Exploring data utilisation in neural networks

DG Haasbroek

orcid.org/0000-0002-9974-3626

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor: Prof MH Davel

Graduation: May 2022

Student number: 27059278

Declaration

I, Daniël Gerbrand Haasbroek, hereby declare that the dissertation entitled "Exploring data utilisation in neural networks" is my own original work and has not already been submitted to any other university or institution for examination.

Daniël Gerbrand Haasbroek
Student number: 27059278
Signed on 10 December 2021 at Potchefstroom.

Acknowledgements

I would like to thank my supervisor, Prof. Marelie Davel, for providing valuable guidance throughout my studies. Your patience and honest feedback are truly appreciated.

I would also like to thank Ulrike Janke for assisting with dreadful administrative tasks and formalities. Your friendly advice and assistance bring peace of mind in the most stressful situations.

To Prof. Albert Helberg I am thankful for the advice provided during the final years of my undergraduate studies. I would not have been able to pursue machine learning research without the opportunities and guidance that you provided.

Finally, to my parents I am thankful for providing support and assistance without hesitation. I am also thankful to my sisters for providing friendship and moral support.

This work was partly funded by the South African National Space Agency (SANSA).

Abstract

The generalisation ability of deep neural networks differs somewhat from that of more traditional models. Specifically, large networks that have the ability to "memorise" the training data can still generalise well, regardless of regularisation. This generalisation ability is still not completely understood. As part of a larger approach to studying the generalisation ability of neural networks by measuring the utilisation of training data, we aim to find a method of measuring mutual information and redundancy, specifically in the context of information processing in neural networks.

To this end, we study various existing redundancy estimators, and, using these as inspiration, we develop a new, nearest-neighbours-based redundancy estimator that can be used with discrete-continuous mixture distributions. We evaluate this new estimator on synthetically generated data and compare its behaviour to that of existing estimators. As a demonstration of the use of this estimator in neural network analysis, we calculate various node-based mutual information estimates in fully connected, feedforward networks trained for classification. Our demonstration reveals interesting regularities and differences between networks with different generalisation characteristics.

Overall, we implement and evaluate several redundancy estimators and show that node-based redundancy estimates can be used to analyse neural networks.

Keywords: deep neural networks, generalisation, mutual information estimation, redundancy estimation, information flow

Contents

List of Figures

List of Tables

List of Acronyms

CE Cross Entropy

DNN Deep Neural Network

LNC Local Non-Uniformity Correction

LSH Locality-Sensitive Hashing

MSE Mean Squared Error

ReLU Rectified Linear Unit

RMS Root Mean Squared

RNG Random Number Generator

List of Symbols

x — a scalar variable

$\mathbf{x}$ — a vector variable

X — a random scalar variable

$\mathbf{X}$ — a random vector variable

x_i — the ith component of the vector $\mathbf{x}$

$X^{(j)}$ — the jth sample point of the random variable X

$X_i^{(j)}$ — the ith component of the vector $\mathbf{X}^{(j)}$

$\mathcal{X}$ — a set

$f(\cdot)$ — a function

$\psi(\cdot)$ — the digamma function

$\mathbb{P}(e)$ — probability of event e

$\mathbb{H}(X)$ — entropy of the random variable X

$\mathbb{I}(X, Y)$ — redundancy between the random variables X and Y

$\mathbb{I}(\mathbf{X})$ — redundancy between the components of the random vector $\mathbf{X}$ (only when $\mathbb{I}(\cdot)$ is applied to a single random vector)

$\mathbb{T}(\cdot)$ — the indicator function: $\mathbb{T}(x) = 1$ if the predicate x is true; $\mathbb{T}(x) = 0$ otherwise

Chapter 1

Introduction

1.1 Generalisation in Deep Learning

Any mathematical modelling exercise aims to capture one or more relationships that exist among variables relevant to some task. In the context of such a task, not all existing relationships are usually of equal importance. In addition, when using machine learning methods to derive models from measurements of these relevant variables, the relationships to be captured are typically not known. Assuming that the set of measurements is finite, it is generally impossible to establish whether relationships observed among the measurements truly exist among the underlying variables; noise might produce spurious relationships, for example. Most machine learning exercises pose the aforementioned two problems — the task-specific problem of establishing which existing relationships are important, and the practical problem of establishing which observed relationships truly exist.

It is usually desirable for a machine learning model not only to model the training data well but also to model relationships that exist in similarly distributed, unseen data. In other words, it is usually desirable for a model to capture only those relationships that truly exist among the underlying variables. In a broad sense, the generalisation ability of a model refers to the degree to which the behaviour of the model on the training data agrees with the behaviour of the model on similarly distributed, unseen data [1], [2].

Given a finite set of measurements of some variables, statistical methods are often used to determine which relationships *probably* exist among the underlying variables. Bayes' theorem can be used to show that, among several functions that explain the

observed data, the simplest function is the most probable if the prior (of the functions considered) is uniform [2], [3]. Statistical learning theory suggests a related concept: the generalisation error of a trained model is bounded above by the complexity of the set of functions that can be represented by the modelling method [4], [5]. Although the complexity of a set of functions can be defined in several ways, a common approach is to relate the complexity to the highest correlation that can be obtained, on average, between a sequence of random values and a sequence of function values at random points, for any function in the set [2], [4]–[6].

The bounds that the complexity of the function class imposes on the generalisation error explain the generalisation ability of many model types, in the sense that an increase in the complexity of the function class, beyond the amount necessary to capture important relationships, leads to poorer generalisation of trained models [5].

Controlling the complexity of the function class is not the only method of controlling generalisation ability. Statistical learning theory suggests that good generalisation can also be obtained by introducing a preference for certain functions in the function class, even if the complexity of the function class is higher than needed [7]. The use of such regularisation techniques also explains the generalisation ability of many model types [6], [7].

For deep artificial neural networks (DNNs), neither the complexity of the function class of a particular network nor the use of regularisation techniques during training can adequately explain generalisation [6]. This was demonstrated in [6] and [8], among others, where the authors obtained good generalisation with networks that could, without any modification, fit random data easily, regardless of the use of regularisation techniques.

The generalisation ability of DNNs is still not completely understood. In this work, we explore tools that can assist us in investigating the generalisation ability of neural networks.

The problem that we address is best understood in the context of a larger approach to investigating generalisation. This larger approach is based on the informal idea of confirmation. Confirmation is based on the idea that the output of a trained artificial neural network should be confirmed by information in the set of sample points used to train the network. Thus, the *confirmation* of the output value of a network in response to a particular input is determined by the amount of task-relevant information used to construct that output. Similarly, the *confirmation ability* of a network refers to the

degree to which the outputs of a network are confirmed by task-relevant information. Since a network can obtain task-relevant information only from the data used to train that network, the confirmation ability of a network is strongly related to the utilisation of available training data. These definitions of confirmation, confirmation ability, and data utilisation should not be taken as precise, technical definitions but rather as broad, intuitive concepts.

A main hypothesis of our confirmation-based approach is that the generalisation ability of neural networks can be understood, at least partially, in terms of confirmation ability. However, no method of measuring confirmation ability is available, and developing such a method is a large task.

The measurement of confirmation ability involves the measurement of both task-relevant- and redundant information in networks. As a starting point for measuring these, we focus on the measurement of the information-theoretic concepts of mutual information and redundancy.

1.2 Problem Statement

As part of a larger goal of understanding the utilisation of training data in neural networks, we need an effective method of measuring mutual information and redundancy, specifically in the context of information processing in neural networks.

1.3 Research Scope

The scope of the work is limited as follows:

- Network architectures are limited to fully connected, feedforward neural networks. The activation functions used in hidden layers are limited to rectified linear units (ReLUs). These architectures are conceptually simple while still sharing important similarities with several high-performance architectures [9].

- Tasks are limited to classification, and we further limit the study to simple, widely used datasets. This allows us to focus primarily on developing effective measurement methods without getting distracted by the complexities of various datasets.

- Development and evaluation of methods for measuring mutual information and redundancy are prioritised. Full-scale applications of these methods, beyond simple demonstrations, are beyond the scope of this work.

1.4 Research Questions

The primary questions that this research aims to address are:

- Which properties of the probability distributions encountered in neural network analysis are relevant to the measurement of mutual information and redundancy?
- Which methods of measuring or estimating mutual information and redundancy are most suited to the types of probability distributions encountered in neural network analysis?
- How accurate are the estimation methods, and how is the accuracy affected by various factors?
- Which measurements of mutual information and redundancy are relevant to the measurement of confirmation ability?

1.5 Research Objectives

The following objectives are pursued to answer the research questions:

- We aim to identify the general properties of the probability distributions encountered in neural networks by studying both literature on probability- and information theory and other work that takes an information-theoretic approach to network analysis.
- We aim to identify methods of measuring or estimating mutual information and redundancy that are most suited to the probability distributions encountered in neural network analysis by studying the assumptions and requirements of various methods and matching these to the properties of the relevant distributions.

- Using the set of suitable measurement- or estimation methods, we aim both to characterise the accuracy of these methods and to select the most appropriate method by evaluating the methods on synthetically generated data.
- We aim to identify some measurements of mutual information and redundancy that are likely to form part of a measurement of confirmation ability by exploring the relationship between these measurements and generalisation ability.

1.6 Research Methodology

The following methodology is followed to complete the research objectives:

Literature Study A literature study is performed to obtain information on neural network generalisation, probability theory, information theory, estimation of mutual information and redundancy, and application of information theory to neural network analysis.

Estimator Development Relevant methods of estimating mutual information and redundancy are identified. The most promising methods are implemented in software.

Empirical Estimator Evaluation Synthetic data is generated, and the implemented estimation methods are evaluated on the synthetic data. Based on the results of this analysis, the method that seems most suitable for our network analysis is selected.

Estimator Application The selected estimation method is used to explore some measurements of mutual information and redundancy in neural networks. The relationship between these measurements and neural network generalisation is also explored.

