國立交通大學

物理研究所

碩士論文

深度學習之訊息理論分析

Information-theoretic analysis on Deep Learning

研 究 生：黃師揚

指導教授：張正宏 博士

中 華 民 國 一 百 零 七 年 九 月

深度學習之訊息理論分析

Information-theoretic analysis on Deep Learning

研 究 生：黃師揚　　Student: Suiong Ng

指導教授：張正宏　　Advisor: Cheng-Hong Chang

國 立 交 通 大 學

物 理 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Physics

College of Science

National Chiao Tung University

in partial fulfilment of the requirements

for the Degree of

Master

in

Physics

August 2018

Hsinchu, Taiwan

中 華 民 國 一 百 零 七 年 九 月

深度學習之訊息理論分析

學生：黃師揚　　指導教授：張正宏 博士

國立交通大學 物理研究所

摘　　要

人工智慧裡的深度學習近年來帶給科技界重大的衝擊，深度神經網路的學習過程本質相同於統計物理的趨向平衡問題。對於深度學習動態系統的探討有助於瞭解與學習機制有關的生物問題，例如腦神經網路的成型及生物新演化論。本論文運用資訊論工具來探討此問題，包含以互訊息剖析深度神經網路動態系統的兩相變換，以及運用訊息瓶頸理論觀查該動態系統的最後收斂點。

Information-theoretic analysis on Deep Learning

Student: Suiong Ng Advisor: Dr. Cheng-Hong Chang

Institute of Physics
National Chiao Tung University

Abstract

The recent breakthrough of Deep Learning has had a great impact on science and technology. The training process of Deep Neural Network is essentially the same as the problem of approaching equilibrium in Statistical Physics. The objective of this thesis is to understand this process by utilizing tools in Information Theory. The content includes using Mutual Information to analyze the phase transition during the training process and applying the Information Bottleneck theory to understand how the training dynamics converges to its final state.

Acknowledgement

My sincere gratitude and appreciation to my thesis advisor, Prof. C.-H. Chang, for making this thesis possible. This thesis wouldn't be completed without his continuous encouragement and support. It had been an amazing experience and honor for being able to take part in this interdisciplinary research.

I also thank my schoolmates, my friends, my family, and T.-Y. Lee for their companion, love and emotional support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 History and motivation

A revolutionary development of science and technology in recent years is the success of Deep Learning, a branch of machine learning and artificial intelligence. Since this achievement, a large amount of resources worldwide has been invested to study this technique. These studies were originally mainly aimed at improving the efficiency and accuracy of implementations in the technique. However, they were followed by increasing recent theoretical works devoted to uncovering its underlying mechanism from various perspective, including Physics, Mathematics, and Information Theory. For examples, the Information Bottleneck (IB) proposed by N.Tishby [1] [2], the renormalization group by P.Mehta [3], Hamiltonian symmetry by H.Lin [4], the stabilizer in group theory by A.Paul [5], and manifold disentanglement by P.Brahma [6]. These signs of progress have led to several cross-disciplinary discussions, such as in the talk of Y.LeCun [7] [8] and in 2017 Cognitive Computational Neuroscience [9] conference.

Among these studies, Tishby et al [1] [2] tried to understand how the structure of a deep neural network (DNN) changes during a learning process. Two issues of concern therein are (i) how this dynamical system evolves and (ii) to which final state does it converge? These two problems are closely related to a physical system approaching thermal equilibrium and fluctuating around that equilibrium, such as a Brownian particle in a potential well. The article [2] entitled “Opening the black box of deep neural networks via information” soon hit the headline of several media and sparked a lot of discussions. One of the main findings in that study was that a trained network will converge to an optimal structure complied with the IB theory. This theory was originally proposed to describe the tradeoff between data compression and preserving meaningful information [10]. It has been applied to many systems including the real neural

networks analyzed by biophysicist W.Bialek [11]. To the best of our knowledge, his concept was first introduced by Tishby's group to study DNNs.

In contrast to the consistent properties governed by the IB, another main finding in Ref. [2] is more controversial. That is, during the learning process a DNN will undergo a phase transition from an empirical error minimization (ERM) phase to a data compression phase. This elegant transition demonstrated in Ref. [2] was hypothesized to be a general feature in Deep Learning. However, after that attractive work, no other similar results were reported for a while. Until recently, some objections began to emerge. People did try to replicate Tishby's results from ArXiv articles, Github, or youtube videos of conference workshops, but encountered several difficulties. For instance, Github users reported issues of missing or not disclosed data. [12]. Academics expressed their doubts. Tishby rebuked these opinions in several comment sections of these blogs [13] [14], forum [15], and youtube [16]. Nevertheless, a conference paper with a dissenting opinion is accepted at the 2018 International Conference on Learning Representations despite Tishby's strong rejection [17]. Aside from the implementation difficulties, there are also concerns about whether the same feature can be seen in different Deep Neural Networks (DNNs) with different activation functions [17]. As a whole, the main difficulty people have is that the phase transition is not so apparent as expected. The manuscript [2] has not been officially published yet and Tishby's claim is still open for debate.

We are one of the early groups trying to understand the statements of Tishby and started our study before objections were reported. We used different measure and DNNs to test the robustness and generality of the properties claimed in Ref. [2]. In our study, Tishby's transition does appear in small networks we tested, but hard to find in larger networks. The latter is also voiced in the several discussions in Ref. [17]. Furthermore, we also showed that the transition feature depends on several factors, including the MI estimators, how data are coarse-grained or averaged. In some trained networks, our study revealed results which are consistent with the IB theory. On the one hand, the results collected in this thesis would directly help clarify some disputes in the community of Deep Learning. On the other hand, the information-theoretical studies in these artificial systems would more widely shed light on the learning mechanism in general complex systems, such as brain functions and a novel evolution theory [18].

1.2 Deep Neural Network

An artificial neural network (ANN) is a human-made neural network with artificial neural nodes mimicking the structure of actual neurons. A DNN is an ANN with a multi-layer structure. Nodes in a layer are connected to nodes in neighboring layers. Each node sums up the input values from other nodes multiplied by some weights. After adding a bias to that sum and converting it by an activation function, the outcome is the output of the node (Fig. 1.1). A typical activation function is tanh, which was used in our study. The output value is then passed on to the next neuron as one of its input values (Fig. 1.2). The output values of the last layer are the result of this DNN.

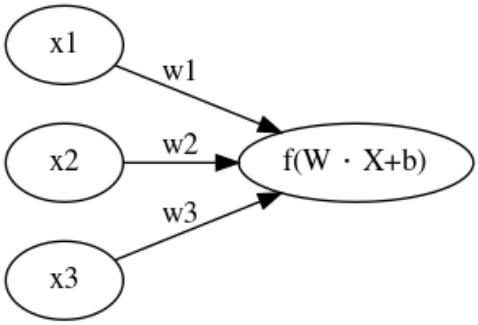


Figure 1.1: An artificial neuron takes inputs from 3 neurons (nodes) in the previous layer. The sum of the weighted input plus a bias b is then transformed by an activation function f, where W and X are vectors of the form

$$W \cdot X = \begin{bmatrix} w1 & w2... \end{bmatrix} \begin{bmatrix} x1 \\ x2... \end{bmatrix}.$$

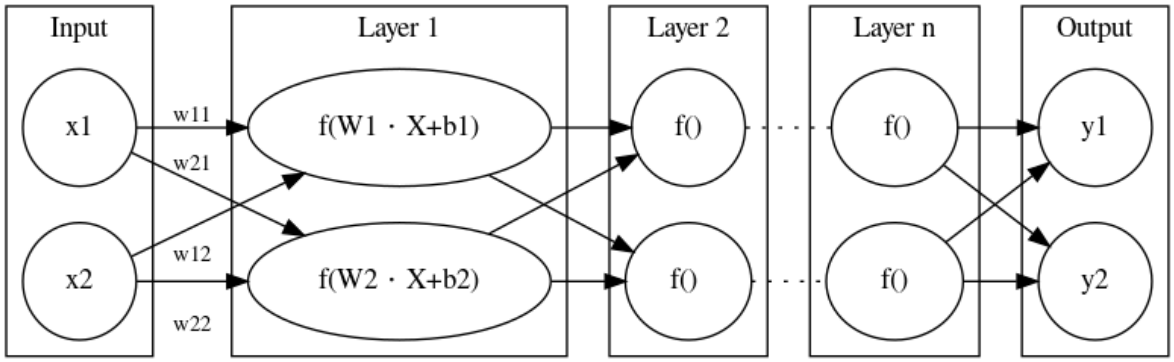


Figure 1.2: Several artificial neurons are chained together to form a DNN of n hidden layers. The outputs of the nodes in a layer are passed rightward on to the neurons in the next layer.

In a computer code, the structures of DNNs can be designed as follows. First, parameters

such as the number of layers and the number of nodes in each layer need to be hand-picked by the designer in advance. By contrast, weights and biases are hyperparameters to be trained and updated through a learning process. The simplest type of learning is a supervised learning process which updates these parameters step by step based on a training dataset. This dataset consists of input data and their corresponding output data (label). During the training process, the input data in the training dataset are fed into a DNN. The weights and biases of that DNN will be adjusted to minimize the difference between the output of the DNN and the label in the training dataset. The function used to characterize that difference is called the cost function. The value evaluated by the cost function is called a loss. The cost function used in this thesis is the categorical cross entropy.

$$H(p,q) = -\sum_{x} p(x) \log q(x) \tag{1.1}$$

where p is the probability distribution of label and q is the probability distribution of output. Stochastic Gradient Descent (SGD) with the back-propagation technique is one of the most popular methods [19] [20] in the learning process to compute gradients to minimize the loss.

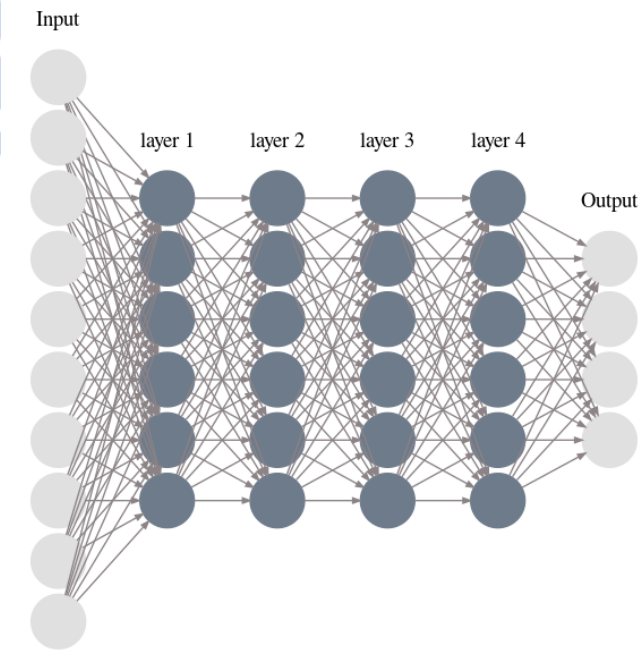


Figure 1.3: The layered structure of a feed-forward, fully-connected DNN with 4 hidden layers. The number of layers and number of nodes in each layer are hyperparameters which need to be selected before training.

In DNNs, the intralayer nodes are not connected, while the interlayer nodes may be connected. In this thesis, the latter are all fully connected and only pass information in one direction (feed-forward DNN, as shown in Fig. 1.3). Besides, only supervised learning for classification tasks was studied. Therein, the default optimizer was SGD, or more specifically, mini-batch gradient descent. SGD is the special case where the batch size of mini-batch is 1. For the mini-batch gradient descent, the weights are tuned according to the average gradient of the cost function of a small part of the training data in the whole epoch.

1.3 SGD and Physics

SGD is the key ingredient for the success of DNNs. There had been studies trying to explain why SGD has such good generalization performance [21] [22], that is, performance of predicting the outcome of the unseen dataset. The evolution of the structure of a DNN guided by SGD can be viewed as the random walk of a Brownian particle confined in a potential funnel and kicked by stochastic forces. The SGD in a DNN training process adjusts the values of weights according to the gradients of the cost function. It is similar to the convergence of the Brownian particle toward the bottom of the potential well driven by the potential gradient. Nevertheless, the convergence of DNN does not follow a smooth path, because it can be disturbed by the random shuffling of training samples. It is like the irregular motion of a Brownian particle jostled by the surrounding water molecules. Despite this similarity, the dimension of the actual potential landscape of a DNN is usually pretty high, with a lot of local minima [23]. Why DNNs can so easily find an optimal minimum during training, or whether what they find are optimal, is less trivial and still actively discussed.

Chapter 2

Concepts and tools in Information Theory

In this chapter, we introduce some basic concepts and tools which are useful for analyzing the evolution and the final structure of DNNs during training.

2.1 Kullbak-Leibler divergence and the Mutual Information

Given two probabilities $P(i)$ and $Q(i)$, the Kullback-Leibler (KL) divergence is defined as

$$D_{KL}(P||Q) \equiv -\sum_i P(i) \log \frac{Q(i)}{P(i)} \tag{2.1}$$

It is a measure to describe the distance between $P(i)$ and $Q(i)$. The KL divergence is valid when "$Q(i) = 0$ implies $P(i) = 0$", because $Q(i)/P(i) = 0$ implies $\log Q(i)/P(i) = -\infty$, which needs to be compensated by $P(i) = 0$ outside the log.

Mutual Information (MI) is a measure of mutual dependence between two random variables. In this thesis, it is used to characterize the relevance between a hidden layer of a DNN and its input data or label. Given two random variables X and Y and the joint probability distribution $p(X, Y)$, the MI of X and Y is defined as

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{2.2}$$

With this quantity, data processing inequality [24] states that for any function f, the MI of X and Y would be larger than the MI of X and $f(Y)$.

$$I(X,Y) \geq I(X, f(Y)) \tag{2.3}$$

This property can be used to justify the reliability of the MI analysis on DNNs and to rescale MI values to compare them between different DNNs (see examples in section 4.3).

2.2 The Information Plane

The activations of nodes in a layer of a DNN can be described by a vector variable. Let X, Y, and T be the variables of the input, the label (desired output), and a hidden layer respectively. One can calculate the MI $I(X;T)$ between X and T as well as the MI $I(T;Y)$ between T and Y. Then the state of the hidden layer during training is represented by a point in the $I(X;T)$-$I(T;Y)$ plane, termed the Information Plane (IP) [2]. An example can be seen in Fig. 2.1

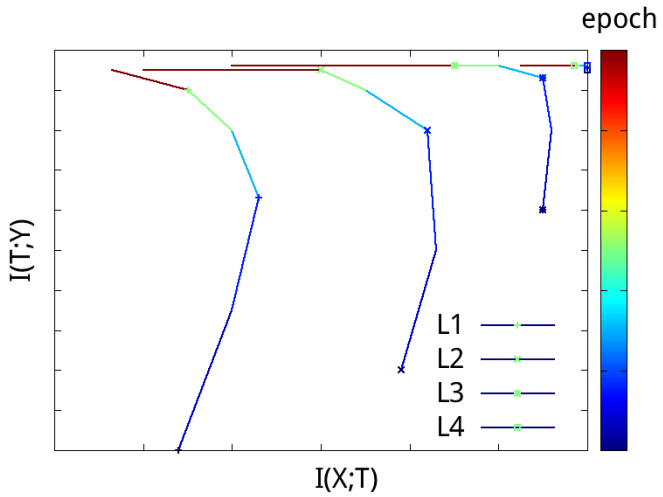


Figure 2.1: In the IP of a DNN, the evolution of each layer follows a certain curve. The curves undergo a phase transition during training, around at the turning points of those curves. This figure is replotted from [2].

Information in the feed-forward neural network passes through the network only in a single direction. e.g. $X \rightarrow T_1 \rightarrow ...T_i... \rightarrow Y$. It satisfies the data processing inequality (eq. 2.3), for which:

$$I(X;Y) \geq I(T_1;Y) \geq I(T_2;Y) \geq I(T_3;Y)... \tag{2.4}$$

$$I(X;T_1) \geq I(X;T_2) \geq I(X;T_3)... \tag{2.5}$$

That is, $I(X;T)$ and $I(T;Y)$ in the IP plots would decrease as the layer goes deeper. As a result, deeper layers will be closer to the left-bottom corner of the IP. The fact that $I(T;Y) < I(X;Y)$ for any hidden layer T is being exploited to rescale $I(T;Y)$ as a percentage value $I(T;Y)/I(X;Y)$. It provides a unified scale to compare the dynamics in the IP between differ-

ent DNNs. In the following IPs, the $I(T;Y)$ axis is rescaled as mentioned above, except for the case of the MNIST dataset, while the $I(X;T)$ axis is not rescaled.

2.3 The Information Bottleneck

The essential information of a data usually does not need to be represented by so many nodes (or bits) as it is stored. That is, the data can be compressed to some extent without heavily distorting the desired information to be retained. The IB theory [10] is aimed at finding the optimal representation (relevant information) of an input data. That representation should keep the information required to appear in the output. "Optimal" here refers to a tradeoff between best compression and minimal distortion. The theory is termed IB, because the process of extracting relevant information of input data is as if forcing it through a bottleneck [10].

DNN is a good candidate for exploring the IB. From the perspective of IB theory, the function of DNN is to reduce the space layers by layers to compress the information. Notably, the key reduction is not the number of nodes, but the dimension of vectors to represent an information stored in nodes. However, the compression should not be too strong to make the output far away from the required label. A strong compression in the variable T of a hidden layer will have a small $I(X;T)$. An accurate prediction will have a strong relation between the layer T and the label, characterized by I(T; Y). Both $I(X;T)$ and $I(T;Y)$ are functions of the encoder $q(T|X)$. This encoder is a conditional probability, representing the probability of finding $t \in T$ on the condition of $x \in X$. Therefore, a tradeoff can be obtained by looking for an optimal $q(T|X)$ to minimize the Lagrangian

$$L[q(T|X)] = I(X;T) - \beta I(T;Y). \tag{2.6}$$

Therein, β is a free parameter to express whether compression is more important or prediction accuracy is more important. Note that the optimal $q(T|X)$ will lead to a large difference between $I(X;T)$ and $\beta I(T;Y)$, instead of a large individual $I(X;T)$ or $\beta I(T;Y)$. Here, T is called the bottleneck variable [25]. Note that there exists an alternative definition for the Lagrangian, $L = \beta I(X;T) - I(T;Y)$ [25]. The L used in our study is eq. 2.6.

Different β give different constraints to L, which leads to different optimal points $(I(X;T), I(T;Y))$ in the IP. The curve formed by those points is called the IB bound, as shown in Fig. 2.2. The $(I(X;T),\ I(T;Y))$ points in a DNN must fall in the area below this bound. For a larger β, $I(T;Y)$ is the dominant term for deciding the minimum L in eq. 2.6. In this case, according to the IB and the data processing inequality, the maximum value to which training can shift $I(T;Y)$ is $I(X;Y)$. Therefore, the upper bound at which $I(T;Y)$ saturates becomes a straight line at large β, as shown in Fig. 2.2. The curve in Fig. 2.2 plays the same role as the curve for the tradeoff between data quality and data size in the lossy data compression technique, such as PSNR-bitrate curve for video compression.

