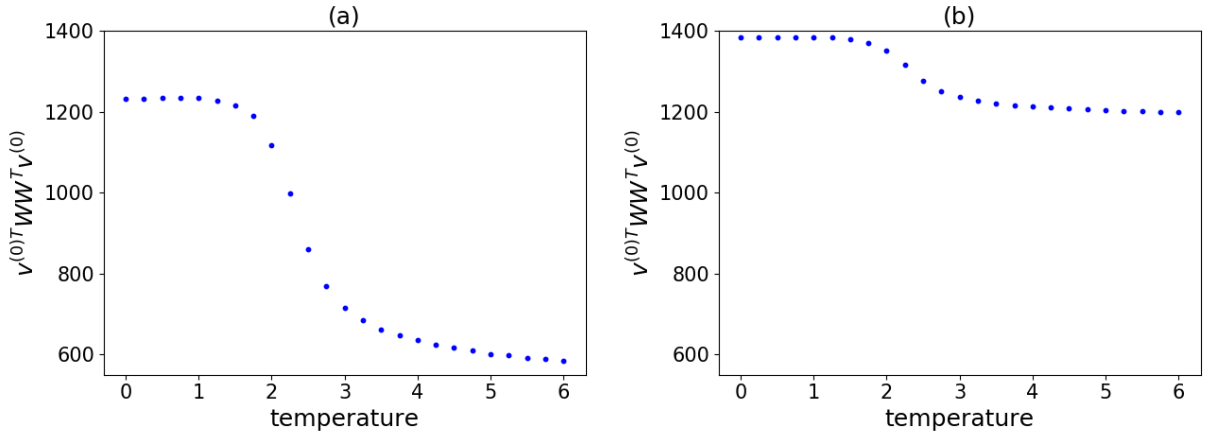


Figure 3.11: Averaged values of $v^{(0)t} WW^t v^{(0)}$ for the configurations $\{v^{(0)}\}$ at each temperature. We use the weight matrix W of type V RBM with 64 hidden neurons (left) and 225 hidden neurons (right).

The figures apparently show a big change near the critical point, and behave like the magnetization. It indicates that the principal eigenvectors[7] with large eigenvalues are related to the magnetization, and the information about the phase transition is surely imported in the weight matrix. Moreover, we find that the RBM with more hidden neurons has much larger values of $v^{(0)t} WW^t v^{(0)}$ at high temperature. This means the principle eigenvectors of such RBM grasp well features of the configurations at high temperature, compared with the RBM with fewer hidden neurons. We can check this tendency in Fig. 3.12 in more detail.

[7]We have investigated the properties of eigenvectors whether they are related to the magnetization vector. However, as far as we saw each eigenvector, we could not get any physically reasonable pictures. We want to come back to this problem in future works.

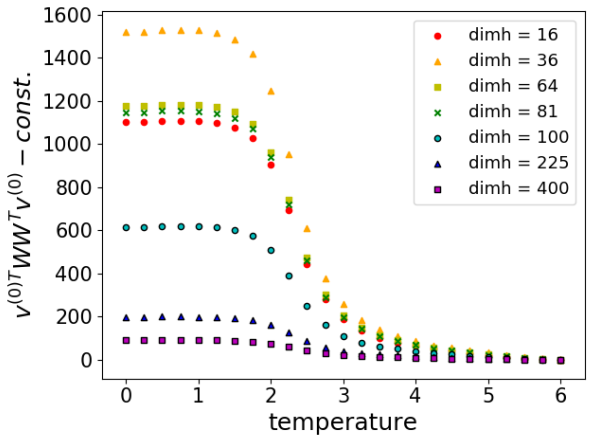


Figure 3.12: Averaged values of $v^{(0)t}WW^tv^{(0)}$ at each temperature. The legend shows the number of hidden neurons N_h. In all the cases, the values at $T = 6$ are subtracted.

We find that the RBM with small N_h has much smaller value at high temperature than at low temperature. The RBM with large N_h, on the other hand, has only little change in the value. This difference must cause different behaviors in the reconstruction of configurations, as shown in Fig. 3.3 ($N_h = 64$) and Fig. 3.6 ($N_h = 225$): The former approaches the critical temperature T_c, while the latter goes to the higher temperature. Therefore, it implies that the RBM with small N_h never fails to learn about higher temperature, but the RBM with large N_h learns too much about higher temperature. We will come back to this point in the next subsection.

3.3.4 Singular value spectrum and information stored in W

Finally, we study the eigenvalue spectrum λ_a of the matrix WW^t. Figs. 3.13 and 3.14 show the eigenvalues in the descending order.

In Fig. 3.13, the red dots shows the eigenvalues of type V RBM trained by configurations at all the temperatures ($T = 0, 0.25, \ldots, 6$), while the blue line is the eigenvalues of type L RBM (only $T = 0$).

These are obviously different. For type L RBM, only several eigenvalues are especially large, and the rest is apparently smaller. On the other hand, for type V RBM, the eigenvalues decrease gradually and there are no jumps or big distinctions between larger and smaller eigenvalues. The behavior indicates that, in type L RBM, the weight matrix holds only small relevant information and only a small number of neurons is required in the hidden layer. In type V RBM, however, since it is trained by configurations at various different temperatures, all the eigenvectors are equally utilized to represent relevant features of spin configurations at various temperatures. Namely, in order to learn features of a wide range of temperatures, larger number of neurons in the hidden layer

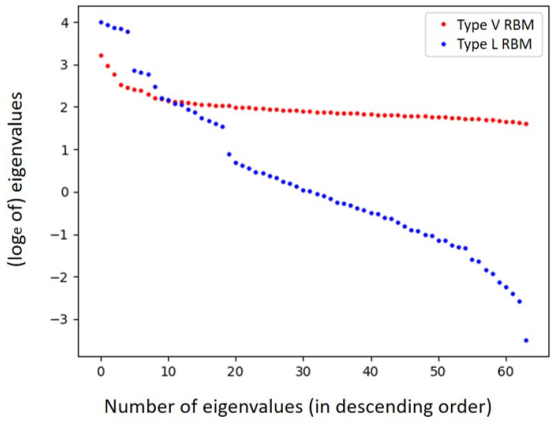


Figure 3.13: Eigenvalues of WW^t for type V RBM (red) and type L RBM (blue). Both RBMs have 64 neurons in the hidden layer.

are necessary. Using such larger degrees of freedom, the weight matrix has learned configurations with various characteristic scales at various temperatures so that the RBM can grasp rich properties of these configurations. This shows that, as we discussed at the end of Sec. 3.2, type V RBM never forgets the configurations away from the critical temperature but learns about all the temperatures.

