FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jakub Černý

Applications of information theory to the study of deep learning

Department of Applied Mathematics

Supervisor of the bachelor thesis: Mgr. David Sychrovský
Study programme: Mathematics for Information Technologies

Prague 2025

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date ....

Author's signature

I dedicate this to my family and friends who helped me along the way.

Title: Applications of information theory to the study of deep learning

Author: Jakub Černý

Department: Department of Applied Mathematics

Supervisor: Mgr. David Sychrovský, Department of Applied Mathematics

Abstract: Despite deep learning's remarkable empirical success, its theoretical underpinnings lag behind. Information theory provides a powerful framework for analyzing internal network representations, particularly through recent advances in information bottleneck (IB) theory and the information plane. This thesis investigates how the structure of the information plane, specifically the clustering behavior of internal representations, influences neural network performance. We introduce Purity theory, a novel framework for quantifying layer-wise clustering, complementing established IB perspectives. Our analysis reveals a significant correlation between information plane structure and generalization performance in binary classification tasks. Leveraging this correlation, we propose a new information-theoretic metric that effectively predicts model generalization capability. Furthermore, we develop a model selection algorithm based on this metric, which demonstrably outperforms selection based solely on training loss.

Keywords: Information Bottleneck Theory, Deep Learning, Machine Learning, Neural Networks

Název práce: Aplikace teorie informace na studium učení hlubokých neuronových sítí

Autor: Jakub Černý

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. David Sychrovský, Katedra aplikované matematiky

Abstrakt: Přes pozoruhodný empirický úspěch hlubokého učení (deep learning) jeho teoretické základy zaostávají. Teorie informace poskytuje účinný rámec pro analýzu vnitřních reprezentací v sítích, zejména díky nedávnému pokroku v teorii informačního hrdla (information bottleneck, IB) a konceptu informační roviny. Tato práce zkoumá, jak struktura informační roviny, konkrétně shlukování vnitřních reprezentací, ovlivňuje výkon neuronových sítí. Představujeme Purity teorii, nový rámec pro kvantifikaci shlukování reprezentací po vrstvách, který doplňuje stávající IB perspektivy. Naše analýza odhaluje významnou korelaci mezi strukturou informační roviny a schopností generalizace v úlohách binární klasifikace. Na základě této korelace navrhujeme novou metriku založenou na teorii informace, která účinně předpovídá schopnost modelu generalizovat. Dále vyvíjíme algoritmus pro výběr modelu využívající tuto metriku, který prokazatelně překonává výběr založený výhradně na trénovací ztrátě (train loss).

Klíčová slova: Hluboké učení, Teorie informačního hrdla, Strojové učení, Neuronové Sítě

Contents

1 Introduction

1.1 Motivation

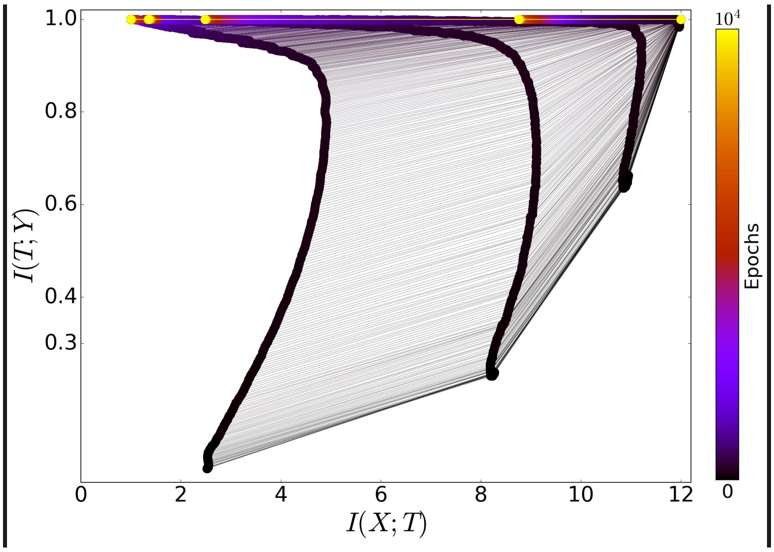
Neural networks are a growing part of the machine learning landscape. As a huge part of large language models, computer vision, and even reinforcement learning, it is immensely important to study their inner workings. This is a challenging numerical problem that becomes increasingly difficult the deeper one looks, so finding the right tools for the job is crucial. Information theory is one of those novel approaches that attempts to tackle the problem of looking behind the curtains of neural networks.

1.2 Information plane of neural networks

When one considers information theory in the context of neural networks, one of the first available tools is mutual information. This concept was first used in the influential paper Deep learning and the information bottleneck principle [1]. The authors consider a layer as the transformed distribution of the input and calculate the mutual information of these different distributions. This then yields interesting results about not only the network, the training process of the network, but also its performance. In this thesis, we will mainly focus on the latter.

The concept of an information plane originates from the notion that we can quantify the amount of information that a layer shares with both the input and output. This can be done for each layer; thus, in theory, mapping the transformation of information from input to output. Intuitively, we can then see how redundant information about the input is lost as we get closer to the output. A natural question then arises: is this transformation relevant to the performance of a neural network? If we try to draw a comparison between the brain and neural networks, we might get an intuition of how the information is transformed. A more gradual transformation of the input may intuitively correlate with better performance, while a rapid loss of information about the input may be detrimental. There may also be less obvious quirks in the mutual-information structure. This is the main point of discussion in this thesis and will be tested in Chapter 6.

Furthermore, we can utilize this method for information mapping for each neural network instance during training. This will produce the information plane map in Figure 1.1 that attempts to study the training process of a network.



.

Figure 1.1 Information plane map as introduced in Opening the Black Box of Deep Neural Networks via Information [2]

In Figure 1.1 $I(T;Y)$ is the mutual information of a layer T with *output* and $I(T;X)$ the mutual information with *input*. Each layer is represented as a point described not only by $I(T;X)$ and $I(T;Y)$, but also by the training epoch for which the mutual information has been computed.

This then showcases not only the final mutual information structure of the trained network, but also the training process and the so-called drift diffusion phase. This is described as the shift to the right when initially gaining mutual information with the output. Intuitively, this means that during the training process, the latter layers first need to gain mutual information with the input before the training process selects which parts of the input are important.

As addressed in Section 3.5, it also holds that two consecutive layers T_1 and T_2 satisfy the relationship $I(X,T_i) \geq I(X,T_{i+1})$. This corresponds to the intuition that mutual information with the input should be greater in layers closer to the input. The question then still stands: do the aforementioned mutual information values and information of layers somehow correlate with the performance of the model? That is going to be the main focus of this thesis. We will first examine the behavior and estimates of mutual information from a theoretical point of view in Chapters 3 and 4 and then examine the practical issues in Chapter 5. Then we can finally test whether there is any correlation between the mutual information structure and the performance of a network. But first, we need to introduce the basic terminology.

2 Definitions and Preliminaries

Notation. Vectors appear in bold (e.g., $\boldsymbol{v}$); sets in calligraphic (e.g., $\mathcal{A}$). Logarithms are natural, unless otherwise stated. The symbol $\triangleq$ denotes the definitional equality.

2.1 Machine learning vocabulary

Before we can start describing mutual information and plug-in-estimator in neural networks in the context of mutual information, we first need to define what a neural network is and what properties we require from it. We also define some basic notation and concepts that are going to be used throughout the next chapters.

We are going to define neural networks from the ground up, starting with a neuron.

Definition 2.1 (Neuron)**.** *A* neuron *is a parametric function $\varphi : \mathbb{R}^d \to \mathbb{R}$ defined by*

$$\varphi(\boldsymbol{z}; \boldsymbol{w}, b) \triangleq g\left(\langle \boldsymbol{w}, \boldsymbol{z} \rangle + b\right) = g\left(\sum_{i=1}^{d} w_i z_i + b\right)$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is called weights*, $b \in \mathbb{R}$ is a* bias*, and $g : \mathbb{R} \to \mathbb{R}$ is a fixed* activation function *(Definition 2.2).*

To ensure that the neural network isn't only a linear regression model, we need to introduce so-called activation functions, which provide the neural network with differentiable nonlinearities.

Definition 2.2 (Activation Function)**.** *We call a function $g : \mathbb{R} \to \mathbb{R}$ an* activation function *if it is a function that is differentiable almost everywhere. For $\boldsymbol{v} \in \mathbb{R}^k$, $g(\boldsymbol{v})$ denotes element-wise application: $[g(\boldsymbol{v})]_i = g(v_i)$.*

Remark 2.1. *In the definition, we do not require the activation function to be non-linear, as in some cases, the output of a regression network does not strictly need to be nonlinear. Common examples of activation functions include:*

$$\begin{aligned}
\mathrm{ReLU}(x) &\triangleq \max\{0, x\} \\
\sigma(x) &\triangleq (1 + e^{-x})^{-1} \quad (\textit{sigmoid}) \\
\tanh(x) &\triangleq \frac{e^x - e^{-x}}{e^x + e^{-x}}
\end{aligned}$$

Now we can define what we mean by a layer in neural networks. These are the basic building blocks of the deep neural network. In this work, we will be interested in measuring mutual information between the layers and data on which the network is trained.

Definition 2.3 (Neural Network Layer)**.** *A* layer *with n neurons is a function $f : \mathbb{R}^d \to \mathbb{R}^n$ defined by*

$$f(\boldsymbol{z}) \triangleq g\left(\mathbf{W}\boldsymbol{z} + \boldsymbol{b}\right)$$

where $\mathbf{W} \in \mathbb{R}^{n \times d}$ is a weight matrix, $\boldsymbol{b} \in \mathbb{R}^n$ is a bias vector and g is an activation function.

This naturally leads to the definition of a feed-forward neural network.

Definition 2.4 (Feed-Forward Neural Network)**.** *A feed-forward neural network with architecture* $\mathcal{A} = (d_0, d_1, \dots, d_L)$ *is a composition*

$$\mathrm{NN} \triangleq f_L \circ f_{L-1} \circ \cdots \circ f_1$$

where each $f_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$ *is a layer (Definition 2.3). The parameters* $\theta = \{\mathbf{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}_{\ell=1}^{L}$ *are learned from the data. We require all activations to ensure that* NN *is differentiable almost everywhere. We call* d_0 *the* input size *and* d_L *the* output size*.*

Remark 2.2. *The type of neural network defined in Definition 2.4 is called feed-forward. Although there are several other types, we do not consider them in this thesis. From now on, neural networks will refer to feed-forward neural networks only.*

Now we will define the function that is used for classification in neural networks. It is an extension of the sigmoid to more than just a binary classification.

Definition 2.5 (Softmax)**.** *Let* $\Delta^{k-1} \triangleq \{\boldsymbol{p} \in \mathbb{R}^k_{\geq 0} : \sum_i p_i = 1\}$*. The function* $\mathrm{softmax} : \mathbb{R}^k \to \Delta^{k-1}$ *is mapped to the probability simplex:*

$$[\mathrm{softmax}(\boldsymbol{s})]_i \triangleq \frac{e^{s_i}}{\sum_{j=1}^{k} e^{s_j}}, \quad i = 1, \dots, k$$

Remark 2.3. *This is typically used in the output layer for multiclass classification as it provides a distribution over output variables.*

Proof of distribution properties:

1. *Non-negativity:* Since $e^{s_i} > 0$ for all $s_i \in \mathbb{R}$ and the denominator $\sum_{j=1}^{k} e^{s_j} > 0$, it follows that $[\mathrm{softmax}(\boldsymbol{s})]_i > 0$ for all i.

2. *Unit sum:* Direct computation shows $\sum_{i=1}^{k}[\mathrm{softmax}(\boldsymbol{s})]_i = \frac{\sum_{i=1}^{k} e^{s_i}}{\sum_{j=1}^{k} e^{s_j}} = 1$.

Thus, $\mathrm{softmax}(\boldsymbol{s})$ is a valid discrete probability distribution. Now we just need to define some basic terms used in machine learning.

Remark 2.4. *To train a neural network, we utilize the concept of gradient descent. We define a loss function, which is a function that captures the error of the current neural network, and by differentiating it and finding the local minima, we "improve" the predictions in the next training step. For this, an optimizer is defined, which is an algorithm that determines the size of the step that we should take to minimize the loss of the neural network. We do not discuss this on a rigorous level, as it is not relevant to the method we are going to explore.*

Definition 2.6 (Dataset)**.** *A* dataset *is a finite multiset* $\mathcal{D} \triangleq \{(\boldsymbol{x}_j, y_j)\}_{j=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$ *where* $\mathcal{X} \subseteq \mathbb{R}^{d_0}$ *is the input space and* $\mathcal{Y}$ *the label space.*

Definition 2.7 (Training and Test Datasets)**.** *Let $\mathcal{D}$ be a dataset as in Definition 2.6, with $\mathcal{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^N$. A* training dataset *(or* training set*) is a subset*

$$\mathcal{D}_{\text{train}} \;=\; \{(\mathbf{x}_i, y_i)\}_{i \in I_{\text{train}}} \subseteq \mathcal{D},$$

used to fit the parameters of a learning algorithm. A test dataset *(or* test set*) is a subset*

$$\mathcal{D}_{\text{test}} \;=\; \{(\mathbf{x}_i, y_i)\}_{i \in I_{\text{test}}} \subseteq \mathcal{D},$$

reserved exclusively for evaluating the generalization performance of the trained model.

Typically, the index sets satisfy $I_{\text{train}} \cap I_{\text{test}} = \varnothing$ and $I_{\text{train}} \cup I_{\text{test}} = \{1, \ldots, N\}$, so that

$$\mathcal{D}_{\text{train}} \;\dot\cup\; \mathcal{D}_{\text{test}} \;=\; \mathcal{D}, \quad \textit{with} \quad N_{\text{train}} + N_{\text{test}} = N.$$

The split is assumed to be independent and identically distributed (i.i.d.) with respect to the underlying data-generating distribution.