Result Assessment The results of the analyses and experiments are assessed and used to derive conclusions and recommendations for further study.

1.7 Dissertation Overview

1.7.1 Layout

In Chapter 2, background information from the relevant literature is reviewed. In Chapter 3, the probability distributions encountered in neural network analysis are discussed. Several methods of estimating mutual information and redundancy are discussed in Chapter 4. Synthetic data is generated in Chapter 5, and the estimation methods are evaluated on the synthetic dataset in Chapter 6. In Chapter 7, we apply the estimators to neural networks. Finally, the results of our exploration are reviewed and conclusions are drawn in Chapter 8.

1.7.2 Notation

Throughout this document, random variables are denoted by uppercase letters (e.g. X or $\mathbf{X}$), while normal variables are denoted by lowercase letters (e.g. x or $\mathbf{x}$). Additionally, vector variables are denoted by letters in boldface (e.g. $\mathbf{x}$ or $\mathbf{X}$), scalar variables are denoted by italicised letters (e.g. x or X), sets are denoted by uppercase letters in calligraphy font (e.g. $\mathcal{X}$), special functions are denoted by uppercase letters in blackboard font (e.g. $\mathbb{H}(\cdot)$), and normal functions are denoted by italicised lowercase letters (e.g. $f(\cdot)$). Subscript indices indicate components of a vector (e.g. x_i, $\mathbf{x}_i$, X_i, or $\mathbf{X}_i$), while superscript indices indicate sample points of a random variable (e.g. $X^{(j)}$ or $\mathbf{X}^{(j)}$). Letters that denote functions are always followed by parentheses or square brackets.

These conventions are described here and listed on page xiii for reference. It should not be necessary to memorise these conventions — symbols are always introduced where they are used.

1.8 Publications

Prior to the work presented in the main body of this document, a pre-study, also related to the generalisation ability of neural networks, was performed as part of the current degree. This pre-study resulted in a conference paper [10] co-authored by prof Marelie H. Davel and presented at the SACAIR 2020 conference. The paper is reproduced in Appendix B. Parts of this paper are reused in Section 1.1.

A release of the implementation of the redundancy estimators presented in Chapter 4, together with an empirical analysis of the estimators, is in preparation.

1.9 Discussion

In this chapter, we introduced the problem of generalisation of neural networks. We introduced our larger approach to studying generalisation, and, within the context of this approach, we stated the problem that this work aims to address. Then, we elaborated on the scope of the research and the research questions, objectives, and methodology. Finally, we gave an overview of this dissertation and mentioned publications that form part of the current degree.

Chapter 2

Background

We review relevant literature and discuss background information and closely related work.

2.1 Chapter Overview

In this chapter, we review relevant literature and discuss background information and other work closely related to our research. In Section 2.2, we review concepts from information theory that are essential to the correct understanding and application of mutual information and redundancy. In Section 2.3, we discuss several methods of estimating mutual information and redundancy from a finite set of sample points. We review work related to the generalisation ability of DNNs, with a focus on work that uses an information-theoretic approach, in Section 2.4. Finally, in Section 2.5, we discuss aspects of the reviewed literature that are particularly relevant to our research.

2.2 Information Theory

In the broadest sense, information theory provides several tools for the measurement of information content [11], [12]. In this section, we review several of these tools, namely entropy (Section 2.2.1), which is useful for measuring the average amount of information in the value of a single variable, mutual information (Section 2.2.2), which is useful for measuring the average amount of information shared between two variables, and

redundancy (Section 2.2.3), which is an extension of mutual information that is useful for measuring the average amount of information shared between several variables.

For the definitions in this section, let $\mathbf{X}$ be a random vector variable taking on possible values from $\mathcal{X}$. Additionally, let $p_{\mathbf{X}}(\cdot)$ be the probability mass- or density function of $\mathbf{X}$. When all components of $\mathbf{X}$ are discrete, $\mathbf{X}$ is discrete, and $p_{\mathbf{X}}(\cdot)$ is a mass function; otherwise, $\mathbf{X}$ is continuous or mixed (as defined in Section 3.2), and $p_{\mathbf{X}}(\cdot)$ is a density function. Let $\mathbf{Y}$, $\mathcal{Y}$, and $p_{\mathbf{Y}}(\cdot)$ be similarly defined. Finally, let $p_{\mathbf{X},\mathbf{Y}}(\cdot,\cdot)$ be the joint mass- or density function of $\mathbf{X}$ and $\mathbf{Y}$. When both $\mathbf{X}$ and $\mathbf{Y}$ are discrete, $p_{\mathbf{X},\mathbf{Y}}(\cdot,\cdot)$ is a mass function; otherwise, $p_{\mathbf{X},\mathbf{Y}}(\cdot,\cdot)$ is a density function.

The expected value of $f(\mathbf{X})$ is, for discrete $\mathbf{X}$,

$$\mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}}}[f(\mathbf{X})] := \sum_{\mathbf{x}\in\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}) f(\mathbf{x})\,, \tag{2.1}$$

and, for continuous or mixed $\mathbf{X}$,

$$\mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}}}[f(\mathbf{X})] := \int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}\,. \tag{2.2}$$

2.2.1 Entropy

The entropy of a random variable is, intuitively, the average amount of uncertainty about the value of the variable, or, equivalently, the average amount of information that is gained when the value of the variable becomes known [11], [13]. Three common variations of entropy relevant to our study are discrete entropy, differential entropy, and relative entropy.

Discrete- and Differential Entropy

Discrete entropy is defined for discrete random variables, while differential entropy is defined for continuous random variables. The (discrete or differential) entropy of $\mathbf{X}$ is

$$\mathbb{H}(\mathbf{X}) := \mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}}}[-\log(p_{\mathbf{X}}(\mathbf{X}))] \quad [11], [13], \tag{2.3}$$

and the conditional entropy of $\mathbf{X}$ given $\mathbf{Y}$ is

$$\mathbb{H}(\mathbf{X} \mid \mathbf{Y}) := \mathbb{E}_{(\mathbf{X},\mathbf{Y})\sim p_{\mathbf{X},\mathbf{Y}}}\left[-\log\left(\frac{p_{\mathbf{X},\mathbf{Y}}(\mathbf{X},\mathbf{Y})}{p_{\mathbf{Y}}(\mathbf{Y})}\right)\right] \quad [11], [14]. \tag{2.4}$$

For discrete random variables, the conditional entropy of $\mathbf{X}$ given that $\mathbf{Y} = \mathbf{y}$ is

$$\mathbb{H}(\mathbf{X} \mid \mathbf{Y} = \mathbf{y}) := \mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}|\mathbf{y}}}[-\log(p_{\mathbf{X}|\mathbf{y}}(\mathbf{X}))] \quad [14], \tag{2.5}$$

where

$$p_{\mathbf{X}|\mathbf{y}}(\mathbf{x}) = \frac{p_{\mathbf{X},\mathbf{Y}}(\mathbf{x},\mathbf{y})}{p_{\mathbf{Y}}(\mathbf{y})} \tag{2.6}$$

is the conditional mass function of $\mathbf{X}$ given that $\mathbf{Y} = \mathbf{y}$, with $p_{\mathbf{Y}}(\mathbf{y}) \neq 0$. For all of these definitions, $0\log(0)$ and $0\log(\frac{0}{0})$ are defined as 0.

Discrete entropy measures the average amount of information in a value of a discrete random variable. Conditional discrete entropy measures the average amount of information in a value of a random variable given that the value of another random variable is known. Discrete entropy is always non-negative [13].

Differential entropy is not directly related to a simple, intuitive concept, but is rather a blind adaptation of discrete entropy to continuous variables [15]. Differential entropy can be any real value, which makes intuitive interpretation of it challenging [11].

Relative Entropy

Relative entropy, or Kullback-Leibler divergence (KL divergence), is defined for discrete and continuous random variables. Let $m_{\mathbf{X}}(\cdot)$ be a mass- or density function for $\mathbf{X}$ such that $p_{\mathbf{X}}(\mathbf{x}) = 0$ whenever $m_{\mathbf{X}}(\mathbf{x}) = 0$. Then, the relative entropy of $\mathbf{X}$ with respect to $m_{\mathbf{X}}(\cdot)$ is

$$\mathbb{H}[\mathbf{X} \mid\mid m_{\mathbf{X}}(\mathbf{X})] := \mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}}}\left[\log\left(\frac{p_{\mathbf{X}}(\mathbf{X})}{m_{\mathbf{X}}(\mathbf{X})}\right)\right] \quad [11], [13]. \tag{2.7}$$

Again, for this definition, $0\log(0)$ and $0\log(\frac{0}{0})$ are defined as 0.

Relative entropy is non-negative, and $\mathbb{H}[\mathbf{X} \mid\mid m_{\mathbf{X}}(\mathbf{X})] = 0$ if and only if either, for discrete distributions, $p_{\mathbf{X}}(\mathbf{x}) = m_{\mathbf{X}}(\mathbf{x})$ for all $\mathbf{x}$ or, for continuous distributions, $p_{\mathbf{X}}(\mathbf{x}) = m_{\mathbf{X}}(\mathbf{x})$ almost everywhere ($p_{\mathbf{X}}(\mathbf{x}) \neq m_{\mathbf{X}}(\mathbf{x})$ only on sets of Lebesgue measure zero) [16], [17].

One approach to using relative entropy to measure the average amount of information in a value of a continuous random variable $\mathbf{X}$ is to construct a discrete distribution for this variable [15]. The limit of the density of the points in this discrete distribution, as the number of these points approaches infinity, is then used as the reference probability density function $m_{\mathbf{X}}(\cdot)$, while the density function $p_{\mathbf{X}}(\cdot)$ of the original distribution remains unchanged [15], [18]. The negative relative entropy calculated using these density functions is viewed by some as the correct extension of discrete entropy to the realm of continuous distributions [15].