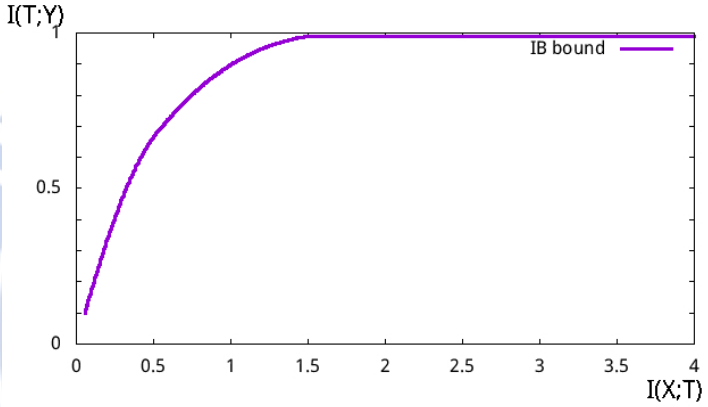


Figure 2.2: The curve of IB bound at different β given by the IB theory based on minimizing eq. 2.6. The vertical axis has been rescaled to the percentage of $I(X;Y)$ for comparing with other DNNs.

2.4 The modified Blahut-Arimoto algorithm

To find the optimal $q(T|X)$, or the optimal representation, of the input variable [10] [26], the IB theory suggests a modified version of the Blahut-Arimoto algorithm [27] [28]. By using this algorithm, the optimal $q(T|X)$ and subsequently the optimal structure which DNNs can achieve can be calculated. The corresponding optimal bound $I(T;Y)$ can be used as a benchmark for analyzing the generalization performance of DNNs. Because eq. 2.6 cannot be solved analytically, the algorithm for finding the optimal $q(T|X)$ is based on an iteration process as follows.

For a known $p(X,Y)$,and a given $\beta \geq 0$, first randomly initialize $q^{(0)}(T|X)$. With that, one

obtains

$$q^{(n)}(t_i) = \sum_x p(x) q^{(n)}(t_i|x) \tag{2.7}$$

Owing to Bayes' Theorem, $g(x|t) = g(t|x)h(x)/h(t)$ and $g(y|t) = \sum_x g(x|t)h(x,y)$, one has

$$q^{(n)}(y_j|t_i) = \frac{1}{q^{(n)}(t_i)} \sum_x p(x, y_j) q^{(n)}(t_i|x) \tag{2.8}$$

They are used to optimize $q^{(1)}(t|x)$ by

$$q^{(n+1)}(t_i|x_j) = \frac{q^{(n)}(t_i)}{Z(x_j, \beta)} e^{-\beta D_{KL}[p(y|x_j)|q^{(n)}(y|t_i)]} \tag{2.9}$$

Here, D_{KL} is the KL divergence in eq. 2.1. $q^{(n)}(T|X)$ is expected to converge to a stable distribution after iterations and $Z(x_j, \beta)$ is introduced to normalize $\sum_i q^{(n)}(t_i|x_j)$. With a known $q(T|X)$, we can determine the probabilities, $q(T)$, at the nodes of layer T by

$$q(t) = \sum_i p(x_i) q(t|x_i) \tag{2.10}$$

Then, $p(T, Y)$ can subsequently be derived. from which it follows MIs $I(X;T)$ and $I(T;Y)$. These two MI values are thought to be the MIs of the optimal DNN. Notice that "optimal" here does not refer to the most accurate nor the best generalization performance, but the most optimal tradeoff between compression and accuracy.

To calculate the KL-divergence in the modified Blahut-Arimoto algorithm in eq. 2.9, one usually encounters the problems of having some bins with zero probability in the histogram $p(X, Y)$ collected from real data. This would give zeros to $P(i)$ or $Q(i)$ at some bin i and lead to a divergence problem to eq. 2.1. To avoid that problem, zero can be replaced by some small probability ϵ. Then, the optimal $q(T|X)$, and the IB bound can be calculated. The latter will differ slightly for different ϵ, as shown in the example in Fig. 2.3.

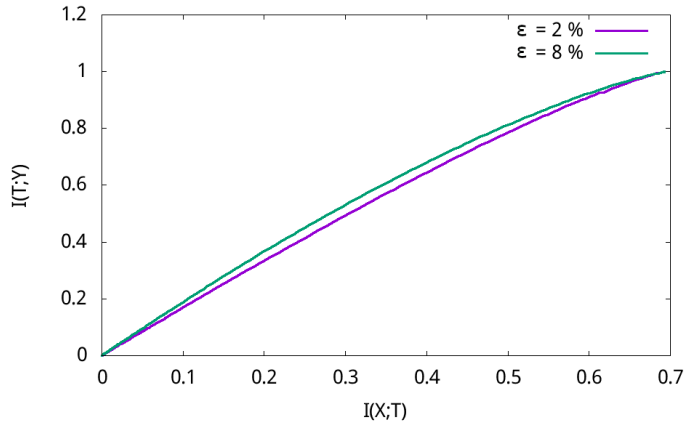


Figure 2.3: The IB bound of a sample distribution $p(x, y)$ with different ϵ.

2.5 Examples for optimal representations

The dimension of T in the encoder can be arbitrary. Given a dimension, the minimization of the Lagrangian in eq. 2.6 solves possible optimal $q(T|X)$, in the sense of best tradeoff between compression and accuracy mentioned above. There can exist infinitely many optimal $q(T|X)$ when the dimension of T is large. As an example, consider a joint probability described by a matrix.

$$p(X,Y) \Rightarrow \begin{bmatrix} p(x_1,y_1) & p(x_1,y_2) \\ p(x_2,y_1) & p(x_2,y_2) \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{2.11}$$

where the entry at the i-th row and the j-th column denotes the probability of finding the i-th value of x and the j-th value of y. If X and T are both 2-dimensional, so are their conditional probabilities $q(T|X)$ and $q(X|T)$. The former can be represented by a 2×2 matrix. If β is large, the modified Blahut-Arimoto algorithm gives us one of the following two distributions regardless of the initial value $q^{(0)}(T|X)$,

$$q(T|X) \Rightarrow \begin{bmatrix} q(T|x_1) \\ q(T|x_2) \end{bmatrix} \Rightarrow \begin{bmatrix} q(t_1|x_1) & q(t_2|x_1) \\ q(t_1|x_2) & q(t_2|x_2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.12}$$

and

$$q(T|X) \Rightarrow \begin{bmatrix} q(T|x_1) \\ q(T|x_2) \end{bmatrix} \Rightarrow \begin{bmatrix} q(t_1|x_1) & q(t_2|x_1) \\ q(t_1|x_2) & q(t_2|x_2) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.13}$$

As an example, let a DNN consist of an input layer X, an output layer Y, and a hidden layer T, each of which has only one node with one bit (see Fig. 2.4). Then the activations in the hidden layer, which are a representation of X, can be optimized. For the 2×2 diagonal matrix $p(X;Y)$ in eq. 2.11, the output of the DNN is always the same as its input. For a large β as considered above, "optimal" refers more to "accuracy", instead of "compression". In this case, there is no room for $q(T|X)$ to be optimized other than the above two in eq. 2.12 and eq. 2.13. They stand for copying the input values twice and inverting them twice, respectively. Thus, the activation of the hidden layer is either the same as or opposite to the input data.

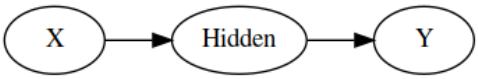


Figure 2.4: A simple DNN with only one node in the input, the output, and the single hidden layer.

If the dimension of T becomes larger, e.g., 2 or 3, the matrix denoting $q(T|X)$ becomes larger, e.g., 3×2 or 4×2, which seems to have countless optimal encoders $q(T|X)$. All these encoders give the same $I(X;T)$ and $I(T;Y)$ values. The countless number of optimal encoders is because when the space of the hidden layer T is larger, the bits in T are more than those required to represent the information in X. Thus, there are infinitely many selections. As an example, let us consider

$$p(X,Y) \Rightarrow \frac{1}{24} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 2 & 1 \end{bmatrix}. \tag{2.14}$$

Here, the first two rows are linearly dependent. Suppose we are interested in $\beta = 10$. Let us take a randomly initialized $q(T|X)$, denoted by a 3×4 matrix

$$q(T|X)^{(0)} \Rightarrow \begin{bmatrix} 0.07 & 0.26 & 0.33 & 0.34 \\ 0.25 & 0.13 & 0.32 & 0.29 \\ 0.29 & 0.23 & 0.38 & 0.10 \end{bmatrix} \tag{2.15}$$

After one iteration, it becomes

$$q(T|X)^{(1)} \Rightarrow \begin{bmatrix} 0.18 & 0.17 & 0.32 & 0.33 \\ 0.18 & 0.17 & 0.32 & 0.33 \\ 0.30 & 0.23 & 0.37 & 0.09 \end{bmatrix} \tag{2.16}$$

After 9 iterations, it converges to

$$q(T|X)^{(9)} \Rightarrow \begin{bmatrix} 0.01 & 0.17 & 0.40 & 0.42 \\ 0.01 & 0.17 & 0.40 & 0.42 \\ 0.90 & 0.02 & 0.04 & 0.04 \end{bmatrix} \tag{2.17}$$

In this example, the information in the X layer is represented by a T layer of a larger dimension via an encoder $q(T|X)$. The two linearly independent rows in $p(X, Y)$ lead to two linearly independent rows in $q(T|X)$. In practice, when we design DNNs, the dimension of layer T is usually smaller than that of layer X. In this case, the possible $q(T|X)$ to reduce the redundant information stored in X will be much fewer.

The encoder in eq. 2.17 yields MIs $I(X;T) = 0.42$ and $I(T;Y) = 0.83$. Besides this encoder, there exist many other possible $q(T|X)$ which give different $q(T)$, but the same $I(X;T)$, and $I(T;Y)$. Interestingly, regardless of what those final distributions $q(T|X)$ look like, the MI values they yield are the same. Table 3.1 is an example of the final $q^{(n)}(T|X)$ of 5000 randomly initialized $q(T|X)$ represented by 4×3 matrix.

Duplications of $q(T\|X)$	Counts
6	1
5	3
4	14
3	81
2	458
1	3764

Table 2.1: Among 500 randomly initialized $q^{(0)}(T|X)$, 3764 final $q^{(n)}(T|X)$ do not appear more than once. 458 final $q^{(n)}(T|X)$ appear twice. One final $q^{(n)}(T|X)$ appears six times. All 5000 $q^{(n)}(T|X)$ yield the same MI values. All $q^{(n)}(T|X)$ converge within $n < 2000$ iterations.

2.6 Kolchinsky's MI Estimator

MI evaluation is sometimes a difficult task. Instead of calculating MI from its definition, it can also be derived from the difference between two Shannon entropies.

$$I(T;Y) = H(T) - H(T|Y) \tag{2.18}$$

$$I(X;T) = H(T) - H(T|X) \tag{2.19}$$

In this thesis, MI is mainly calculated from its original definition, but the result is also compared with MIs in eq. 2.18 and eq. 2.19. An implementation of A.Kolchinsky's entropy estimator [25] [29] is available at [30] in the function "entropy_estimator_kl()" and "kde_condentropy()" in the file "kde.py". We ported this implementation into a C++ code and used this code to estimate MIs. These two functions give a tight upper bound of MI. For example, given a collection of N activation patterns in the T layer (see examples in Fig. 3.2), their Shannon entropy can be calculated by

$$H(T) = -\left[\left\langle \log(\sum_j e^{D_{ij}}) \right\rangle - \log(N) - \frac{n}{d}\log(2\pi v)\right] + \frac{d}{2} \tag{2.20}$$

where v is a noise variance, N is the number of activation patterns, d is the dimension of activation patterns (number of nodes in layer T), and D_{ij} is the square of the distance between two activation patterns i and j divided by $2v$. Note that activation patterns i and j are both

1-dimensional vectors, which yields $D_{ij} = (|i|^2 + |j|^2 - 2i \cdot j^T)/2v$. The conditional Shannon entropy in eq. 2.18 is

$$H(T|Y) = \sum_i p(y_i) H(T|y_i) \tag{2.21}$$

Here, $H(T|y_i)$ is the conditional Shannon entropy of T under the condition of y_i, which can also be calculated by eq. 2.20 but only taking the specific pattern of y_i. In the analogy to eq. 2.21,

$$H(T|X) = \frac{d}{2} \log(2\pi v + 1) \tag{2.22}$$

when T is a multivariate Gaussian. The noise variance v here are selected to 0.1, the same as the one in Ref. [30]. This value is chosen to always produce positive entropy and MI values, and to produce similar MI values as other MI calculation methods.

Chapter 3

DNN tasks, Models and Methods

In this chapter, a DNN model was designed to judge whether an input integer n is divisible by a number m or not. In other words, it decided whether the following statement is true.

Statement: (True or False ?)

$$n \bmod m = 0 \tag{3.1}$$

A set of input integers and their correct answers were chosen as the training dataset. Another independent set of integers without answers was chosen as the testing dataset. The DNN was trained with the training dataset and evaluated by the testing dataset. Recall that training is a process of passing an input data from the training dataset through the DNN and adjusting the weights and biases to minimize the difference between the outcome and the label in the training dataset. When all data in the training dataset have been used once to train the DNN, it is called an epoch. In practice, a training process will be repeated for several epochs.

In this thesis, right after each training epoch, another data selected from the testing or training dataset were fed into the DNN. The resultant activations of all nodes in each layer were recorded. The training process was temporarily frozen during this analysis process, in which no parameter values were changed in the DNN. Training was then resumed after that analysis. The recorded activation values were later used to calculate MIs.

3.1 Two divisibility tasks

The simple mathematical tasks chosen in this section does not involve complex computation. Such a choice is important because DNNs are not good at computing general mathematically complex tasks [31]. The following task belongs to a classification task rather than an algebraic one, such as the addition between two numbers. This is crucial for why the DNN to be trained can achieve good performance.

Two concrete tasks to be used to analyze the dynamics of DNNs are

Task A: (True or False ?)

$$n \bmod 3 = 0 \qquad (512 > n \geq 0) \tag{3.2}$$

Task B: (True or False ?)

$$n \bmod 10 = 0 \qquad (4096 > n \geq 0) \tag{3.3}$$

Here, the integer n was represented in binary format and fed into the input layer of the DNN. Then the DNN should decide whether this integer is divisible by the divisor m or not.

3.2 DNN model design

Although the above tasks are binary classification tasks which can be answered by a single output node of one bit, we still used a DNN with two nodes in the output layer to solve the tasks. One of those nodes denotes divisible and the other non-divisible. Using softmax in the output layer, the node with a larger activation in that layer is regarded as the answer. The reason for that design was that other more complex DNNs were also analyzed in this thesis. These DNNs require an output layer having more than one node. Examples are like those for tackling MNIST digits recognition. To compare with those complex DNNs, it is convenient to have similar output layers.

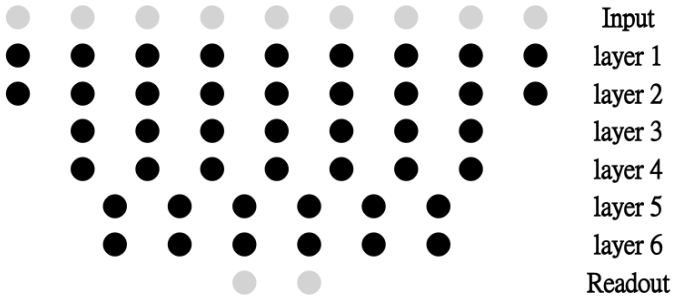


Figure 3.1: A DNN model of eight layers, with node numbers 9-9-9-8-8-7-7-2 in different layers. The input layer is at the top and the output layer at the bottom. This DNN is for solving eq. 3.1.

3.3 Methods of analysis

The MI between a hidden layer and the input data or the label of data is of concern. Therein, the variable to represent the activations in a layer contains information from more than two nodes and is a vector rather than a number. There exist different ways to calculate MIs from these vectors.

3.3.1 Representations of activations in each layer

In the following, the status of each layer was analyzed independently. The activations of nodes in the same layer formed an activation pattern, which was treated as a single vector variable. The 2-dimensional histogram of finding a pattern in a hidden layer and an input pattern was collected to build a 2-dimensional joint probability to calculate the correlation between a hidden layer and the input as shown in Fig. 3.2. Likewise, the joint probability of finding some patterns in a hidden layer and the label was also obtained. Notice that since activations are float numbers, it is rare to find the same output activation pattern from two different input data. Thus, patterns from two layer are generally a one-to-one correspondence.

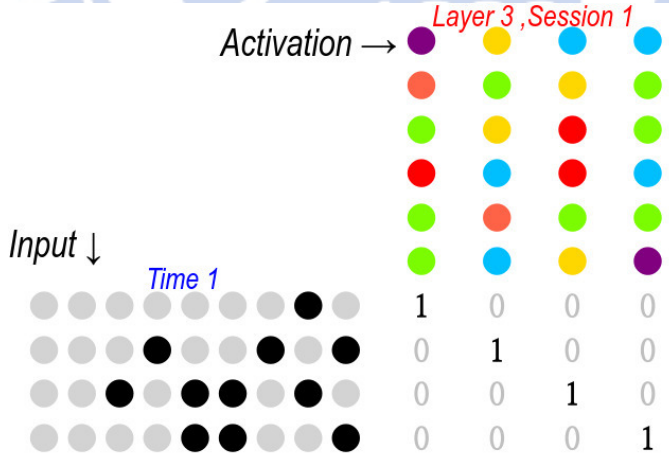


Figure 3.2: The histogram for finding an activation pattern in the input layer of 9 nodes (left part) and another pattern in a layer of 6 nodes (upper part). Different colors denote different values of activations. Here, 4 patterns for both layers are collected.

3.3.2 Ensemble Averages

Even when the training dataset is the same, the deterministic training process looks like a stochastic process. This stochasticity may arise from (i) the shuffling of the data in the training dataset in each epoch (default setup of Keras), (ii) the initial weights randomly generated by an initializer, (iii) the Dropout layer which randomly sets some weights before a certain layer to zero, or (iv) other means to improve the convergence speed or to prevent overfitting. It's like that a Brownian motion is stochastic, although each collision between that particle and the water molecules follows Newton's law and is deterministic. As we are usually more interested in the average of a Brownian motion. Here, we are more concerned with the average of the training process. Thus, several DNNs with the same structure, in layers and nodes, but with different initial weights were trained. The training of each of these DNNs is a session. In the end, an ensemble average over all these sessions was calculated and analyzed.

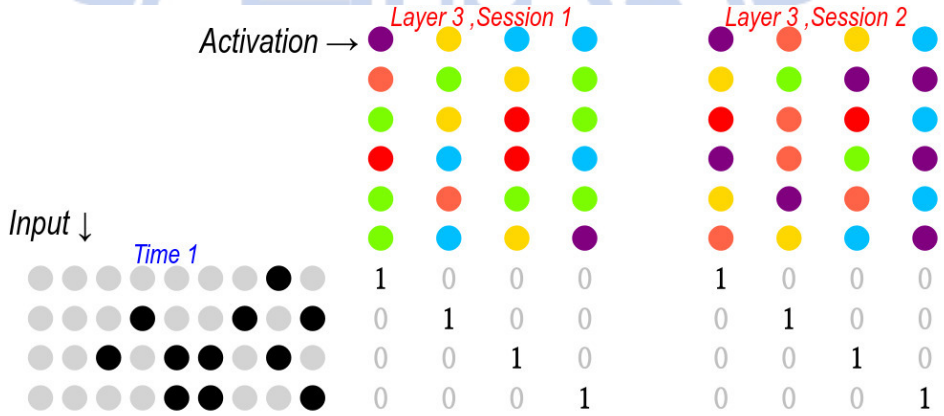


Figure 3.3: The same input activation pattern can generate different activation patterns in different sessions, which form distinct histograms in those sessions.