Remark 2.5. *We define $\mathcal{X}$ as a finite subset of a possibly infinite input space, as it is possible to have an infinite generator of such a space. But even in practice, we have to use a finite subset of this space, thus the reason for the definition.*

Definition 2.8 (Internal Representation)**.** *For input $\boldsymbol{x} \in \mathcal{X}$ and layer $\ell \in \{1, \ldots, L-1\}$, the* internal representation *at layer ℓ is*

$$\boldsymbol{h}^{(\ell)}(\boldsymbol{x}) \;\triangleq\; (f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1)(\boldsymbol{x}) \in \mathbb{R}^{d_\ell}.$$

For dataset $\mathcal{D}$, the ℓ-th layer representations are $\mathcal{H}^{(\ell)} \triangleq \{\boldsymbol{h}^{(\ell)}(\boldsymbol{x}_j)\}_{j=1}^N$.

2.2 Information-theoretic basics

Now that we have defined the basic terminology in machine learning and neural networks, we need to determine the tools that we will use throughout the thesis, that is, the tool of mutual information. For that, we need to define the information content and entropy.

Definition 2.9 (Information Content)**.** *For a discrete random variable Z with a probability mass function p_Z, the* information content *of the result z is*

$$\mathcal{I}_Z(z) \;\triangleq\; -\log p_Z(z).$$

Definition 2.10 (Entropy)**.** *The* Shannon entropy *of discrete random variable Z is*

$$H(Z) \;\triangleq\; \mathbb{E}[\mathcal{I}_Z(Z)] = -\sum_{z \in \mathcal{Z}} p_Z(z) \log p_Z(z).$$

Definition 2.11 (Joint Entropy)**.** *For discrete random variables X, Y with joint distribution $p_{X,Y}$,*

$$H(X,Y) \;\triangleq\; -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{X,Y}(x,y) \log p_{X,Y}(x,y).$$

a

Definition 2.12 (Mutual Information)**.** *For discrete random variables X and Y,*

$$I(X;Y) \triangleq H(X) + H(Y) - H(X,Y) = \sum_{x,y\in\mathcal{X},\mathcal{Y}} p_{X,Y}(x,y)\log\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}.$$

Definition 2.13 (Plug-in Estimator)**.** *Given i.i.d. samples $\{z_1,\ldots,z_N\}$ from discrete random variable Z, the plug-in entropy estimator is*

$$\widehat{H}(Z) \triangleq -\sum_{z\in\mathcal{Z}} \hat{p}_Z(z)\log\hat{p}_Z(z), \quad \hat{p}_Z(z) \triangleq \frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\{z\}}(z_j)$$

where $\mathbb{1}_A$ is the indicator function. For paired samples $\{(x_1,y_1),\ldots,(x_N,y_N)\}$ from (X,Y), the plug-in mutual information estimator is

$$\widehat{I}(X;Y) \triangleq \widehat{H}(X) + \widehat{H}(Y) - \widehat{H}(X,Y)$$

with $\widehat{H}(X,Y)$ computed via the empirical joint distribution

$$\hat{p}_{X,Y}(x,y) \triangleq \frac{1}{N}\sum_{j=1}^{N}\mathbb{1}_{\{(x,y)\}}((x_j,y_j)).$$

Theorem 2.6 (Consistency of Plug-in Estimators)**.** *For discrete random variables with finite support:*

1. *$\widehat{H}(Z) \xrightarrow{a.s.} H(Z)$ as $N\to\infty$ (entropy estimator)*
2. *$\widehat{I}(X;Y) \xrightarrow{a.s.} I(X;Y)$ as $N\to\infty$ (mutual information estimator)*

Proof. The result follows from:

1. *Glivenko-Cantelli theorem [3]*: Empirical distributions converge uniformly to true distributions.
2. *Continuous mapping theorem*: Entropy and mutual information are continuous functions of the probability mass function
3. *Strong law of large numbers*: Empirical averages converge almost surely

Specifically for mutual information:

$$\begin{aligned}\widehat{I}(X;Y) &= \widehat{H}(X) + \widehat{H}(Y) - \widehat{H}(X,Y)\\ &\xrightarrow{a.s.} H(X) + H(Y) - H(X,Y) = I(X;Y)\end{aligned}$$

Since each entropy estimator converges to its true value. □

Remark 2.7. *While consistent, plug-in estimators exhibit a* finite sample downward bias*. For entropy, one obtains the Miller–Madow correction*

$$\mathbb{E}[\widehat{H}(Z)] = H(Z) - \frac{|\mathcal{Z}|-1}{2N} + \mathcal{O}\big(N^{-2}\big) \quad [4],$$

so $\mathbb{E}[\widehat{H}(Z)] \le H(Z)$. For mutual information, a similar expansion gives

$$\mathbb{E}[\widehat{I}(X;Y)] = I(X;Y) - \frac{\big(|\mathcal{X}|-1\big)\big(|\mathcal{Y}|-1\big)}{2N} + \mathcal{O}\big(N^{-2}\big) \quad [5].$$

Intuitively, the joint space $\mathcal{X}\times\mathcal{Y}$ is larger than either marginal, so the negative bias in $\widehat{H}(X,Y)$ outweighs the two marginal biases, making $\widehat{I}(X;Y)$ underestimate $I(X;Y)$.

Now we can define a term used later in Chapter 6. This describes the flow of the information through one specific neural network and could be understood as one instance of a neural network plotted in Figure 1.1.

Definition 2.14 (Mutual Information (MI) Structure). *Let $\mathcal{N}$ be a feed-forward neural network with input random variable X, output random variable Y, and hidden-layer random variables $T_1, \ldots, T_{L-1}$. The* Mutual Information structure *of $\mathcal{N}$ is the ordered list*

$$\mathcal{I}(\mathcal{N}) \;=\; \Big\{ \Big(I(X;T_i),\; I(T_i;Y) \Big) \Big\}_{i=1}^{L-1}.$$

2.3 Discretization

If we want to calculate mutual information via the plug-in estimator, we will need the probabilities of each internal representation of a hidden layer. That is why we need a binning function that maps the vector of real values onto a vector of finite space, where we can easily calculate its probability.

Definition 2.15 (Binning Function). *A* binning function $T : \mathbb{R}^d \to \mathcal{M}$ *assigns every vector $\boldsymbol{v}$ a* bin label $m \in \mathcal{M}$ *such that each vector receives exactly one label:*

$$T(\boldsymbol{v}) = m.$$

Applying T to the layer-ℓ representations $\mathcal{H}^{(\ell)}$ turns every continuous vector $\boldsymbol{h}^{(\ell)}(\boldsymbol{x})$ into a symbol $T(\boldsymbol{h}^{(\ell)}(\boldsymbol{x})) \in \mathcal{M}$, so we can plug these counts into the entropy and mutual-information estimators from Definition 2.13.

Remark 2.8. *The choice of a binning function is a crucial step in calculating the mutual information. These functions will eventually create clusters of binned vectors, which we will study in Chapter 4. As the sizes of a layer can range from a few to thousands of neurons, we will need to carefully choose a proper binning function for each layer separately.*

3 Classical Theorems on Mutual Information Estimation

3.1 Introduction

Now that we have defined the basic terminology, we investigate the theoretical guarantees that *plug-in* (maximum likelihood) estimators of mutual information (MI) have. A plug-in estimator depends on two quantities that we typically control:

- The size of the support N, or in other words, the number of observations used to calculate the mutual information. This value is typically bound by the size of a dataset, but does not have to be as addressed in Remark 2.5;
- the *effective alphabet size* K (number of discrete states created by the binning function).

Because we choose both the binning rule and the size of the dataset, rigorous guidance on how N should scale with K is indispensable; this will be discussed in Section 3.3. We also used more practical guidance in the form of calculating the unseen mass when binning as described in Section 4.2.
Although we can typically control N and K, it is essential to consider that manipulating them can lead to severe drawbacks, such as impure clustering in Definition 4.2 and computational complexity that can grow uncontrollably, as shown in Section 5.4.

This chapter is structured as follows.

§3.2 Classical results on bias, variance, consistency, and minimax rates.

§3.3 Consistency of discretization *sieve* (data-driven).

§3.4 Guarantees of the existence of the best upper bound estimator.

§3.5 Neural networks viewed as Markov chains and implications of data processing inequality.

§3.6 Deterministic networks, clustering interpretation, and regularized noise estimators.

3.2 Classical guarantees of the plug-in estimator

Throughout this section, X and Y are discrete random variables with finite supports of sizes K_X and K_Y, and we denote $K := K_X K_Y$. As addressed earlier, we can nudge these numbers in the desired direction, but it is hard to precisely control these numbers.

3.2.1 Bias and variance

Theorem 3.1 (Leading bias: Miller–Madow–Paninski [4]). *For fixed K_X, K_Y and sufficiently large N,*

$$\mathbb{E}[\hat{\mathrm{I}}_N] - \mathrm{I} = \frac{K_X K_Y - K_X - K_Y + 1}{2N \ln 2} \; + \; O\Big(\frac{1}{N^2}\Big).$$

Proof Idea. Taylor-expand the empirical entropy around the true probability mass function and compute multinomial moments [4]. □

In the paper Estimation of entropy and Mutual Information [4], it is distinguished between true support sizes and observed support sizes; here we understand it as the observed support size. This result may seem arbitrary, as even small neural networks face the curse of dimensionality. But it proves to be rather useful in practice. For example, in classification tasks, we can understand the binning of the output layer as a simple function to a space of labels of size D_ℓ. The usefulness of this theorem is then instantaneously apparent, as calculating the mutual information of any layer with the output layer is strictly bounded by the size of the output space. For binary tasks, we then immediately get:

Theorem 3.2 (Binary tasks and MI with output). *Consider a neural network NN with output size $d_\ell = 2$ trained on a binary problem, the mutual information of the hidden layer T_i and output layer Y is:*

$$\mathbb{E}[\hat{\mathrm{I}}(T_i, Y)_N] - \mathrm{I}(T_i, Y) = \frac{K_{T_i} - 1}{2N \ln 2} \; + \; O\Big(\frac{1}{N^2}\Big),$$

where K_{T_i} is the number of observed bins in layer K_{T_i} via a chosen binning function.

Proof. This is a direct application of Theorem 3.1, but only for a binary case. □

Remark 3.3. *As most of our hypotheses will be tested on binary tasks, this result is very important. The guarantees for mutual information with input are worse, but if binning functions are chosen accordingly, they can reach similar levels,*

Theorem 3.4 (Asymptotic normality [4]). *Let $K = o(\sqrt{N})$. Then*

$$\sqrt{N}\Big(\hat{\mathrm{I}}_N - \mathrm{I}\Big) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma_I^2),$$

where σ_I^2 is an explicit variance depending on $p(x, y)$. Hence $\mathbb{E}[(\hat{\mathrm{I}}_N - \mathrm{I})^2] = O(1/N)$.

Proof Idea. Apply the delta method to the entropy functional. The asymptotic normality of the empirical distribution and the functional derivative of entropy yield the result from Estimation of entropy and Mutual Information as in Estimation of entropy and Mutual Information [4]. □

Remark 3.5 (Practical implications of Theorem 3.4). *The central-limit theorem shows that* whenever the number of occupied bins satisfies $K = o(\sqrt{N})$, *the plug-in MI estimator is asymptotically normal with variance σ_I^2/N. This implies:*

a) ***Confidence intervals.*** *Report $\hat{I}_N \pm z_{\alpha/2}\sigma_I/\sqrt{N}$, where $z_{\alpha/2}$ is the Gaussian quantile, avoiding costly bootstrap resampling.*

b) ***Hypothesis tests.*** *Standard z-tests for differences in mutual information across layers or epochs are justified when $K = o(\sqrt{N})$.*

c) ***Design guidance.*** *If K grows too quickly (e.g., fine binning in wide layers), normality breaks down. Increase N or reduce K (via coarser bins or adaptive merging) before using CLT.*

3.2.2 Impossibility in the large-alphabet regime

Definition 3.1 (Consistent estimator)**.** *An estimator $\hat{\theta}_N$ is consistent for θ if $\hat{\theta}_N \xrightarrow{P} \theta$ (convergence in probability) as $N \to \infty$.*

Theorem 3.6 (Paninski impossibility bound [4])**.** *Suppose $K \to \infty$ and $N = O(K^{1-\alpha})$ for some $0 < \alpha < 1$. Then* no *estimator, plug-in or otherwise, can be consistent for entropy (and hence for mutual information).*

3.2.3 Sharp sample complexity

Theorem 3.7 (Fundamental limits of entropy estimation [6])**.** *For entropy estimation over K-supported distributions, there exist constants $c, C > 0$ such that when $N \geq cK/\log K$:*

1. ***(Lower bound)*** *For any estimator $\hat{H}$, there exists a distribution p with $|\operatorname{supp}(p)| = K$ satisfying:*

$$\mathbb{E}[(\hat{H} - \mathrm{H}(p))^2] \geq C \left[\left(\frac{K}{N \log K} \right)^2 + \frac{(\log K)^2}{N} \right]$$

2. ***(Upper bound)*** *There exists an estimator $\hat{H}^*$ such that for all distributions p with $|\operatorname{supp}(p)| = K$:*

$$\mathbb{E}[(\hat{H}^* - \mathrm{H}(p))^2] \leq \frac{1}{C} \left[\left(\frac{K}{N \log K} \right)^2 + \frac{(\log K)^2}{N} \right]$$

Remark 3.8. *Theorem 3.6 exhibits a phase transition: Estimation is impossible when $N \ll K/\log K$, but feasible above this threshold (Theorem 3.7). This is an important realization when calculating MI via plug-in estimator, as it showcases its limitations.*

3.3 Sieve consistency for data-driven discretization

While binning a hidden layer in a neural network, different binning functions are tested and refined. We take inspiration from the concept of *sieves*.