2.2.2 Mutual Information

The mutual information between two random variables is, intuitively, the average reduction in uncertainty about the value of one variable resulting from an observation of the value of the other variable, or, equivalently, the average amount of information that is gained about the value of one variable when the value of the other variable becomes known [11], [14]. The mutual information between $\mathbf{X}$ and $\mathbf{Y}$ is

$$\mathbb{I}(\mathbf{X},\mathbf{Y}) := \mathbb{E}_{(\mathbf{X},\mathbf{Y})\sim p_{\mathbf{X},\mathbf{Y}}}\left[\log\left(\frac{p_{\mathbf{X},\mathbf{Y}}(\mathbf{X},\mathbf{Y})}{p_{\mathbf{X}}(\mathbf{X})p_{\mathbf{Y}}(\mathbf{Y})}\right)\right] \quad [11], [14]. \tag{2.8}$$

For this definition, $0\log(0)$ and $0\log(\frac{0}{0})$ are defined as 0.

Mutual information is a special case of relative entropy and, therefore, has the same properties as relative entropy [11]; notably, mutual information is non-negative. Additionally, $\mathbb{I}(\mathbf{X},\mathbf{Y}) = 0$ if and only if either, for discrete distributions, $\mathbf{X}$ and $\mathbf{Y}$ are independent or, for continuous distributions, $\mathbf{X}$ and $\mathbf{Y}$ are independent almost everywhere.

2.2.3 Redundancy

Redundancy, or total correlation, is one extension of mutual information to finite sets of random variables [19]. (There are other extensions as well, and some are analysed in [19].) From the similarity between mutual information and redundancy, we can intuit that the redundancy in a set of random variables is the average amount of redundant information among the values of the variables. Our use of the term "redundancy" should

not be confused with the use in [13], where the term refers to a simple modification of discrete entropy.

Let $\mathbf{X}_1, \ldots, \mathbf{X}_d$ be the d vector-valued components of $\mathbf{X}$ respectively taking on possible values from $\mathcal{X}_1, \ldots, \mathcal{X}_d$. Additionally, let $p_{\mathbf{X}_1}(\cdot), \ldots, p_{\mathbf{X}_d}(\cdot)$ be the respective marginal mass- or density functions of $\mathbf{X}_1, \ldots, \mathbf{X}_d$. Then, the redundancy between the variables $\mathbf{X}_1, \ldots, \mathbf{X}_d$ is

$$\mathbb{I}(\mathbf{X}_1, \ldots, \mathbf{X}_d) := \mathbb{E}_{\mathbf{X} \sim p_{\mathbf{X}}} \left[\log \left(\frac{p_{\mathbf{X}}(\mathbf{X})}{p_{\mathbf{X}_1}(\mathbf{X}_1) \ldots p_{\mathbf{X}_d}(\mathbf{X}_d)} \right) \right] \quad [19]. \tag{2.9}$$

Again, for this definition, $0 \log(0)$ and $0 \log(\frac{0}{0})$ are defined as 0.

The relation

$$\mathbb{I}(\mathbf{X}_1, \ldots, \mathbf{X}_d) = \sum_{i=1}^{d} \mathbb{H}(\mathbf{X}_i) - \mathbb{H}(\mathbf{X}) \tag{2.10}$$

is also sometimes used to define redundancy [19], [20]. If these entropy values all exist, then this definition is equivalent to the one given initially. This relation provides some insight into redundancy — intuitively, the redundancy in a set of variables is the average reduction in information attained when the variables in the set are represented collectively rather than separately.

Redundancy, like mutual information, is a special case of relative entropy, and the same properties regarding non-negativity and independence apply to redundancy.

As mentioned in Section 1.1, we are interested both in measuring task-relevant information and in quantifying redundancies in neural networks. Mutual information can be used to measure task-relevant information in a random variable, as discussed in Section 2.4.6, and redundancy can be used to measure redundant information between random variables. Since mutual information is just a special case of redundancy, both of these requirements can be addressed by a single redundancy estimation technique.

2.3 Redundancy Estimation

Often, for a random vector $\mathbf{X}$ in $\mathbb{R}^d$ with components $X_1, \ldots, X_d$, the true distribution is unknown, but a finite sample of n independent, identically distributed points $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(n)}$ is available. In such a case, the available points can be used to estimate

the redundancy $\mathbb{I}(\mathbf{X}) = \mathbb{I}(X_1, \ldots, X_d)$.

In this section, we review several methods of estimating redundancy from such a finite sample. Specifically, we discuss 3H estimators (Section 2.3.1), partitioning estimators (Section 2.3.2), the ensemble dependence-graph estimator (Section 2.3.3), bounds estimation (Section 2.3.4), and nearest-neighbours estimators (Section 2.3.5).

2.3.1 3H Estimators

Several methods of estimating the redundancy in $\mathbf{X}$ requires direct estimation of the entropy $\mathbb{H}(\mathbf{X})$ of $\mathbf{X}$ and the entropies $\mathbb{H}(X_1), \ldots, \mathbb{H}(X_d)$ of the components of $\mathbf{X}$. The relation in Equation 2.10 can then be used to calculate a redundancy estimate [20], [21].

As discussed in Section 3.4, entropy is generally not defined for the variables that we consider. Therefore, any estimation method that requires direct estimation of an entropy value is not suitable for our purposes.

2.3.2 Partitioning Estimators

One of the simplest methods of directly estimating redundancy requires partitioning the values of each component X_i to form the discrete random variable $\bar{X}_i$ [20], [22]. The d discrete variables $\bar{X}_1, \ldots, \bar{X}_d$ then form the components of a discrete vector $\bar{\mathbf{X}}$. The probability $\mathbb{P}(\bar{\mathbf{X}} = \bar{\mathbf{x}})$ that the vector $\bar{\mathbf{X}}$ has value $\bar{\mathbf{x}}$ is set equal to the fraction of sample points with discretised value $\bar{\mathbf{x}}$. Using this construction, the redundancy in $\mathbf{X}$ is estimated as

$$\mathbb{I}(\mathbf{X}) \approx \mathbb{I}(\bar{\mathbf{X}}) = \mathbb{I}(\bar{X}_1, \ldots, \bar{X}_d) \quad [20], [22]. \tag{2.11}$$

In the limit as the number n of sample points and the number of partitions tend to infinity, the estimated redundancy converges to the true value if all underlying densities are proper functions [20]. If the densities are not functions, this limit might diverge [20]. As discussed in Chapter 3, the densities of the distributions that we consider cannot be expressed as proper functions.

The partitioning required to calculate the redundancy estimate is equivalent to adding noise to the random variables [22]. One relevant disadvantage of this is that the redundancy between the noisy variables is not invariant under invertible transformations of the true noiseless variables, and, therefore, when this estimation technique is used to

analyse neural networks, estimates from different networks are difficult to compare [22]. A more detailed discussion is presented in [22].

Because of these shortcomings, partitioning estimators are unsuitable for our purposes.

2.3.3 Ensemble Dependence-Graph Estimator

Another redundancy estimator uses locality-sensitive hashing (LSH) to map sample points to nodes in a dependency graph [23]. Each component X_i is mapped to a node with index $\bar{X}_i$ using a randomised LSH function. The d indices $\bar{X}_1, \ldots, \bar{X}_d$ form the components of a discrete index vector $\bar{\mathbf{X}}$. The probability $\mathbb{P}(\bar{\mathbf{X}} = \bar{\mathbf{x}})$ that the index vector $\bar{\mathbf{X}}$ has value $\bar{\mathbf{x}}$ is set equal to the fraction of sample points with index vector $\bar{\mathbf{x}}$. Using this construction, the base redundancy estimate is

$$\mathbb{I}(\mathbf{X}) \approx \sum_{\bar{\mathbf{x}} \in \bar{\mathcal{X}}} \left[f\left(\frac{\mathbb{P}(\bar{\mathbf{X}} = \bar{\mathbf{x}})}{\prod_{i=1}^{d} \mathbb{P}(\bar{X}_i = \bar{x}_i)} \right) \cdot \prod_{i=1}^{d} \mathbb{P}(\bar{X}_i = \bar{x}_i) \right] \quad [23], \tag{2.12}$$

where $\bar{\mathcal{X}}$ is the set of possible values of the index vector $\bar{\mathbf{X}}$, $\bar{x}_i$ is the ith component of $\bar{\mathbf{x}}$,

$$f(x) = \min\{x \log(x), u\}, \tag{2.13}$$

and u is an upper bound on $f(\cdot)$ [23]. The final ensemble estimator is obtained by using an ensemble estimation technique to improve the convergence rate of the base estimator [23].

In [23], this estimator is presented only for $d = 2$, and the probabilities associated with the index values are viewed as weighted connections in a graph. Although we suspect that this estimator will work in cases where $d \geq 2$, we make no attempt to verify this, and it is unclear how the estimator can be related to a dependency graph in such cases. Therefore, this estimator is not immediately suitable for our purposes.

2.3.4 Bounds Estimation

Instead of estimating redundancy directly, upper and lower bounds on the true redundancy can be calculated [24]. One such set of bounds, for $d = 2$, requires that one of the random variables be a mixture of finitely many component distributions, where

closed-form expressions are known for the entropy of each component distribution, the Chernoff α-divergence in each pair of component distributions, and the KL divergence in each pair of component distributions [24]. These expressions are typically obtained by assuming that the component distributions are well-known distributions (e.g. normal or uniform distributions) [24].

Calculating such bounds on the redundancy is more computationally efficient than estimating the redundancy, but, due to the assumption about the structure of the variables, the bounds have an estimation bias, which is bounded [24]. In addition, these bounds depend on the existence of the conditional entropy of one variable given the other, and it is not clear whether these bounds can be extended to cases where $d \geq 2$ [24]. As discussed in Section 3.4, the entropy of the variables that we consider is generally not defined. These shortcomings make this technique unsuitable for our purposes.