Importantly, there are two ways of calculating MIs. The first one is averaging all MIs calculated from individual histograms in Fig. 3.3. The second one is pooling together the data from all histograms to form a big histogram to calculate its MI. These two calculations will generally lead to different results, especially when the distributions in two histograms are not centered around at the same point. For distribution in A and B in Fig. 3.4, two kinds of average lead to different MI values. As a schematic example,

$$I(A) + I(B) \neq \frac{I(A+B)}{2} \tag{3.4}$$

Generally, the data points in the sum of two histograms will be more uniformly distributed than those in the individual histograms. Thus, pooling data from different histograms will usually generate a smaller MI than the first one and at least reveals the lower bound of the desired MIs, which was mainly used in this thesis. But sometimes un-pooled procedure was also taken to test the deviation from the pooling procedure.

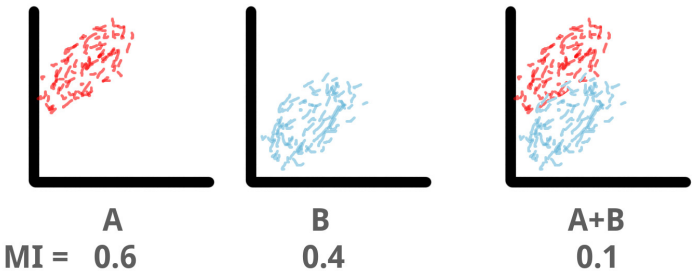


Figure 3.4: Two histograms collected from two sessions A and B are pooled together to form a big histogram. The MIs calculated from three different histograms are listed at the bottom.

3.3.3 Coarse-graining and data preprocessing

Activations of nodes in the input layer could only take values 0 or 1. However, activations of the nodes in other layers were generally float numbers. Because all real values can appear, there are infinitely many activation patterns. Therefore, coarse-graining data in the histogram is necessary before MI calculations. In [2], the data were transformed by the arctan function before being analyzed. This manipulation was however not used in this thesis.

If colors in Fig. 3.3 are divided into two groups, represented by two new colors, a pattern of multiple colors in Fig3.3 is reduced to a pattern of two colors in Fig. 3.5. In this case, two similar multiple-color patterns may be projected to the same two-color pattern. This leads to the degeneracy of some two-color pattern, as shown in Fig. 3.5. A pattern with a large degeneracy is more likely to be found in different sessions. When pooling together all these projected activation patterns, we have a big histogram as shown in Fig. 3.6. More generally, if n_1-color patterns are reduced to n_2-color patterns, the size of that histogram will decrease. In

the following, this coarse-graining procedure is termed binning, and the number n_2 is called the number of bins.

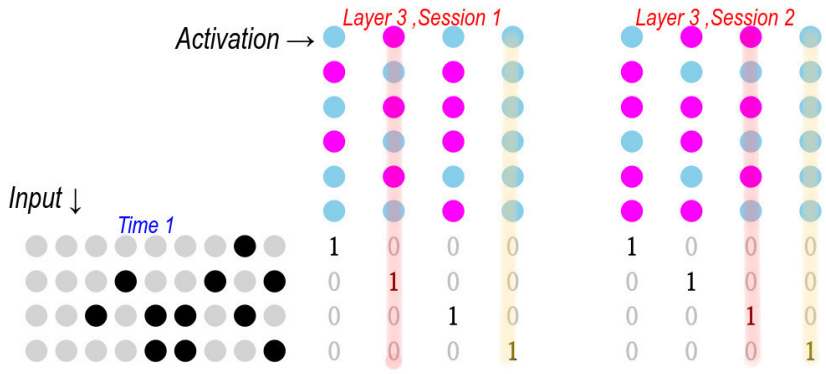


Figure 3.5: Coarse-graining leads to the degeneracy of the activation patterns. In this example, the 2nd and 3rd input-patterns has a degenerate pattern in the 3rd hidden layer in two different sessions.

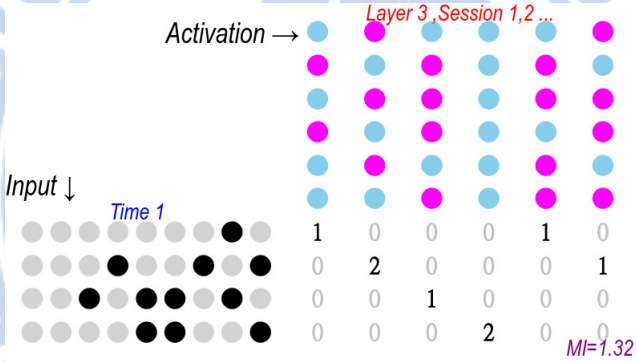


Figure 3.6: The final big histogram for calculating MIs.

3.3.4 MI calculations

Several estimators, such as the kernel density estimator [32] and the pair-wise distance based estimators [33], can be used to evaluate MIs. However, they are limited in some conditions on the data [29]. For instance, the above pair-wise distance based estimator requires a concrete distance measure for sample points. For different activation patterns in Fig. 3.5, the distance concept does not exist. Instead of defining extra "artificial" distance for those patterns, the MIs in the next chapter were mainly calculated by their fundamental definition in eq. 2.2. In this case, most of the (x, y) points to be inserted into eq. 2.2 in the 2-dimensional histogram were 0

or 1, as explained in the last subsection. As a comparison, we also used an entropy estimator [30] in section 4.6, 4.10 and 4.9 to evaluate MIs to see the difference.

Note that shuffling the activation patterns of "continuous" colors of a hidden layer T can usually bring the histograms $p(X,T)$ for individual sessions to a form like an "identity matrix", as those in Fig. 3.2 or 3.3, because the chance of finding two identical activations are extremely rare.

Suppose that "matrix" is $n \times n$, as that in Fig. 3.2 for n = 4. After pooling data from two sessions, the resultant big histogram will be generally $n \times 2n$, because an input pattern usually generates two distinct patterns for layer T, unless the two DNNs have completely the same weights. A simple calculation shows that the MIs for the individual histograms in Fig. 3.3 and that for the big histogram are the same, as shown in eq. 3.5 and 3.6. That MI value is independent of how we shuffle the patterns in X or T, but only depends on the number of the collected patterns in X. Therefore, the MIs of two DNNs are always identical as long as the number of the collected input patterns from those DNNs are the same. It raises the doubt whether MI is a meaningful metric to reflect the identity of a DNN. Fortunately, this problem disappears, when the activations are coarse-grained to few colors, as two colors in Fig. 3.5 and 3.6. In this case, histograms from individual sessions generally cannot be brought to an "identity matrix". The values in the corresponding big histogram will become more irregular, which makes the MI values more diverse.

$$I\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) = 1.09861 \tag{3.5}$$

$$I\left(\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}\right) = I\left(\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}\right) = 1.09861 \tag{3.6}$$

Since the data points in the big histogram can have more than 10^8 elements, the computation is rather memory-consuming. Thus, Monte Carlo sampling was sometimes used to reduce the

amount of data to extract the key behaviors of the system. In this sampling, data points are randomly picked up and deleted, which leads to a smaller histogram. It is clear that this sampling will affect the result if too many points are deleted. In the example in Fig. 3.7 the states of hidden layers in the IP calculated with and without the Monte Carlo sampling follow the same tread and converge to the same final states.

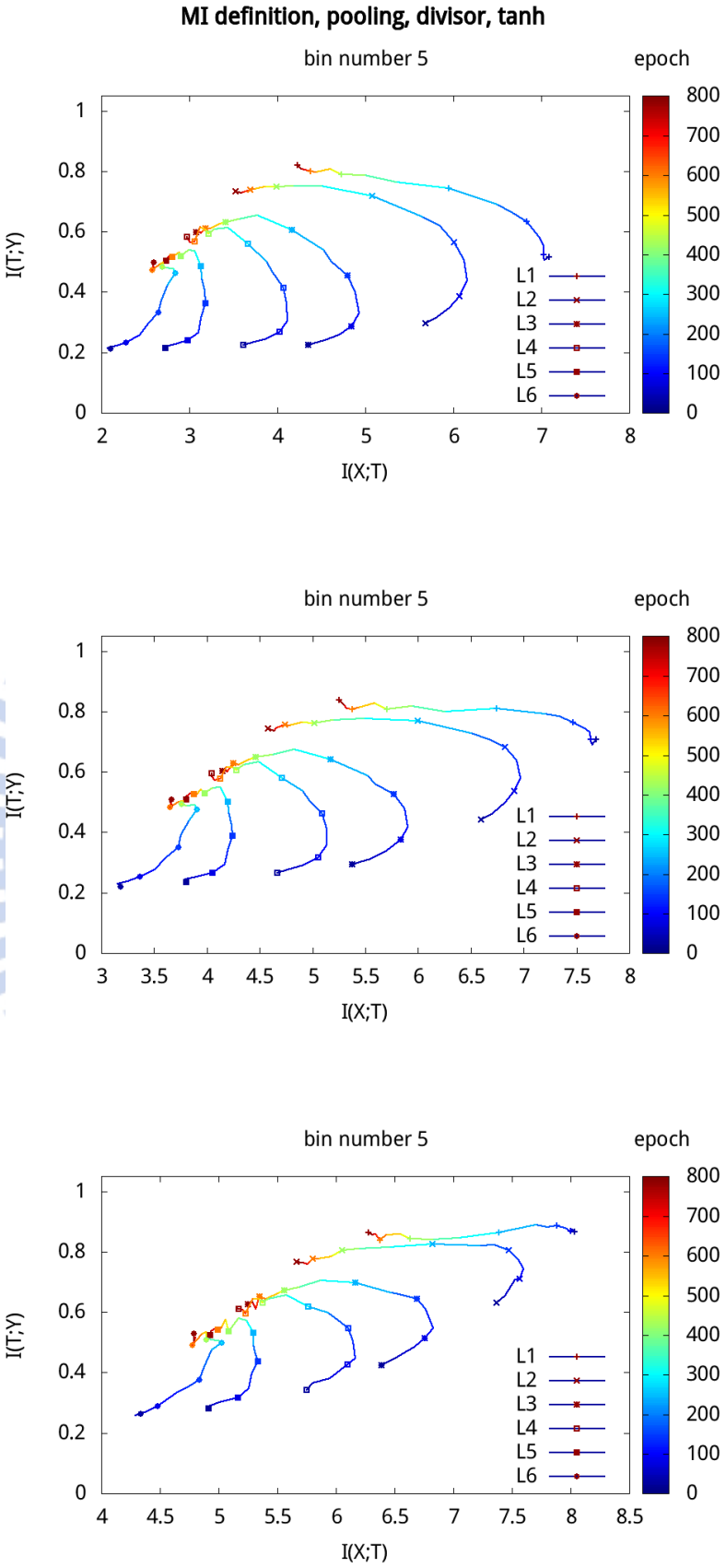


Figure 3.7: During a training process, the evolutions of the states of different layers were represented by different trajectories in the IP. These trajectories were calculated without (top) or with (middle, bottom) the Monte Carlo sampling for the case of eq. 3.3. The data sizes from top to bottom are 100%, 33%, 10% in the original data respectively.

Chapter 4

The dynamics of trained DNN state

In this chapter, two types of DNNs were trained to solve eq. 3.1. Each type contains a large ensemble of DNNs, which form a large number of sessions. All training histories were recorded and analyzed. The analysis was focused on how the state of DNNs in the IP evolves as they converge to their final state.

4.1 The accuracy distributions

In our trained DNNs, the accuracy against the testing dataset is high, which indicates the success of these DNNs. For example, blindly answering false to eq. 3.2 would yield a success rate of 66%. But our trained DNNs can normally achieve 96%, as collected in Fig. 4.1. Even in the case of eq. 3.3, in which blindly answering false has reached a higher success rate of 90% , the trained DNNs still outperformed that rate, as shown in Fig. 4.2.

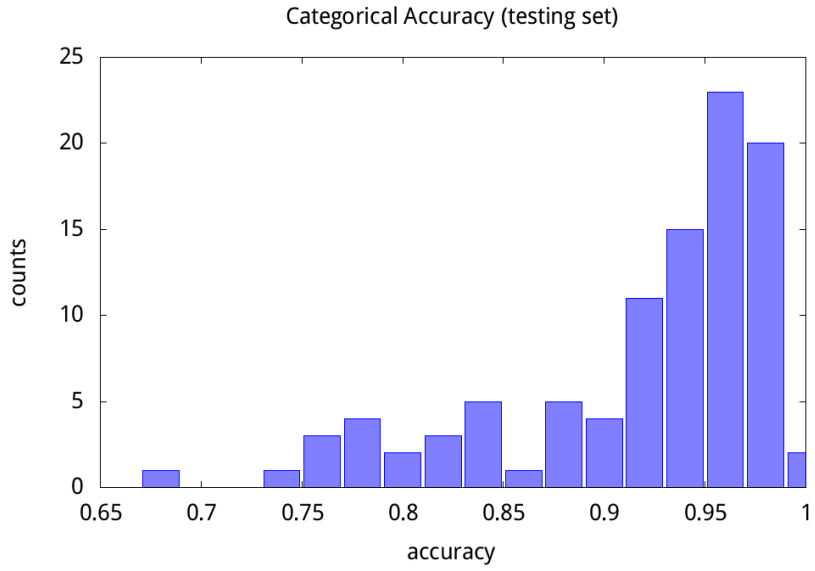


Figure 4.1: The distribution of the categorical accuracy of the trained DNNs. (testing dataset, for the task in eq. 3.1 with n = 512).

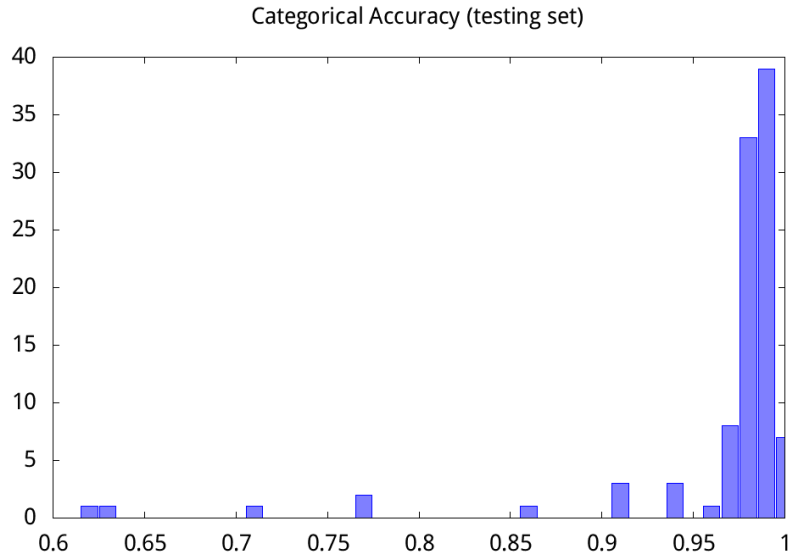


Figure 4.2: The distribution of the categorical accuracy of the trained DNNs. (testing dataset, for the task in eq. 3.1 with n = 4096).

4.2 The evolution of activations

During a training process to solve eq. 3.2, the evolution of the activations of three layers in a 6-hidden-layer DNN were collected in the histograms in Fig. 4.3. Each histogram contains 100 training sessions and 4 epochs are shown. At the beginning of training, the weights were selected to be normal or uniform distribution (uniform and initialized by VarianceScaling in Fig. 4.3). Given uniform activations distributed for the input layer, activations in each layer before training were depicted in the first row. The deeper the layer, the more unimodal is the distribution. After training, a clear trend along the axis of epoch is that the activations of a trained DNN were more concentrated at the two ends of a histogram.

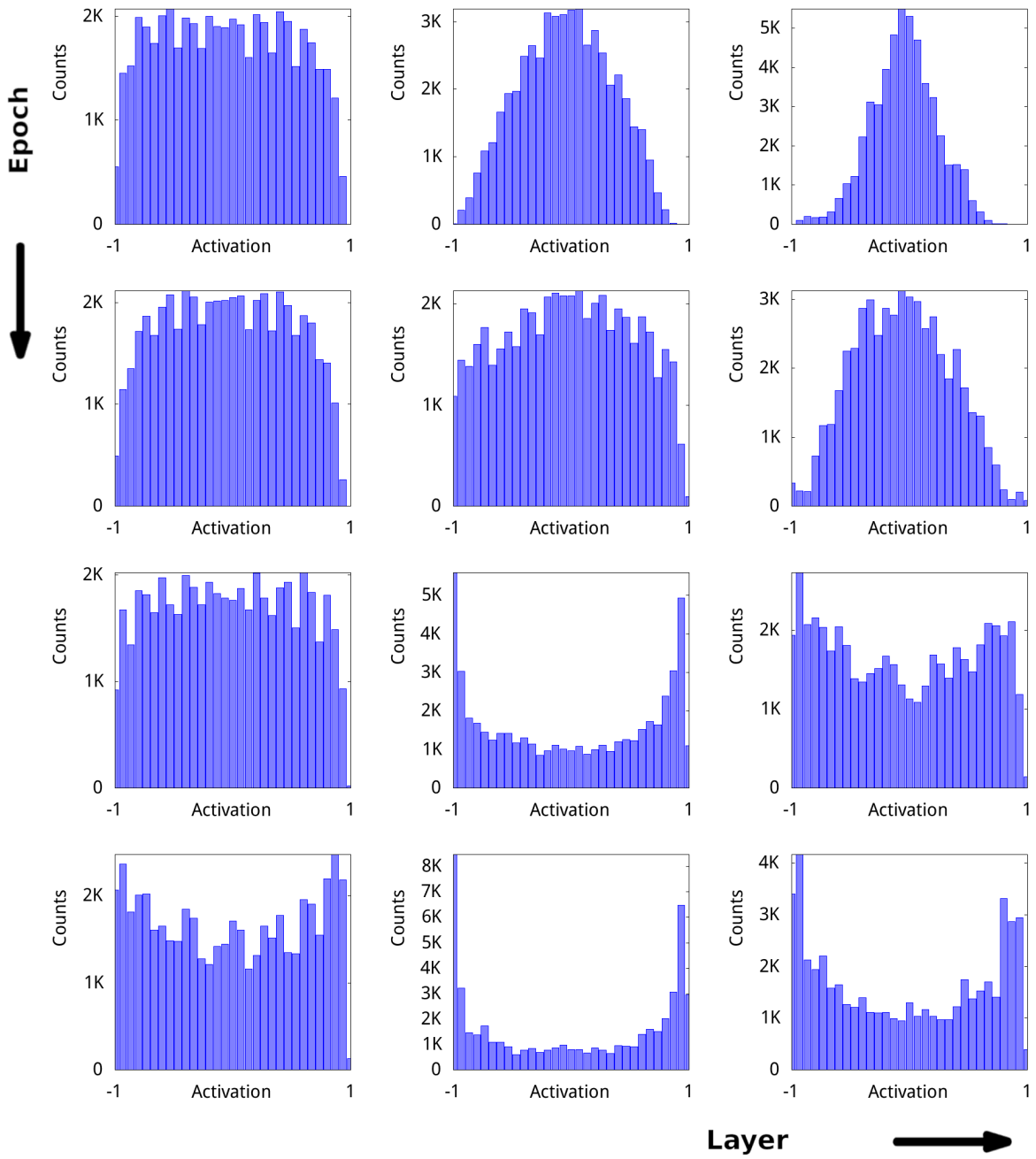


Figure 4.3: The time evolutions of the distributions of activations in three different layers in a 6-layer DNN.