Definition 3.2 (Sieve of partitions[4])**.** *A sequence of discretizations $\{\mathcal{P}_N\}_{N \geq 1}$, where each $\mathcal{P}_N$ is a finite partition of $\mathbb{R}^d$, is a* sieve *if:*

1. *$\mathcal{P}_N$* refines *$\mathcal{P}_{N-1}$ ($\forall N \geq 2$), meaning every cell in $\mathcal{P}_N$ is contained in some cell of $\mathcal{P}_{N-1}$; and*

2. The maximal cell diameter $\|\mathcal{P}_N\|_\infty \to 0$ as $N \to \infty$.

Theorem 3.9 (Almost-sure consistency under sieves)**.** *[4] Let X, Y be continuous random variables with a bounded density. Apply the plug-in MI estimator using a sieve $\{\mathcal{P}_N\}$ and assume:*

1. *The number of occupied cells satisfies $K_N = o(\sqrt{N})$ a.s.; and*
2. *$\|\mathcal{P}_N\|_\infty \to 0$.*

Then $\hat{\mathrm{I}}_N(X;Y) \xrightarrow{a.s.} I(X;Y)$.

Proof Idea. Binning bias vanishes by Riemann-sum approximation as $\|\mathcal{P}_N\|_\infty \to 0$. The variance vanishes because $K_N = o(\sqrt{N})$ (Theorem 3.4). Apply the Borel-Cantelli lemma [7]. □

Remark 3.10 (Exploiting sieve consistency)**.** *The approach of adaptive binning used in this thesis is inspired by the construction of sieve consistency. The algorithm is described in Section 5.1.*

3.4 Distribution-free high-confidence guarantees

Theorem 3.11 (Best upper bound (BUB) estimator [4])**.** *There exists an entropy estimator $\hat{H}_{BUB}$ such that for* all *probability mass functions:*

$$\left|\mathbb{E}[\hat{\mathrm{H}}_{BUB}] - \mathrm{H}\right| \;\le\; C\sqrt{\tfrac{\log N}{N}},$$

where C is universal. The induced MI estimator has the same uniform bound.

Remark 3.12. *We cannot guarantee results of* BUB, *but it is good to know that there exists such an estimator when constructing the plug-in estimator and corresponding binning function.*

3.5 Neural networks as Markov chains

Consider a feed-forward network with layers $(T_0 := X, T_1, \ldots, T_L := Y)$. The mapping is deterministic, so:

$$X \longrightarrow T_1 \longrightarrow T_2 \longrightarrow \cdots \longrightarrow T_L$$

forms a **Markov chain**. By the data-processing inequality as described in Elements of Information Theory ([8]), we have:

$$I(X;T_1) \ge I(X;T_2) \ge \cdots \ge I(X;Y).$$

Equality holds if and only if each map $T_{\ell+1} = f_{\ell+1}(T_\ell)$ is injective on $\mathrm{supp}(T_\ell)$.

Remark 3.13. *This is later used as an important sanity check. If we do not choose the binning functions carefully, we might get an estimation of the information plane that does not exhibit this aforementioned property. If this occurs, the binning process has failed.*

Proposition 3.14 (Injective layer implies infinite MI[9])**.** *Suppose X is absolutely continuous and $x \mapsto T_\ell(x)$ is a.s. injective and locally Lipschitz. Then $I(X; T_\ell) = \infty$.*

Sketch. Smooth injective maps preserve differential entropy modulo Jacobian. For absolutely continuous X, $H(X, T_\ell) = -\infty$, so $I(X; T_\ell) = \infty$. □

Discretization alleviates this: binning collapses activation vectors into symbols, making T_ℓ noninjective and yielding finite empirical MI.

Proposition 3.15 (DPI for deterministic layers)**.** *If every $f_{\ell+1}$ is injective a.s., then $I(X; T_\ell) = I(X; T_{\ell+1})$ $\forall \ell$. Thus $I(X; T_\ell) = I(X; Y)$ for all layers.*

Proof. For deterministic f, $I(X; f(X)) = H(f(X))$ if X is discrete, and ∞ if X is continuous and f injective. Mutual information cannot drop. □

Consequences for empirical MI. Any *observed* decrease in plug-in estimates across layers stems from discretization or stochastic regularization (e.g., dropout), not deterministic computation. This explains why binning-based MI curves often show *compression*: discretization makes layers many-to-one, breaking injectivity.

3.6 Neural-network-specific analysis

We focus on deterministic feed-forward networks (Definition 2.4). For input $x \in X$, layer output is $T_\ell = f_\ell \circ \cdots \circ f_1(x)$.

3.6.1 Information saturation in deterministic networks

The following theorem formalizes the observation that a purely deterministic A feed-forward network cannot lose information about its input unless the mapping ceases to be injective.

Theorem 3.16 (Deterministic information saturation [10])**.** *Let X be the network input layer and $(T_\ell)_{\ell=1}^{L}$ the hidden layers of a deterministic feed-forward network with $T_\ell = f_\ell(T_{\ell-1})$.*

(i) If X is absolutely continuous and every f_ℓ is injective a.s., then $I(X; T_\ell) = \infty$ for all ℓ.

(ii) If X takes values in a finite set $\mathcal{D}$ and every composite map $x \mapsto T_\ell(x)$ is injective on $\mathcal{D}$, then $I(X; T_\ell) = \log_2 |\mathcal{D}|$ for all ℓ.

Proof sketch. Case (i) follows from Proposition 3.14. For case (ii) the map $X \mapsto T_\ell$ is a bijection on $\mathcal{D}$, hence $H(X \mid T_\ell) = 0$ and $I(X; T_\ell) = H(X) = \log_2 |\mathcal{D}|$. □

Remark 3.17. *Theorem 3.16 implies that any observed drop in empirical mutual information within a deterministic network must be attributed to* estimation procedure *(e.g., binning, numerical precision) rather than a true loss of information.*

3.6.2 Binning-induced compression and clustering

Proposition 3.18 (Goldfeld–Melnyk–Srikant [9])**.** *Let T_ℓ be a deterministic hidden representation discretized by a fixed partition $\mathcal{P}$. If M_N denotes the number of occupied cells after N samples and $\hat{I}_N(X;T_\ell)$ be the approximation of mutual information using the plug-in estimator, then*

$$\hat{I}_N(X;T_\ell) = \log_2 M_N - \hat{H}_N(X \mid T_\ell).$$

Hence any decrease in the plug-in estimate $\hat{I}_N$ is equivalent to a reduction in M_N—i.e., to geometric clustering *of samples in the representation space.*

Proof. Fix the N observed pairs $\{(x_i, T_\ell(x_i))\}_{i=1}^N$ and write

$$\mathcal{C} \;=\; \{C_1, \dots, C_{M_N}\} \subset \mathcal{P}, \qquad n_m \;=\; \#\{\, i : T_\ell(x_i) \in C_m\}, \qquad \sum_{m=1}^{M_N} n_m = N.$$

Step 1: write the empirical distributions. Since every sample of input is seen exactly once, it would not make sense to use it twice, the plug-in joint law is $\widehat{p}_{X,T_\ell}(x_i, C_m) = \frac{1}{N}$ whenever $T_\ell(x_i) \in C_m$ and 0 otherwise. Hence, the empirical marginal $\widehat{p}_{T_\ell}(C_m) = n_m/N$.

Step 2: plug into $\widehat{I}_N(X;T_\ell)$. Because T_ℓ is deterministic, $\widehat{H}_N(T_\ell \,|\, X) = 0$ and

$$\widehat{I}_N(X;T_\ell) = \widehat{H}_N(T_\ell) = -\sum_{m=1}^{M_N} \frac{n_m}{N} \log_2\!\Big(\frac{n_m}{N}\Big) = \sum_{m=1}^{M_N} \frac{n_m}{N} \log_2\!\Big(\frac{N}{n_m}\Big). \tag{1}$$

Step 3: separate the cell count M_N. Add and subtract $\log_2 M_N$ inside the logarithm in (1):

$$\frac{N}{n_m} \;=\; \frac{M_N}{n_m} \cdot \frac{N}{M_N} \quad\Longrightarrow\quad \log_2\!\Big(\frac{N}{n_m}\Big) = \log_2 M_N - \log_2 n_m.$$

Replacing this in (1) yields

$$\widehat{I}_N(X;T_\ell) = \underbrace{\log_2 M_N}_{\text{cell count}} \;-\; \underbrace{\sum_{m=1}^{M_N} \frac{n_m}{N} \log_2 n_m}_{\widehat{H}_N(X|T_\ell)},$$

which is exactly the claimed identity. □

This result explains the conflicting empirical reports on "compression". Networks with saturating nonlinearities (e.g., `tanh`) promote stronger clustering and thus show pronounced drops in M_N, whereas ReLU networks frequently maintain piecewise-linear structure and exhibit flat I curves as described in the paper On the information bottleneck theory of deep learning [11].

3.6.3 Making mutual information finite

Because $I(X;T_\ell)$ is either infinite or constant in deterministic networks, meaningful information-theoretic analysis requires deliberately breaking injectivity. Three practical options are:

a) **Additive noise.** Inject Gaussian noise $Z_\ell \sim \mathcal{N}(0, \sigma^2 I)$ into each layer. The resulting AWGN channel admits closed-form MI bounds and $I(X;T_\ell + Z_\ell) < \infty$ [9].

b) **Quantisation.** Constrain activations to a finite alphabet (low-precision arithmetic). For example eight-bit quantisation already yields stable MI estimates while preserving accuracy for layers of size 32.

Design takeaway. In fully deterministic networks, raw mutual information is ill-defined as a compression metric; one must inject noise or impose binning to obtain finite, interpretable values. As we are using the plug-in estimator that requires binning, we will stick to option b.

3.6.4 Noise-regularized networks

Definition 3.3 (Gaussian-noised layer)**.** *For $\sigma > 0$, set $T_\ell^\sigma = T_\ell + Z$ with $Z \sim \mathcal{N}(0, \sigma^2 I)$ independent of X.*

Theorem 3.19 (Finite MI and parametric rate [9])**.** *For any $\sigma > 0$, $I(X;T_\ell^\sigma)$ is finite. If T_ℓ^σ is discretized with a sieve satisfying $K_N = o(\sqrt{N})$, then:*

$$\mathbb{E}\Big[(\hat{I}_N - I)^2\Big] = O(1/N), \quad \textit{independent of } d_\ell.$$

Idea. Additive Gaussian noise ensures T_ℓ^σ has finite differential entropy. Apply Theorem 3.4 for the $O(1/N)$ rate. □

3.6.5 Direct estimation without noise

Practitioners often compute plug-in MI without noise injection. The estimator is defined only after discretization, causing:

i) **Bias growth.** If $K \approx b^{d_\ell}$ grows too quickly, plug-in estimators suffer severe negative bias (Theorem 3.6). Even though this potential exponential growth is possible, in practice, the populated alphabet tends to be manageable even for higher bit quantizations.

ii) **Variance inflation.** Many near-empty bins inflate variance, making confidence intervals unreliable. Bootstrapping is thus used to address this issue.

Takeaways. Use adaptive binning/clustering to enforce $K_N = o(\sqrt{N})$, apply Paninski bias corrections, report jackknife errors, and use bootstrapping, or use variational MI lower bounds that bypass discretization.

3.7 Practical guidelines

1. **Use sieves:** Refine discretization with $K_N = o(\sqrt{N})$ for consistency (Theorem 3.9).
2. **Bias correction:** Apply Miller–Madow or Paninski corrections; prefer jackknife estimators for large K.
3. **Interpreting MI curves:** Decreasing empirical MI signals clustering (Proposition 3.18), not information loss.
4. **Ensure finite MI:** Inject small Gaussian noise (Theorem 3.19).
5. **Phase-transition aware design:** Ensure $N \geq cK \log K$; else use variational estimators.
6. **Deterministic caveat:** In a strictly deterministic network the true $I(X; T_\ell)$ does not decrease. Interpret empirical MI curves only as clustering metrics, unless explicit noise or quantization is introduced.

3.8 Conclusion

Classical information theory provides sharp finite-sample error bounds for plug-in MI estimators and identifies regimes where estimation is impossible. In deterministic deep networks, empirical MI primarily measures clustering rather than loss of information. Mild-noise injection restores finite MI with parametric convergence. These insights guide the principled choices for discretization, data set size, and estimator corrections in deep learning pipelines. Most importantly, the theory points us to study the clustering process in order to fully interpret what mutual information in deterministic neural networks means, that is, what the next chapter will focus on.

4 Clustering Theory and Practical Metrics

One of the most important aspects of accurately calculating mutual information is the *binning* of hidden layers as described in Theorem 3.16. It is only natural to seek a theory that describes how well we can bin these real-valued vectors and, in turn, calculate their probabilities. The biggest issue is that the number of variables, and even their distributions, vary depending on the network architecture. Another challenge is that each model is initialized differently; in practice, this changes the size and number of clusters even for networks with identical architectures. Given this variability, we must employ techniques that do not impose tight constraints but provide meaningful insights.

That is why we introduce two practical methods designed specifically to quantify how well we sample in a vector space: the Good-Turing estimator and the *purity* estimator, a method created specifically for use in this work. Both of these metrics will be described in the following sections. We will use both methods to calculate a metric that measures how well we have *binned* a hidden layer.