The difference of the eigenvalues between type V and type L is also phrased that type V has a scale invariant eigenvalue spectrum[8]. On the other hand, type L has such scale invariant behavior only for a few eigenvalues. This must be related to our previous numerical results, shown in Figs. 3.3, 3.4 and 3.5, that type V RBM generates a flow toward the critical point where the configurations have scale invariance.

Finally, let us look at a difference of eigenvalue spectrum between different number of hidden neurons N_h, shown in Fig. 3.14. If the RBM has more neurons than $N_v = 100$ in the hidden layer, the eigenvalues tend to have similar values. It is compared to the case with a smaller N_h in which the eigenvalues are more hierarchical between large and small eigenvalues. This difference indicates that the RBM with larger N_h than N_v seems to have learned unnecessary noises of input configurations. In other words, it has learned too much irrelevant features which are especially specific to configurations at higher temperature. This corresponds to what we pointed out at the end of Sec. 3.3.3. It may explain our numerical results shown in Fig. 3.6, which is apparently different from Fig. 3.3, that the flow first approaches the critical point but passes there and eventually goes away to higher temperature.

To summarize, we find that type V RBM with smaller N_h than N_v can learn configurations at all the temperature, without learning too much features at a specific temperature. This RBM

[8]Exactly speaking, it is not completely scale invariant, but compared to type L, there is at least no jump between larger and smaller eigenvalues.

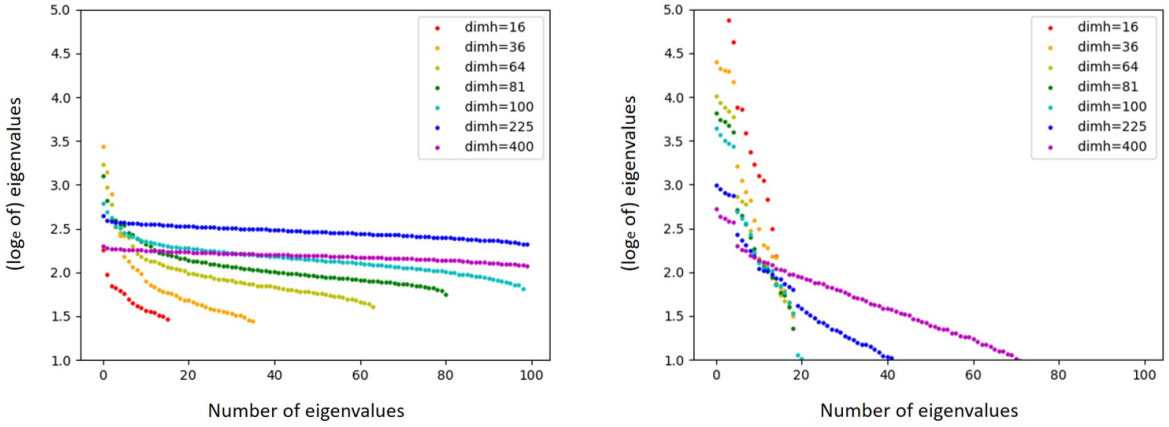


Figure 3.14: Eigenvalues of WW^t for type V RBM (left) and type L RBM (right). The legend shows the number of hidden neurons N_h.

learns them using all the neurons in the hidden layer, and then has a scale invariant spectrum of eigenvalues of the weight matrix. As a result, after a lot of iterations of this RBM, all the configurations are transformed into the configurations at near the critical temperature T_c. This flow of transformation is oriented in the completely opposite direction to the RG flow.

Chapter 4

Autoencoder

Is it universal that the fixed point of the flow generated by an unsupervised network which learns spin configurations at various temperatures is critical point? In order to check this we train Autoencoder (AE) and generate temperature flow by AE.

In Sec.4.1, we explain the setup of AE and how we get the temperature flow by trained AE. In Sec.4.2, we show the numerical results for fixed point temperatures of 1-layer AE. We study the relation between the SVD of weight matrix and the progress of training of AE.

4.1 Setup

4.1.1 Setup of AE

We explain the setup of AE. We train AE by using the algorithm explained in Sec.2.2.2 . We prepare a 1-layer AE and its structure is shown in Fig.4.1. We take sigmoid function $\sigma(x)$ as activation function defined by

$$\sigma(x) := \frac{1}{1+e^{-x}} \,. \tag{4.1}$$

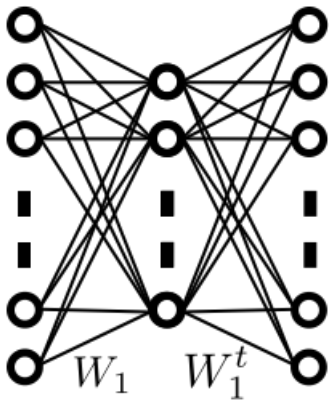


Figure 4.1: Structure of 1-layer AE

Training data set is composed of spin configurations with size 64×64 and temperatures $T = 1.0, 1.1, 1.2, \cdots, 5.0$ generated by MC method. In AE case, we encode spin variables as $v_i = 0, 1$. We train AEs which have various numbers of hidden units through datasets which are composed of various combinations of temperatures.

4.1.2 Generation of AE flows

By using trained AE, we can generate the temperature flow. Let us write the i-th component of output layer as

$$y_i = \sigma\left(\sum_\alpha W_{i\alpha} h_\alpha + b_i\right), \tag{4.2}$$

where h_α is the output of the hidden layer and W_{ij} and b_i are the last weight matrix and bias respectively. Then we get the new spin configuration $\mathbf{v}(1) = (v(1)_i)$ by

$$v(1)_i = \begin{cases} 1 & (y_i \geq \frac{1}{2}) \\ 0 & (y_i < \frac{1}{2}), \end{cases} \tag{4.3}$$

because AE is trained so as to reconstruct the same images as input images. We can use this $\mathbf{v}(1)$ as a new input of AE and we get $\mathbf{v}(2)$. By repeating this process, we get the flow of spin configurations

$$\mathbf{v}(0) \to \mathbf{v}(1) \to \mathbf{v}(2) \to \cdots \to \mathbf{v}(n). \tag{4.4}$$

As in RBM case, we can translate this configuration flow to temperature flow by a supervised network trained so as to estimate the temperature of spin configurations correctly.