In Fig. 4.3, most activations fall in the [-1,1] region, because the activation function was tanh in these DNNs. According to Fig. 1.1, the activation at a node of a layer is a function of the sum of many weighted inputs from all nodes in the previous layer. If biases are ignored (bias were initialized to zero in Fig. 4.3) and input and weights are independent variables, the activation of a node is like the sum of random variables in the Central Limit Theorem (CLT). Thus a unimodal distribution similar to the Gaussian distribution in the CLT is expected, as shown in the middle

and right panels of the first row in Fig. 4.3

4.3 The dynamics in the Information Plane

Calculating $I(X;T)$ and $I(T;Y)$ from the collected activations as introduced in the last chapters, without using MI estimators, yielded trajectories in the IP similar to those observed in Ref. [2]. However, the result depends on the level of binning in Fig. 3.5, or equivalently, the number of bins (colors) selected for the patterns of the histogram $p(X;Y)$. When the bin number is small, as in the first plot in Fig. 4.4, the $I(X;T)$ of a trajectory in the IP first increase and then decrease. The increase is termed the ERM phase, while the decrease is termed the compression phase. The change from the former to the latter is regarded as a general trend for the training process of DNNs [2]. Although this trend is clearly seen in the first plot of Fig. 4.4, it becomes less obvious, if the bin number gradually increases. For instance, the ERM phase becomes smaller in the second plot and even disappears in the third plot. Despite that, all points on those trajectories agree with the data processing inequality. That is, for any epoch, the $I(X;T)$ and $I(T;Y)$ of deeper layers are smaller. Notice that although the ERM phase is not seen in the third plot, one cannot say this DNN did not undergo that phase. The structures of the DNN in all the three plots of Fig. 4.4 are the same. The difference is not in their structure, but in how the data were processed. Furthermore, according to these plots, MI values decrease with smaller bin number. It is expected because a smaller bin number will squeeze more data points on $p(X;Y)$ into a bin, which will suppress the fine relation between variables before binning, and thus generally reduce the MI value.

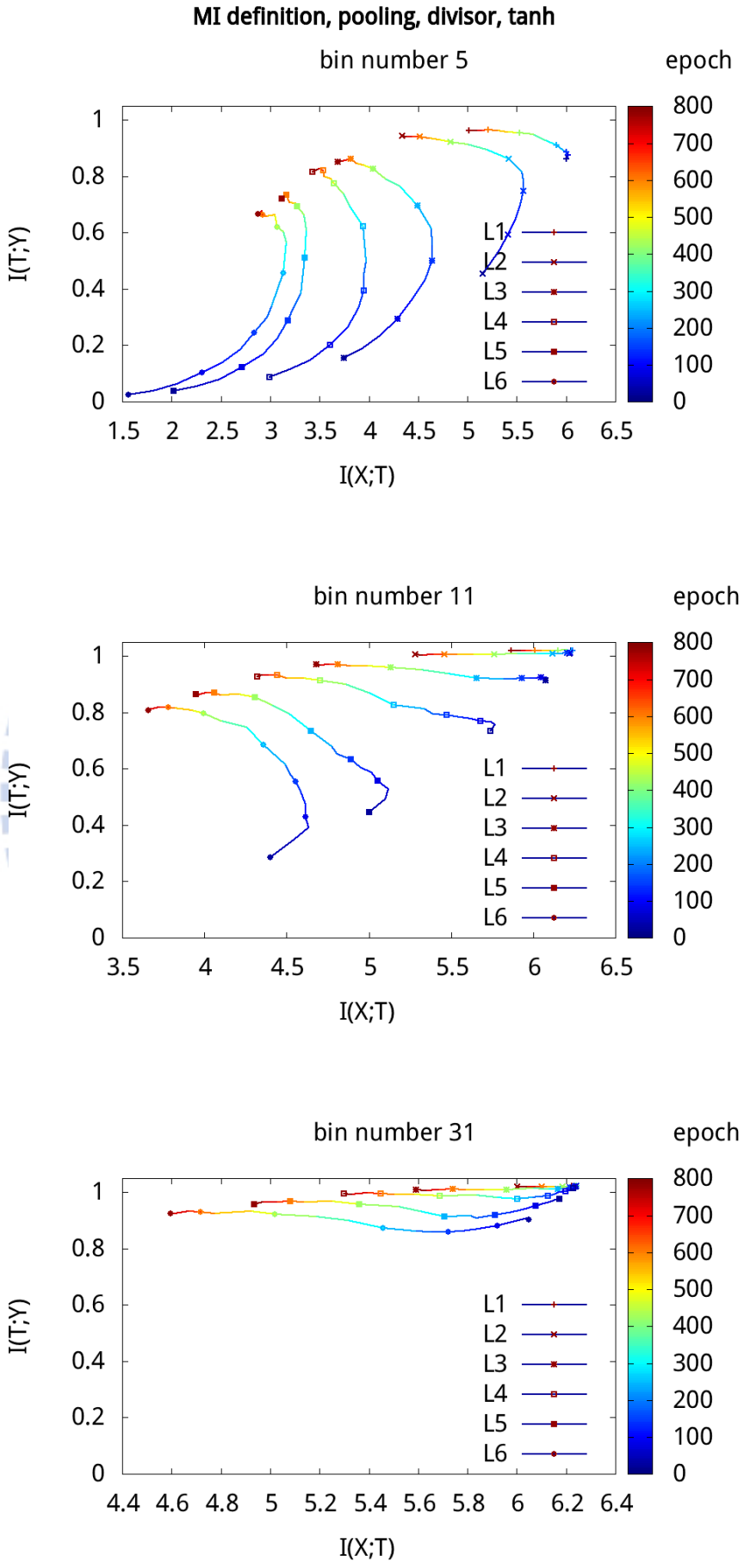


Figure 4.4: Trajectories in the IP with different bin numbers, 5, 11, and 31, for the task in eq. 3.1 with n = 512.

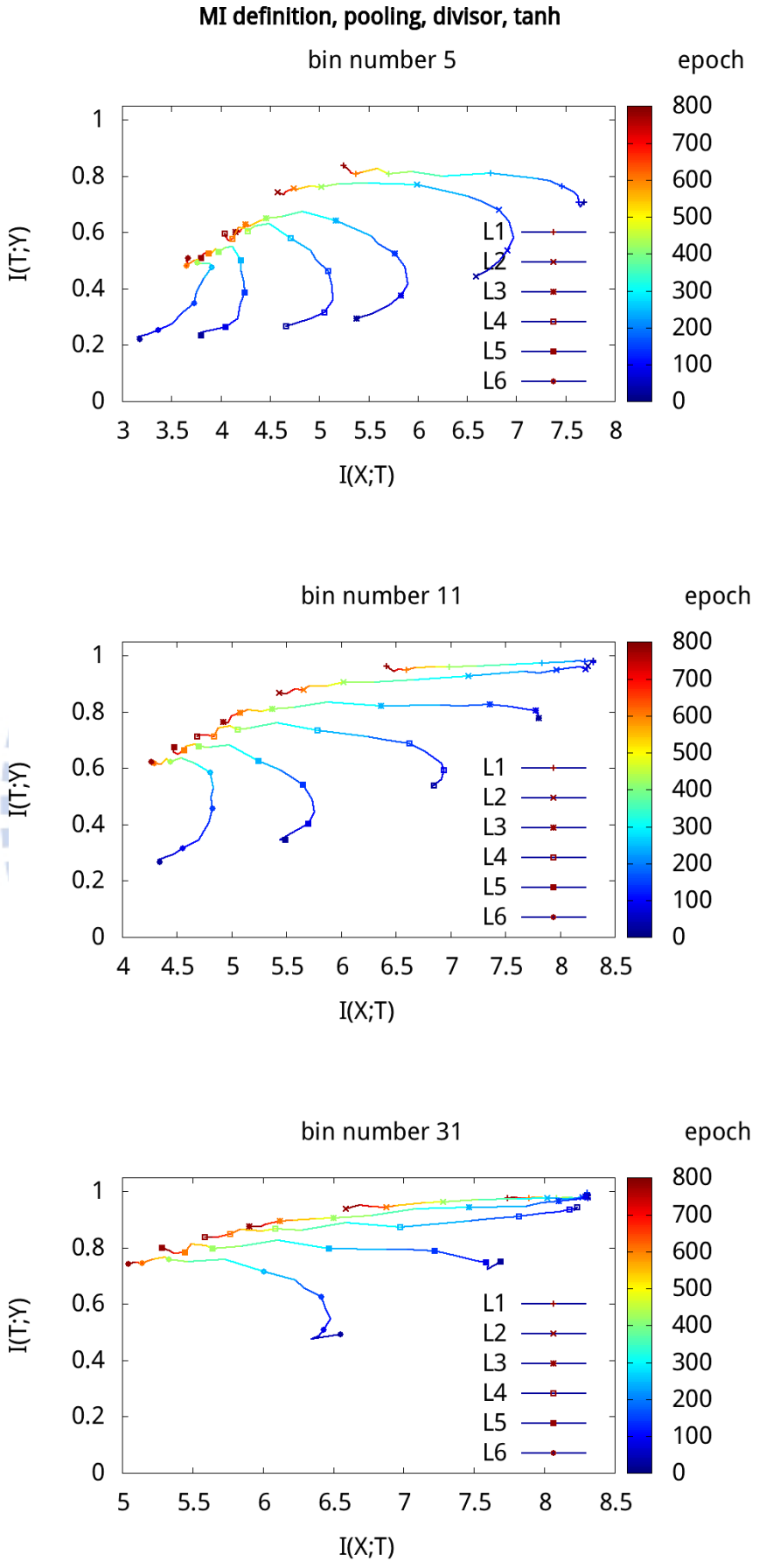


Figure 4.5: Trajectories in the IP with different bin numbers, 5, 11, and 31 for the task in eq. 3.1 with n = 4096. Here, Monte Carlo sampling was used.

As for the task in eq. 3.3, the dataset for $p(X, Y)$ becomes significantly larger. Thus, Monte Carlo re-sampling was used to delete part of the data to approximate MI. The result, as shown in Fig. 4.5, follows the same trend as that in Fig. 4.4.

4.4 Theoretical IB bound

Figure 4.6 shows the IB bound for task eq. 3.2 calculated by the algorithm in eq. 2.7-2.9. A comparison between those bound and the final $I(T;Y)$ values in Fig. 4.5 and fig.4.4 lead to the following conclusions. First, $I(T;Y)$ values are all lower than the upper bound predicted by the IB theory. Second, although our DNN had been well trained to perform accurate classification, the MI values in Fig. 4.5 are still far from the upper bound in Fig. 4.6. That is, the structure of the network is still not optimal from the aspect of information theory. Notice that on the way to approach an IB bound, we do not know which β the DNNs has selected.

To dissect this discrepancy, recall that training is a process aimed at reducing the loss calculated by cost function. However, minimizing the L in eq. 2.6 is aimed at reducing the difference between the compression level $I(X;T)$ and the accuracy level $\beta I(T;Y)$. Certainly, they have different goals. A DNN good at solving classification task has a small loss. There is no guarantee supporting that small loss implies small $I(X;T) - \beta I(T;Y)$. Even when β is large, such that compression contribution can be ignored, accuracies denoted by a small loss and a large $I(T;Y)$ are still not equivalent. The relation between the cost function and $I(T;Y)$ requires a deeper analysis. Even when small loss and large $\beta I(T;Y)$ are equivalent, there still exist other possibilities for the above discrepancy:

- The trained DNN has only arrived at a local minimum, not the global minimum, of the cost function. At the global minimum, $I(T;Y)$ will also arrive at its upper bound.
- The trained DNN has reached its global minimum of the cost function. The DNN has stopped to change its structure during further training. Thus, the L cannot be further minimized to reach the IB bound, estimated by the modified Blahut-Arimoto algorithm. In this DNN, the IB bound is only a weak bound. The IB theory does not say each system can come close to that bound.

- The iterative algorithm for solving eq. 2.6 has a problem with a histogram $p(X, Y)$ collected from real data, which contains several bins with zero values. This makes the theoretical bound of $I(T; Y)$ overestimated.
- The MIs calculated from their definition are not accurate enough, which misestimate the MI value.

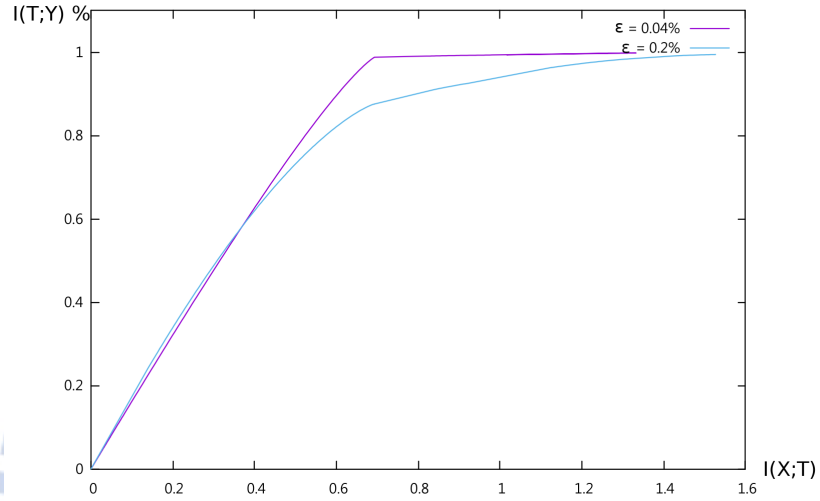


Figure 4.6: The IB bound for a DNN for task eq. 3.2 with n=512, calculated by eq. 2.9.

4.5 The effect of the size of the dataset

In [2], the existence of the turning points of trajectories in Fig. 2.1 was regarded as a general trend of DNNs during training. To understand how robust is that trend, [2] also explored the effect of different sizes of the dataset in their DNNs. In the following, we studied the same size effect in our DNNs. To this end, we divided the dataset into a training dataset and a testing dataset and changed the ratio of the amounts of data in these two sets. The dynamics in the IPs in Fig. 4.7, 4.8 and 4.9 reveal that smaller amount of training data leads to less compression phase and lower $I(T; Y)$ values. It seems to be reasonable, because generally the compression phase starts later than the ERM phase. Thus, DNNs requires a sufficient amount of training data to enter that phase. It explains our intuition that more training data will enhance the performance of DNNs.

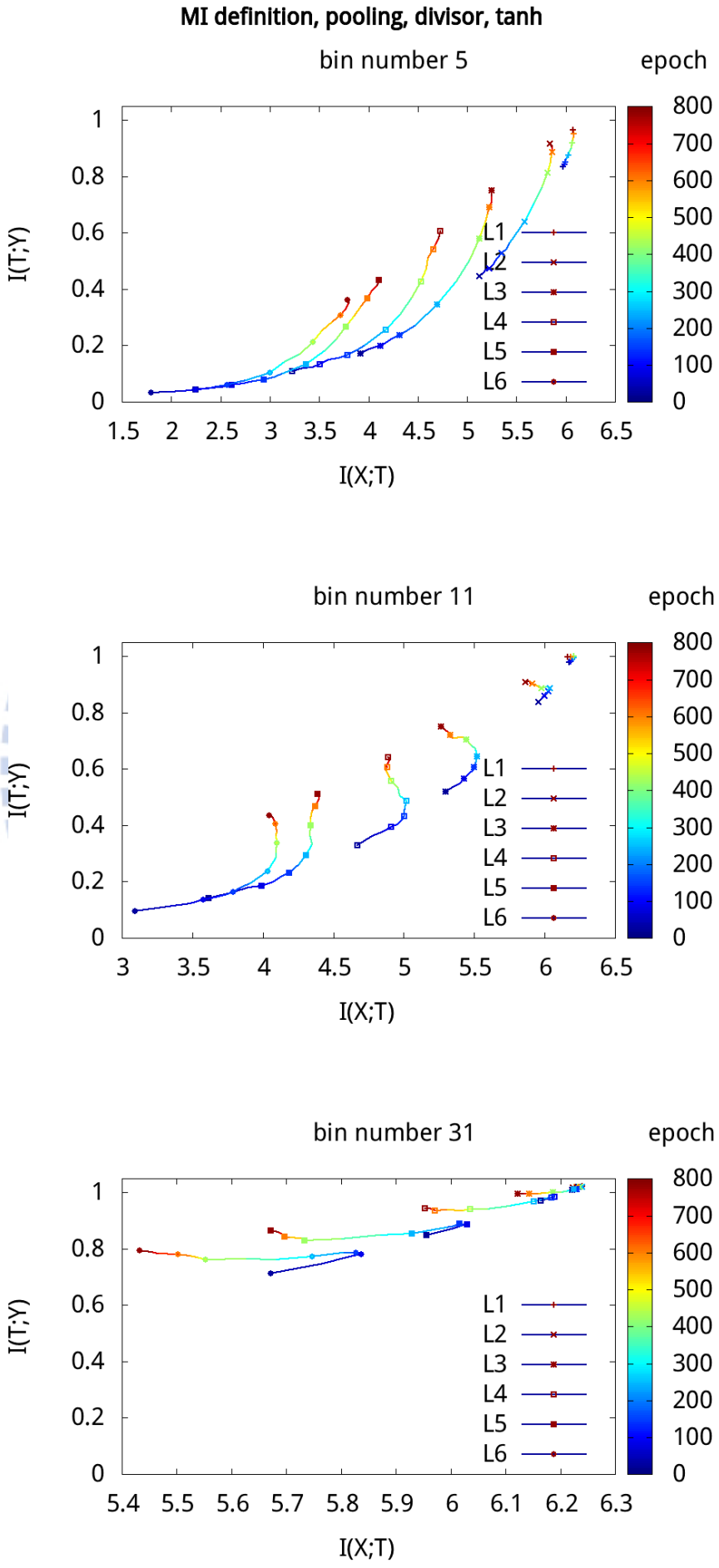


Figure 4.7: Trajectories in the IP with 25%:75% training to testing dataset ratio.