Remark 4.1. *As described in Deep learning and the information bottleneck principle [1] and discussed repeatedly in Chapter 3, it is important to remember that the actual metric that we calculate is not the real continuous mutual information. That is easily seen from the relationship between Propositions 3.14 and 3.18. The actual MI of a deterministic neural network is almost always infinite, while we calculate a finite metric, that is bounded by the size of our dataset. What we are doing is measuring the clustering of different layers according to the binning functions chosen by us. That is why we focus on clustering in this Chapter.*

4.1 Cluster definitions and purity

Cluster. Let NN be a neural network with architecture $\mathcal{A}$ and H_i one of its layers. Consider a data set L, its internal representation in the layer H_i denoted $\mathcal{H}^{(i)}$, and a binning function T_i. A *cluster* is any subset of $\mathcal{H}^{(i)}$ that is mapped to the same vector (called a *marker*) by T_i. If a cluster contains only one vector, we call it a *mono-cluster*.

Label. The label of a vector $\boldsymbol{v} \in \mathcal{H}^{(i)}$ is defined as the label of the output produced by NN when processing the corresponding input. If we consider regression models, then a corresponding cluster in the output layer will be considered as the label.

Intuitively, when designing any binning method, we would like the groups in an inner layer to maintain a meaningful relationship with those in the subsequent layer. The perfect one-to-one correspondence is unrealistic, as that would imply a bijection between clusters for every layer. This would then mean that the binning function itself is solving the task of the neural network by clustering in earlier layers. Therefore, we can relax this perfect one to one correspondence to a weaker restriction. If two vectors are placed in the same cluster of H_i, we would prefer them to remain together in H_{i+1}. This motivates a notion we call *perfect purity.*

Definition 4.1 (Perfect Purity). *Let T_i and T_{i+1} be the binning functions applied to consecutive hidden layers H_i and H_{i+1}. A cluster $C \subseteq \mathcal{H}^{(i)}$ possesses* perfect purity *with respect to the pair (T_i, T_{i+1}) if*

$$\forall\, \boldsymbol{v}, \boldsymbol{w} \in C, \quad T_{i+1}\Big(\boldsymbol{h}^{(i+1)}(\boldsymbol{v})\Big) \;=\; T_{i+1}\Big(\boldsymbol{h}^{(i+1)}(\boldsymbol{w})\Big).$$

In words, every two vectors that are in a cluster H_i remain in the same *cluster after transformation to H_{i+1} and subsequent binning by T_{i+1}. The pair (T_i, T_{i+1}) is* perfectly pure *if the clusters* all *of H_i satisfy this property.*

As shown later, this requirement is too strong for real networks because it disallows growth in the number of clusters between layers, a phenomenon observed in almost every trained model. We therefore relax the perfect purity to a weaker notion of *label purity.*

Definition 4.2 (Label Purity). *Let $C \subseteq \mathcal{H}^{(i)}$ be a cluster with marker m and denote $y^*(m)$ as the label of m. The* label-purity *of C is*

$$\mathrm{Pur}(C) \;=\; \frac{\Big|\{\boldsymbol{v} \in C : \mathrm{label}(\boldsymbol{v}) = y^*(m)\}\Big|}{|C|}.$$

A cluster is pure *(or* labeled*) if $\mathrm{Pur}(C) = 1$ and $|C| > 1$. We define the weighted average purity of the layers as:*

$$\bar{\rho} = \sum_{k=1}^{K} \frac{|C_k|}{N}\, \mathrm{Pur}(C_k)$$

Remark 4.2. *The weighted average purity can be interpreted as the probability that an internal representation will be mislabeled into a wrong bin. If we then consider the rest of the neural network to be a function f and a specific internal representation P_i, then the output is $f(P_i) = Y$. Then, if we consider f as a channel we get the following result using Fano's inequality.*

Theorem 4.3 (Bias of Plug-in MI Estimator). *Let f be a binning function partitioning N samples into K clusters in a layer T, and Y a discrete label with $|\mathcal{Y}| < \infty$ and $\mathcal{T}$ be the support of binned T. The plug-in mutual information estimator $\hat{I}_N(T;Y)$ satisfies:*

$$|\mathbb{E}[\hat{I}_N(T;Y)] - I(T;Y)| \leq | - \frac{|\mathcal{T}| - 1}{2N} + O\left(\frac{1}{N^2}\right)| + H_B(\bar{\rho}) + (1 - \bar{\rho}) \log |\mathcal{T}|,$$

where $O(1/N^2)$ absorbs higher-order terms in $1/N^2$.

Proof. Decompose the bias:

$$|\mathbb{E}[\hat{I}_N] - I| = |\underbrace{\mathbb{E}[\hat{H}(T)] - H(T)}_{B_1} - \Big(\mathbb{E}[\hat{H}(T|Y)] - H(T|Y)\Big)|.$$

Part 1: Standard entropy estimation in Coverage adjusted entropy estimation [12]:

$$B_1 = -\frac{|\mathcal{T}| - 1}{2N} + O\left(\frac{1}{N^2}\right).$$

Part 2: By Fano's inequality [8] and the definition of $\bar{\rho}$, where H_B is the binary entropy:

$$H(T|Y) \leq H_B(\bar{\rho}) + (1 - \bar{\rho}) \log |\mathcal{T}|.$$

Part 3: Combine results, then from the non-negativity of all the expected values of plug-in-estimated entropy we get:

$$\begin{aligned}
\left|\mathbb{E}[\hat{I}_N] - I\right| &= |B_1 - \left(\mathbb{E}[\hat{H}(T|Y)] - H(T|Y)\right)| \leq |B_1 + H(T \mid Y)| \\
&\leq |B_1| + |H(T \mid Y)| \\
&\leq |-\frac{|\mathcal{T}| - 1}{2N} + O\left(\frac{1}{N^2}\right)| + |H_B(\bar{\rho}) + (1 - \bar{\rho}) \log |\mathcal{T}|| \\
&= |-\frac{|\mathcal{T}| - 1}{2N} + O\left(\frac{1}{N^2}\right)| + H_B(\bar{\rho}) + (1 - \bar{\rho}) \log |\mathcal{T}|
\end{aligned}$$

□

Now that we have defined the basic terminology, we can show that there exist binning functions satisfying both label purity and perfect purity via construction.

Theorem 4.4 (Existence of Perfect Purity via Functional Binning)**.** *Let* NN *be a depth L neural network processing inputs from a labeled dataset. For any layer H_i, define a* functional binning $T_i^{\mathsf{func}} : \mathcal{H}^{(i)} \to \mathcal{Y}$ *as:*

$$T_i^{\mathsf{func}}(\boldsymbol{v}) = \textit{output label of } \mathrm{NN} \textit{ when } \boldsymbol{h}^{(i)} = \boldsymbol{v}.$$

Then:

(i) ***Perfect Purity:*** *For any consecutive layers H_i, H_{i+1}, the pair $(T_i^{\mathsf{func}}, T_{i+1}^{\mathsf{func}})$ satisfies perfect purity (Definition 4.1).*

(ii) ***Cluster Collapse:*** $|\mathrm{Im}(T_i^{\mathsf{func}})| \leq |\mathrm{Im}(T_{i+1}^{\mathsf{func}})| \leq \cdots \leq |\mathcal{Y}|$*, where* $|\mathrm{Im}(\cdot)|$ *denotes the number of clusters.*

(iii) ***Label Preservation:*** *Every cluster $C \subseteq \mathcal{H}^{(i)}$ is label-pure (Definition 4.2).*

Proof. As every internal representation has the same label in every layer, all of the above is satisfied trivially. □

Remark 4.5 (Geometric Binning vs. Functional Binning)**.** *Theorem 4.4 shows that perfect purity is achievable with* functional *binning, but this is impossible for* geometric *binning (e.g., k-means, grids, or density-based clustering). The fundamental obstruction is that neural networks must* reorganize *representations across layers to solve complex tasks:*

- ***Early layers mix classes:*** *Intuitively, geometric clusters in shallow layers often contain multiple labels because they capture low-level features (edges, textures) shared across classes.*

- ***Nonlinear transformations split clusters:*** *Activation functions (ReLU, sigmoid) and weight transformations deliberately separate mixed groups from layer H_i to H_{i+1} to allow classification.*

- ***Choice of task:*** *Consider a binning function τ that is only geometric. Suppose that this function bins two different real vectors into one cluster. We can always imagine a classification task that would split these two vectors into two separate labels.*

Theorem 4.6. *Let* NN *be a neural network of depth L with binning functions $T_1, \dots, T_L$ (one per layer). If each consecutive pair (T_i, T_{i+1}) is perfectly pure for $i = 1, \dots, L-1$, then the clusters in* every *layer are label-pure.*

Proof. Fix a layer index $i < L$ and a cluster $C \subseteq \mathcal{H}^{(i)}$. By perfect purity of (T_i, T_{i+1}), the set $\widetilde{C} := \left\{\boldsymbol{h}^{(i+1)}(\boldsymbol{v}) : \boldsymbol{v} \in C\right\}$ satisfies

$$T_{i+1}(\boldsymbol{u}) = T_{i+1}(\boldsymbol{u}') \quad \forall\, \boldsymbol{u}, \boldsymbol{u}' \in \widetilde{C}.$$

Thus $\widetilde{C}$ is contained in a single cluster $C_{i+1} \subseteq \mathcal{H}^{(i+1)}$. Applying perfect purity of (T_{i+1}, T_{i+2}) to C_{i+1} yields

$$T_{i+2}\Big(\boldsymbol{h}^{(i+2)}(\boldsymbol{u})\Big) = T_{i+2}\Big(\boldsymbol{h}^{(i+2)}(\boldsymbol{u}')\Big) \quad \forall\, \boldsymbol{u}, \boldsymbol{u}' \in C_{i+1}.$$

Inductively, the entire set C is mapped to a single cluster C_L in the output layer H_L.

By construction, clusters of the output layer correspond one-to-one with labels, so every vector in C receives the *same* label y^*. Hence $\mathrm{Pur}(C) = 1$. Because C was arbitrary, all clusters in layer H_i are label-pure; repeating for $i = 1, \dots, L-1$ completes the proof. □

The original purity metric measures the semantic coherence *within* a single layer, while perfect purity enforces *structural coherence* across successive layers, precisely the consistency described above. Experiments show that perfect purity is rarely achieved, whereas label purity is attainable within an acceptable margin of error or even without error.

4.2 Good–Turing estimator

When binning an internal representation, we would naturally be interested in quantifying how many internal representations with non-zero probability we have not observed using this one particular binning function. The unobserved mass is then the sum of the probabilities of these unobserved internal representations. Let a binning function T induce clusters $C_1, \dots, C_K$ (as defined in Section 4.1) on a sample of N vectors. Let n_r be the number of clusters of size r, n_1 being mono clusters. Then the Good–Turing estimate[13] of the unobserved probability mass is

$$\hat{p}_0 = \frac{n_1}{N}. \tag{4.1}$$

To assign probabilities to observed events more accurately, we can use a smooth estimate introduced as an improvement in Good-Turing frequency estimation without tears[14]:

$$r^* = (r+1) \cdot \frac{n_{r+1}}{n_r}. \tag{4.2}$$

The adjusted counts r^* can then be normalized. If we then define $n_0 = N$ Equation (4.1) is simply the $r = 0$ case of Equation (4.2). This estimate allows us to check whether the binning and sample size are sufficient: a large $\hat{p}_0$ indicates poor coverage, while a small value suggests a reliable sampling of the distribution.

4.3 Balancing purity and coverage

We now have two complementary metrics for judging a binning strategy: *label purity* reflects semantic coherence, while the Good–Turing estimate $\hat{p}_0$ reflects coverage. A sensible strategy is to choose the coarsest binning that reaches an acceptable purity threshold while keeping $\hat{p}_0$ below, say 5%. This ensures clusters maintain meaningful label associations while adequately representing the underlying distribution.

Summary. The interplay of coverage ($\hat{p}_0$) and label purity gives a two-way trade-off. Our practical analysis in the next chapter will show how these metrics guide the choice of binning in real-world neural networks.

5 From Theory to Practice

The preceding chapters established (1) the impossibility of reliable plug-in estimation when the effective alphabet K grows too fast (Theorem 3.6); (2) finite–sample guarantees inside the *sieve regime* $K_N = o(\sqrt{N})$ (Theorem 3.9); and (3) operational surrogates—Good–Turing unseen mass and purity as defined in Definition 4.2 that allow *data–driven* control of K. The present chapter turns those results into a concrete pipeline that transforms a mini-batch $\mathcal{B} = \{(x_j, y_j)\}_{j=1}^{N}$ and a trained network NN into layer-wise estimates of the mutual information $\Big(I(X;T_\ell),\ I(Y;T_\ell)\Big)$. Only concepts introduced earlier (binning, purity, Good–Turing, jackknife, bootstrap) are used; no additional heuristics are introduced.

5.1 Adaptive neuron-wise discretization

Quantile codebooks. For neuron j in the hidden layer ℓ, let $A_{\ell,j}$ denote its activation. To bin $A_{\ell,j}$ we construct for every $a \in A_{\ell,j}$:

$$T_{\ell,j}(a) = k \quad \text{iff} \quad Q_{\ell,j}(k/\tau) \le a < Q_{\ell,j}\Big((k+1)/\tau\Big), \qquad k = 0, \dots, \tau - 1,$$

where $Q_{\ell,j}(p)$ is the empirical p–quantile of the current mini batch of internal representations and $\tau \in \{2, 3, 4, \dots\}$ will be chosen adaptively. The vector quantiser

$$T_\ell := \Big(T_{\ell,1}, \dots, T_{\ell,d_\ell}\Big) : \mathbb{R}^{d_\ell} \longrightarrow \{0, \dots, \tau - 1\}^{d_\ell}$$

produces the binned representation $V_\ell := T_\ell(A_\ell)$ that feeds the mutual–information estimator.