4.1.3 Temperature measurement by a supervised-learning NN

Let us see the results of the NN for measuring temperatures. In the left side of Fig. 4.2, we plot the behavior of the loss function (2.14) as we iterate the renewal of the weights and biases. The

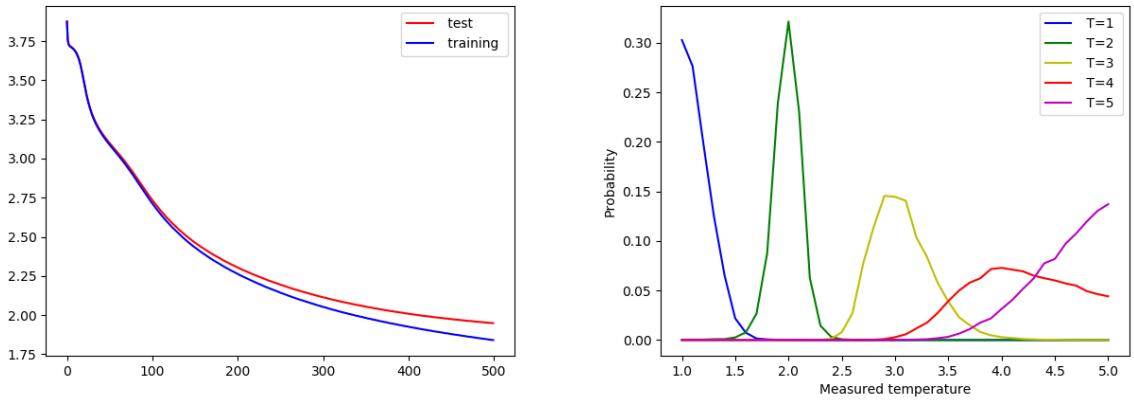


Figure 4.2: (Left) Training error and test error up to 5000 epochs. (Right) Probability distribution of measured temperature when the configurations at $T = 1, 2, 3, 4, 5$ are input.

blue line shows the training error, namely values of the loss function after iterations of training using the 25000 configurations. It is continuously decreasing, even after 5000 epochs.

On the right side of Fig. 3.2 we show the probability distribution of temperature this NN measures. Here we input the configurations at $T = 1, 2, 3, 4, 5$ which are not used for the training. Thus we can confirm the NN can correctly measure the temperature of configurations, even if it does not learn those configurations.

4.2 Numerical results of 1-layer AE

4.2.1 Fixed point temperatures

Using the methods in Sec. 4.1.2, we generate a sequence of reconstructed configurations. We prepare two different sets of initial configurations. These configurations are not used for the trainings of the RBM. Then by using the supervised NN in Sec. 4.1.3, we translate the flow of configurations to a flow of temperature distributions.

AE trained by various temperatures from low to high $T = 1.0, 1.1, \cdots, 5.0$

First, we show the results of AE trained by various temperatures corresponding to type V in Sec.3.1.2. The temperatures are $T = 1.0, 1.1, 1.2, \cdots, 4.9, 5.0$. The number of configurations at each temperature is 500.

In Fig. 4.3, we plot temperature distributions of configurations generated by AE trained by 1000 epochs. The left panel shows a flow of temperature distributions starting from $T = 1$. The right panel starts from $T = 5$. In all the figures, the black lines are the measured temperature distributions of the original configurations. The colored lines show temperature distributions of the reconstructed configurations $\{v_i^{(n)}\}$ after various numbers of iterations. The number of iterations n is shown in the legends. These results indicate that the critical temperature T_c is a stable fixed point of the flows of this AE.

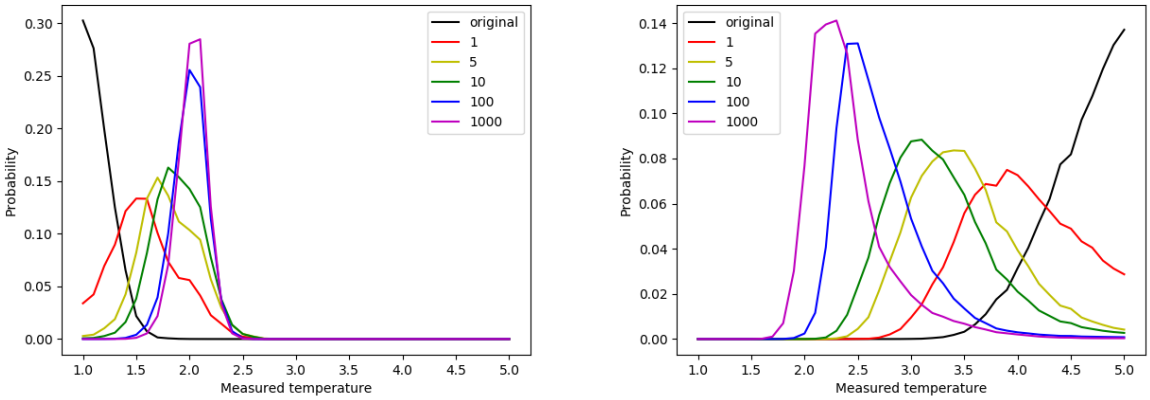


Figure 4.3: Temperature distributions after various numbers of iterations of AE trained by 1000 epochs. The training data set is composed of the configurations at $T = 1, 1.1, 1.2, \ldots, 5$. In the left panel, the original configurations are generated at $T = 1$.In the right panel, the original configurations are generated at $T = 5$.

In Fig. 4.4, we plot temperature distributions of configurations generated by AE trained by 5000 epochs. The left panel shows a flow of temperature distributions starting from $T = 1$. The right panel starts from $T = 5$. In this case, a stable fixed point of the flows of this AE is lower than T_c.