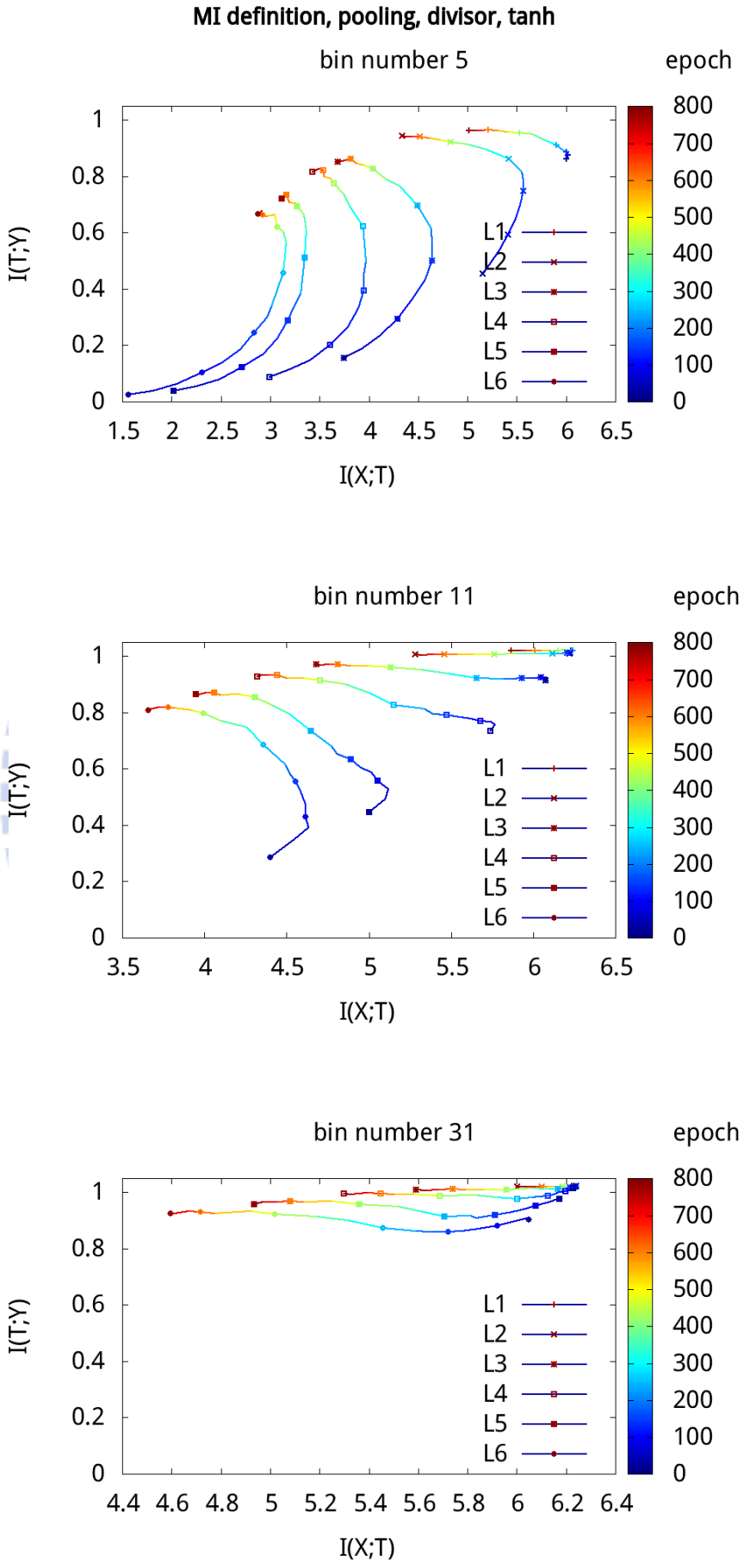


Figure 4.8: Trajectories in the IP with 50%:50% training to testing dataset ratio.

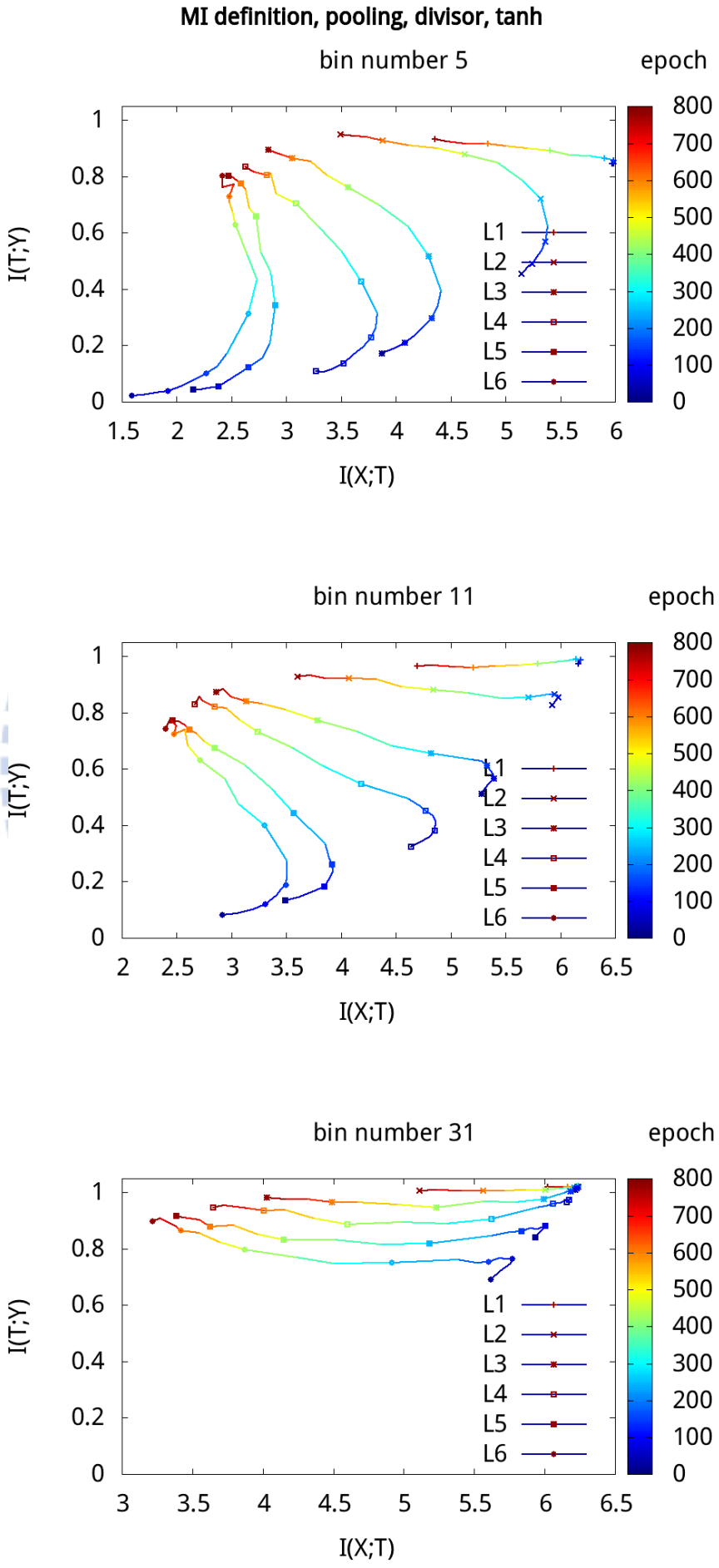
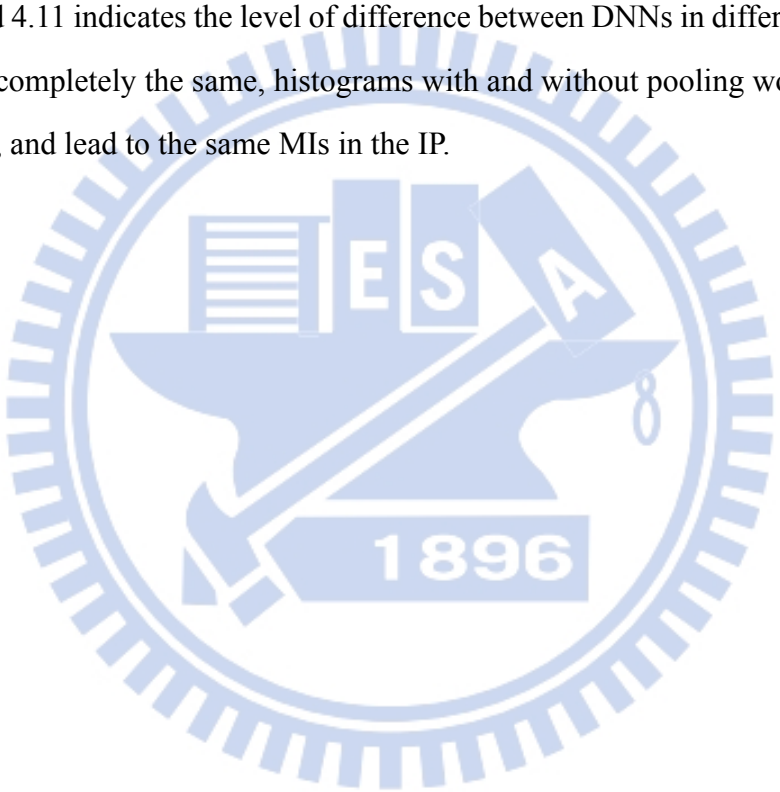


Figure 4.9: Trajectories in the IP with 75%:25% training to testing dataset ratio.

4.6 Calculating MIs with and without Pooling

The two kinds of MI evaluation mentioned in section 3.3.2 lead to different but similar dynamics in the IP. For the above divisor task, calculating the MIs of the large histogram $p(X;Y)$ with data pooled from all session gives Fig. 4.10. Averaging the MIs calculated from the small histograms of individual sessions yields Fig. 4.11. Interestingly, Fig. 4.10 and 4.11 are similar to the first and the second plots in Fig. 4.9. That is, pooling data has the same effect as reducing the bin number, which makes the trajectories turn more smoothly. The level of discrepancy between Fig. 4.10 and 4.11 indicates the level of difference between DNNs in different sessions. If these DNNs were completely the same, histograms with and without pooling would follow the same distributions, and lead to the same MIs in the IP.



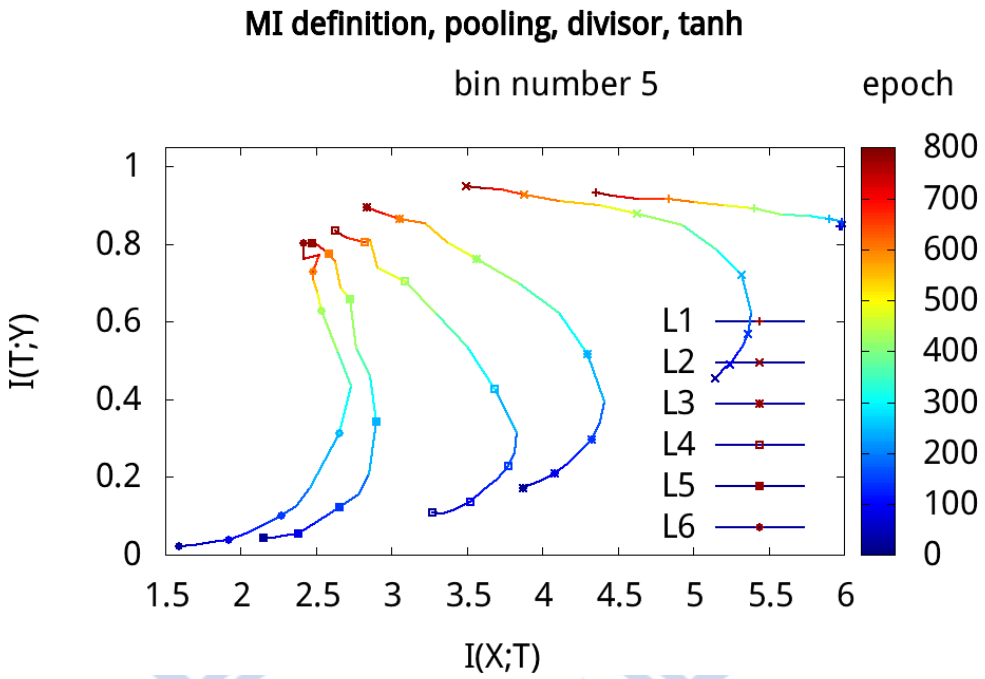


Figure 4.10: Calculating MIs after pooling the activation data. (75:25 training to testing dataset ratio, bin number: 4, n=512).

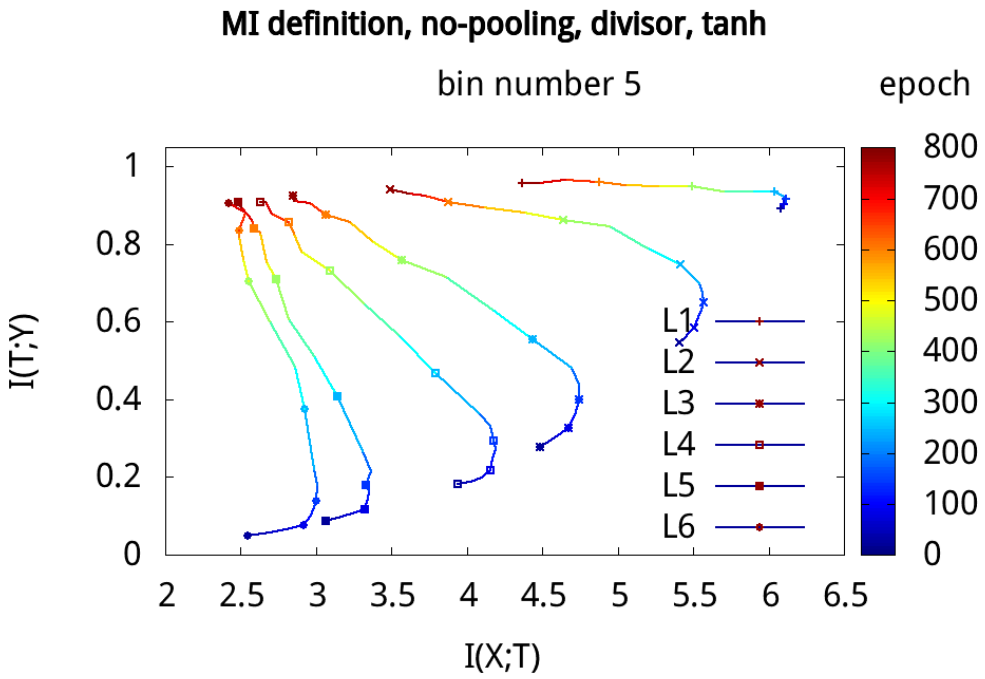


Figure 4.11: Averaging the MI values from individual sessions. (75:25 training to testing dataset ratio, bin number:4, n=512).

4.6.1 Approximating MIs by an estimator

The above MIs were also estimated by Kolchinsky's estimator, eq. 2.18 $\sim$ 2.22. Applying this estimator, procedures for calculating Fig. 4.10 and 4.11 lead to Fig. 4.12 and 4.13. respectively. A common change by using this estimator is that the compression phase is weakened.

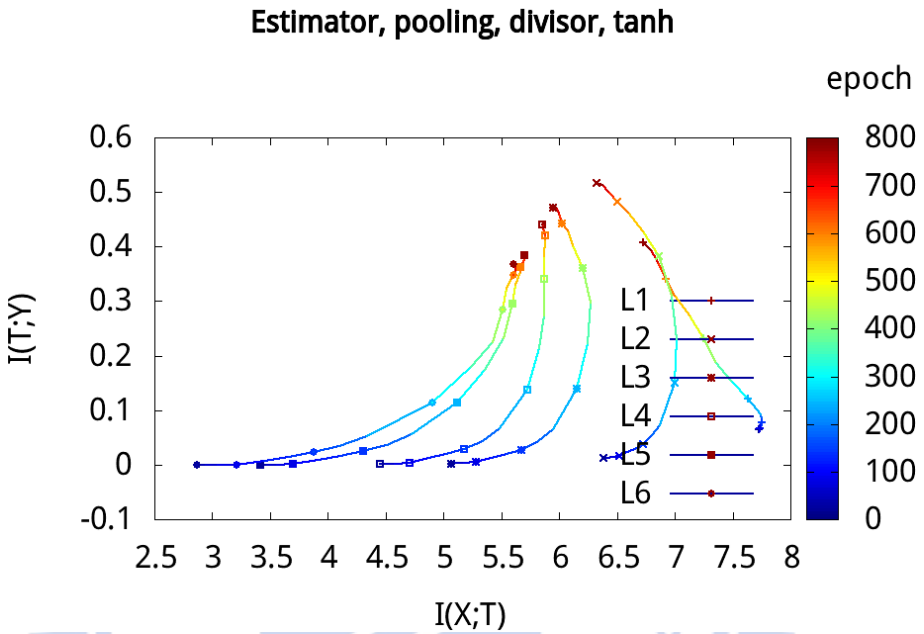


Figure 4.12: The MI curves in Fig. 4.10, calculated by Kolchinsky's estimator instead.

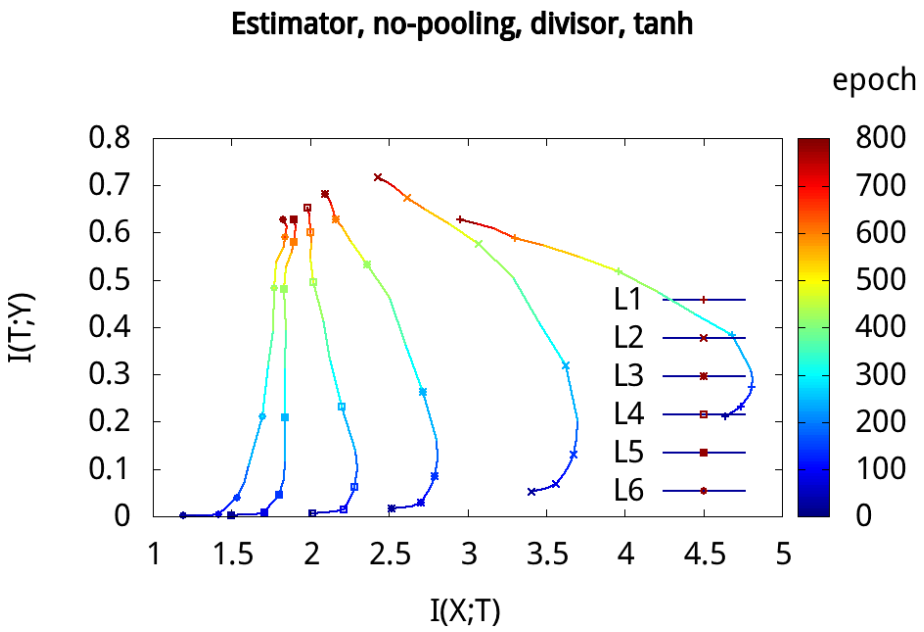


Figure 4.13: The MI curves in Fig. 4.11, calculated by Kolchinsky's estimator instead.

4.7 A space extended from the IP

Since the accuracy of DNNs determined by the cost function is different from that characterized by $I(T;Y)$, why not let us consider the former as a coordinate to describe the evolution of the state of a DNN, in additional to the two coordinates of the IP. In the following, we add the accuracy as an extra dimension to the IP and extend the 2-dimensional IP to a 3-dimensional space, as shown in Fig. 4.14. Notice that whereas each layer has its own MI values, $I(X;T)$ and $I(T;Y)$, the accuracy only refers to the whole DNN. Denoting accuracy by α, the points of different trajectories (layers) in the ($I(T;Y)$,$I(X;T)$, α) space have the same accuracy value at the same epoch.