Three safety gauges. After binning a layer we compute

(G1) **Unseen mass** $\hat{p}_0$ via Eq. (4.1);

(G2) **Label purity** as in Definition 4.2;

(G3) **Sieve ratio** $K_\ell/\sqrt{N}$. (Only as a sanity check)

The adaptive rule is deliberately minimal:

$$\tau \leftarrow 2\tau \quad \text{until} \quad \hat{p}_0 \le 0.05,\ \text{purity} \ge 0.95,\ K_\ell \le 0.5\sqrt{N}. \tag{5.1}$$

As the norm of our maximal bin size cannot be zero, the sieve ratio will be more of a sanity check than an actual metric. We will mainly focus on the unseen mass and the purity of the label.

Formal description. The algorithm selects the maximal τ satisfying the safety criteria:

Algorithm 1 ADAPTIVE BINNING for one layer ℓ

Require: Activations $\{a_j\}_{j=1}^{N}$ in layer ℓ; initial $\tau \leftarrow 2$

1: **while** τ satisfies Eq. (5.1) **do**
2: $\quad \tau \leftarrow \tau + 1$ ▷ Increase bin count
3: **end while**
4: $\tau \leftarrow \tau - 1$ ▷ Revert to last valid τ
5: Build quantile codebook T_ℓ with τ equal-mass bins
6: **return** binned vector Z_ℓ

Because each refinement is nested and the cell diameters shrink geometrically, the sequence $\{\mathcal{P}_N\}$ forms a sieve; therefore, $\hat{I}_N$ is consistent with Theorem 3.9.

5.2 Plug-in mutual information with bias correction

For discrete variables U and V with joint probability mass function $\hat{p}_{UV}$ the basic plug-in estimator is

$$\hat{I}(U;V) = \sum_{u,v \in U,V} \hat{p}_{UV}(u,v) \log \frac{\hat{p}_{UV}(u,v)}{\hat{p}_U(u)\,\hat{p}_V(v)}.$$

Because $K = |\operatorname{supp}(U,V)|$ can be large, two classical bias-reduction techniques are employed.

(J) Delete–d jackknife. With $d = 2$ the bias-corrected estimate is $\hat{I}_{\mathrm{J}} := \hat{I}_{\text{plug}} - \left(\bar{I}_{(-)} - \hat{I}_{\text{plug}}\right)$, where $\bar{I}_{(-)}$ averages plug-in values after deleting every possible d–subset.

(B) Non–parametric bootstrap. From resamples $\{\mathcal{B}^{(b)}\}_{b=1}^{B}$ compute $\hat{I}_{\mathrm{B}} := 2\hat{I}_{\text{plug}} - \frac{1}{B}\sum_{b=1}^{B} \hat{I}_{\text{plug}}^{(b)}$, and use the percentile rule for confidence intervals.

These corrections inherit the convergence rate $O(N^{-1})$ ensured by Theorem 3.4, provided Eq. (5.1) holds.

5.3 Layer-wise pipeline

Combining Sections 5.1 and 5.2 yields the routine below.

Algorithm 2 Mutual Information Layer Scan

Require: Mini–batch $\mathcal{B}$; trained network NN; bias mode $\in \{\mathrm{J}, \mathrm{B}\}$

$A_0, \ldots, A_L \leftarrow \mathrm{NN}(\mathcal{B})$ ▷ forward pass
for $\ell = 0$ **to** L **do**
  $Z_\ell \leftarrow$ Adaptive Binning(A_ℓ)
end for
Initialize table $\mathcal{T} \leftarrow [\,]$
for $\ell = 1$ **to** $L - 1$ **do**
  $\hat{I}_\ell^{(X)} \leftarrow I_{\mathrm{mode}}\Big(Z_0, Z_\ell\Big)$
  $\hat{I}_\ell^{(Y)} \leftarrow I_{\mathrm{mode}}\Big(Z_L, Z_\ell\Big)$
  Append $(\hat{I}_\ell^{(X)}, \hat{I}_\ell^{(Y)})$ to $\mathcal{T}$
end forreturn $\mathcal{T}$

Every hidden representation Z_ℓ satisfies the sieve condition by construction; hence, the bias-corrected plug-in estimator attains the error bounds derived in Chapter 3.

This framework enables the calculation of mutual information between any two layers. As established in Chapter 1, our primary interest lies in the mutual information between the distributions of hidden layers and the input and output layers. The subsequent chapter will examine various hypotheses and tests, but first, we establish computational guarantees.

5.4 Computational complexity analysis

In the next section, we will denote:

N	Number of data points (mini-batch size)
L	Number of neurons in the hidden layer
B	Number of bins (τ) for clustering

5.4.1 Binning Complexity

The binning step transforms continuous activations $\mathbf{V} \in \mathbb{R}^{N \times L}$ into discrete bins via three steps, which have the time complexity:

1. Compute per-neuron statistics: $\Theta(NL)$
2. Calculate bin thresholds: $\Theta(LB)$
3. Discretize activations: $\Theta(NLB)$

Total: $\boxed{\Theta(NLB)}$

We test this empirically on different layer sizes and present our findings in the Tables 5.1,5.2,5.3, and 5.4.

Data Size	$\tau = 10$	$\tau = 20$	$\tau = 40$	$\tau = 80$
1024	0.001347	0.001085	0.000996	0.001200
2048	0.001422	0.001440	0.001617	0.002003
4096	0.001618	0.001803	0.002176	0.003051
8192	0.002061	0.002332	0.003062	0.004901
16384	0.003156	0.003944	0.005565	0.009356

Table 5.1 Execution times (seconds) for Layer Size 8. τ is the number of bins.

Data Size	$\tau = 10$	$\tau = 20$	$\tau = 40$	$\tau = 80$
1024	0.001609	0.001836	0.002267	0.003141
2048	0.001979	0.002431	0.003241	0.005338
4096	0.003040	0.004134	0.005542	0.008551
8192	0.006573	0.006692	0.010676	0.016799
16384	0.007657	0.011429	0.018288	0.032412

Table 5.2 Execution times (seconds) for Layer Size 32. τ is the number of bins.

Data Size	$\tau = 10$	$\tau = 20$	$\tau = 40$	$\tau = 80$
1024	0.002702	0.011003	0.006023	0.008936
2048	0.004955	0.005755	0.009489	0.017348
4096	0.009347	0.014080	0.022867	0.038862
8192	0.017517	0.023315	0.034512	0.076798
16384	0.031424	0.036826	0.087541	0.193508

Table 5.3 Execution times (seconds) for Layer Size 128. τ is the number of bins.

Data Size	$\tau = 10$	$\tau = 20$	$\tau = 40$	$\tau = 80$
1024	0.021181	0.020205	0.031293	0.094120
2048	0.028104	0.038241	0.093796	0.209669
4096	0.053613	0.107792	0.216142	0.410790
8192	0.162647	0.271366	0.524692	1.101788
16384	0.391145	0.554729	0.833604	2.696633

Table 5.4 Execution times (seconds) for Layer Size 1024. τ is the number of bins.

Remark 5.1. *As the computational tests were done on a GPU, smaller dataset sizes may be inconsistent, as the GPU can calculate the whole dataset in one go for smaller batch sizes. Below we present the following graphs, where we fix two out of three variables to better showcase the linearity. As to further prove our point, we present three figures, where in every graph, two of the variables are fixed, and we plot the computational time in relation to one remaining moving variable.*

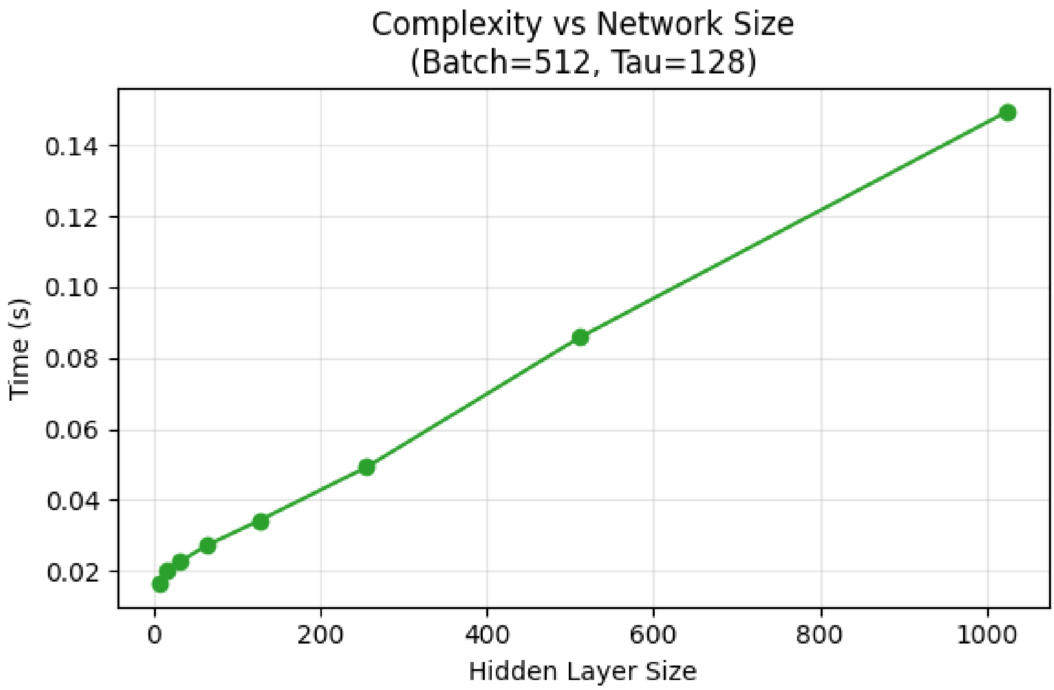


Figure 5.1 Linearity of binning complexity in regards to network size

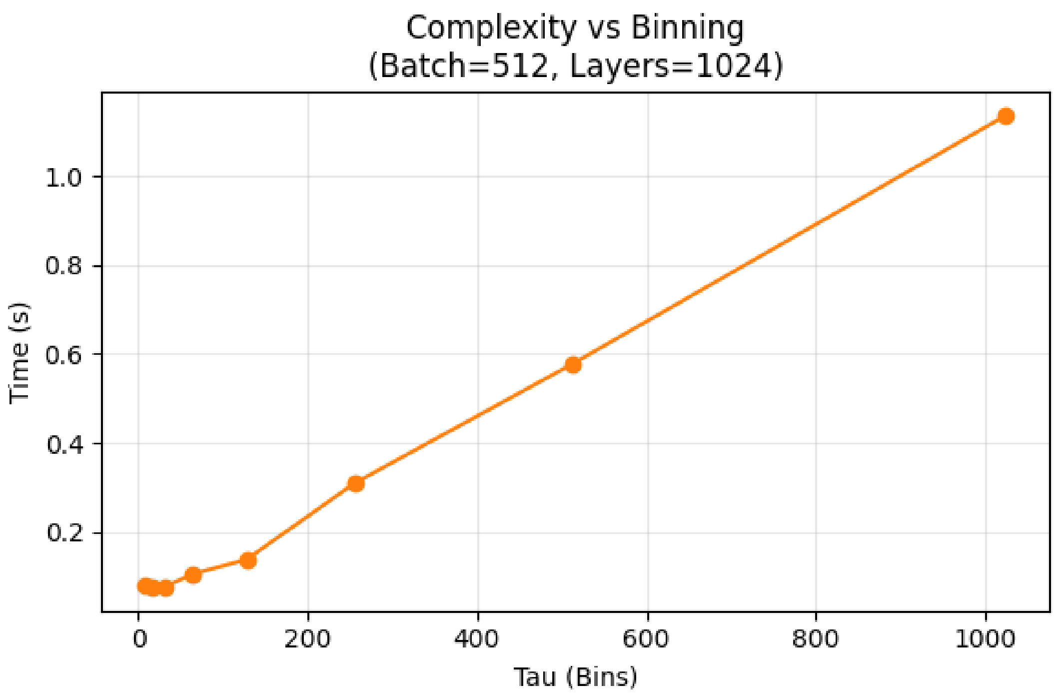


Figure 5.2 Linearity of binning complexity in regards to bin size

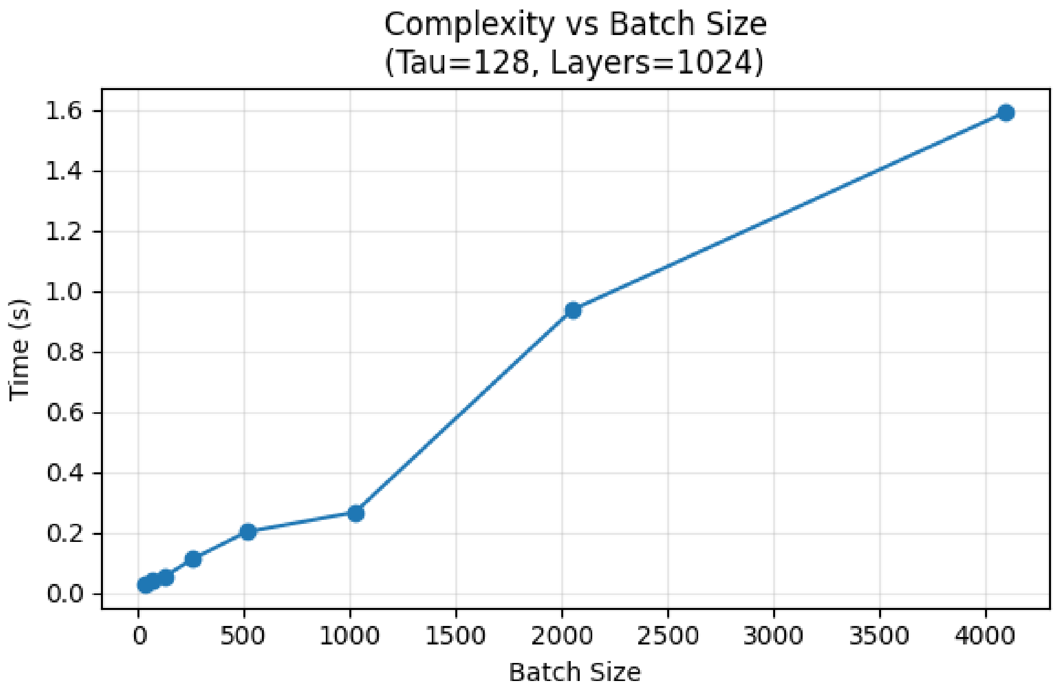


Figure 5.3 Linearity of binning complexity in regards to batch size

5.4.2 Replication of results

Remark 5.2. *As we can see, the trend in all three variables in Figures 5.1, 5.2, and 5.3 is linear. This is one of the main advantages of a plug-in estimator, as it is extremely fast and computationally scales very well. Its time complexity is good, so that in comparison to the training times of neural networks, it's almost negligible.*