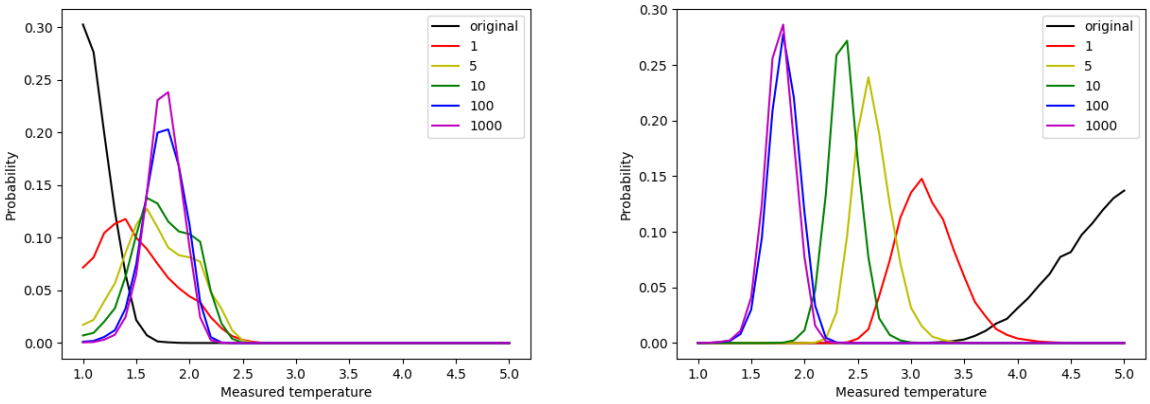


Figure 4.4: Temperature distributions after various numbers of iterations of AE trained by 5000 epochs. The Training data set is composed of the configurations at $T = 1, 1.1, 1.2, \ldots, 5$. In the left panel, the original configurations are generated at $T = 1$. In the right panel, the original configurations are generated at $T = 5$.

Fig.4.5 shows how fixed point temperature depends on the number of hidden units and training epochs. In Fig.4.5, the horizontal axis is the number of hidden units and the vertical axis is the measured temperature of fixed point. Blue dots show the results of AE trained by 1000 epochs. Red dots show the results of AE trained by 5000 epochs. In AE trained by 1000 epochs, its fixed point temperature monotonically increases and can be higher than T_c even if the number of hidden units is smaller than visible units. On the other hand, in AE trained by 5000 epochs, its fixed point temperature is lower than T_c.

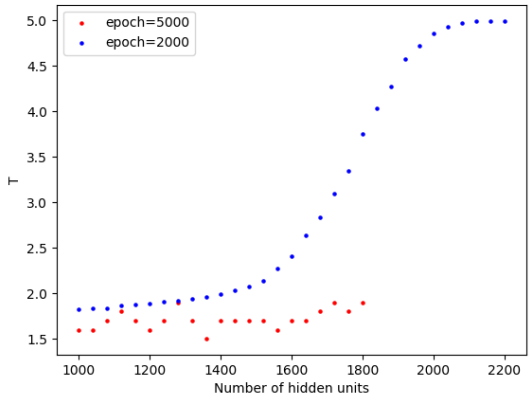


Figure 4.5: Temperatures of fixed point of the flows generated by AEs which have different numbers of hidden units.

AE trained by narrow range temperatures

We also prepare data sets corresponding to type H and type L in Sec.3.1.2. Type H is composed of temperatures $T = 4.6, 3.7, \cdots, 5.0$. Type H is composed of temperatures $T = 1.0, 1.1, \cdots, 1.5$. The number of configurations at each temperature is 1500.

The results of the flows by type H and type L AE are drawn in Fig. 4.6. In both cases, hidden layer has 1600 units. The left panel of Fig. 4.6 is the result of type H AE. In this case, the measured temperature starts from $T = 1.0$, passes the critical point and goes away towards higher temperature as in type H RBM. The right panel of Fig. 4.6 is the result of type L AE. In this case, the measured temperature starts from $T = 5.0$, passes the critical point and goes away towards lower temperature as in type L RBM.

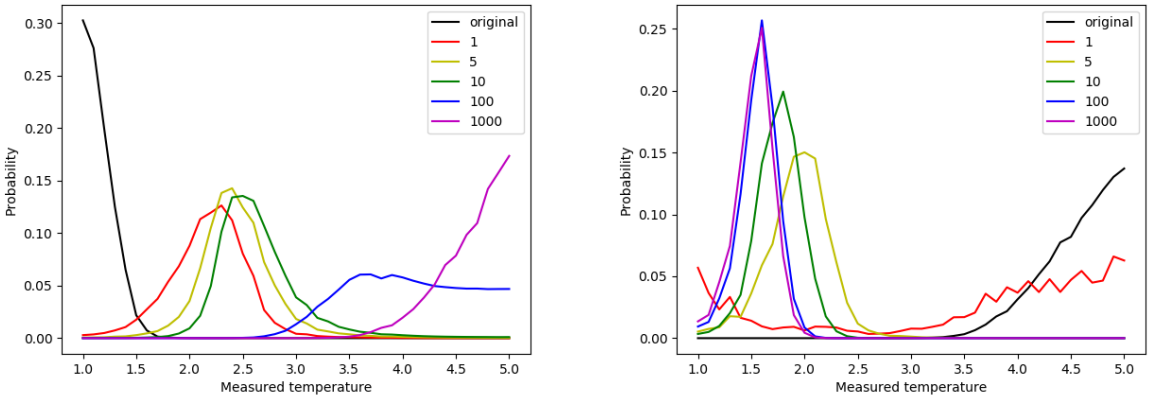


Figure 4.6: Temperature distributions after various numbers of iterations of type H AE (left) and type L AE (right). Type H AE is trained at $T = 4.6, 4.7, \cdots, 5.0$. Type L AE is trained at $T = 1.0, 1.1, \cdots, 1.5$.

Finally, we show the flows by AE trained around critical temperature $T = 2.2, 2.3$. In Fig. 4.7, we plot temperature distributions. The left panel shows a flow of temperature distributions starting from $T = 1$. The right panel starts from $T = 5$. These results indicate that the critical temperature T_c is a stable fixed point of the flows of this AE.

These results suggest that AE can learn the specific temperature of the data set. The fixed point of AE flow corresponds to the feature which AE learns.

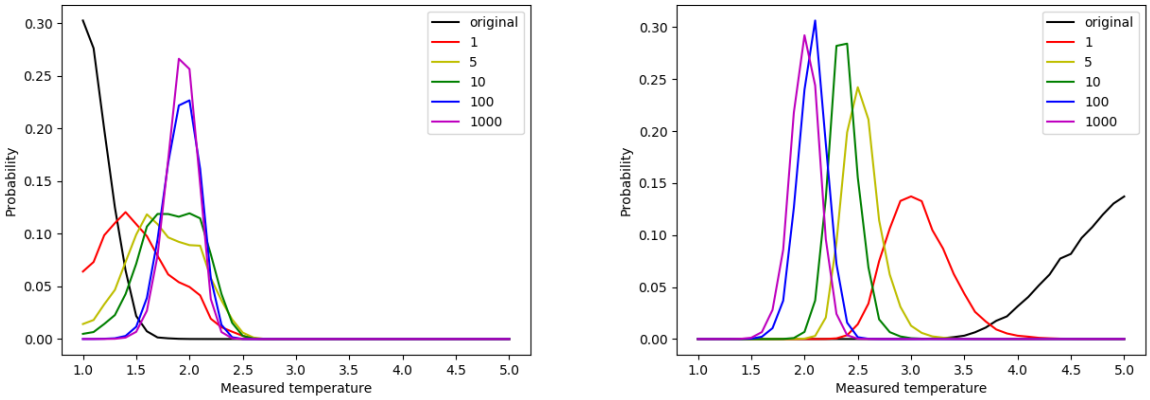


Figure 4.7: Temperature distributions after various numbers of iterations of AE trained at $T = 2.2, 2.3$. In the left panel, the original configurations are generated at $T = 1$. In the right panel, the original configurations are generated at $T = 5$.