2.3.5 Nearest-Neighbours Estimators

The nearest neighbours of each sample point $\mathbf{X}^{(j)}$ can be used in the estimation of redundancy. Estimators that use this technique exist for continuous random variables [20], [25] and for more general random variables [21]. Such estimators are also applicable to cases where $d \geq 2$. These estimators seem most suitable for our purposes and are described in detail in Sections 4.2 to 4.4.

2.4 Related Work

In this section, we discuss work closely related to our research. We briefly review work concerned with the prediction of generalisation ability (Sections 2.4.1 to 2.4.5) as general background information. Then, we discuss work that informs our approach, namely information-theoretic analysis of generalisation (Sections 2.4.6 and 2.4.7) and node-based network analysis (Section 2.4.8).

2.4.1 Hypothesis Space Complexity

As discussed in Section 1.1, the complexity of the hypothesis space, which is determined by network architecture, can be used to construct an upper bound on the generalisation

error of a given network [26]. Although this provides an upper bound, it does not explain how networks achieve generalisation errors significantly lower than the upper bound; that is, low complexity is sufficient, but not necessary, for generalisation [26].

2.4.2 Training Algorithm Stability

In a weaker sense, the generalisation error of a network is also bounded by the stability of the algorithm used to train the network (the worst-case error of a network across all datasets is not bounded) [26]. Stability is a measure of the sensitivity of a training algorithm to changes in the training set, and, for many stability measures, this change amounts to the removal of a single data point from the training set [27]. Typically, some form of expected value of the change in the loss is measured to determine the stability of the algorithm [27]. Again, stability is sufficient, but not necessary, for generalisation [26].

2.4.3 Training Algorithm Robustness

The robustness of a training algorithm is similar to its stability and imposes similar bounds on the generalisation error [26]. Where stability implies small changes in loss when the training set is slightly modified, robustness implies small changes in loss at any single data point when that point is moved by a small amount [28]. More formally, an algorithm is robust if there exists a fixed partition of any training set, which is independent of the training set, such that the difference in loss at any two points in the same partition is bounded [28]. As is the case for stability, robustness is sufficient, but not necessary, for generalisation [26].

2.4.4 Loss Landscape Geometry

Another approach considers the behaviour of the loss as a function of the network parameters, specifically focusing on the sharpness of the minimum on which the training algorithm converged. Based on the theory that models which require less information to describe generalise better, it is proposed that networks with parameters at flatter minima lead to better generalisation, since they can be specified with less precision [29]. However, reparameterisation of DNN models allows the flatness of minima to be altered with no effect on generalisation performance [26], [30].

2.4.5 Margin Distributions

Margin distributions can also be used to study generalisation. When using a DNN to perform classification, the distance from any sample point $\mathbf{X}$ or intermediate representation $\bar{\mathbf{X}}$ to the decision boundary of the network can be precisely defined, but the computation of this distance is intractable [31]. The authors of [31] propose a method of estimating this distance, and this method is used in [32] to estimate the distances of training points and their intermediate representations to the relevant decision boundaries. The distributions of these distances, called margin distributions, are then used to predict generalisation ability with surprising accuracy [32].

2.4.6 Information Bottleneck

The information bottleneck method provides a way of obtaining a compressed representation $\bar{\mathbf{X}}$ of some data $\mathbf{X}$ while preserving the information relevant to some task. Where rate-distortion theory allows the task relevance to be explicitly specified with a distortion function, the information bottleneck method allows the relevance to be implicitly specified with an additional relevance variable Y [33].

To maximise compression of $\mathbf{X}$, the redundancy $\mathbb{I}(\mathbf{X}, \bar{\mathbf{X}})$ between the data $\mathbf{X}$ and the compressed representation $\bar{\mathbf{X}}$ must be minimised; to ensure that $\bar{\mathbf{X}}$ still contains most of the relevant information in $\mathbf{X}$, the redundancy $\mathbb{I}(\bar{\mathbf{X}}, Y)$ between $\bar{\mathbf{X}}$ and the relevance variable Y must be maximised [33]. Thus, the information bottleneck method proposes minimising the functional

$$l\left[\mathbb{P}(\bar{\mathbf{X}} = \bar{\mathbf{x}} \mid \mathbf{X} = \mathbf{x})\right] = \mathbb{I}(\mathbf{X}, \bar{\mathbf{X}}) - \beta\mathbb{I}(\bar{\mathbf{X}}, Y) \quad [33], \tag{2.14}$$

where β is a parameter that controls the importance of preserving relevant information relative to the importance of compressing the representation [33].

In the analysis of neural networks, the input to the networks is typically viewed as the data $\mathbf{X}$ to be compressed, and the internal representations formed by the layers of the network are viewed as compressed representations $\bar{\mathbf{X}}$ of $\mathbf{X}$. The desired output of the network is viewed as the relevance variable Y.

In [34], the authors analyse fully connected, feedforward networks using concepts from the information bottleneck method. They note that the input layer of a DNN contains

the most information about the relevance variable Y, but the representation of this information is too complex to allow good generalisation. From this they conclude that compression is required for generalisation.

An empirical analysis of the training of fully connected, feedforward networks using stochastic gradient descent reveals two phases of training: a learning phase driven by large gradients with relatively low noise, and a compression phase driven by small gradients with relatively high noise [35]. It is suggested that the noisy gradients, which cause compression in the internal representations, are at least partially responsible for the generalisation ability of DNNs [35]. Other work claims that the appearance of the two training phases is largely determined by the choice of activation function [22]. This work also claims that compression is not necessary for generalisation and that noisy gradients are not necessary for compression [22].

In [36], the authors analyse feedforward networks with noisy activation functions. They show that the redundancy between the input data $\mathbf{X}$ and the internal representation $\bar{\mathbf{X}}$ at some hidden layer is related to the clustering of sample points at that layer — at a hidden layer, the compression of the input data is caused by the clustering of sample points. They further show that a lack of clustering and compression does not necessarily affect the generalisation ability of a network.

U-nets [37] are analysed in [38] using a similar information bottleneck approach. In contrast to other architectures analysed with this approach, U-nets have skip connections, and, therefore, information can flow through multiple paths in the network. The authors use estimates of $\mathbb{I}(\mathbf{X}, \bar{\mathbf{X}})$ and $\mathbb{I}(\bar{\mathbf{X}}, Y)$ to identify skip connections that are necessary for optimal network performance. By evaluating several modified networks, they then confirm the necessity, or lack thereof, of several skip connections.

2.4.7 Information Transfer

The concept of the information transfer of a network is introduced in [39]. Intuitively, information transfer measures the amount of task-relevant information contained in a network, with the additional property that only information which allows the network to generalise is measured. The authors demonstrate the utility of information transfer in the comparison and analysis of neural networks.

2.4.8 Node-Based Network Analysis

Network nodes with ReLU activation functions can be interpreted as showing discrete behaviour — if the activation value of a node is zero, that node is not activated; if the activation value is non-zero, the node is activated [9], [40].

The authors of [9] show how this discrete interpretation can be combined with entropy estimates to reveal class-specific clusters of sample points in fully connected, feedforward networks. They also show that such a network can be viewed as consisting of interacting discrete and continuous systems, and they evaluate the classification performance of each system at different layers in the network throughout training. Finally, based on the results of this analysis, they suggest that the generalisation ability of a neural network can be attributed to the collaboration of diverse classifiers formed by individual nodes in the network.

With this discrete view of ReLU nodes, the authors of [8] show that, in networks trained with various forms of noisy data, individual nodes tend to activate either for noisy sample points or for noiseless sample points of a given class, but more rarely for both. Based on this observation, they argue that a network consists of several sub-components that are responsible for modelling overlapping subgroups of sample points. They suggest that this organisation of nodes prevents an increase in network capacity from deteriorating generalisation ability.

2.5 Discussion

In this chapter, we reviewed concepts from information theory, discussed various methods of estimating redundancy, and summarised work related to our research.

Of the existing redundancy estimators discussed in this chapter, one estimator, which is based on nearest neighbours, satisfies the following three properties:

1. The estimator is directly applicable to points sampled from a mixture distribution (as defined in Section 3.2);

2. The estimator is directly applicable to points with $d \geq 2$;

3. The estimates calculated on different sets of points are readily comparable.

Since only one existing estimator satisfies these properties, we do not further compare estimators based on classical properties like consistency or convergence rate.

In the majority of studies where information theory is used to analyse networks, the main focus is placed on network layers. In contrast to this, we aim to combine the node-based approach of the work reviewed in Section 2.4.8 with the information-theoretic approach of the work reviewed in Sections 2.4.6 and 2.4.7.

In the following chapter, we describe the properties of the probability distributions encountered in neural network analysis.

Chapter 3

Mixture Distributions

We briefly review the probability distributions relevant to the network architectures that we consider. We discuss how the characteristics of these distributions affect the calculation of entropy and redundancy.

3.1 Chapter Overview

In this chapter, we describe the properties of the probability distributions relevant to the network architectures that we consider, namely fully connected, feedforward networks with ReLU activation functions in the hidden layers. We then discuss the effects of these properties on the calculation of entropy and redundancy.

A few important definitions, including the definition of a mixture distribution, are given in Section 3.2. In Section 3.3, we describe the mathematical representation of mixture distributions. The effects of this representation on the calculation of entropy and redundancy are discussed in Sections 3.4 and 3.5, respectively. In Section 3.6, we summarise the ideas presented in this chapter.

3.2 Definitions

Continuous probability distributions are typically represented with a probability density function, where the value of the density function at any point gives the probability per unit volume at that point. The probability that a random variable will have a value

within a specific region can be obtained by integrating the density function across that region. However, the probability that the random variable will have a specific value is zero, since there is an infinite number of possible values that the variable could have.

Because of the use of ReLU activation functions in our networks, many nodes have at least one activation value that has a non-zero probability of being produced by the node, while the remaining activation values are continuously distributed. We refer to the values with non-zero probability as the *discrete points* of the distribution; the part of the distribution that assigns non-zero probabilities to the discrete points is the *discrete component* of the distribution, while the remaining part is the *continuous component*. We also use the term *mixture distribution* to refer to a distribution with both continuous and discrete components, and a random variable with such a distribution is a *mixed random variable*.