5.4.3 Mutual information calculation complexity

After binning, the MI computation involves the following, where $U_X, U_Y, U_L \leq N$ denote unique vector counts.:

1. **Marginal distributions:** $\Theta(N \log N)$ for input/output, $\Theta(LN \log N)$ for hidden layers
2. **Joint distributions:** $\Theta(N) + \Theta(U_X U_L + U_Y U_L)$
3. **MI reduction:** $\Theta(U_X U_L + U_Y U_L)$

Total: $\boxed{\Theta\left(LN \log N + (U_X + U_Y)U_L + N\right)}$

3. Complexity regimes

- **Worst-case:** $U_X, U_Y, U_L = \Theta(N) \Rightarrow \Theta\left(LN \log N + N^2\right)$
- **Typical-case:** $U_X, U_Y = \mathcal{O}(1)$, $U_L = \mathcal{O}(N^\alpha) \Rightarrow \Theta\left(LN \log N\right)$

Remark 5.3. *As we can clearly see in Figures 5.1, 5.2 and 5.3, the trend is linear as shown in Section 5.4.*

Practical considerations

- Binning dominates for large B ($\Theta(NLB)$)
- Joint distributions dominate for large N ($\Theta(N^2)$)
- Memory: Joint tables require $\Theta((U_X + U_Y)U_L)$ entries
- Optimal B: Small bin counts ($B \leq 10$) control U_L and avoid N^2 scaling

Remark 5.4. *It is important to realize that the number of unique vector counts doesn't grow linearly with B. This growth is very difficult to describe, as even for the same neural network architectures, the clustering varies. Even so, this showed that the time complexity grows in a polynomial fashion in regard to unique vectors counts.*

5.5 Methodological foundations

5.5.1 Data source selection

The choice of data set used as a source for mutual information estimation requires careful consideration. When studying the training process, it is essential to ensure consistency across the training trajectory, and a fixed dataset should be used for all mutual information calculations during a single training instance to reduce bias. There are three potential data sources:

1. Sampled training data
2. Sampled test data
3. Combined training and test data

For experimental rigor, we consistently select between the first two options, with the specific source (training or test) explicitly documented in each experiment.

5.6 Replicating the Information bottleneck method

To validate our methodology, we replicate the information plane analysis from [2]. The procedure consists of four phases:

1. **Model Training:** Execute standard training while periodically saving model snapshots at predetermined epochs $\{t_1, t_2, \ldots, t_k\}$
2. **MI Calculation:** For each snapshot and layer ℓ:
   - Compute $I(X; T_\ell)$ using Algorithm 5.3.
   - Compute $I(Y; T_\ell)$ using Algorithm 5.3.
3. **Visual Encoding:** Construct the information plane:

- **X-axis:** $I(X; T_\ell)$ (input mutual information).
- **Y-axis:** $I(Y; T_\ell)$ (output mutual information).
- **Color:** Training epoch (temporal encoding).
- **Line types:** Solid lines connect same-layer trajectories across epochs; dashed lines connect layers within single epochs.

4. **Aggregation:** Repeat across multiple initializations and average Figure 1.1 illustrates the resulting information plane visualization.

5.6.1 Experimental setting

We verify our implementation against the original findings from Opening the Black Box of Deep Neural Networks via Information [2] using the following experimental setup:

- **Models:** 100 independent initializations
- **Architecture:** 6-layer fully connected network with layers 12-10-7-5-4-3-2 (matches the architecture [2])
- **Dataset:** Synthetic 12-bit spherical harmonic dataset ($N = 1024$) used in Opening the Black Box of Deep Neural Networks via Information [2]
- **Hyperparameters:** Learning rate $\eta = 0.005$, batch size 64
- **MI Calculation:** Performed on fixed test set subsample

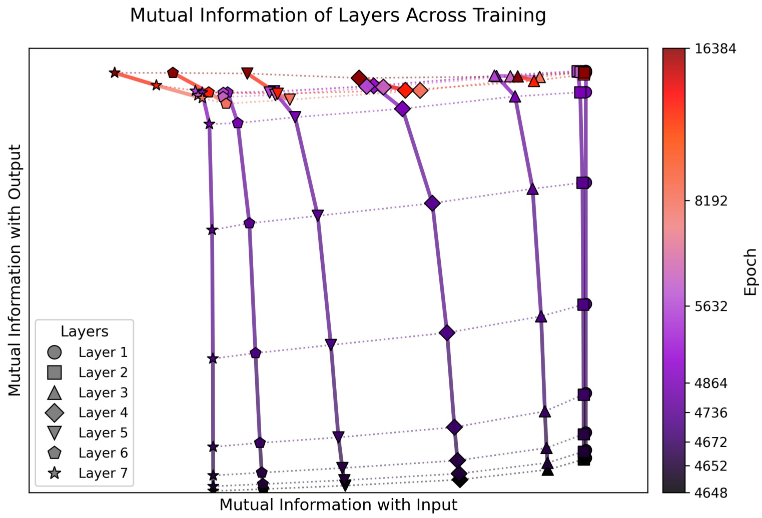


Figure 5.4 Replication of 1.1 using our methods

Our results (Figure 5.4) exhibit strong qualitative agreement with the original visualization of the information plane shown in Figure 1.1. The characteristic drift-diffusion phases described in Opening the Black Box of Deep Neural Networks via Information [2] emerge consistently across trials. Although exact quantitative replication proves challenging due to differences in optimization details (learning rate schedules, initialization schemes), the core phenomenon, progressive compression of input information while preserving predictive information, is captured.

Remark 5.5. *Two fundamental phenomena emerge in Figure 5.4:*

1. ***Data Processing Inequality:*** $I(X; T_{\ell+1}) \leq I(X; T_\ell)$ *holds throughout training, verifying information monotonicity.*

2. ***Drift-Diffusion Phase:*** *The characteristic rightward drift followed by downward diffusion identified in Opening the Black Box of Deep Neural Networks via Information [2] is detectable by our estimator.*

While our focus remains on final model analysis, the plug-in estimator successfully captures these training dynamics.

5.7 Conclusion

We have successfully replicated the results from Opening the Black Box of Deep Neural Networks via Information [2] using the plug-in estimator and our binning algorithm. This has been done with very good time complexity guarantees. We now have a method to calculate the MI structure of a neural network in a very fast and reliable manner. Therefore, it is time to utilize the theory and practice to finally perform experiments on a larger scale.

6 Empirical Analysis of Mutual Information and Generalization

6.1 Motivation

One of the main problems in machine learning is predicting how well a model is going to generalize to unseen data. If the test data set is unknown or has an unknown distribution, train loss can be used as a metric to estimate this performance, but this is often not sufficient, as the distributions of different datasets may vary. In this chapter, we will explore whether mutual information structure, as defined in Definition 2.14 correlates with the performance of models.

6.2 Introduction

Having established a practical information–estimation pipeline in Chapter 5, we now turn to empirical validation of the claim introduced in the Introduction.

Chapter road map

- §6.3 Formal notation and hypotheses
- §6.3.2 Statistical methodology
- §6.3.5 Unconditional MI–test-loss association with test loss on `make_moons`
- §6.4 Unique predictive power *beyond* training loss
- §6.5 Testing the intuition from Chapter 1 and introducing new metric
- §6.7 Predicting test loss using LIC

Hypothesis 1 ($\mathcal{H}_1$)**.** *The mutual information (MI) structure of a fully trained neural network exhibits a statistically significant association with generalization performance (test loss) that* persists *after accounting for the predictive power of training loss.*

Remark 6.1. *If $\mathcal{H}_1$ were true, we could use this to evaluate models and their performance without needing any test set. It could also be used to prune models while training to achieve a more efficient training process.*

6.3 Formal framework

6.3.1 Data representation

For a network NN instance trained to convergence, where train loss adjustments and accuracy are virtually unchanging, let M be the mutual information structure as defined in Definition 2.14 $\mathcal{I}(NN) = M$ and

$$L_{\text{train}},\ L_{\text{test}} \in \mathbb{R}_{\geq 0},$$

6.3.2 Statistical methodology

This section provides formal definitions of statistical concepts used in our analysis followed by citations of works where they were introduced. All metrics are defined for paired observations $\{(x_i, y_i)\}_{i=1}^n$ unless otherwise specified.

Correlation Measures

- **Pearson correlation (r) [15]**: Measures linear dependence between variables:
$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2}\sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$
where $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$, $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$.

- **Distance correlation ($\mathcal{R}$) [16]**: Measures both linear and nonlinear dependencies. Define distance matrices:
$$\begin{aligned} a_{ij} &= \|x_i - x_j\|, \quad b_{ij} = \|y_i - y_j\| \\ A_{ij} &= a_{ij} - \frac{1}{n}\sum_{k=1}^n a_{ik} - \frac{1}{n}\sum_{m=1}^n a_{mj} + \frac{1}{n^2}\sum_{k,m=1}^n a_{km} \\ B_{ij} &= b_{ij} - \frac{1}{n}\sum_{k=1}^n b_{ik} - \frac{1}{n}\sum_{m=1}^n b_{mj} + \frac{1}{n^2}\sum_{k,m=1}^n b_{km} \end{aligned}$$
Then distance covariance and correlation are:
$$\begin{aligned} \mathcal{V}^2(X,Y) &= \frac{1}{n^2}\sum_{i,j=1}^n A_{ij}B_{ij} \\ \mathcal{R}(X,Y) &= \sqrt{\frac{\mathcal{V}^2(X,Y)}{\sqrt{\mathcal{V}^2(X,X)\mathcal{V}^2(Y,Y)}}} \quad \text{for } \mathcal{V}^2(X,X)\mathcal{V}^2(Y,Y) > 0 \end{aligned}$$
with $\mathcal{R} = 0$ if $\mathcal{V}^2(X,X)\mathcal{V}^2(Y,Y) = 0$.

Regression Metrics

- **Coefficient of determination (R^2) [17]**: For linear model $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$:
$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

- **Adjusted R^2 [17]**: Penalizes for number of predictors p. By predictor, we mean one of the features we calculate the regression metric with, for example, it could be one specific $I(X, T_i)$ for a layer T_i:
$$R^2_{\text{adj}} = 1 - \left(\frac{SS_{\text{res}}}{SS_{\text{tot}}}\right)\left(\frac{n-1}{n-p-1}\right)$$

- **F-statistic [18]**: Tests joint significance of predictors ($H_0 : \beta_1 = \cdots = \beta_p = 0$):
$$F = \frac{(SS_{\text{tot}} - SS_{\text{res}})/p}{SS_{\text{res}}/(n-p-1)} = \frac{\text{Explained variance}}{\text{Unexplained variance}}$$

- **Incremental R^2 (ΔR^2) [19]**: Additional variance explained by new predictors:

$$\Delta R^2 = R^2_{\text{full}} - R^2_{\text{reduced}}$$

Significance tested via partial F-test:

$$F_{\text{partial}} = \frac{(SS_{\text{res, reduced}} - SS_{\text{res, full}})/(p_{\text{full}} - p_{\text{reduced}})}{SS_{\text{res, full}}/(n - p_{\text{full}} - 1)}$$

Statistical Significance

- **p-value**: For test statistic T under null hypothesis H_0:

$$p \;=\; \mathbb{P}\Big(|T| \geq |t_{\text{obs}}| \;\mid\; H_0\Big),$$

where t_{obs} is the observed value of T and the probability is taken with respect to the theoretical null distribution.

- **Permutation p-value (p_{perm}) [20]**: When the null distribution is unknown or intractable (e.g. for $\mathcal{R}$ or $\mathcal{R}^*$), we approximate it by randomly permuting one margin of the data B times. Let $\mathcal{T}^{\text{obs}}$ be the observed statistic and $\mathcal{T}^{(b)}$ the value from the b-th permutation. Then

$$p_{\text{perm}} \;=\; \frac{1 + \sum_{b=1}^{B} \mathbb{I}\Big\{\mathcal{T}^{(b)} \;\geq\; \mathcal{T}^{\text{obs}}\Big\}}{1 + B},$$

where the "+1" continuity correction guarantees $0 < p_{\text{perm}} < 1$ even in edge cases and $\mathbb{I}$ is the indicator function. In our experiments we use $B = 10^3$ permutations, giving a resolution of roughly 10^{-3}.