4.2.2 Singular value spectrum of weight matrices

We show a singular value spectrum of weight matrices of 1-layer AE in Fig.4.8. The left panel is the result of AE trained by 1000 epochs. The right panel is AE trained by 5000 epochs. The number of hidden units is 1600 in both cases. The spectrum of 5000 epochs case has larger gapped structures than 1000 epochs case.

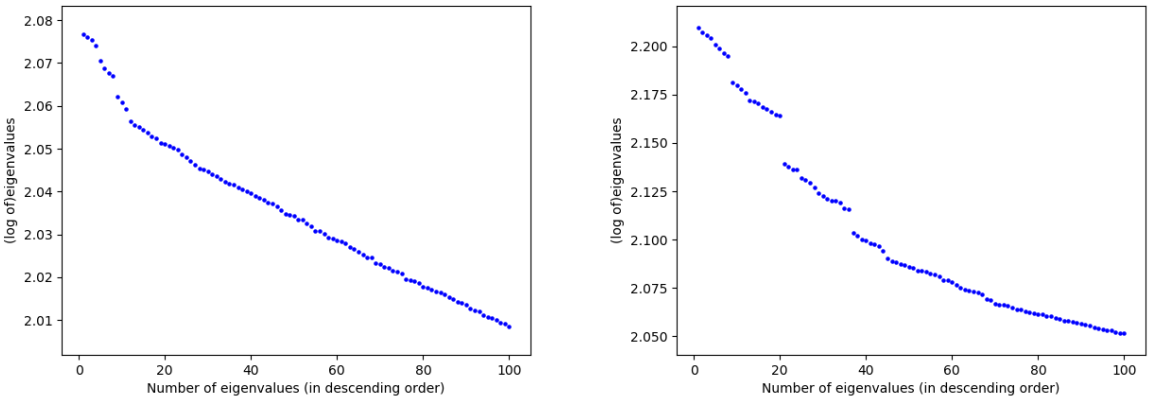


Figure 4.8: Singular values of weights of AE trained by 1000 epochs (left). Singular values of weights of AE trained by 5000 epochs (right).

The gapped structure of the spectrum relates to fixed point temperature. The spectrum of AE

of which fixed temperature is less than T_c has gapped structure. As the number of hidden units, this gap becomes smaller and when this gap vanishes the fixed point of AE flow is the critical point. The right panel of Fig.4.9 clarifies this fact. We plot the difference between the 8-th and the 9-th singular value. This difference decreases as the number of hidden units gets closer to zero around $N_h = 1600$ in which fixed point temperature is T_c (See the right panel of Fig.4.9).

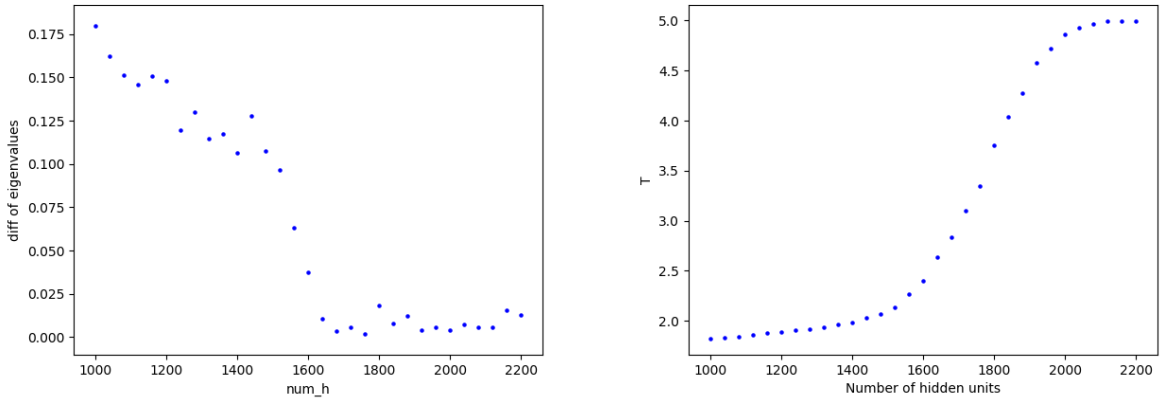
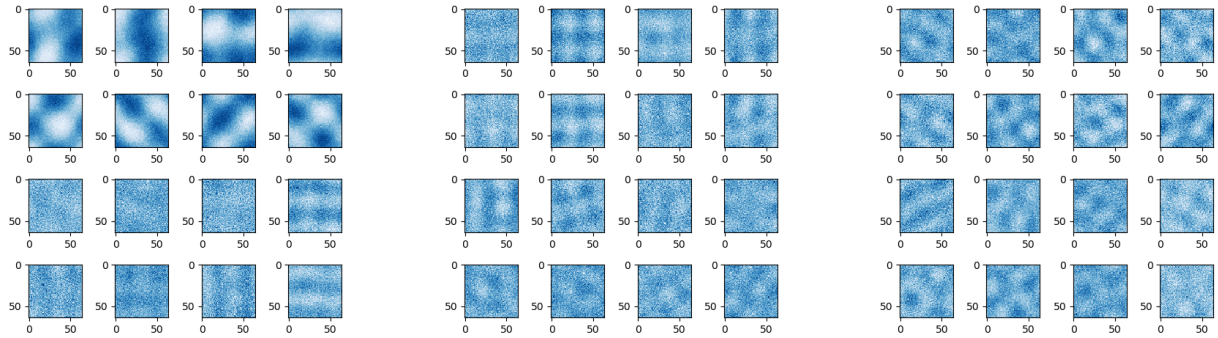


Figure 4.9: The difference between the 8-th and the 9-th singular value (left). Temperatures of fixed point of the flows generated by AEs which have different numbers of hidden units (right).