3.3 Representation

The density of a mixture distribution at a discrete point is not defined — a non-zero probability is distributed across a region with zero volume. The probability of any value or range of values, however, is still well defined.

The Dirac delta "function" is an object that can be used to represent the undefined density values of the discrete component of a mixture distribution while keeping all probabilities well defined. Although this object is not a true function, we refer to it as such for simplicity. The Dirac delta function $\delta(\cdot)$ has two heuristic properties [41], [42] that also define it:

$$\delta(\mathbf{x}) := 0 \quad \text{for all } \mathbf{x} \neq \mathbf{0}\,; \tag{3.1}$$

$$\int_{\mathcal{X}} \delta(\mathbf{x}) d\mathbf{x} := 1 \quad \text{if } \mathbf{0} \in \mathcal{X}\,. \tag{3.2}$$

These properties are only heuristics because they are contradictory — for any function with the first property, either the integral in the second property does not exist, or that integral is zero [17], [41]. Because of this contradiction, the Dirac delta function is not a true function, and special care must be taken when working with it.

In the density function of a mixture distribution, each discrete point has a correspond-

ing term that consists of a shifted and scaled Dirac delta function. Consequently, the density at each discrete point is not defined, but integrating the density function yields a well defined probability value. As an example, consider a distribution with discrete points $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_n$, where each $\bar{\mathbf{x}}_i$ has a probability of $\bar{p}_i$. If the normalised density function of the continuous component is $p_c(\cdot)$, then the density function of the complete distribution is

$$p(\mathbf{x}) = \left(1 - \sum_{i=1}^{n} \bar{p}_i\right) p_c(\mathbf{x}) + \sum_{i=1}^{n} \bar{p}_i \delta(\mathbf{x} - \bar{\mathbf{x}}_i)\,. \tag{3.3}$$

Although the Dirac delta function allows us to represent mixture distributions conveniently, any density function that contains it is no longer a true function, and this complicates the calculation of entropy and redundancy.

3.4 Entropy Calculation

3.4.1 Differential Entropy

Recall (from Section 2.2.1) that the differential entropy of a random variable $\mathbf{X}$ distributed according to the density function $p_{\mathbf{X}}(\cdot)$ is defined as

$$\begin{aligned} \mathbb{H}(\mathbf{X}) &:= \mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}}}[-\log(p_{\mathbf{X}}(\mathbf{X}))] \\ &= -\int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}) \log[p_{\mathbf{X}}(\mathbf{x})] d\mathbf{x}\,. \end{aligned} \tag{3.4}$$

Because the density function of a mixture distribution is defined in terms of the Dirac delta function, care must be taken when using this definition for a variable with such a distribution. If the density function of a mixture distribution were a true function, the density at each discrete point could be ignored when calculating the integral, since the set of discrete points is of Lebesgue measure zero [17]. However, this density function is not a true function, and ignoring the densities at the discrete points would result in a density function that is not properly normalised.

Therefore, to use this definition for a mixed random variable, either the value of $\log[\delta(\mathbf{0})]$ must be defined, or the integral of $\log[\delta(\cdot)]$ must be defined. Neither of these are defined, and differential entropy is not defined for random variables with mixture distributions [21].

3.4.2 Relative Entropy

Recall (from Section 2.2.1) that, for a random variable $\mathbf{X}$ with possible density functions $p_{\mathbf{X}}(\cdot)$ and $m_{\mathbf{X}}(\cdot)$, the relative entropy of $\mathbf{X}$ with respect to $m_{\mathbf{X}}(\cdot)$ is

$$\begin{aligned}\mathbb{H}[\mathbf{X} \mid\mid m_{\mathbf{X}}(\mathbf{X})] &:= \mathbb{E}_{\mathbf{X}\sim p_{\mathbf{X}}}\left[\log\left(\frac{p_{\mathbf{X}}(\mathbf{X})}{m_{\mathbf{X}}(\mathbf{X})}\right)\right] \\ &= \int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}) \log\left(\frac{p_{\mathbf{X}}(\mathbf{x})}{m_{\mathbf{X}}(\mathbf{x})}\right) d\mathbf{x}\,.\end{aligned} \tag{3.5}$$

For mixed random variables, the same type of difficulties seem to arise with relative entropy as with differential entropy. However, relative entropy can be defined using a Radon-Nikodym derivative [43]. Since we do not have the required background in measure theory to work with this definition, we do not explore relative entropy involving mixture distributions further.

3.5 Redundancy Calculation

In addition to the definition of redundancy given in Section 2.2.3, redundancy can be defined in terms of probability measures and a Radon-Nikodym derivative [21]. Since a probability measure is only concerned with probability values, not density values, probability measures can be used to represent discrete, continuous, or mixture distributions without the use of the Dirac delta function. Therefore, redundancy involving discrete, continuous, or mixture distributions is well defined [21].

3.6 Discussion

In this chapter, we introduced mixture distributions and discussed their relevance to the network architectures that we consider. We then briefly discussed how variables with mixture distributions interact with the definitions of differential entropy, relative entropy, and redundancy. Because we lack a solid background in measure theory, we do not explore the use of relative entropy further. However, existing work on redundancy estimators allows us to explore various forms of redundancy in neural networks. These estimators are discussed in detail in the following chapter.

Chapter 4

Redundancy Estimators

We discuss various existing redundancy estimators in detail. We combine some of these estimators into a new estimator, which is used throughout the rest of the study.

4.1 Chapter Overview

In order to estimate redundancy in neural networks, we require a redundancy estimator capable of handling discrete, continuous, and mixture distributions. We propose such an estimator in this chapter. This estimator, however, is heavily inspired by existing estimators. The existing estimators, which use a nearest-neighbours approach, are first discussed in detail in Sections 4.2 to 4.4 before our estimator is introduced in Section 4.5.

4.1.1 Estimator Overview

In Sections 4.2 and 4.3, respectively, we describe the KSG1 and KSG2 estimators, named after the authors of the paper [20] in which they are introduced. These estimators are intended to be used with continuous distributions. The KSG1 estimator has a smaller variance but larger systematic error compared to the KSG2 estimator, which makes the KSG2 estimator more suitable for high-dimensional applications [20]. The KSG2 estimator has an associated correction term, also described in Section 4.3, that reduces the number of sample points required to obtain an accurate estimate. Although it might be possible to adapt this correction term for use with the KSG1 estimator, we

do not explore this possibility, since we suspect that such a correction term would not significantly reduce the larger systematic error of the KSG1 estimator.

In Sections 4.4 and 4.5, respectively, we describe the KSG1m and KSG2m estimators. The KSG1m estimator is a modified version of the KSG1 estimator that can be used with discrete, continuous, and mixture distributions. Similarly, the KSG2m estimator is our modified version of the KSG2 estimator which can be used with discrete, continuous, and mixture distributions. The correction term of the KSG2m estimator, which we derive from that of the KSG2 estimator, reduces the number of sample points required to obtain an accurate estimate.

The implementation of the estimators is discussed in Section 4.6. In Section 4.7, we briefly review the contents of this chapter.

4.1.2 Notation

For all of the estimators in this chapter, let $\mathbf{X}$ be a random vector in $\mathbb{R}^d$ with components $X_1, \ldots, X_d$, and let $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(n)}$ be n independent samples of $\mathbf{X}$. Additionally, let $X_1^{(i)}, \ldots, X_d^{(i)}$ be the d components of $\mathbf{X}^{(i)}$. Denote the uniform norm by $||\cdot||$, such that $||\mathbf{X}|| = \max\{|X_1|, \ldots, |X_d|\}$, and let $\mathbf{X}^{(i,1)}, \ldots, \mathbf{X}^{(i,k)}$ be the k nearest neighbours of $\mathbf{X}^{(i)}$, where distance is measured using the metric induced by the norm $||\cdot||$. Let $X_1^{(i,j)}, \ldots, X_d^{(i,j)}$ be the d components of $\mathbf{X}^{(i,j)}$.

All of the estimators in this chapter take k as a parameter and produce an estimate of the redundancy $\mathbb{I}(\mathbf{X}) = \mathbb{I}(X_1, \ldots, X_d)$.

The symbols introduced to describe one estimator are reused to describe the other estimators. Such symbols refer to the same concepts, and, therefore, this reuse should not cause confusion.

4.2 KSG1 Estimator

The KSG1 estimator, introduced in [20], requires the construction of a "max-norm rectangle" (terminology introduced in [25]) around each sample point. The max-norm rectangle of $\mathbf{X}^{(j)}$ is axis-aligned and centered on $\mathbf{X}^{(j)}$. This rectangle is further constructed to be the smallest rectangle such that the k neighbours of the point are either inside or on an edge of the rectangle. Additionally, for the KSG1 estimator, the max-norm

rectangle is constrained to be a cube. Thus, the perpendicular distance from $\mathbf{X}^{(j)}$ to the side of the rectangle along dimension i is given by

$$B_i^{(j)} := \left|\left|\mathbf{X}^{(j)} - \mathbf{X}^{(j,k)}\right|\right| , \quad i = 1, \ldots, d\,; \tag{4.1}$$

see Figure 4.1 for an illustration. Note that, although this rectangle is a d-dimensional cube and requires its length along only one dimension to be defined, its length along each dimension is defined for consistency with the KSG2 estimator.

After the construction of the max-norm rectangle, a vector $\mathbf{C}^{(j)}$ is constructed for each $\mathbf{X}^{(j)}$ such that the ith component of $\mathbf{C}^{(j)}$ contains the number of points, excluding $\mathbf{X}^{(j)}$ itself, between the edges of the max-norm rectangle in dimension i only:

$$C_i^{(j)} := \sum_{\substack{m=1 \\ m \neq j}}^{n} \mathbb{T}\left(\left|X_i^{(j)} - X_i^{(m)}\right| < B_i^{(j)}\right), \tag{4.2}$$

where $\mathbb{T}(\cdot)$ is the indicator function; that is, $\mathbb{T}(x) = 1$ if the predicate x is true, and $\mathbb{T}(x) = 0$ otherwise.