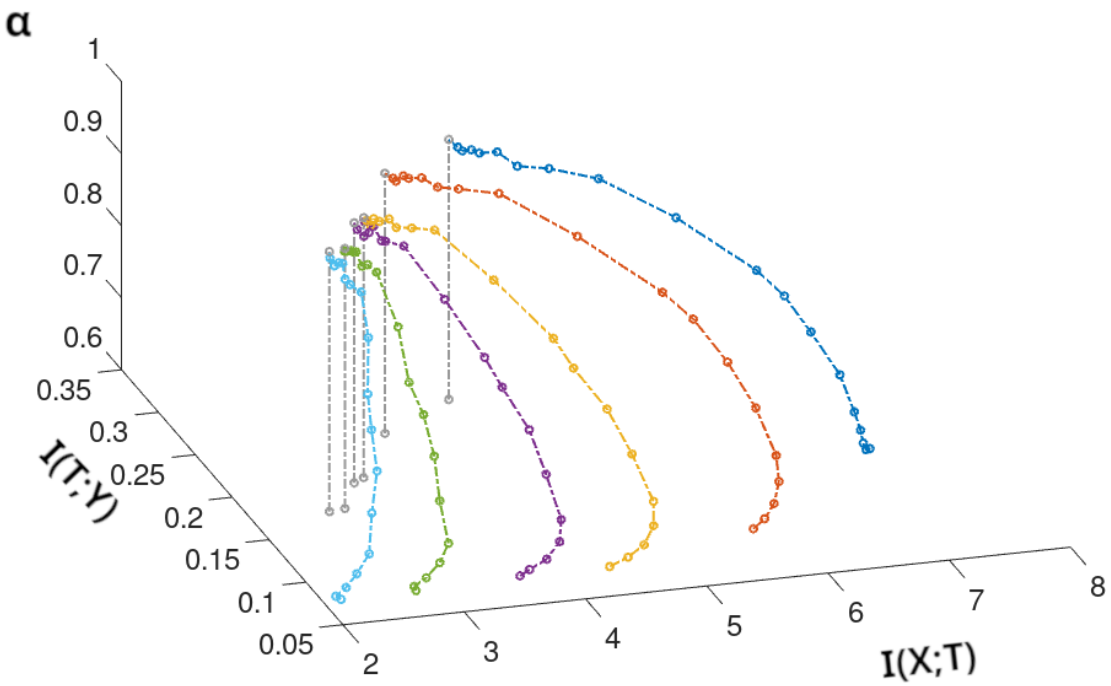


Figure 4.14: Trajectories in the 3-dimensional space ($I(T;Y)$, $I(X;T)$, α) for the task in eq. 3.1 with n=512. Different colors denote different layers.

In the first plot in Fig. 4.15, the turning points in the 2-dimensional IP are regarded as a transition between two phases of training. If we consider another 2-dimensional plane ($I(X;T)$,α), projected from the (($I(T;Y)$, $I(X;T)$, α) space, as in Fig. 4.15, the trajectories show the phase transition from a different perspective. In the original IP, how closely a hidden layer T is related to the label Y, $I(T;Y)$, is compared with how closely this hidden layer is related to the

input layer X, $I(X;T)$. In the ($I(X;T)$,α) plane, the former is replaced by how accurately the DNN can predict a mathematical task. Both the original IP and the ($I(X;T)$,α) plane show that a shallower hidden layer (a T close to X) compresses more strongly than a deeper layer (a T far from X) does. Both figures also show that the compression phase takes more time than the initial ERM phase. A special aspect revealed only by the $I(X;T)$-α plane is that the improvement of accuracy does not stop at the ERM phase. During the compression phase, the accuracy continues to rise.

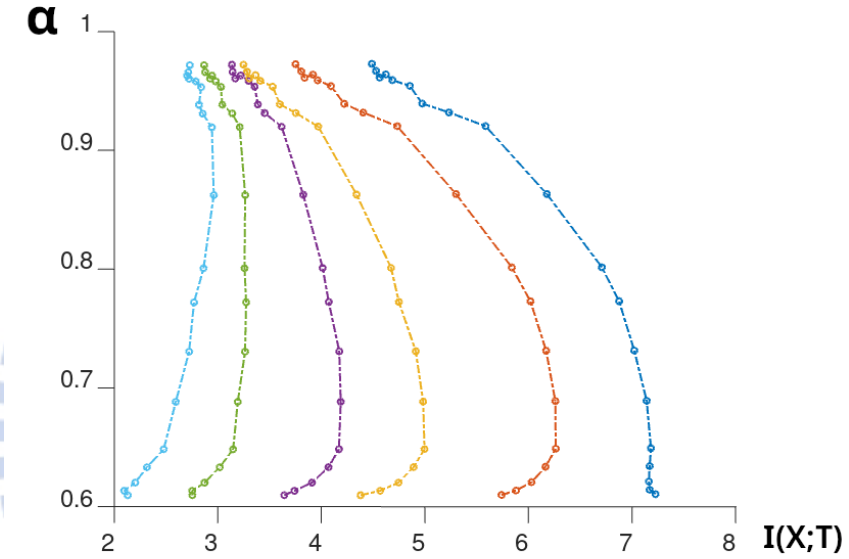


Figure 4.15: Trajectories in the plane ($I(X;T)$,α) projected from those in Fig. 4.14.

When averaging the α over all sessions and plotting it versus epoch, the accuracies of DNNs in both mathematical task in eq. 3.2 and 3.3 are shown in Fig. 4.16 and 4.17 respectively. Interestingly, both DNNs reach good performance within 300 epochs, independent of the difficulties of the task.

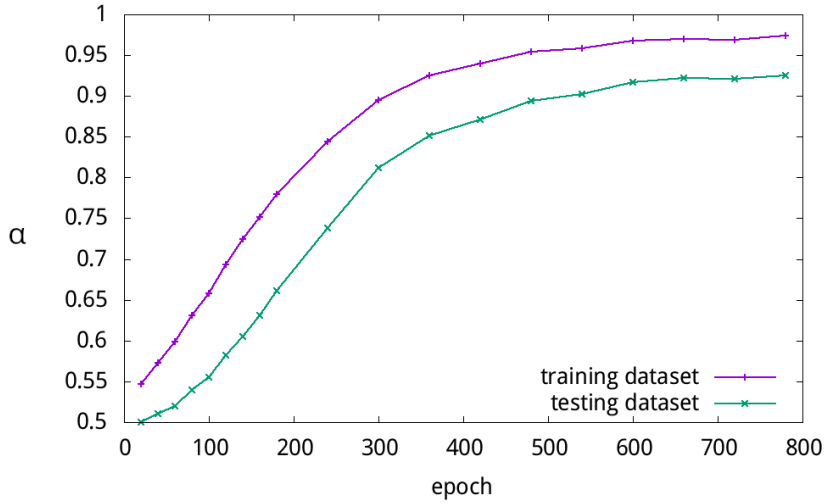


Figure 4.16: The evolution of average accuracy (n = 512).

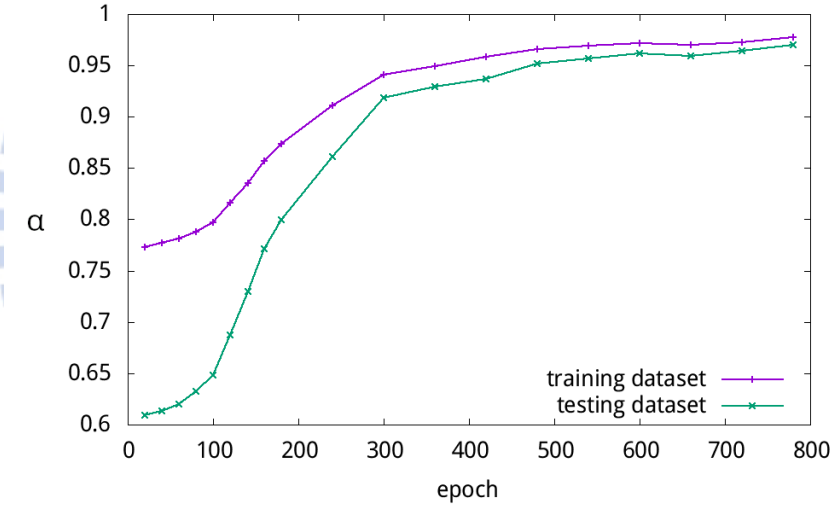


Figure 4.17: The evolution of average accuracy (n = 4096).

4.8 The drift and diffusion phases of SGD optimization

In [2], the authors proposed to use the "*normalized mean and standard deviation of the weights' stochastic gradients for every layer of the DNN as function of the stochastic epoch.*" This gradient of weight is the gradient calculated by the SGD optimizer. However, the meanings of these above two metrics are not precisely defined, which causes confusion and results in distinct implementations from different groups. For example, in Ref. [30], the mean is the L2 norm of a time average of the gradients of weights over all mini-batches in an epoch,

$$\text{norm}_{L2}(\text{mean}(\text{several gradients of weights})). \tag{4.1}$$

Here, the "gradients of weights" are the "gradients of the cost function with respect to the change of weights and the "mean" inside the norm is a time average. After taking that average, we still have a vector. The L2 norm of that vector is a number. In Ref. [30], the standard deviation is the L2 norm of the standard deviation of the above gradients around their mean gradient,

$$\text{norm}_{L2}(\text{standard\_deviation}(\text{several gradients of weights})). \tag{4.2}$$

As the standard deviation inside the norm is a vector, its norm is a number.

In our study, we considered similar means and standard deviations of the above gradients, but for the full training dataset, instead of mini-batches,

$$\frac{\text{norm}_{\text{C}}(\text{single gradient of weights})}{\text{RSS}(\text{weights})} \tag{4.3}$$

$$\frac{\text{standard\_deviation}(\text{single gradient of weights})}{\text{RSS}(\text{weights})} \tag{4.4}$$

Here, the "norm_{C}" in eq. 4.3 is an average over all components of a vector, while the denominator of eq. 4.4 is the root sum square (RSS) of all weights, which is also called a norm in "numpy". Using eq. 4.3 and 4.4, we obtained Fig. 4.18 and 4.19, respectively. These plots do not have the transition feature observed in Ref. [2].

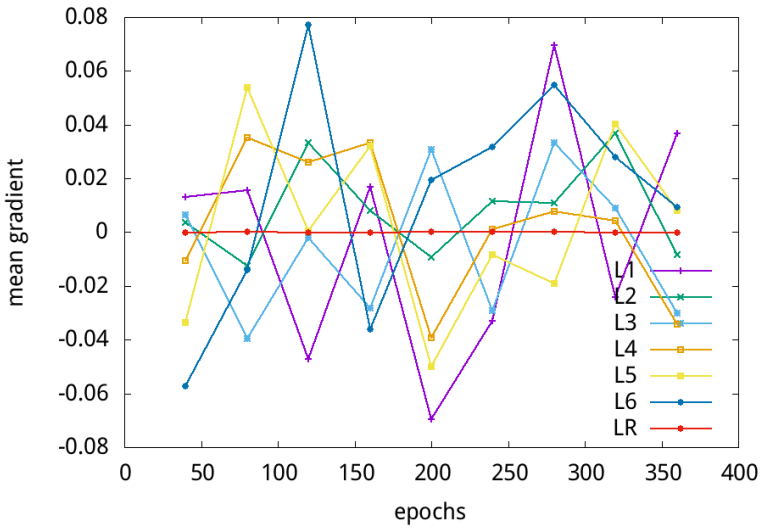


Figure 4.18: The time evolution of the mean defined in 4.3, averaged over 60 DNNs.

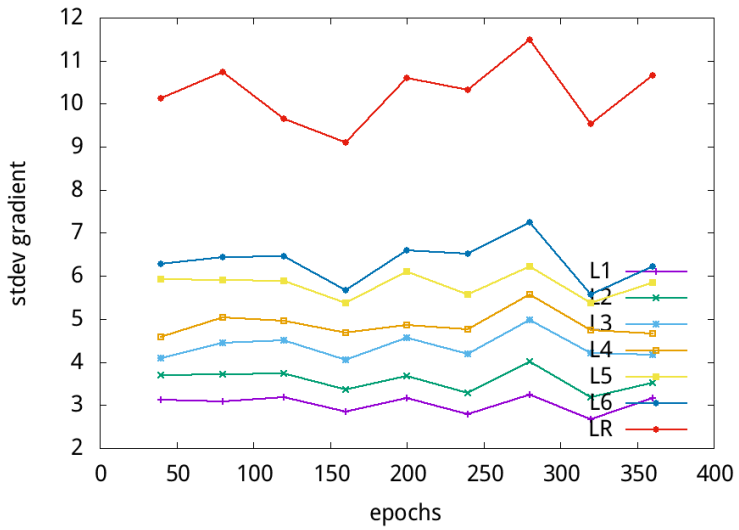


Figure 4.19: The time evolution of the standard deviation of defined in eq. 4.4, averaged over 60 DNNs.

In addition to the mean in eq.4.3 and the standard deviations in eq.4.4, we also calculated the mean of the loss averaged over different sessions and its fluctuation in Fig. 4.20 and 4.21 respectively. In this study, the whole DNN was analyzed, rather than individual layers in Fig. 4.18 and 4.19. In Fig. 4.20, the evolution of the loss of only a single training session was recorded, where an apparent sharp drop is readily seen. In Fig. 4.21, an ensemble of sessions like that in Fig. 4.20 were averaged, which yields a slightly fluctuating decreasing curve. The increasing fluctuating curve in the same plot is the difference between the fluctuating decreasing curve and its smoothed curve, calculated by the Savitzky—Golay filter. That indicates that the

fluctuation grows steadily from the beginning and saturates as the DNN converges to its final state. Analyzing a DNN by the loss of the whole network, instead of by the gradient of loss in each layer, is similar to analyzing a DNN by the accuracy in Fig. 4.15, instead of by the MIs of each layer in the IP.

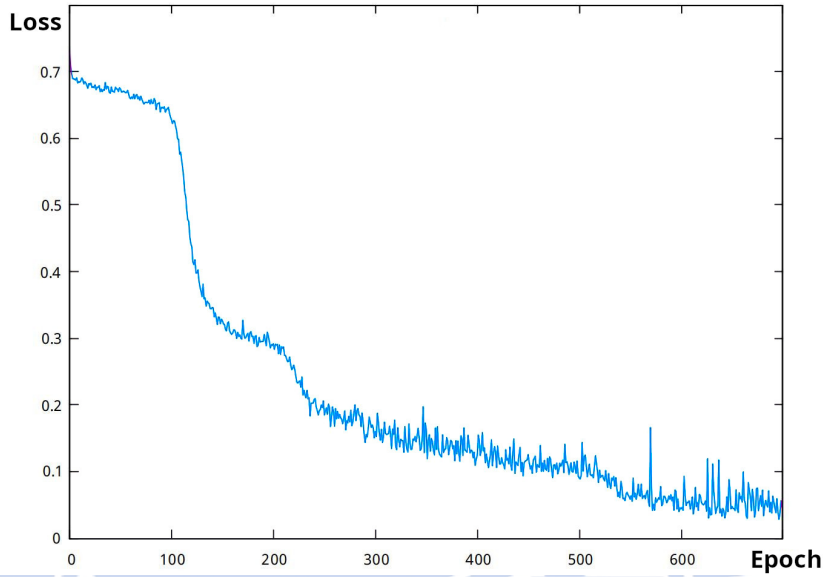


Figure 4.20: The time evolution of the loss of a DNN during training.

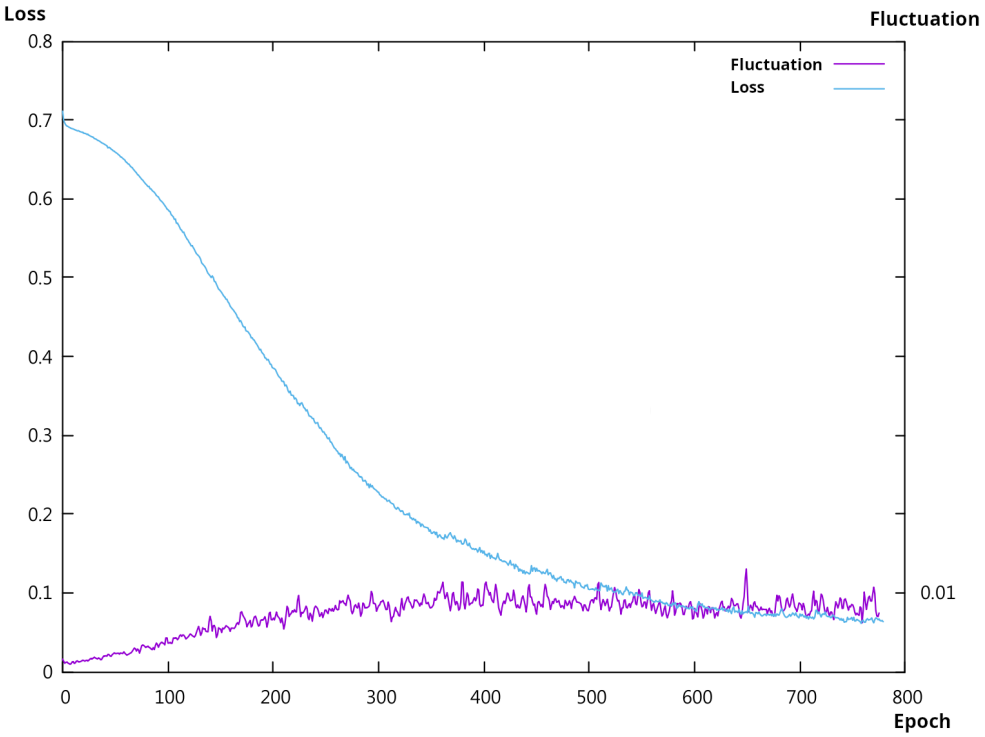


Figure 4.21: The time evolution of the mean loss of DNNs and magnitude of their fluctuations.

Why are the two metrics in Fig.4.21 interesting? The convergence of the loss reduction process to its final state and the fluctuation of loss around that state are determined by the SGD. These behaviors are like the convergence of a physical system to its equilibrium or steady states. A simple example is a Brownian particle attracted by an optical tweezer to approach the minimum of the quadratic potential well of that tweezer. When the particle is closer to the minimum, the force exerted on the particle becomes smaller and thus the steps of random movement also become smaller. By contrast, the fluctuations will become larger, because the bottom of the well is generally flatter as long as the bottom is not extremely narrow. The reduced movements above and increased fluctuations are exactly what occurs during the training process via SGD optimization as seen in Fig. 4.21.

4.9 The Rectified Linear Unit activation function

To understand whether activation functions affect the IP trajectories, the tanh function studied so far was replaced by the Rectified Linear Unit (ReLU). The DNN for eq. 3.2 was reconsidered. All other setups were kept the same for the sake of comparison. As a result, the trained DNNs tend to give a slightly lower accuracy, as shown in Fig. 4.22, when compared with Fig.4.1. Because the activation values of ReLU do not fall within the [-1,1] region, the number of bins to be selected is indefinite. The activation of ReLU was instead multiplied by an integer m and then rounded up. The number m takes over the role of previous bin number. In this way, the IP of ReLU shows very small compression phase, as shown in Fig. 4.23, where data pooling was Fig.3.4 is used to calculate MIs. A strange finding here was that the first hidden layer did not follow the data processing inequality in eq. 2.3.