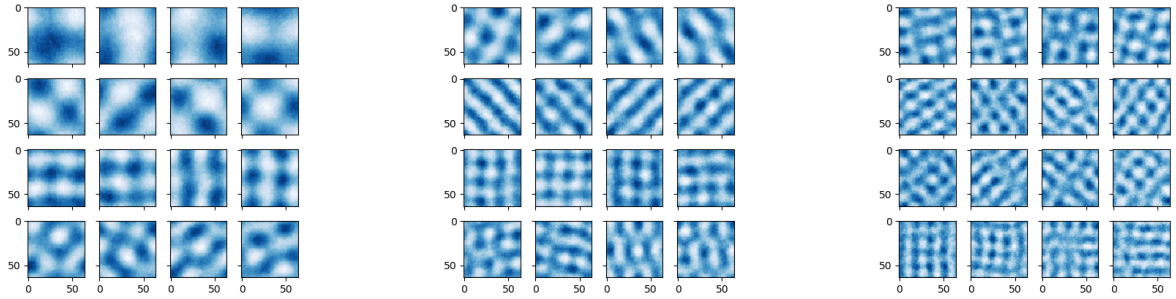
4.2.3 Eigenvectors of WW^t

Next, we show eigenvectors of WW^t. Figs.4.10 and 4.11 show eigenvectors of AE in descending order of their eigenvalues. Fig.4.10 is AE trained by 1000 epochs and Fig.4.11 is AE trained by 5000 epochs. The 5000 epochs case has more eigenvectors which have clear structures than 1000 epochs cases. In 1000 epochs case, clear structures vanish in first six eigenvectors. On the other hand, in 5000 epochs case, the clear structure remains even in the 48th eigenvector. This means that the structure of eigenvector of WW^t reflects the progress of the training.



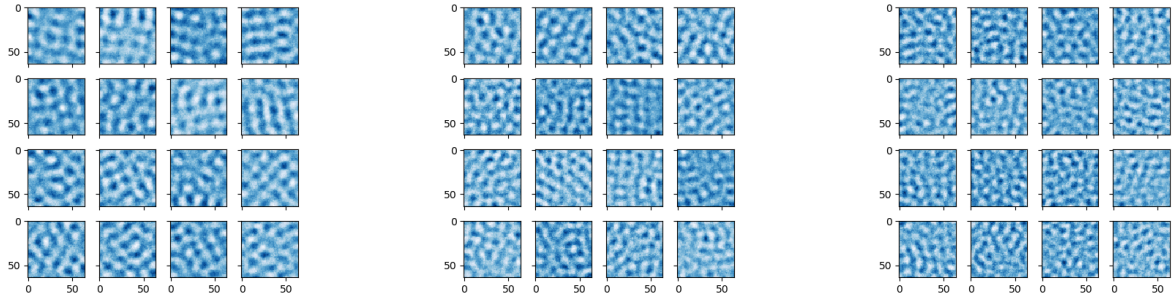
(a) 1st to 16th (b) 17th to 32nd (c) 33rd to 48th

Figure 4.10: Eigenvectors of WW^t of AE trained by 1000 epochs in descending order of the eigenvalues.



(a) 1st to 16th (b) 17th to 32nd (c) 33rd to 48th

Figure 4.11: Eigenvectors of WW^t of AE trained by 5000 epochs in descending order of the eigenvalues

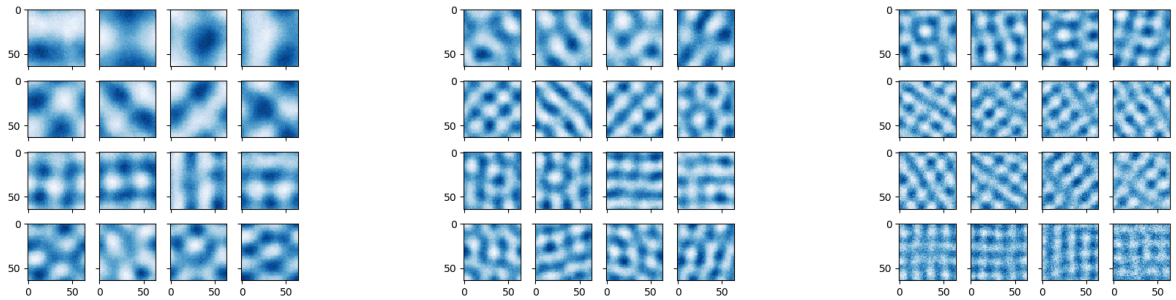


(a) 49th to 64th (b) 65th to 90th (c) 91st to 106th

Figure 4.12: Eigenvectors of WW^t of AE trained by 5000 epochs in descending order of the eigenvalues

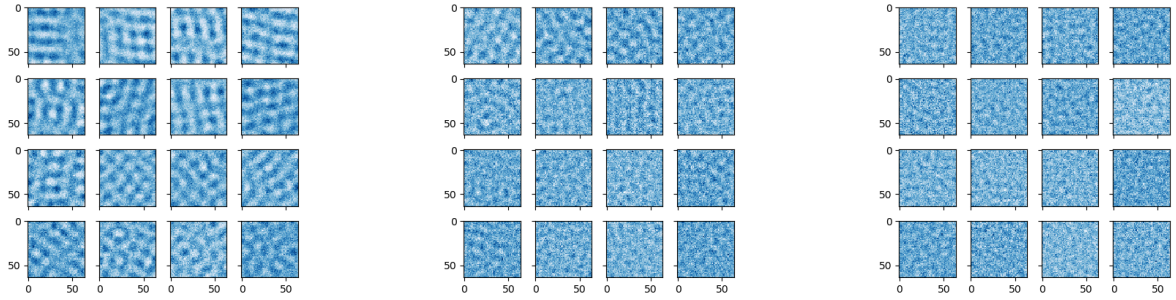
Fig.4.13 shows eigenvectors of type L AE trained at $T = 1.0, \cdots, 1.5$. The structures of first

48 eigenvectors are similar to the type V case (See Fig.4.11). After the 64th eigenvector, the clear structures vanish gradually as shown in Fig.4.14. This reflects that spin configurations at $T = 1.0, 1.1, \cdots, 1.5$ have only large scale structures. On the contrary, in type V case, small structure remains even in the 106th eigenvector. This reflects that data set composed of spin configurations at $T = 1.0, 1.1, \cdots, 5.0$ have various scale structures. Fig.4.15 shows the eigenvectors of WW^t trained at $T = 4.6, 4.7, \cdots, 5.0$. There are no eigenvectors which have a clear strcture. This reflects that $T = 5$ spin configurations have only small noisy fluctuations.