Finally, the KSG1 estimator, given in [20], can be expressed as

$$\hat{\mathbb{I}}_{\mathrm{KSG1}}(\mathbf{X}) := (d-1)\psi(n) + \psi(k) - \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{d}\psi(C_i^{(j)} + 1)\,, \tag{4.3}$$

where $\psi(\cdot)$ is the digamma function.

4.3 KSG2 Estimator

4.3.1 Main Estimator

For the KSG2 estimator, also introduced in [20], the max-norm rectangle is not constrained to be a cube. Thus, the perpendicular distance from each point $\mathbf{X}^{(j)}$ to the side of its max-norm rectangle along dimension i is given by

$$B_i^{(j)} := \left|X_i^{(j)} - X_i^{(j,k)}\right| , \quad i = 1, \ldots, d\,. \tag{4.4}$$

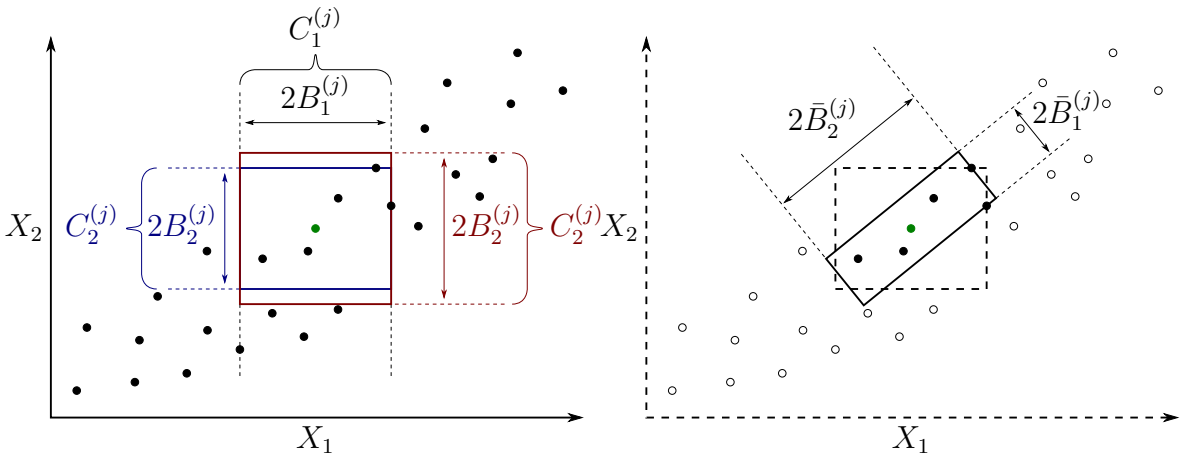

Figure 4.1: Construction of the max-norm rectangles (left) and LNC rectangle (right) for $d = 2$ and $k = 5$. $\mathbf{X}^{(j)}$ is shown in green in the center of the rectangles. **Left:** The max-norm rectangle for the KSG1 and KSG1m estimators is shown in red, while that for the KSG2 and KSG2m estimators is shown in blue. In this figure, $C_1^{(j)} = 7$ and $C_2^{(j)} = 11$ for KSG1, $C_1^{(j)} = 9$ and $C_2^{(j)} = 11$ for KSG1m, and $C_1^{(j)} = 9$ and $C_2^{(j)} = 9$ for KSG2 and KSG2m. **Right:** The LNC rectangle is shown with solid lines, while the KSG2 max-norm rectangle is shown with dashed lines.

To construct the vector $\mathbf{C}^{(j)}$, points coinciding with the edges of the max-norm rectangle in each dimension are also counted. Thus, the ith component of this vector is given by

$$C_i^{(j)} := \sum_{\substack{m=1 \\ m \neq j}}^{n} \mathbb{T}\left(\left|X_i^{(j)} - X_i^{(m)}\right| \leq B_i^{(j)}\right). \tag{4.5}$$

Note that, in contrast to the construction of $C_i^{(j)}$ for the KSG1 estimator, the inequality used here is not strict. The process of finding $B_i^{(j)}$ and $C_i^{(j)}$ is illustrated in Figure 4.1.

Finally, the KSG2 estimator, given in [20], can be expressed as

$$\hat{\mathbb{I}}_{\text{KSG2}}(\mathbf{X}) := (d-1)\left(\psi(n) - \frac{1}{k}\right) + \psi(k) - \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{d}\psi(C_i^{(j)}). \tag{4.6}$$

4.3.2 Local Non-Uniformity Correction

To obtain a redundancy estimate with a fixed accuracy, the KSG estimators require that the number n of sample points grows exponentially with the value of the true

redundancy [25]. A local non-uniformity correction (LNC) term $\mathbb{C}_{\text{LNC}}(\mathbf{X})$, introduced in [25], can be added to the KSG2 redundancy estimate to relax this requirement. The correction term takes a non-negative parameter $\alpha_{k,d}$.

The correction term requires the construction of a d-dimensional rectangle around each point $\mathbf{X}^{(j)}$. Like the max-norm rectangle, this LNC rectangle is centered on $\mathbf{X}^{(j)}$, with the k nearest neighbours $\mathbf{X}^{(j,1)}, \ldots, \mathbf{X}^{(j,k)}$ of $\mathbf{X}^{(j)}$ either inside the rectangle or on an edge of the rectangle. Unlike the max-norm rectangle, the LNC rectangle is not necessarily axis-aligned.

To construct the LNC rectangle, the authors of [25] perform a modified[1] principal component analysis on the k neighbours $\mathbf{X}^{(j,1)}, \ldots, \mathbf{X}^{(j,k)}$ and transform these points to the PCA space to obtain $\bar{\mathbf{X}}^{(j,1)}, \ldots, \bar{\mathbf{X}}^{(j,k)}$. They additionally transform $\mathbf{X}^{(j)}$ to the PCA space to obtain $\bar{\mathbf{X}}^{(j)}$. The perpendicular distance, in PCA space, from $\bar{\mathbf{X}}^{(j)}$ to the side of the rectangle along dimension i is then given by

$$\bar{B}_i^{(j)} := \max_m \left| \bar{X}_i^{(j)} - \bar{X}_i^{(j,m)} \right|, \quad i = 1, \ldots, d\,. \tag{4.7}$$

Figure 4.1 shows an example of an LNC rectangle.

The volume $V^{(j)}$ of the max-norm rectangle and the volume $\bar{V}^{(j)}$ of the LNC rectangle are given by

$$V^{(j)} := 2^d \prod_{i=1}^{d} B_i^{(j)}\,; \qquad\qquad \bar{V}^{(j)} := 2^d \prod_{i=1}^{d} \bar{B}_i^{(j)}\,. \tag{4.8}$$

These volumes are used to construct the term

$$T^{(j)} := \begin{cases} \log\left(\frac{\bar{V}^{(j)}}{V^{(j)}}\right) & \text{if } \frac{\bar{V}^{(j)}}{V^{(j)}} < \alpha_{k,d} \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

for each point $\mathbf{X}^{(j)}$, and these terms can be used to express the LNC term, given in [25], as

$$\mathbb{C}_{\text{LNC}}(\mathbf{X}) := -\frac{1}{n} \sum_{j=1}^{n} T^{(j)}\,. \tag{4.10}$$

[1] When calculating the covariance matrix for PCA, the authors use $\mathbf{X}^{(j)}$, rather than the sample mean, as the mean of the k points.

The corrected redundancy estimate is

$$\hat{\mathbb{I}}_{\mathrm{LNC}}(\mathbf{X}) := \hat{\mathbb{I}}_{\mathrm{KSG2}}(\mathbf{X}) + \mathbb{C}_{\mathrm{LNC}}(\mathbf{X})\,. \tag{4.11}$$

We refer to the combination of the KSG2 estimator and the LNC term as the LNC estimator.

The KSG2 estimator assumes that the probability density inside each max-norm rectangle is constant. The LNC term assumes that the density inside each LNC rectangle is constant. The use of PCA to find the LNC rectangles turns the local uniformity assumption of the KSG2 estimator into a local linearity assumption [25]. To avoid a large correction when the underlying distribution is uniform, the parameter $\alpha_{k,d}$ is used to detect violations of the local uniformity assumption. Thus, $\alpha_{k,d}$ should be sufficiently small such that $T^{(j)} = 0$ with high probability when the underlying distribution is uniform, but sufficiently large such that a meaningful correction is applied when the underlying distribution is not uniform.

4.3.3 Alpha Estimation

In the supplementary material of [25], the authors discuss a method for finding suitable values of $\alpha_{k,d}$. In addition to k and d, this method requires the number n_α of random rectangles to sample and the error probability ϵ_α for uniform distributions as parameters.

The method requires sampling n_α d-dimensional rectangles randomly. For the jth sampled rectangle, the authors sample k points $\mathbf{X}^{(j,1)}, \ldots, \mathbf{X}^{(j,k)}$ from the uniform distribution with that rectangle as support. With $V^{(j)}$ and $\bar{V}^{(j)}$ as in the definition of the LNC term, they set

$$\alpha^{(j)} := \frac{\bar{V}^{(j)}}{V^{(j)}}\,. \tag{4.12}$$

Finally, $\alpha_{k,d}$ is selected as the $\lceil \epsilon_\alpha n_\alpha \rceil$-th smallest $\alpha^{(j)}$ value.