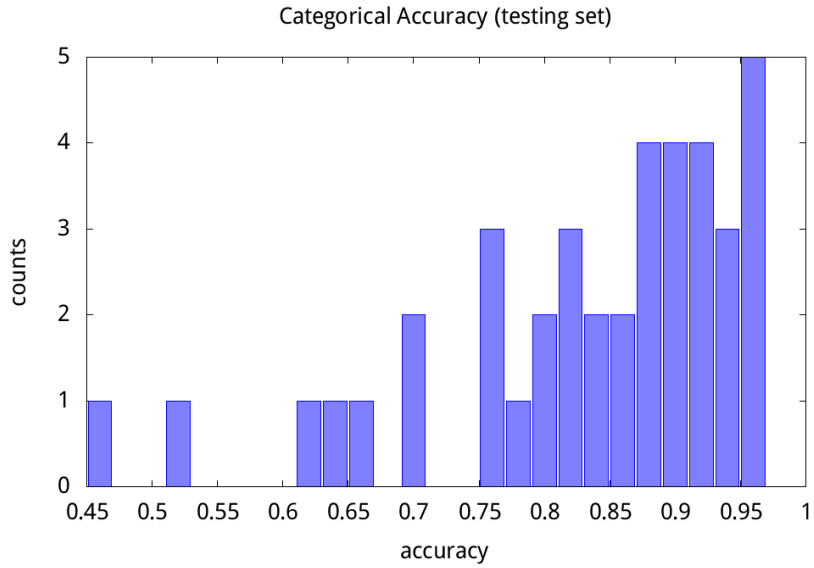


Figure 4.22: The accuracy distribution of trained DNNs with the ReLU activation function (n=512).

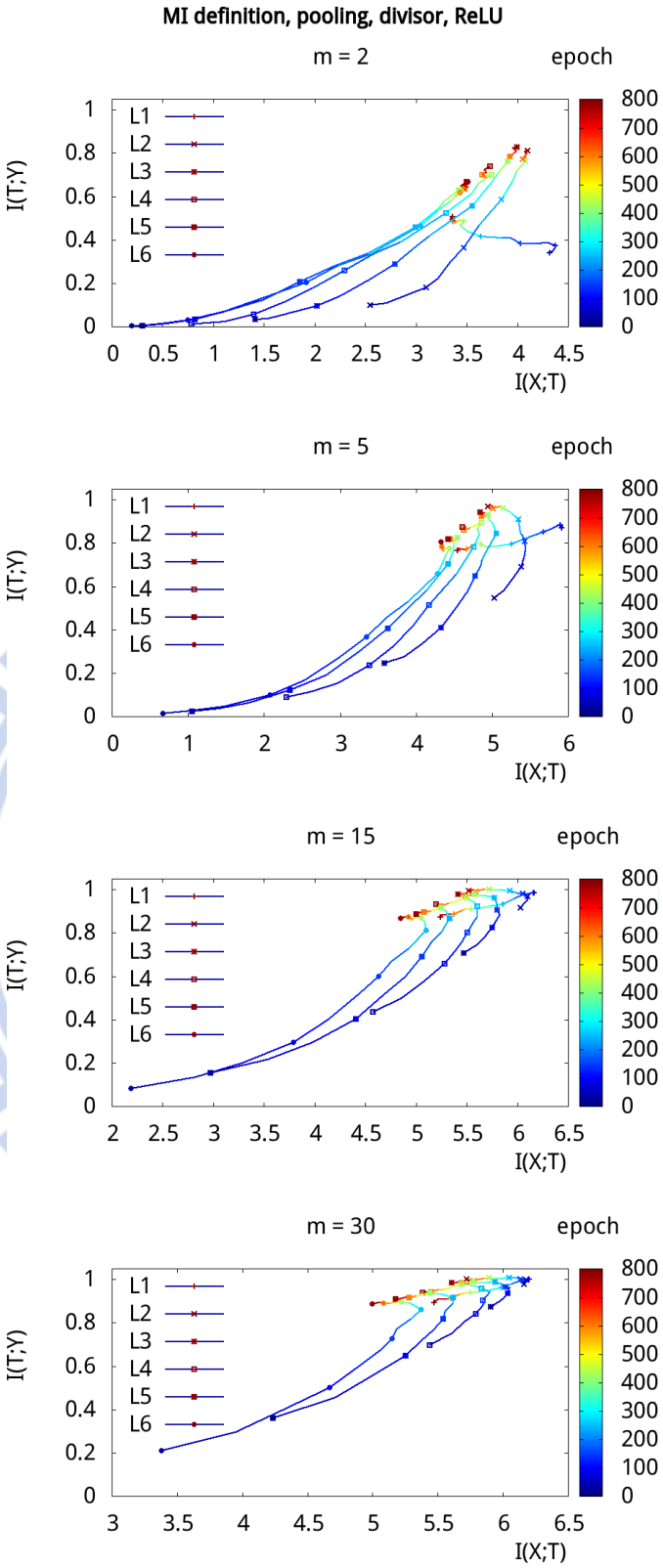


Figure 4.23: The IP trajectories of DNNs with the ReLU activation function by using data pooling (n=512).

If without pooling, averaging over MI values yields a similar result to Fig. 4.23, except for the first layer, as shown in Fig. 4.24. In these plots, the violation of the data processing

inequality in Fig. 4.23 is lifted.

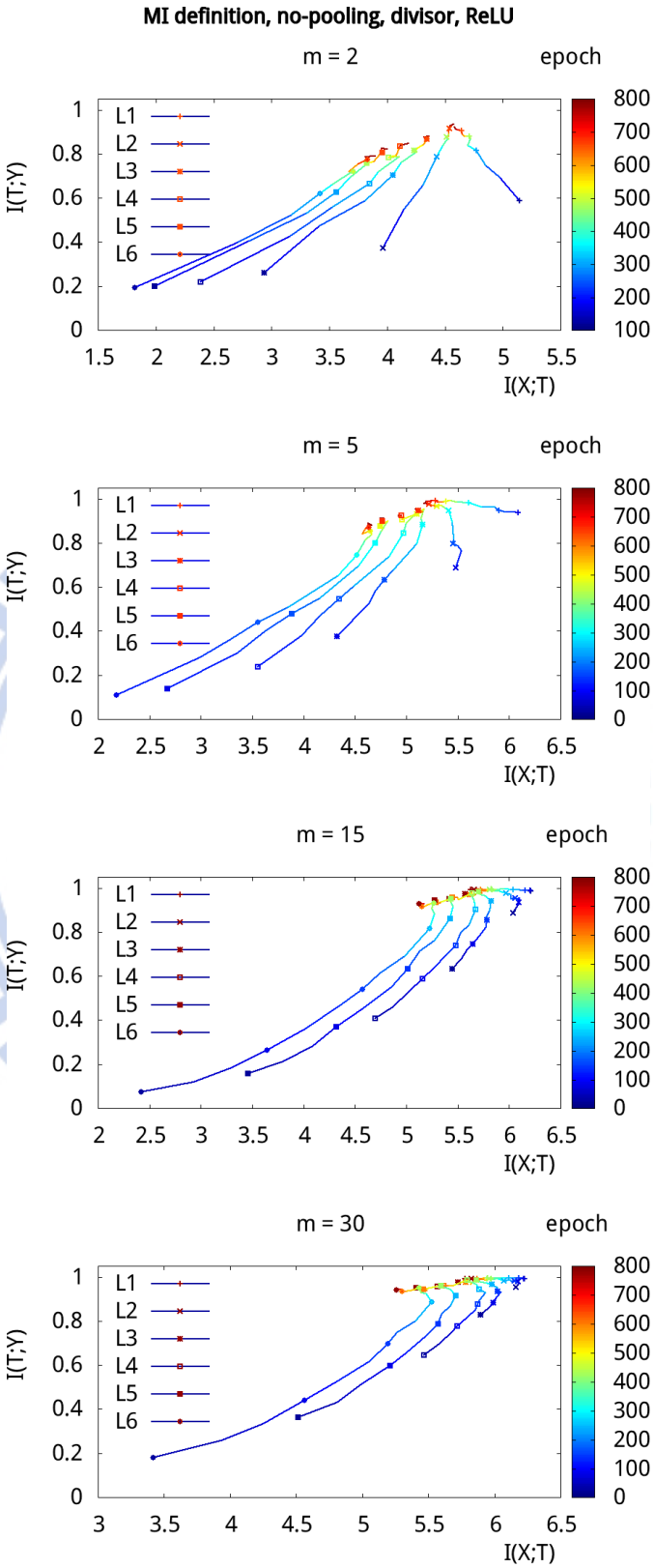


Figure 4.24: The IP trajectories of DNNs with the ReLU activation function, calculated from the same data as those in Fig. 4.23, but without pooling data.

When using Kolchinsky's estimator, the trajectories in Fig. 4.23 and 4.24 become Fig. 4.25 and 4.26.

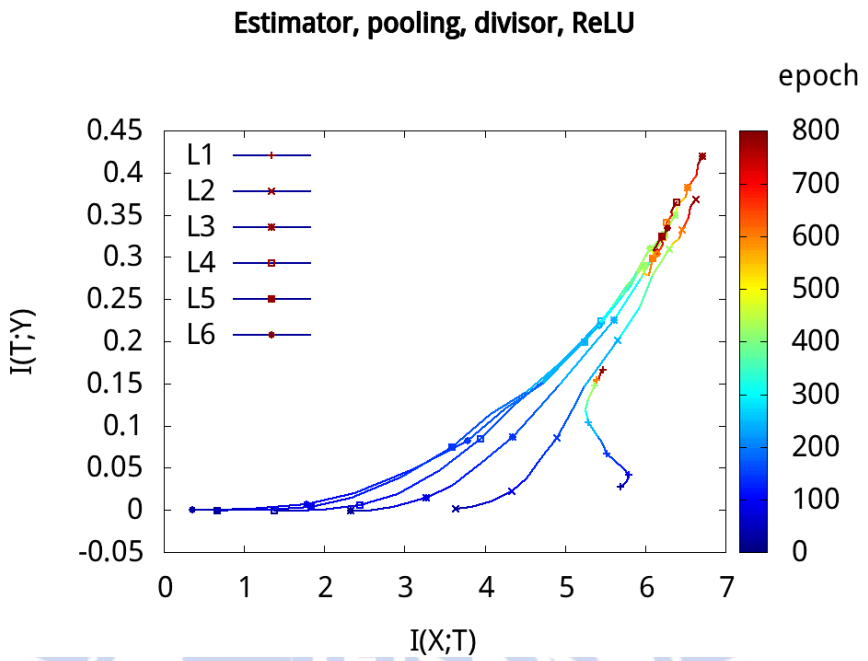


Figure 4.25: The IP trajectories calculated from the same data as those in Fig. 4.23, but by Kolchinsky's estimator instead.

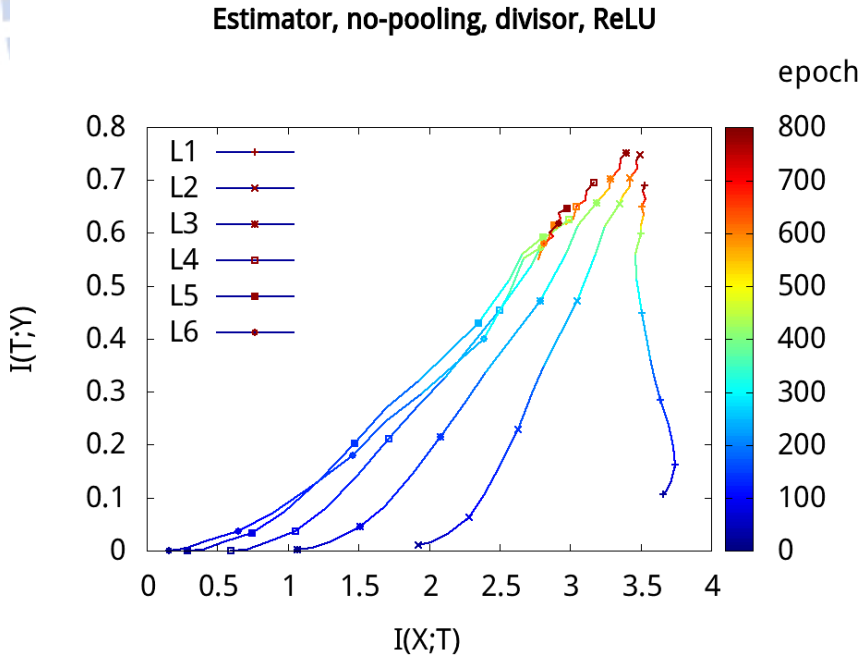


Figure 4.26: The IP trjaectories calculated from the same date as those in Fig. 4.24, but by Kolchinsky's estimator instead.

Recall that if the activation function is tanh, the change of IP trajectories from "pooling data" to "no pooling data" is significant. Examples include the change from Fig. 4.10 to 4.11

(MI definition) and from Fig. 4.12 to 4.13 (MI estimator). By contrast, if the activation function is ReLU, the change of IP trajectories from "pooling data" to "no pooling data" is insignificant. Examples include the change from Fig. 4.23 to 4.24 (MI definition) and from Fig. 4.25 to 4.26 (MI estimator).

4.10 MNIST

To understand whether the general turning trend in Fig. 2.1 exists in other more complicated tasks, a DNN for MNIST hand-writing recognition was also analyzed. This DNN is significantly more complex than those used for the mathematical task in eq. 3.1. To reduce computational complexity, this DNN was designed to be shallow and small, with an Input-16-16-16-Output structure. The input image data has 28×28 pixels with an 8-bit grayscale, which were used to train the DNN. However, when MIs were calculated, the sizes of the images were down-scaled to 4×4 and the grayscale was converted to black-and-white to ease the computation complexity.

In Fig. 4.27, the trajectories in the IP for the above MNIST task were calculated for different bin numbers, where MIs were calculated after pooling data, as explained in Fig.3.4. Unfortunately, they do not have the same features as those in Fig. 4.4 and 4.5 for eq. 3.1. This finding leads to two possibilities: (i) The trend claimed in Fig. 2.1 in [2] is still not general. (ii) This trend is general, but can be observed only by a better data representation without down-scaling. Interestingly, we are not the only group which cannot find an apparent trend on MNIST. This trend was neither found in another recent work [25] from Santa Fe Institute, Massachusetts Institute of Technology, and Arizona State University.

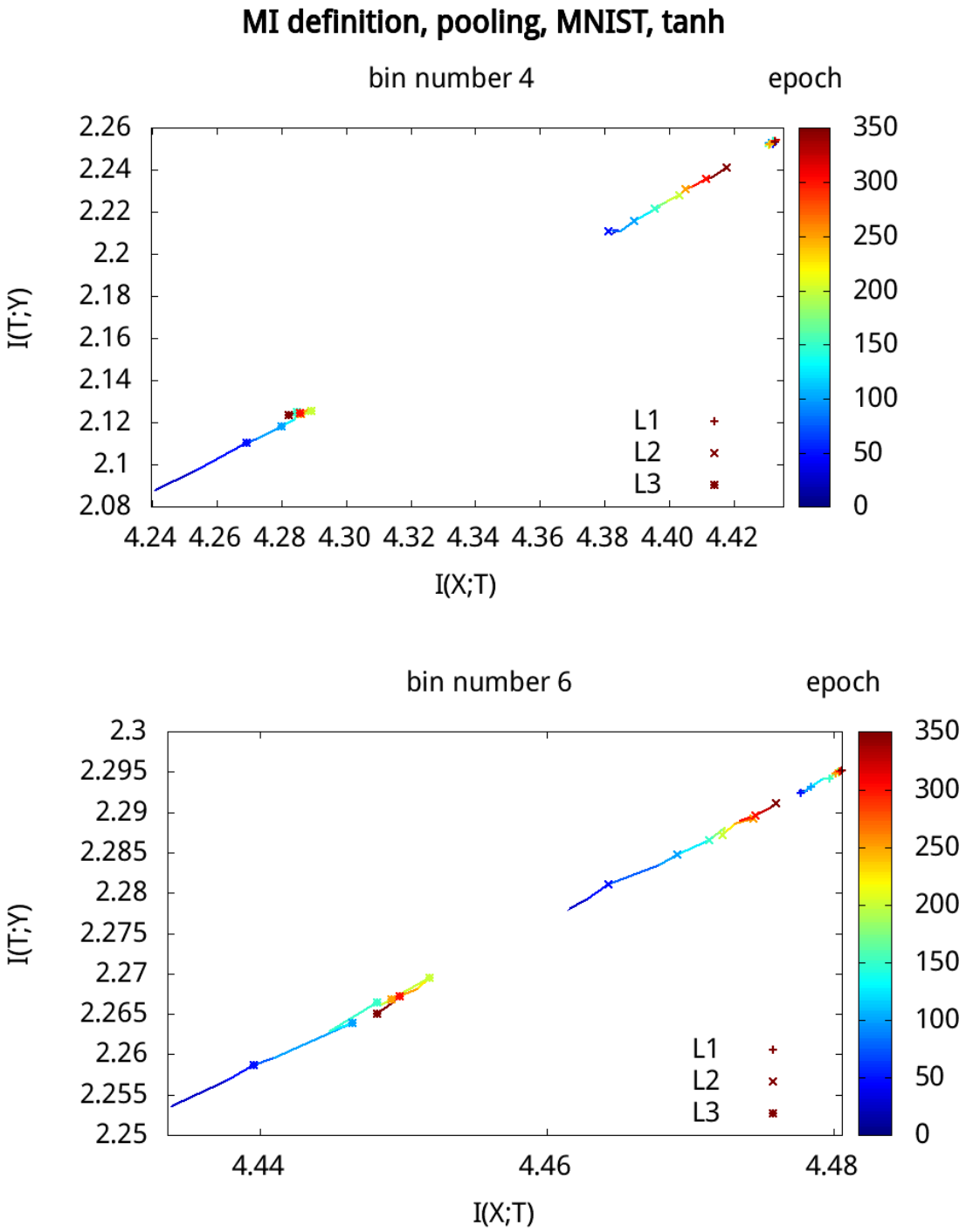


Figure 4.27: The trajectories in the IP of a DNN for MNIST hand-writing recognition. The vertical axis is not rescaled to $I(X;T)/I(X;Y)$ as before because the estimated $I(X;Y)$ value is not very precise for this large system.

To test how representative is Fig.4.27, we studied how sensitively these plots change with pooling data. In contrast to the result in section 4.6, averaging MIs of individual sessions without pooling will significantly change the IP trajectories for the MNIST task, as shown in Fig. 4.28. It indicates that the DNNs in different sessions have very different structures, even at the end of

the training, because the final MIs in Fig. 4.28 are not those in Fig. 4.27.