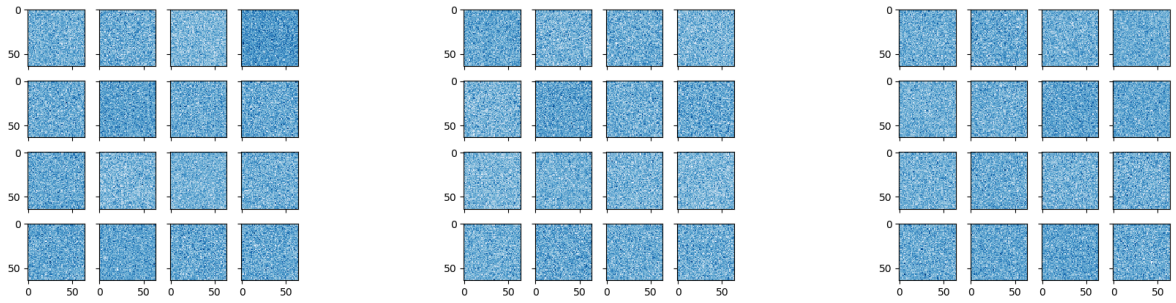
(a) 1st to 16th (b) 17th to 32nd (c) 33rd to 48th

Figure 4.13: Eigenvectors of WW^t of AE trained by $T = 1.0, 1.1, \cdots, 1.5$ spin configurations in descending order of the eigenvalues



(a) 49th to 64th (b) 65th to 90th (c) 91st to 106th

Figure 4.14: Eigenvectors of WW^t of AE trained by $T = 1.0, 1.1, \cdots, 1.5$ spin configurations in descending order of the eigenvalues



(a) 1st to 16th (b) 17th to 32nd (c) 33rd to 48th

Figure 4.15: Eigenvectors of WW^t of AE trained by $T = 4.6, 4.7, \cdots, 5.0$ spin configurations in descending order of the eigenvalues

To observe above properties for the eigenvectors of WW^t more quantitatively, we can use the snapshot entropy (2.55). The singular values of the images which have only large scale structures are hierarchical, so the snapshot entropy becomes small. On the other hand, the singular values of the images which have only small scale structures degenerate, so the snapshot entropy becomes large. Then, we can estimate how clear structures the eigenvectors have by culclating its snapshot entropy. The results are shown in Fig.4.16. In Fig.4.16, We show the results of four different cases. The red dots are the snapshot entropy of the eigenvectors of WW^t trained at $T = 1.0, 1.1, \cdots, 5.0$. The blue dots are $T = 1.0, 1.1, \cdots, 1.5$. The green dots are $T = 2.1, 2.2, \cdots, 2.5$. The magenta dots are $T = 3.6, 3.7, \cdots, 4.0$. As the temperature of the data set increases, the snapshot entropy saturates faster. $T = 1.0, 1.1, \cdots, 5.0$. case saturates at the latest. This results well describe the observation on the eigenvectors.

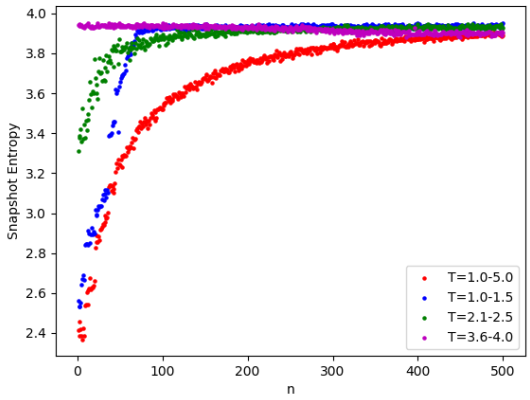


Figure 4.16: The snapshot entropy of eigenvectors of WW^t trained by various temperature sets.

Chapter 5

Conclusions

In this thesis, in order to see what the restricted Boltzmann machine (RBM) learns in the process of training, we investigated flow of configurations generated by the weight matrices of the RBM. We prepare three types RBMs; type V, type H and type L. Type V is trained by configurations at temperatures ranging widely from low to high, $T = 0, 0.25, \ldots, 6$. Type H is trained by configurations at higher temperatures $T = 4, 4.25, \ldots, 6$ than $T_c \simeq 2.27$. Type L is trained by configurations at the lowest temperatures $T = 0$.

In type V RBM, we found that the temperature of an initial configuration flows towards the critical point $T = T_c$ where the system becomes scale invariant. The result suggests that the configurations at $T = T_c$ are *attractors* of the type V RBM flow. In type H/L case, the attractor of the RBM flow is higher/lower than T_c as expected. If the number of the hidden units is larger than the visible units, the attractor of RBM flow is higher than T_c even if RBM is type V, because RBM learns noisy structures of input data.

In order to understand the numerical results of the RBM flows and to find a clue of what the machine has learned, we explored the properties of the weight matrix W_{ia} , especially those of the product WW^t . WW^t reflects the spin correlation in input configurations. We show that $v^t WW^t v$ behaves like the order parameter (magnetization). It indicates that the principal eigenvectors with large eigenvalues are related to the magnetization, and the information about the phase transition is surely imported in the weight matrix. We also study singular value spectrum of weight matrix. Its structure reflects the information about scales which RBM learns.

We also investigate flows of configurations generated by the Autoencoder in order to check the universality of the fact that fixed point of the flow of RBM is the critical point of the 2-dimensional Ising model. We compare the fixed point temperatures of AEs trained by different learning epochs. Unlike RBM case, the temperature of the fixed points of AE trained sufficiently is lower than T_c if AE is sufficiently trained (5000 epochs). On the other hand, if the learning epochs are 1000, the fixed point temperature monotonically increase with the number of hidden units.

We prepare the datasets which are narrow range of temperatures: higher temperatures case

corresponding to type H ($T = 4.6, 4.7, \cdots, 5.0$), lower temperatures case corresponding to type L ($T = 1.0, 1.1, \cdots, 1.5$) and around T_c case ($T = 2.2, 2.3$). The fixed point temperatures are higher than T_c, lower than T_c and around T_c, respectively. This result suggests that AE can learn the specific temperature of the dataset and the fixed point of the AE flow corresponds to the features which AE learns.