The authors seem to suggest that they use the volume of the jth sampled rectangle instead of $V^{(j)}$. It is also not clear whether they use $\bar{V}^{(j)}$ or the volume of the smallest rectangle that contains the k points in PCA space (the difference being whether $\bar{\mathbf{X}}^{(j)}$ is the center of the rectangle). Since $V^{(j)}$ and $\bar{V}^{(j)}$ are used in the calculation of the LNC term, we assume that these are the correct values to use. In addition, we assume that $\mathbf{X}^{(j)}$ is the center of the jth sampled rectangle, since the definitions of $V^{(j)}$ and $\bar{V}^{(j)}$

require $\mathbf{X}^{(j)}$ and $\bar{\mathbf{X}}^{(j)}$. These assumptions are revisited in Section 4.6.

This method of finding $\alpha_{k,d}$ effectively samples n_α points from the distribution of the ratio $\frac{\bar{V}^{(j)}}{V^{(j)}}$ for uniform distributions. The value of $\alpha_{k,d}$ is then selected such that, for a uniform distribution, this ratio is smaller than $\alpha_{k,d}$ with probability ϵ_α. Therefore, the term $T^{(j)}$ will be zero with probability $(1 - \epsilon_\alpha)$ when the jth point is sampled from a uniform distribution.

4.4 KSG1m Estimator

The KSG estimators assume that the components of $\mathbf{X}$ are continuously distributed [20] and, as confirmed in Section 6.2.1, these estimators do not work well with mixture distributions. An estimator that does work for these distributions is introduced in [21]. This estimator, which we call the "KSG1m" estimator, is a combination of the KSG1 estimator and the continuous-discrete estimator introduced in [44].

The continuous-discrete estimator introduced in [44] can be used to estimate the redundancy between a discrete variable and a continuous variable. The KSG1m estimator includes a test to determine whether sample points are likely from the discrete component of the underlying distribution, and, if this is the case, techniques from the continuous-discrete estimator are employed to obtain a better redundancy estimate [21].

For the KSG1m estimator, the max-norm rectangles are constructed as for the KSG1 estimator:

$$B_i^{(j)} := \left|\left|\mathbf{X}^{(j)} - \mathbf{X}^{(j,k)}\right|\right| , \quad i = 1, \ldots, d\,. \tag{4.13}$$

The vector $\mathbf{C}^{(j)}$, however, is constructed as for the KSG2 estimator (the inequality is not strict):

$$C_i^{(j)} := \sum_{\substack{m=1 \\ m \neq j}}^{n} \mathbb{T}\left(\left|X_i^{(j)} - X_i^{(m)}\right| \leq B_i^{(j)}\right). \tag{4.14}$$

An additional value $K^{(j)}$ is computed for each point $\mathbf{X}^{(j)}$. If $B_i^{(j)} = 0$ for each dimension i, then $K^{(j)}$ is set to the number of points exactly equal to $\mathbf{X}^{(j)}$; otherwise, $K^{(j)} = k$.

Using these values, the KSG1m estimator, given in [21], can be expressed as

$$\hat{\mathbb{I}}_{\text{KSG1m}}(\mathbf{X}) := (d-1)\log(n) + \frac{1}{n}\sum_{j=1}^{n}\left[\psi(K^{(j)}) - \sum_{i=1}^{d}\log(C_i^{(j)}+1)\right]. \tag{4.15}$$

Although the authors claim that this estimator recovers the KSG1 estimator when used with a continuous distribution, this cannot be the case, since $\mathbf{C}^{(j)}$ is constructed differently for the two estimators. In addition, the KSG1m estimator uses $\log(n)$ and $\log(C_i^{(j)}+1)$ in the final expression, whereas the KSG1 estimator uses $\psi(N)$ and $\psi(C_i^{(j)}+1)$, respectively. This use of $\log(\cdot)$ instead of $\psi(\cdot)$ also disagrees with the definition of the continuous-discrete estimator in [44].

Because of these disagreements, we use $\psi(\cdot)$ instead of $\log(\cdot)$ in the implementation of this estimator. The difference in the construction of $\mathbf{C}^{(j)}$, however, is crucial to the correct performance of the estimator at discrete points. Therefore, the construction of $\mathbf{C}^{(j)}$ cannot be altered.

4.5 KSG2m Estimator

Although the KSG1m estimator works well with mixture distributions, the difference in the construction of $\mathbf{C}^{(j)}$ causes inaccuracies when the estimator is used with continuous distributions. In addition, the LNC term cannot be used with this estimator, since the LNC term is only defined for the KSG2 estimator. To address these shortcomings, we construct a new estimator, which we call the "KSG2m" estimator, and a modified LNC term, which we call the "LNCm" term.

4.5.1 Main Estimator

Our KSG2m estimator is a combination of the continuous-discrete estimator introduced in [44] and the KSG2 estimator. Like the KSG1m estimator, the KSG2m estimator includes a test to determine whether sample points are likely from the discrete component of the underlying distribution. If this is the case, estimation techniques from the continuous-discrete estimator are employed. If sample points are likely from the continuous component of the underlying distribution, the normal KSG2 estimation techniques

are used.

The max-norm rectangle and the vector $\mathbf{C}^{(j)}$ for each point $\mathbf{X}^{(j)}$ are constructed exactly as for the KSG2 estimator:

$$B_i^{(j)} := \left| X_i^{(j)} - X_i^{(j,k)} \right| , \quad i = 1, \ldots, d ; \tag{4.16}$$

$$C_i^{(j)} := \sum_{\substack{m=1 \\ m \neq j}}^{n} \mathbb{T} \left(\left| X_i^{(j)} - X_i^{(m)} \right| \leq B_i^{(j)} \right) . \tag{4.17}$$

For each point we introduce the value $K^{(j)}$. For both the KSG1m and KSG2m estimators, we interpret this value as the number of points, excluding $\mathbf{X}^{(j)}$ itself, either inside the max-norm rectangle of $\mathbf{X}^{(j)}$ or on the border of this rectangle. Therefore, this value can be calculated as for the KSG1m estimator: if $B_i^{(j)} = 0$ for *each* dimension i, then $K^{(j)}$ is set to the number of points exactly equal to $\mathbf{X}^{(j)}$; otherwise, $K^{(j)} = k$.

In order to recover the continuous-discrete estimator when our estimator is used with a mixture distribution, we also introduce the value $D^{(j)}$ for each point. If $B_i^{(j)} = 0$ for *any* dimension i, we set $D^{(j)} = 0$; otherwise, $D^{(j)} = 1$.

Using these values, our KSG2m estimator can be expressed as

$$\hat{\mathbb{I}}_{\mathrm{KSG2m}}(\mathbf{X}) := (d-1)\psi(n) + \frac{1}{n} \sum_{j=1}^{n} \left[\psi(K^{(j)}) - \frac{(d-1)D^{(j)}}{k} - \sum_{i=1}^{d} \psi(C_i^{(j)}) \right] . \tag{4.18}$$

4.5.2 Local Non-Uniformity Correction

Our LNCm term is very similar to the original LNC term. The LNCm term takes a non-negative parameter $\alpha_{k,d}$ and a parameter $\epsilon_\infty \in [0, 1]$.

Rather than a modified principal component analysis, we perform standard PCA on the $k+1$ points $\mathbf{X}^{(j)}, \mathbf{X}^{(j,1)}, \ldots, \mathbf{X}^{(j,k)}$ and transform these points to the PCA space to obtain $\bar{\mathbf{X}}^{(j)}, \bar{\mathbf{X}}^{(j,1)}, \ldots, \bar{\mathbf{X}}^{(j,k)}$. Using these points, the LNC rectangle is constructed as for the original LNC term:

$$\bar{B}_i^{(j)} := \max_m \left| \bar{X}_i^{(j)} - \bar{X}_i^{(j,m)} \right| , \quad i = 1, \ldots, d . \tag{4.19}$$

The volume $V^{(j)}$ of the max-norm rectangle and the volume $\bar{V}^{(j)}$ of the LNC rectangle

are given by

$$V^{(j)} := 2^d \prod_{i=1}^{d} B_i^{(j)}\,; \qquad\qquad \bar{V}^{(j)} := 2^d \prod_{i=1}^{d} \bar{B}_i^{(j)}\,, \tag{4.20}$$

and these volumes are used to construct the term

$$T^{(j)} := \begin{cases} \log\left(\frac{\bar{V}^{(j)}}{V^{(j)}}\right) & \text{if } V^{(j)} \neq 0 \text{ and } \bar{V}^{(j)} \neq 0 \text{ and } \frac{\bar{V}^{(j)}}{V^{(j)}} < \alpha_{k,d} \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

for each point $\mathbf{X}^{(j)}$.

The value of the final LNCm term depends on the $V^{(j)}$ and $\bar{V}^{(j)}$ values. If either of the following conditions hold, then our LNCm term is $\mathbb{C}_{\text{LNCm}}(\mathbf{X}) := \infty$:

- the fraction of points for which both $V^{(j)} \neq 0$ and $\bar{V}^{(j)} = 0$ is greater than ϵ_∞, when $\epsilon_\infty < 1$;
- $V^{(j)} \neq 0$ and $\bar{V}^{(j)} = 0$ for all points, when $\epsilon_\infty = 1$.

If none of these conditions are true, our LNCm term can be expressed as

$$\mathbb{C}_{\text{LNCm}}(\mathbf{X}) := -\frac{1}{n} \sum_{j=1}^{n} T^{(j)}\,. \tag{4.22}$$

The corrected redundancy estimate is

$$\hat{\mathbb{I}}_{\text{LNCm}}(\mathbf{X}) := \hat{\mathbb{I}}_{\text{KSG2m}}(\mathbf{X}) + \mathbb{C}_{\text{LNCm}}(\mathbf{X})\,. \tag{4.23}$$

We refer to the combination of the KSG2m estimator and the LNCm term as the LNCm estimator.

Unlike the LNC term, the LNCm term explicitly handles the case in which $V^{(j)} = 0$. In this case, $B_i^{(j)} = 0$ for at least one dimension i, indicating that the jth point is likely sampled from the discrete component of the underlying distribution. Since we want to add a correction to the redundancy estimate only when the jth point is sampled from the continuous component of the underlying distribution, we set $T^{(j)} = 0$ whenever $V^{(j)} = 0$.