- **Partial distance correlation [21]**: Measures dependence between X and Y while controlling for Z (denoted dCor). Define doubly centered distance matrices A for (X, Z), B for (Y, Z), C for Z. Then

$$\mathcal{R}^*(X, Y \mid Z) \;=\; \frac{\langle A - P_C A,\, B - P_C B\rangle_F}{\|A - P_C A\|_F\, \|B - P_C B\|_F},$$

where $P_C = C(C^\top C)^{-1} C^\top$ is the projection matrix and $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product.

6.3.3 Implementation specifics

Our Python codebase follows the algorithmic flow outlined in Sections 5.1–5.3 and supports all experiments reported in this dissertation. This includes the unconditional association study on the standard benchmark dataset `make_moons`. Key ingredients are grouped below for clarity. We plan to release the repository publicly to our Github repository once completed.

- **Software architecture**
  - *Modular design:* independent packages for binning, MI estimation, regression/statistics, and plotting.

  - *Batch processing:* all estimators accept mini-batch tensors to exploit GPU parallelism.
  - *Adaptive binning:* quantile-threshold implementation of Algorithm 1.
  - *Bias correction:* jackknife and bootstrap routines selectable via a single flag.

- **Datasets**
  - `make_moons` ($N = 1024$) and `make_circles`, from SCIKIT-LEARN [22]. For brevity, in our tables, we are going to call these datasets moons and circles, respectively.
  - `Synthetic spherical` dataset, similarly as before, we are going to call this dataset spherical in tables for brevity[2].

- **Neural architectures**
  - Feed-forward networks denoted `L_N`: L hidden layers, each with N neurons (*e.g.*, `5L_32N`).
  - *Decreasing* variants: layer i contains $(L - i)N$ neurons, producing linearly tapering widths.

- **Training protocol**
  - 100 independent random seeds per architecture.
  - Stochastic Gradient Descent (SGD) with default momentum 0.9.
  - Early-stopping disabled to ensure equal epoch budgets.

- **MI estimation**
  - Kernel estimator of Chapter 5 evaluated *only at the final epoch* (i.e. we analyse the fully trained network) and its MI structure.

- **Statistical tests**
  - Correlation measures (see Section 6.3): Pearson r, distance correlation $\mathcal{R}$, partial distance correlation $\mathcal{R}^*$.
  - Significance via analytic p-values where available, otherwise permutation p_{perm} with $B = 10^3$ shuffles. We also use F-statistics.

- **Compute environment**
  - Single workstation: NVIDIA RTX 4070 (CUDA 12.4), Intel 13$^{\text{th}}$-Gen CPU, 32 GiB RAM.
  - Typical run time for a full `make_moons` sweep (all six architectures, 100 seeds) is ≈ 3.5 hours.

These design choices keep the information-plane analysis reproducible across datasets and models while remaining computationally tractable, cf. the complexity discussion in Section 5.4.

6.3.4 Baseline: predictive power of the training loss

We first regress the test loss on the training loss only; this is the natural method to predict how well a model generalizes on unseen data, so this will be a baseline for our predictions. We assume the following dependence:

$$L_{\text{test}} = \alpha_0 + \alpha_1 L_{\text{train}} + \varepsilon \tag{6.1}$$

and fit α_0, α_1 from several random initializations of the weights of the neural network and subsampled data sets. We then calculate the p-value to show statistical significance.

Architecture	Pearson correlation	adj. R^2	p-value
5L_32N	0.1905	0.0323	2.70×10^{-3}
4L_32N	0.1707	0.0251	8.03×10^{-3}
5L_4N	0.9954	0.9909	3.81×10^{-206}
6L_8N	0.2719	0.0683	3.94×10^{-4}
4L_8N	0.2376	0.0517	6.63×10^{-4}
5L_4N_dec	0.8975	0.8044	2.62×10^{-72}

Table 6.1 Correlation of test loss and regression metrics when *only* L_{train} is used as in equation 6.1

Observation. The training loss alone predicts test loss between models well. It is a useful tool, but not always accurate, as the Pearson correlation can be very low and even negative for heavily overfitting cases as shown in Table 6.6. Thus, we explore the possibility of MI structure predicting test loss further. First we need to establish whether there is any correlation present at all.

6.3.5 Unconditional MI–test dependence

Now that we have a baseline for predicting the test loss, we analyze the hypothesis $\mathcal{H}_1$, see Section 6.2 by performing two sets of statistical experiments in Tables 6.2 and 6.3. As the MI structure is not a single scalar, but rather a list of tuples, we need to use tools other than Pearson correlation. We use multiple linear regression and

Distance correlation and permutation-based inference To detect statistical dependence between mutual information (MI) features and test loss L_{test}, we use the *distance correlation* (dCor), as described in Section 6.3.2. To assess statistical significance, we compute a permutation-based p-value as also described in Section 6.3.2. This nonparametric procedure simulates the null distribution under independence.

Multiple linear regression We model L_{test} as a linear function of all MI features:

$$L_{\text{test}} = \beta_0 + \sum_{j=1}^{p} \beta_j \mathrm{MI}_j + \varepsilon$$

where l is the total number of MI features (input & output MI across layers). We report:

- **Adjusted** R^2: Variance explained (penalized for model complexity)
- F**-statistic**: Tests joint significance $H_0 : \beta_1 = \cdots = \beta_l = 0$

These approaches are complementary: dCor detects any dependence structure, while regression quantifies linear predictive power. We will then use p_{perm} and F statistics respectively to measure statistical significance.

Architecture	$\widehat{\text{dCor}}$	p_{perm}
5L_32N	0.362	0.001
4L_32N	0.282	0.012
5L_4N	0.324	0.002
6L_8N	0.250	0.181
4L_8N	0.378	0.001
5L_4_decreasing	0.278	0.049

Table 6.2 Unconditional distance correlation (dCor), and its associated p-value (`make_moons`)

Architecture	adj. R^2	F	p_F
5L_32N	0.471	45.2	9.4×10^{-14}
4L_32N	0.245	17.3	3.4×10^{-6}
5L_4N	0.958	1220	3.4×10^{-60}
6L_8N	0.191	7.5	3.5×10^{-3}
4L_8N	0.238	16.1	4.0×10^{-5}
5L_4dec	0.077	2.032	5.1×10^{-2}

Table 6.3 Unconditional multiple regression (MI only) (`make_moons`)

Take-away. In Tables 6.2 and 6.3 We see a correlation between the structure of mutual information in a neural network and test loss in every used architecture. There seems to be both linear and non-linear dependencies, and both experiments reported statistical significance in all except one architecture. Multiple linear regression also reports steady R^2. This correlation supports that $\mathcal{H}_1$ may be true, but we now need to perform experiments that account for train loss and its possible connection to MI structure.

6.4 Unique predictive power beyond training loss

6.4.1 Conditional predictive power of mutual information

As described in the previous Section, we already showed that there is a correlation between the MI structure and the test loss of a netowrk. But in order to confirm or reject hypothesis $\mathcal{H}_1$, we now need to control for the predictive power of train loss. This analysis tests whether the MI structure provides unique predictive power for test loss beyond what is captured by training loss alone. This will be done via two already introduced metrics in Section 6.3.2. This will be done in the manner as in the last experiments and as described in Section 6.3.3.

6.4.2 Conditional test results

Architecture	**Incremental R²**	**Partial dCor**
5L_32N	$\Delta R^2_{\text{adj}} = 0.462$ $F(10, 111) = 11.95$ $p = 8.78 \times 10^{-14}$	$\text{dCor} = 0.278$ $p_{\text{perm}} = 0.046$
4L_32N	$\Delta R^2_{\text{adj}} = 0.217$ $F(8, 110) = 5.21$ $p = 1.58 \times 10^{-5}$	$\text{dCor} = 0.258$ $p_{\text{perm}} = 0.038$
5L_4N	$\Delta R^2_{\text{adj}} = 0.009$ $F(10, 90) = 8049.04$ $p = 3.05 \times 10^{-128}$	$\text{dCor} = 0.800$ $p_{\text{perm}} = 0.001$
6L_8N	$\Delta R^2_{\text{adj}} = 0.157$ $F(10, 72) = 2.65$ $p = 0.008$	$\text{dCor} = 0.304$ $p_{\text{perm}} = 0.025$
4L_8N	$\Delta R^2_{\text{adj}} = 0.238$ $F(8, 92) = 5.17$ $p = 2.45 \times 10^{-5}$	$\text{dCor} = 0.376$ $p_{\text{perm}} = 0.002$

Table 6.4 Unique predictive power of MI beyond training loss (`make_moons`)

Key Findings from Table 6.4:

1. **Significant unique predictive power:** All architectures show statistically significant unique predictive power from MI in both metrics.

2. **Variance explained:** Adding MI features explains substantial additional variance in test loss, reaching to 46.2% beyond training loss. Only architecture 5L_4N reports lower linear dependencies, but still reports high nonlinear dependencies.

3. **Outstanding results:**
    - Strongest linear effect: 5L_32N ($\Delta R^2_{\text{adj}} = 0.462$)
    - Strongest nonlinear effect: 5L_4N (dCor* = 0.800)

Interpretation: These results from Table 6.4 confirm $\mathcal{H}_1$ in our setting. I structure provides statistically significant and practically meaningful predictive power for test loss beyond training loss. The strength and nature of this relationship vary depending on architecture, suggesting MI captures fundamental generalization mechanisms complementary to loss-based metrics.

6.5 Practical generalization metric

6.5.1 Metric definition and theoretical motivation

We operationalize $\mathcal{H}_1$ as presented in Hypothesis 1 into a concrete generalization predictor. Until now, we have used metrics that computed correlation and explained variance using more than one value. This is theoretically useful, but in practice, we would like to have a single scalar metric that would intuitively show how well a model generalizes. We utilize intuition from Chapter 1 and reward a bigger spread of mutual information with input.

Definition 6.1. *We define the **Layerwise Information Compression (LIC)** metric as:*

$$\text{LIC} := (I(X, T_1))^2 - \sum_{k=2}^{L} (I(X, T_i)^2$$

where $\mathrm{I}(X, T_k)$ *denotes the mutual information between input* X *and activations at layer* k*, and* L *is the total layer count.*

Theoretically, LIC operationalizes the intuition that more gradual information transformation from input to output benefits the performance of the model. Optimal generalization should correlate with high LIC values, reflecting balanced preservation and compression.

Hypothesis 2. *LIC of a fully trained neural network exhibits a statistically significant association with generalization performance (test loss).*

6.6 Empirical validation

We evaluate LIC's predictive power through Pearson correlation with test loss. We are going to separately analyze the Pearson correlation of the LIC metric with test loss and train loss with test loss. The latter was already discussed in Section 6.3.4, but we present it here for better interpretation of the results. We will denote the Pearson correlation of test loss and LIC as LIC Cor and the correlation between train loss and test loss as Loss Cor. We will also calculate the corresponding p-value for both cases and name them accordingly, as with correlation. This whole experiment will be done on two separate datasets `make_moons` and `make_circles`. For brevity, we will refer to these datasets as moons and circles, respectively.

Dataset	Model	LIC Cor	LIC p	Loss Cor	Loss p
moons	5L_32N	0.5073	5.0×10^{-8}	0.2110	3.7×10^{-3}
moons	4L_8N	0.4423	3.2×10^{-6}	0.2376	6.6×10^{-4}
moons	5L_8N	0.4501	1.7×10^{-5}	0.2719	3.9×10^{-4}
moons	5L_32N	0.3883	6.82×10^{-5}	0.9040	3.7×10^{-5}
moons	5L_4N	0.5201	2.9×10^{-12}	0.4137	1.4×10^{-9}
moons	3L_16N	0.0388	7.1×10^{-1}	0.9040	1.1×10^{-35}
moons	2L_32N	-0.0365	7.37×10^{-1}	0.4940	4.28×10^{-12}
circles	5L_20N	0.1514	1.27×10^{-1}	-0.1171	9.37×10^{-2}
circles	5L_8N	0.1984	4.57×10^{-2}	-0.0444	5.28×10^{-1}
circles	5L_4N_dec	0.0801	4.30×10^{-1}	-0.0948	1.84×10^{-1}
circles	4L_20N	0.2839	1.34×10^{-2}	0.2501	7.85×10^{-2}
circles	5L_7N	-0.0265	8.20×10^{-1}	-0.0198	8.10×10^{-1}

Table 6.5 Pearson correlations between LIC/training loss and test loss

Key Findings and Interpretation of Table 6.5

1. **Statistical significance**: In our experiments, 7/12 were statistically significant in both train loss and LIC correlation as shown in Table 6.5. However, the fact that one metric was statistically significant did not imply that the other was too. Also, none of the statistically significant experiments implied a negative correlation. Neither metric seems to correlate more strongly with the test loss than the other.

2. **Architectural dependencies**: Here we make observations about the performance of our metrics in the context of the architecture.

   - LIC seems to correlate more strongly with test loss for higher capacity models, i.e. more and larger layers. This may be because we inherently have fewer data points to detect complex relationships between layers of a model.
   - The correlation of both the LIC and train loss seem to differ between datasets. The `make_moons` seemed to report a stronger correlation with both metrics than `make_circles`.

3. **Conclusion**: The comprehensive experiment suggested that there is non-negative and statistically significant correlation between LIC and the test loss. This confirms $\mathcal{H}_2$ in our setting. The magnitude is similar to the correlation with train loss, which showed a similar correlation. These two metrics seem to report non-overlaping performances for models, suggesting that both could be used in fusion when selecting the best-performing model. This will be discussed in the next section.

6.7 Generalization predictor

As our Layerwise Information Compression (LIC) is computed via the train dataset and the results suggest a non-negligible non-negative correlation. The natural next step is to utilize the training dataset and LIC to predict the performance of networks on unseen datasets. This performance will again be measured by test loss.