We study SVD of weight of trained AE. Singular value spectrum has larger gapped structures in 5000 epochs case than in 1000 epochs case. The gapped structure in the singular value spectrum of the weight matrix is related to the fixed point temperature. Especially, the fixed point temperature becomes T_c when the gap just vanishes. The number of eigenvectors which have clear structures is more in 5000 epochs case than in 1000 epochs case. This means that the structures encoded in eigenvectors of WW^t correspond to the features which AE learns. In facts, if AE is trained by spin configurations at $T = 1.0, 1.1, \cdots, 1.5$, eigenvectors have only large structures. On the other hand, if AE is trained by $T = 4.6, 4.7, \cdots, 5.0$ spin configurations, no large structures are encoded in eigenvectors. These properties can be described by the snapshot entropy of the eigenvectors of WW^t quantitatively.

Acknowledgments

I would like to thank my supervisor Satoshi Iso for teaching me various things and the collaboration which this thesis is mainly based on. I also would like to thank another collaborator Shotaro Shiba Funai for valuable advice and discussions. I would like to thank the participants for I-URIC frontier colloquium 2017, especially Shunichi Amari, Taro Toyoizumi and Shinsuke Koyama for fruitful discussions. I am also very thankful to all members of KEK theory group, especially my colleagues, Toshihiro Aoki, Daiki Ueda, Hikaru Ohta, Naoto Kan, Ryota Kojima and Takuya Hasegawa for stimulating discussions and continuous encouragement.

Bibliography

[1] L. Wang, "Discovering phase transitions with unsupervised learning," Phys. Rev. B **94** (2016) 195105 [arXiv:1606.00318 [cond-mat.stat-mech]].

[2] G. Torlai and R. G. Melko, "Learning thermodynamics with Boltzmann machines," Phys. Rev. B **94** (2016) 165134 [arXiv:1606.02718 [cond-mat.stat-mech].

[3] A. Tanaka and A. Tomiya, "Detection of phase transition via convolutional neural network," J. Phys. Soc. Jap. **86** (2017) no.6, 063001 [arXiv:1609.09087 [cond-mat.dis-nn]].

[4] A. Morningstar and R. G. Melko, "Deep Learning the Ising Model Near Criticality," arXiv:1708.04622 [cond-mat.dis-nn].

[5] Domingos S. P. Salazar, "Nonequilibrium Thermodynamics of Restricted Boltzmann Machines," Phys. Rev. E **96**, 022131 (2017) [arXiv:1704.08724 [cond-mat.stat-mech]].

[6] Sebastian J. Wetzel, "Unsupervised learning of phase transitions: from principal component analysis to variational autoencoders," Phys. Rev. E **96**, 022140 (2017) [arXiv:1703.02435 [cond-mat.stat-mech]].

[7] Wenjian Hu, Rajiv R.P. Singh, and Richard T. Scalettar, "Discovering Phases, Phase Transitions and Crossovers through Unsupervised Machine Learning: A critical examination," Phys. Rev. E **95**, 062122 (2017) [arXiv:1704.00080 [cond-mat.stat-mech]].

[8] A. Kritzhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems,* 2012.

[9] Q. V. Le, "Building high-level features using large scale unsupervised learning," *Acoustics, Speech and Signal Processing* (*ICASSP*), 2013.

[10] C. Beny, "Deep learning and the renormalization group," arXiv:1301.3124 [quant-ph].

[11] S. Saremi and T. J. Sejnowski, "Hierarchical model of natural images and the origin of scale invariance," Proceedings of the National Academy of Sience **110** (2013) 3071

[12] P. Mehta and D. J. Schwab, "An exact mapping between the Variational Renormalization Group and Deep Learning," arXiv:1410.3831 [stat.ML].

[13] A. Paul and S. Venkatasubramanian, "Why does Deep Learning work? - A perspective from Group Theory," arXiv:1412.6621 [cs.LG].

[14] H. W. Lin, M. Tegmark and D. Rolnick, "Why Does Deep and Cheap Learning Work So Well?," J. Stat. Phys. **168** (2017) 1223-1247 [arXiv:1608.08225 [cond-mat.dis-nn].

[15] M. Sato, "Renormalization Group Transformation for Hamiltonian Dynamical Systems in Biological Networks," arXiv:1609.02981 [q-bio.OT].

[16] M. Koch-Janusz and Z. Ringel, "Mutual Information, Neural Networks and the Renormalization Group," arXiv:1704.06279 [cond-mat.dis-nn].

[17] K. G. Wilson, "Renormalization group and critical phenomena. 1. Renormalization group and the Kadanoff scaling picture," Phys. Rev. B **4** (1971) 3174.

K. G. Wilson, "Renormalization group and critical phenomena. 2. Phase space cell analysis of critical behavior," Phys. Rev. B **4** (1971) 3184.

[18] G. E. Hinton and R. R. Salakhutdinov, "Reduction the dimensionality of data with neural networks," Sience **313** (2006) 504.

[19] S. Amari, "Theory of adaptive pattern classifiers", IEEE Trans. Elect. Comput. EC-**16** (1967) 299-307.

[20] Y. Imura, T. Okubo, A. Morita, and K. Okunishi, "Snapshot Spectrum and Critical Phenomenon for Two-Dimensional Classical Spin Systems" J. Phys. Soc. Jpn. **83**, 114002(2014)

[21] H. Matsueda, C. H. Lee, and Y. Hashizume, "Coment on "Snapshot Spectrum and Critical Phenomenon for Two-Dimensional Classical Spin Systems"" J. Phys. Soc. Jpn. **85**, 086001(2016)

[22] H. Matsueda and D. Ozaki, "Proper encoding for snapshot-entropy scaling in two-dimensional classical spin models" Phys. Rev. E **92**, 042167(2015)

[23] K. Seki, K. Okunishi, "Snapshot spectra in the world-line quantum Monte Crlo for one-dimensional quantum spin systems"[arXiv:1706.09257]

[24] C. H. Lee, Y. Yamada, T. Kumamoto and H. Matsueda, "Exact Mapping from Singular Value Spectrum of Fractal Images to Entanglement Spectrum of One-Dimensional Quantum Systems," J. Phys. Soc. Jap. **84** (2015) no.1, 013001 [arXiv:1403.0163 [cond-mat.stat-mech]].

[25] T. Kumamoto, M. Suzuki, H. Matsueda, "Singular-Value-Decomposition Analysis of Associative Memory in a Neural Network," J. Phys. Soc. Jap. **86** (2017) no.2, 024005 [arXiv:1608.08333 [cond-mat.stat-mech]].

[26] S. Iso, S. Shiba, and S. Yokoo, "Scale-Invariant Feature Extraction of Neural Network and Renormalization Group Flow," Phys. Rev. E **4** (2018) 053304.