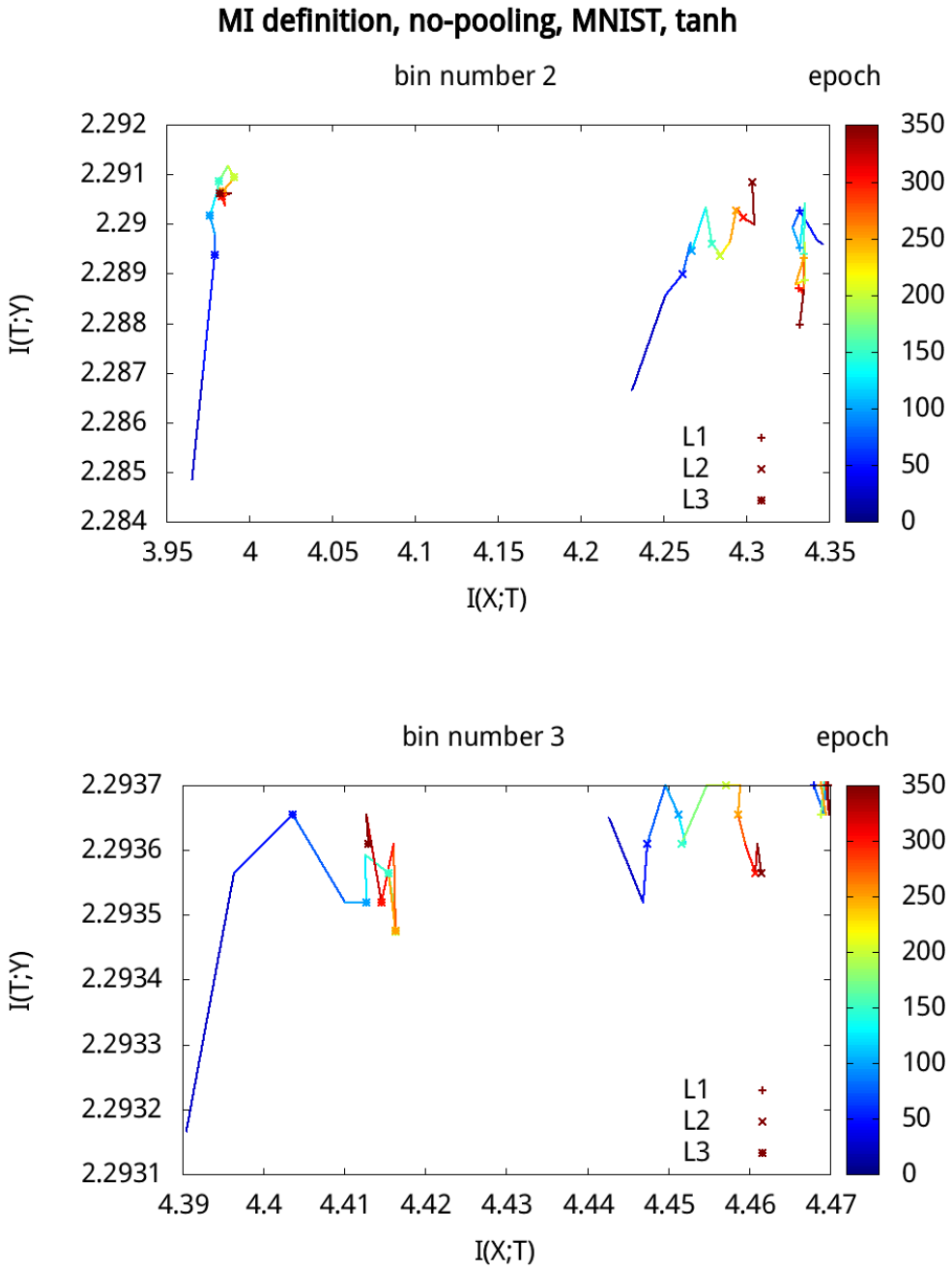


Figure 4.28: These two plots are calculated from the same data as in Fig. 4.27, but with MIs averaged from those of individual sessions without pooling. Recall that pooling has the same effect of reducing bin number. For the sake of an easier comparison to Fig. 4.27, the bin number is reduced in these figures.

Furthermore, when using Kolchinsky's MI estimator, the trajectories in Fig. 4.27 and 4.28 change to those in Fig. 4.29 and 4.30. With these figures, it is clear that the IP trajectories for MNIST calculated by that estimator are in consistent with those calculated by the original definition of MI.

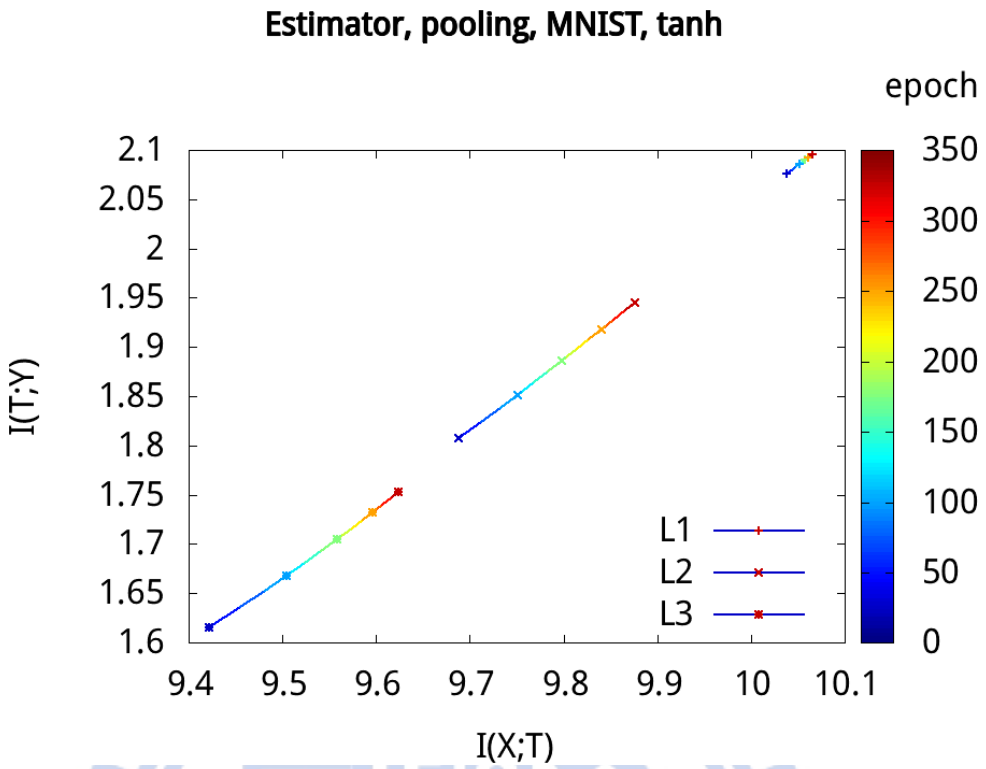


Figure 4.29: The MI curves in Fig. 4.27 calculated by Kolchinsky's estimator instead.

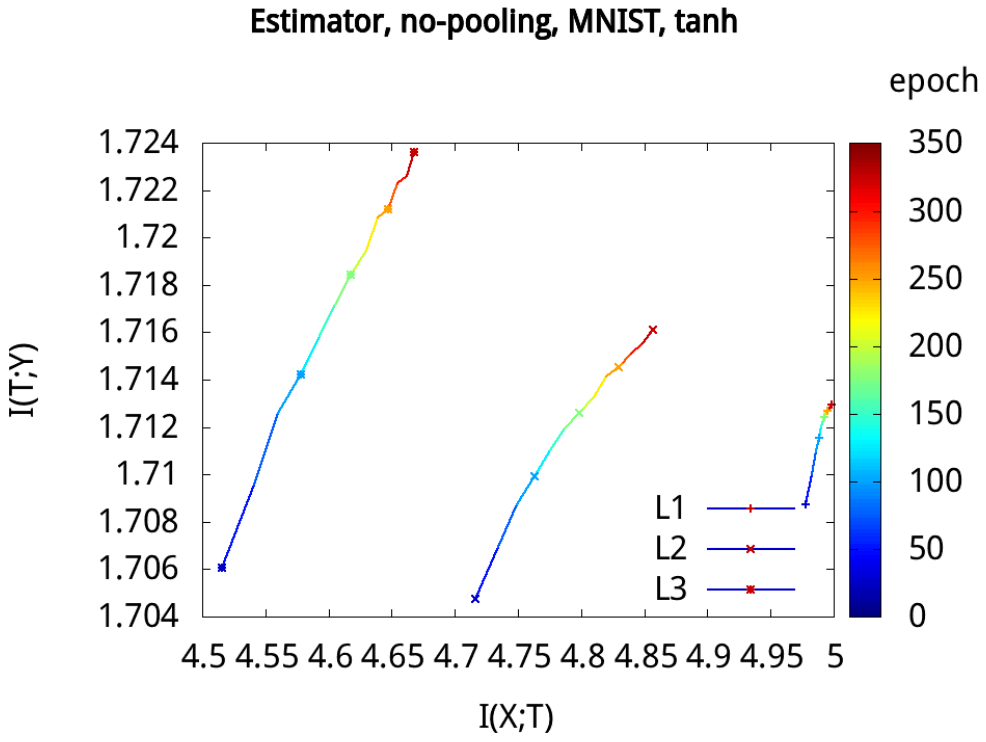


Figure 4.30: The MI curves in Fig. 4.28 calculated by Kolchinsky's estimator instead.

Chapter 5

Conclusion

After the recent breakthrough of the techniques of DNNs, several approaches have been suggested to analyze this incredible tool. Among others, Tishby proposed to track the dynamics of a trained DNN in the IP and discussed its optimal performance by the IB theory. It was claimed that the observed features in his studies are general in DNNs. However, objections appeared after Tishby's eye-catching findings [17]. Some people had reported that they could not replicate Tishby's results in the same or other DNNs [13] [14], which raised a lot of discussions and debates. At the moment, this problem is not yet settled down. To understand the sensitivity, robustness, and generality of Tishby's findings, it is thus instructive to examine different tasks, different DNNs, and even different measures to analyze the systems. This is what this thesis is devoted to.

To this end, we studied two kinds of tasks:

- (P1) the number classification task in eq. 3.2 and eq. 3.3
- (P2) the MNIST hand-writing recognition.

The DNNs used to solve these tasks include

- (N1) Input-9-9-8-8-7-7-Output (to tackle (P1))
- (N2) Input-16-16-16-Output (to tackle (P2))

  * check appendix for more detail.

The measures employed in the current and previous works to analyze DNNs during training are collected in Table 5.1. Different measures provided different perspectives to understand the dynamics of DNN training process.

Current methods	Previous methods
MI directly calculated or by Monte Carlo	MI calculated from estimator
Loss	Gradient of loss
Fluctuation of loss	Standard deviations of the gradient of loss
Extended space from IP	IP

Table 5.1: A comparison between the current and previous measures.

According to these studies, we do see a consistency between a trained DNN and the IB theory. Although the studied DNNs have not reached the optimal tradeoff of compression and accuracy, they are indeed constrained by the IB bound, as predicted. Regarding the training process, we do see the transition from the ERM phase to the compression phase, as claimed in Ref. [2]. However, this trend occurs only in small DNNs in our studies, such as those for solving the simple classification task in eq. 3.1. It is absent in DNNs for solving larger tasks, such as the digit recognition in MNIST. Furthermore, our analysis shows that the observed transition features depend on several factors, such as the measure used and the data collected. Thus, the claimed general trend may exist in certain classes of DNNs or general DNNs under certain data collection and preparation procedures. There is still a lot to do to understand whether and how much a DNN compresses information during training. The result collected in this thesis added more information for this community and sheds light on this contested issue.

5.1 Future Work

According to the above conclusion, applying MI analysis to more complex DNNs is still very challenging. Phase transitions are not yet observed in all DNNs. Whether this is a limitation of the theory itself or a limitation of the methods to inspect DNNs is still unclear. This leaves much room for improvement, both in theoretical models and the numerical techniques. For example, in the coarse-graining problem, the bin number of activation patterns and the down-scaling of MNIST image are all data representing problem which can be further improved. There might also exist other advanced tools and MI estimators for calculating MIs. All these might make the whole study more systematic and the result more conclusive.

References

[1] N. Tishby and N. Zaslavsky, "Deep Learning and the Information Bottleneck Principle," *"IEEE Information Theory Workshop (ITW)"*, pp. 1–5, 2015.

[2] R. Shwartz-Ziv and N. Tishby, "Opening The Black Box of Deep Neural Networks via Information," *arXiv:1703.00810*, 2017.

[3] P. Mehta and D. J. Schwab, "An exact mapping between the variational renormalization group and deep learning," *arXiv:1410.3801*, 2014.

[4] H. W. Lin, M. Tegmark, and D. Rolnick, "Why Does Deep and Cheap Learning Work So Well?" *Journal of Statistical Physics*, vol. 168, pp. 1223–1247, 2017.

[5] A. Paul and S. Venkatasubramanian, "Why does Deep Learning work? - A perspective from Group Theory," *arXiv:1412.6621*, 2014.

[6] P. P. Brahma, D. Wu, and Y. She, "Why Deep Learning Works: A Manifold Disentanglement Perspective," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, 2016.

[7] Y. LeCun, "Yann LeCun - How Does The Brain Learn So Quickly?" https://www.youtube.com/watch?v=WUZhLzaD3b8, [Online; accessed 20-July-2018].

[8] Y. LeCun, "Yann LeCun - Power & Limits of Deep Learning," https://www.youtube.com/watch?v=0tEhw5t6rhc, [Online; accessed 20-July-2018].

[9] "Cognitive Computational Neuroscience 2017," http://ccneuro.org/2017/.

[10] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing*, p. arXiv preprint physics/0004057, 1999.

[11] S. E. Palmer, O. Marre, M. J. B. II, and W. Bialek, "Predictive information in a sensory population," *Proceedings of the National Academy of Sciences of the United State of America*, vol. 122, pp. 6908–6913, 2015.

[12] R. Shwartz-Ziv, "Idnns," https://github.com/ravidziv/IDNNs, [Online; accessed 20-July-2018].

[13] Aaron Schumache, "Failure to replicate Schwartz-Ziv and Tishby," https://planspace.org/20180213-failure_to_replicate_schwartz-ziv_and_tishby/, 2018, [Online; accessed 20-July-2018].

[14] Emin Orhan, "No, information bottleneck (probably) doesn't open the "black-box" of deep neural networks," https://severelytheoretical.wordpress.com/2017/09/28/no-information-bottleneck-probably-doesnt-open-the-black-box-of-deep-neural-networks/, 2017, [Online; accessed 20-July-2018].

[15] "On the Information Bottleneck Theory of Deep Learning (Disagrees with Tishby)," https://www.reddit.com/r/MachineLearning/comments/79efus/r_on_the_information_bottleneck_theory_of_deep/, 2017, [Online; accessed 20-July-2018].

[16] "Information Theory of Deep Learning. Naftali Tishby," https://www.youtube.com/watch?v=bLqJHjXihK8, 2017, [Online; accessed 20-July-2018].

[17] ICLR 2018 Conference Program Chairs, "On the Information Bottleneck Theory of Deep Learning," https://openreview.net/forum?id=ry_WPG-A-, 2018, [Online; accessed 20-July-2018].

[18] R. A. Waston and E. Szathmáry, "How Can Evolution Learn?" *Cell*, vol. 31, pp. 147–157, 2016.

[19] L. Bottou, "Online Learning and Stochastic Approximations," *On-line learning in neural networks*, pp. 9–42, 1998.

[20] L. Bottou, *Neural Networks: Tricks of the Trade*. Springer, 2012.

[21] C. Baldassi, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, "Subdominant Dense Clusters Allow for Simple Learning and High Computational," *Physical Review Letters*, vol. 115, 2015.

[22] C. Baldassi, C. Borgs, J. Chayes, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, "Unreasonable Effectiveness of Learning Neural Networks: From Accessible States and Robust Ensembles to Basic Algorithmic Schemes," *PNAS*, vol. 113, 2016.

[23] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp Minima Can Generalize For Deep Nets," *arXiv:1703.04933*, 2017.

[24] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 2012.

[25] A. Kolchinsky, B. D. Tracey, and D. H. Wolpert, "Nonlinear Information Bottleneck," *arXiv:1705.02436*, 2017.

[26] D. Strouse and D. J. Schwab, "The deterministic information bottleneck," *arXiv: 1604.00268v2*, 2016.

[27] S. Arimoto, "An algorithm for computing the capacity of arbitrary discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 18, 1972.

[28] R. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE Transactions on Information Theory*, vol. 18, 1972.

[29] A. Kolchinsky and B. D. Tracey, "Estimating Mixture Entropy with Pairwise Distances," *Entropy MDPI*, vol. 19, 2017.

[30] A. Kolchinsky, "Code for on the information bottleneck theory of deep learning," https://github.com/artemyk/ibsgd, [Online; accessed 20-July-2018].

[31] Łukasz Kaiser and I. Sutskever, "Neural GPUs Learn Algorithms," *International Conference on Learning Representations*, 2016.

[32] P. Hall and S. C. Morton, “On the estimation of entropy,” *Annals of the Institute of Statistical Mathematics*, vol. 45, pp. 69–88, 1993.

[33] A. Kraskov, d Stögbauer, and P. Grassberger, “Estimating mutual information,” *Physical Review E*, vol. 69, 2004.

Appendix A

A.1 Abbreviations and Glossary table

MI	Mutual Information
IB	Information Bottleneck
IP	Information Plane
SGD	Stochastic Gradient Descent
ERM	Empirical Error Minimalization
label	desired output data of certain input data
epoch	the process of training a DNN one time with a complete training dataset
session	the whole training process of a DNN
activation	the output of a node (artificial neuron)
loss	the difference between the label and the output

Table A.1: Common abbreviations and glossary in this thesis.

A.2 DNN model parameters

The parameters in our numerical studies are collected in Table A.2 and A.3.

DNN Model for eq.3.1	
Layers	Input-9-Dropout(0.1)-9-8-8-7-7-Output
Total Epochs	800
Activation function	tanh
Optimizer	SGD
Batchsize	8

Table A.2: Parameters used in keras/tensorflow for compiling DNN model for eq.3.1

DNN Model for MNIST	
Layers	Input-16-Dropout(0.2)-16-Dropout(0.2) -16-Output
Total Epochs	156
Activation function	tanh
Optimizer	SGD
Batchsize	256

Table A.3: Parameters used in keras/tensorflow for compiling DNN model. (MNIST)

A.3 Pseudo-codes for machine learning and data saving

The following python pseudo-code is used to train a DNN for the classification task for eq.3.1. It calculates and saves data, such as the activations for one session. This code needs to be executed several times to obtain the data of multiple sessions for ensemble average.

Because the numbers of different labels in the dataset are unbalanced, in that they are not equal. The dataset is split into training dataset and testing dataset first. Then the training dataset is randomly resampled to balance out the labels. By contrast, the testing dataset are unbalanced.

```
# using tensorflow backend to evaluate values of tensors
import keras.backend as K

# Function for getting activations with respect a certain dataset
def save_activations(model, act_test)
    for layer in model.layers
        output = layer.output

        # Keras functions for evaluating value of tensor
        # Note that "output" and "model.input" are placeholder tensors only, with no values
        # Adding tensor "K.learning_phase" to input tensor for setting the learning phase
        func = K.function(model.input + [K.learning_phase()], output)

        # Evaluate Keras function for values of output tensors (activations)
        # Appending 0 in inputs for setting learning phase to 0 (testing phase)
        activation = func([ act_test, 0 ])
        save activation to file

# Main program for training and recording data for analysis
# In this thesis, an additional dataset is needed for analysising a DNN.
# This dataset is however not requied for DNN training.
```

```
load training dataset (x_train, y_trian) from file
load testing dataset (x_train, y_trian) from file
load activation-testing dataset from file

convert all dataset to binary class matrices by Keras' "to_categorical()"

# build a DNN model layer by layer and then compile it
model = Sequential()
model.add( Dense( nodes, activation='tanh' ...))
....
model.compile(loss='categorical_crossentropy', optimizer=SGD(), metrics=['accuracy'])

# Training and recording data
while not converged
    # Train for short period of time (only a few epochs)
    history = model.fit(x_train, y_train, .....)
    save trainig history for all epochs

    # save activation for activation-testing dataset after a few epoch
    save_activation( model, activation_testing )
```