To assess the predictive utility of LIC, we evaluated its ranking quality using various datasets of trained networks of the same architecture. Specifically, we sort networks by:

1. training loss alone,
2. LIC alone, and
3. the lexicographic sum of training loss and LIC.

We then compare these rankings to the ground truth ranking based on test loss.

To quantify the quality of the ranking, we use two well-established and complementary metrics from statistics and information retrieval. Spearman rank correlation coefficient (ρ) and normalized discounted cumulative gain at cutoff k (nDCG@k). Let us briefly describe both.

Spearman rank correlation coefficient $\boldsymbol{\rho}$ is a non-parametric measure of statistical dependence between two rankings introduced in the Paper The proof and measurement of association between two things [23]. Given two permutations $\pi, \sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ representing predicted and ground-truth ranks respectively, Spearman ρ is defined as:

$$\rho(\pi, \sigma) = 1 - \frac{6 \sum_{i=1}^{n} (\pi(i) - \sigma(i))^2}{n(n^2 - 1)}. \tag{6.2}$$

It ranges from -1 (perfect inverse correlation) to $+1$ (perfect correlation), with 0 indicating no rank correlation. It is ideal for measuring the overall monotonic agreement between predicted and actual orders.

Normalized Discounted Cumulative Gain (nDCG@k) focuses on the accuracy of top-ranked predictions as introduced in Cumulated gain-based evaluation of IR techniques [24]. Let rel_i denote the relevance of ground truth (e.g. inverse test loss) of the item ranked i-th in the predicted list. The discounted cumulative gain at position k is defined as:

$$\mathrm{DCG}\,@k = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)}, \tag{6.3}$$

and its normalized version is:

$$\mathrm{nDCG}\,@k = \frac{\mathrm{DCG}\,@k}{\mathrm{IDCG}\,@k}, \tag{6.4}$$

where IDCG $@k$ is the ideal DCG@k obtained from the perfect ordering. nDCG@k lies in $[0, 1]$ and provides fine-grained information on the quality of the top k ranking. In short, the classical correlation assesses the overall ranking, whereas the nDCG@k measures how well we ranked the list with a focus on the top k places. As we usually choose very few models, the nDCG will be more of a focus for us.

The test is going to be carried out on neural network architectures not yet used in previous tests to prevent any bias. It will also be performed on three binary tasks: `make_moons`, `make_circles`, and the synthetic spherical harmonic dataset used in the replication Section 5.6.1. We are going to utilize two separate ranking algorithms, which are going to predict the true ordered list according to test loss. These two algorithms are then evaluated by Pearson's correlation and nDCG. The two ranking algorithms are:

Methods of testing

1. **According to train loss** Classical method of predicting test loss, the lower the train loss, the lower the predicted test loss.

2. **Fusion analysis** We are going to utilize both training loss and LIC, creating two lists according to both training loss and LIC. Then we are going to create a new list, where the models will be ranked according to the sum of their placements in the separate orderings.

For clearer interpretation, we present the results in two tables quantifying the improvement of our fusion method (training loss + LIC) over the baseline (training loss alone). The difference metric is defined as:

$$\Delta_{\text{metric}} = (\text{fusion metric}) - (\text{train metric})$$

with positive values indicating fusion superiority.

Dataset	Model	Train ρ	Fusion ρ	$\Delta\rho$
moons	4L_32N	-0.155	0.157	+0.312
moons	4L_8N	0.230	0.319	+0.089
moons	5L_12N	0.297	0.294	-0.003
moons	5L_27N	0.148	0.284	+0.136
moons	2L_16N	0.454	0.287	-0.167
circles	4L_20N	0.027	-0.033	-0.060
circles	4L_12N	0.048	0.020	-0.028
spherical	5L_20N_dec	-0.869	-0.803	+0.066
spherical	6L_12N	0.246	-0.084	-0.330
Summary				
Avg $\Delta\rho$				+0.001
Fusion win rate				5/9
Max gain				+0.312

Table 6.6 Spearman ρ comparison between train loss ordering and fusion ordering (higher is better)

Dataset	Model	Train nDCG@5	Fusion nDCG@5	ΔnDCG
moons	4L_32N	0.244	0.472	+0.228
moons	4L_8N	0.338	0.574	+0.236
moons	5L_12N	0.472	0.611	+0.139
moons	5L_27N	0.688	0.550	-0.138
moons	2L_16N	0.656	0.658	+0.002
circles	4L_20N	0.529	0.629	+0.100
circles	4L_12N	0.482	0.536	+0.054
spherical	5L_4N_dec	0.044	0.115	+0.071
spherical	6L_12N	0.335	0.389	+0.054
Summary				
Avg ΔnDCG				+0.082
Fusion win rate				7/9
Max gain				+0.236

Table 6.7 nDCG@5 comparison between train loss ordering and fusion ordering (higher is better)

6.7.1 Key findings and interpretation

1. **Results**: Our fusion model performed better in 7/9 tests according to the nDCG@5 metric and performed better in 5/9 experiments according to the Pearson correlation. This shows a noticeable improvement in the top five rankings, while the overall correlation rankings seem to be unchanged.

2. **Pearson correlation**: Even though there is no widespread improvement noticeable, we observe that our fusion metric performs better on models where the train loss completely fails to be a good predictive feature. But this could be explained by the fact that when a model overfits heavily, even random sampling is better than sampling using train loss. This would be an interesting area of future research.

3. **nDCG@5**: This experiment shows noticeable improvement when choosing the best-performing models. There is only one occurrence when the usual train loss ranking outperforms our fusion model. This would suggest a better performance on binary tasks using our fusion model.

4. **Different Datasets**: Performance on datasets other than `make_moons` seems to be noticeable but needs to be studied more. Performance seems to deteriorate according to correlation statistics, but the top five rankings still show noticeable improvement even on other datasets. The more overfitting-prone models of the `spherical_binary` dataset suggest that both methods are very comparable.

5. **Correlation and LIC**: The correlation seems to be larger when considering the multivariate correlation with MI structure than when considering the Pearson correlation with LIC. This implies that LIC is not the perfect metric

to capture the full correlation from the MI structure. A more comprehensive analysis for more models is needed and will be an interesting area of research.

6.8 Conclusion

All experiments yielded positive results for our hypotheses. There is a nonzero correlation between the clustering and the mutual information structure of a neural network and its performance on unseen data. Although LIC does not produce the full correlation suggested by previous experiments, it can still be used to locate models that will perform better on unseen data. This method yields slightly better results for identifying the top 5 performing models. This would be an interesting are for more research with larger computational resources, larger models, and datasets.

7 Conclusion

7.1 Theoretical results

The whole topic of mutual information in the context of neural networks is a very complicated theoretical endeavor that is highly dependent on the estimator of choice. As we have chosen the plug-in estimator, robust information theory had to be introduced to have any theoretical guarantees. As this is a relatively new research area, these guarantees are often not practical enough for frequent use. Thus, we introduced a new concept of clustering purity. The purity of a layer, as introduced in this thesis, seems to be a good metric for measuring how well a binning function has been chosen. Not only is it a valuable metric when looking through the lens of channel theory, but it also yields good results in practice and is very useful when choosing a binning function, as it controls the clustering process to separate labeled internal representations.

In contrast, the Good-Turing estimator was used to control how well we have sampled the space of clustered internal representations. As the purity grows with finer binning functions, so does the estimated unobserved mass of the clustered space. Choosing the optimal binning function is then a trade-off between these two values. Because it is easy to implement, controlling the average purity of a layer is very useful.

7.2 Practical results

As experimental testing in Chapter 6 on binary data sets has shown, there is a clear correlation between the structure of the mutual information of a neural network and its performance on unseen data. This correlation is stronger for networks with more layers. This is intuitive as the more layers we have to analyze, the more complex relationships can be formed between the layers. When a network has varying amounts of neurons in each layer, it is even more important to choose the binning function by the binning functions of other layers, as the binned vector spaces are inherently different. This correlation is an interesting phenomenon and should be studied more, not only in the context of larger networks but also in the context of more difficult tasks, where the internal representations will be more complex and harder to quantify.

We have managed to utilize this theoretical correlation with LIC. This metric was inspired by the intuition of the introduction Chapter 1 and rewards a faster transformation of input. This metric manages to preserve some of the theoretical correlation and is utilized in practice. With binning functions adaptively chosen by the algorithm defined in Section 5.1, the experiments have shown a non-negative correlation for all statistically significant tests on binary problems.

This was then used to create an algorithm to choose models with the lowest test loss, and our tests have shown small to significant gains in choosing the top five models with the smallest test loss. This algorithm has been tested in three different binary settings, and all have shown at least minor improvements, with the biggest gains in neural networks with more layers. This has shown that the theoretical correlation can be utilized to assess the performance of neural networks. More comprehensive experiments are needed to examine the performance of not only this algorithm, but also LIC on larger models and tasks.

Bibliography

1. Tishby, Naftali; Zaslavsky, Noga. Deep learning and the information bottleneck principle. *arXiv preprint arXiv:1503.02406*. 2015. Seminal work on information bottleneck in DNNs.
2. Ravid Shwartz-Ziv, Naftali Tishby. Opening the Black Box of Deep Neural Networks via Information. *arXiv preprint arXiv:1703.00810*. 2017.
3. Vaart, Aad W. van der. *Asymptotic statistics*. Cambridge University Press, 2007.
4. Paninski, Liam. Estimation of entropy and Mutual Information. *Neural Computation*. 2003, vol. 15, no. 6, pp. 1191–1253. Available from doi: 10.1162/089976603321780272.
5. Treves, Alessandro; Panzeri, Stefano. The upward bias in measures of information derived from limited data samples. *Neural Computation*. 1995, vol. 7, no. 2, pp. 399–407. Available from doi: 10.1162/neco.1995.7.2.399.
6. Jiao, Jiantao; Venkat, Kartik; Han, Yanjun; Weissman, Tsachy. Minimax Estimation of Functionals of Discrete Distributions. *IEEE Transactions on Information Theory*. 2015, vol. 61, no. 5, pp. 2835–2885. Available from doi: 10.1109/TIT.2015.2412945. Minimax bounds for entropy estimation.
7. Durrett, Richard. *Probability: Theory and examples*. Cambridge University Press, 2019.
8. Cover, Thomas M.; Thomas, Joy A. *Elements of Information Theory*. 2nd. Wiley-Interscience, 2006.
9. Goldfeld, Ziv; Van Den Berg, Ewout; Greenewald, Kristjan; Melnyk, Igor; Nguyen, Nam; Kingsbury, Brian; Polyanskiy, Yury. Estimating Information Flow in Deep Neural Networks. *arXiv:1810.05728*. 2019. Finite MI in noise-regularized networks.
10. Amjad, Rana Ali; Geiger, Bernhard C. Learning representations for neural network-based classification using the Information Bottleneck Principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2020, vol. 42, no. 9, pp. 2225–2239. Available from doi: 10.1109/tpami.2019.2909031.
11. Saxe, Andrew M; Bansal, Yamini; Dapello, Joel; Advani, Madhu; Kolchinsky, Artemy; Tracey, Brendan D; Cox, David D. On the information bottleneck theory of deep learning*. *Journal of Statistical Mechanics: Theory and Experiment*. 2019, vol. 2019, no. 12, p. 124020. Available from doi: 10.1088/1742-5468/ab3985.
12. Vu, Vincent Q.; Yu, Bin; Kass, Robert E. *Coverage adjusted entropy estimation*. 2007. Available from doi: 10.21236/ada472999.
13. Good, I. J. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*. 1953, vol. 40, no. 3–4, pp. 237–264.
14. Gale, William A.; Sampson, Geoffrey. Good-Turing frequency estimation without tears*. *Journal of Quantitative Linguistics*. 1995, vol. 2, no. 3, pp. 217–237. Available from doi: 10.1080/09296179508590051.

15. Pearson, Karl. Vii. note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*. 1895, vol. 58, no. 347–352, pp. 240–242. Available from DOI: 10.1098/rspl.1895.0041.

16. Székely, Gábor J.; Rizzo, Maria L.; Bakirov, Nail K. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*. 2007, vol. 35, no. 6. Available from DOI: 10.1214/009053607000000505.

17. Ezekiel, Mordecai. 2.—Methods of Correlation Analysis. *Journal of the Royal Statistical Society*. 2018, vol. 94, no. 2, pp. 302–303. ISSN 0952-8385. Available from DOI: 10.1111/j.2397-2335.1931.tb01015.x.

18. Fisher, Sir Ronald. Statistical methods for research workers. *Statistical Methods, Experimental Design, and Scientific Inference*. 1990, pp. xxxv–xxxviii. Available from DOI: 10.1093/oso/9780198522294.002.0003.

19. Cohen, Jacob. *Statistical Power Analysis for the behavioral sciences*. 2013. Available from DOI: 10.4324/9780203771587.

20. Good, Phillip. Permutation, parametric and bootstrap tests of hypotheses. *Springer Series in Statistics*. 2005. Available from DOI: 10.1007/b138696.

21. Székely, Gábor J.; Rizzo, Maria L. Partial distance correlation with methods for dissimilarities. *The Annals of Statistics*. 2014, vol. 42, no. 6. Available from DOI: 10.1214/14-aos1255.

22. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

23. Spearman, C. The proof and measurement of association between two things. *Studies in individual differences: The search for intelligence*. 1961, pp. 45–58. Available from DOI: 10.1037/11491-005.

24. Järvelin, Kalervo; Kekäläinen, Jaana. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*. 2002, vol. 20, no. 4, pp. 422–446. Available from DOI: 10.1145/582415.582418.

List of Figures

List of Tables