The LNCm term also handles the case in which $V_{(j)} \neq 0$ and $\bar{V}^{(j)} = 0$. In this case,

there might be a functional relationship between the variables $X_1, \ldots, X_d$. If there is such a relationship, the true redundancy is infinite [25]. To avoid producing an infinite value when $V_{(j)} \neq 0$ and $\bar{V}^{(j)} = 0$ for only a few points, the parameter ϵ_∞ controls the fraction of points for which this must hold before an infinite value is produced.

4.5.3 Alpha Estimation

Again, our method of finding suitable $\alpha_{k,d}$ values for the LNCm term is very similar to the method discussed in the supplementary material of [25]. Our method also requires the number n_α of random rectangles to sample and the error probability ϵ_α for uniform distributions as parameters.

We sample n_α d-dimensional rectangles by sampling, for each dimension of each rectangle, the two endpoints of the rectangle in that dimension from a continuous uniform distribution with the interval $[-2^{31}, 2^{31} - 1]$ as support. For the jth rectangle, we set $\mathbf{X}^{(j)}$ to the center of the rectangle and sample k points $\mathbf{X}^{(j,1)}, \ldots, \mathbf{X}^{(j,k)}$ from the uniform distribution with that rectangle as support. With $V^{(j)}$ and $\bar{V}^{(j)}$ as in the definition of the LNCm term, we then set

$$\alpha^{(j)} := \frac{\bar{V}^{(j)}}{V^{(j)}} \tag{4.24}$$

for only those $\bar{n}_\alpha$ points with $V^{(j)} \neq 0$. We leave the remaining $\alpha^{(j)}$ values undefined. If $\epsilon_\alpha \in [0, 1)$, we select $\alpha_{k,d}$ as the $\lfloor \epsilon_\alpha \bar{n}_\alpha + 1 \rfloor$-th smallest defined $\alpha^{(j)}$ value. If $\epsilon_\alpha = 1$, we select $\alpha_{k,d}$ as the largest defined $\alpha^{(j)}$ value. (For this discussion, indexing starts at 1 — the 1st smallest value is the smallest value.)

As for the LNCm term, we explicitly handle the case in which $V^{(j)} = 0$. In this case, we simply leave $\alpha^{(j)}$ undefined. By selecting $\alpha_{k,d}$ from the defined $\alpha^{(j)}$ values, we ensure that, for a distribution with a uniform continuous component, the term $T^{(j)}$ will be zero with probability ϵ_α given that the jth point is sampled from the continuous component of the underlying distribution.

4.6 Implementation

We implement our KSG2m estimator and the associated LNCm term in Python. We use the `scikit-learn` library for k-nearest-neighbours queries, the `NumPy` library for

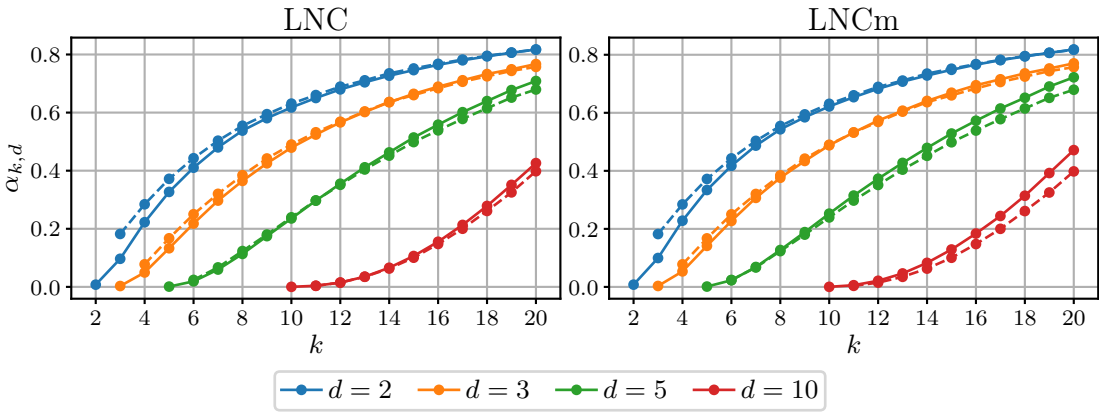

Figure 4.2: Values of $\alpha_{k,d}$ estimated with $n_\alpha = 500\,000$ and $\epsilon_\alpha = 0.005$ for the LNC and LNCm terms. The values given in the supplementary material of [25] are shown with dashed lines.

numerical computation, the `SciPy` library for computation of the digamma function, and the `Numba` library for parallel computation. Since the KSG1, KSG2, and KSG1m estimators are similar to our KSG2m estimator, we also implement these estimators. We implement the LNC term for similar reasons.

Note that an implementation of the KSG2 estimator with the LNC term is provided[2] by the authors of [25]. This implementation does not include the alpha estimation procedure. We evaluate this implementation in Section 6.3. We were not able to find implementations of the other estimators.

The authors of [25] provide values of $\alpha_{k,d}$ calculated with $n_\alpha = 500\,000$ and $\epsilon_\alpha = 0.005$ for the LNC term. To verify the correctness of our alpha estimation implementation, we generate $\alpha_{k,d}$ values with the same parameters for both the LNC and LNCm terms. The calculated values are shown in Figure 4.2.

As can be seen in Figure 4.2, we are not able to reproduce the exact $\alpha_{k,d}$ values given in the supplementary material of [25]. This might be caused by the assumptions about the alpha estimation procedure discussed in Section 4.3.3. The LNC estimator with our $\alpha_{k,d}$ values is one of the most accurate estimators that we evaluate, as reported in Section 6.2.1. This fact, in combination with the similarities between the values that we generate and those provided in [25], strongly suggests that we are sampling $\alpha_{k,d}$ from the correct distribution.

[2] `https://github.com/BiuBiuBiLL/MIE`

Table 4.1: Summary of the redundancy estimators that we evaluate. The non-uniformity correction, where available, reduces the number of sample points required to obtain an accurate estimate.

Estimator	Supports Mixture Distributions	Non-Uniformity Correction
KSG1 [20]	No	None
KSG2 [20]	No	Available [25] (see LNC estimator)
LNC [20], [25]	No	Included
KSG1m [21]	Yes	None
KSG2m	Yes	Available (see LNCm estimator)
LNCm	Yes	Included

In the following chapters, as in [25], $\alpha_{k,d}$ values estimated with $n_\alpha = 500\,000$ and $\epsilon_\alpha = 0.005$ are used in the calculation of the LNC and LNCm terms. In addition, $\epsilon_\infty = 0.005$ is used for the LNCm term.

4.7 Discussion

In this chapter, we described the KSG1, KSG2, and KSG1m estimators, as well as the LNC term for the KSG2 estimator, in detail. Using these as inspiration, we derived the KSG2m estimator and the LNCm term. We implemented the estimators and correction terms listed in Table 4.1 in Python. In the following chapter, we generate the synthetic data that will be used to evaluate these estimators.

Chapter 5

Synthetic Dataset

We generate synthetic data to evaluate the behaviour of the redundancy estimators. We calculate the true redundancy between several features of the synthetic dataset.

5.1 Chapter Overview

In this chapter, we generate synthetic data to evaluate the behaviour of the redundancy estimators described in Chapter 4. We also calculate the true differential entropy of several of the synthetic features and the true redundancy between several features. The results of this are used in Chapter 6 to evaluate the accuracy of the estimators.

The structure of the synthetic dataset is described in Section 5.2. In Section 5.3, we discuss the calculations required to obtain the true redundancy between several of the features. In Section 5.4, we describe the methods used to sample the synthetic data. We summarise the contents of this chapter in Section 5.5.

5.2 Structure

To ensure that the estimators are evaluated on different types of distributions, we include features with uniform, unimodal, bimodal, discrete, and mixture distributions. In order to introduce dependencies among the features, we combine several of these features to produce new features. A hidden feature is also introduced into the combinations to eliminate the possibility of infinite redundancy values.

Our synthetic dataset consists of 6 continuous features, 1 discrete feature, and 1 feature with a mixture distribution. We also generate a continuous hidden feature, which is not included in the synthetic dataset. The continuous features $X_1, \ldots, X_6$ and the hidden feature X_H are distributed as follows:

$$X_H \sim \text{Uniform}(a_H, b_H)\,; \tag{5.1}$$

$$X_1 \sim \text{Uniform}(a_1, b_1)\,; \tag{5.2}$$

$$X_2 \sim \text{Unimodal-Exponential}(\mu_2, s_2)\,; \tag{5.3}$$

$$X_3 \sim \text{Bimodal-Exponential}(a_3, b_3, s_3)\,; \tag{5.4}$$

$$X_4 = X_1 + X_2 + X_H\,; \tag{5.5}$$

$$X_5 = X_1 + X_3 + X_H\,; \tag{5.6}$$

$$X_6 = X_2 + X_3 + X_H\,, \tag{5.7}$$

where "unimodal-exponential" refers to a bi-exponential (Laplace) distribution. The probability density functions of X_1, X_2, and X_3 are, respectively,

$$p_1(x) = \frac{1}{b_1 - a_1}\,, \qquad a_1 \le x \le b_1\,; \tag{5.8}$$

$$p_2(x) = \frac{\exp\left(-\frac{|x-\mu_2|}{s_2}\right)}{2s_2}\,; \tag{5.9}$$

$$p_3(x) = \frac{\exp\left(-\frac{x-a_3}{s_3}\right) + \exp\left(\frac{x-b_3}{s_3}\right)}{2s_3\left[1 - \exp\left(\frac{a_3-b_3}{s_3}\right)\right]}\,, \qquad a_3 \le x \le b_3\,, \tag{5.10}$$

where the functions are zero outside of the specified intervals [45]. These distributions are chosen for the ease with which their density functions can be integrated. We use $a_H = -5$, $b_H = 5$, $a_1 = -25$, $b_1 = 25$, $\mu_2 = 0$, $s_2 = 12.5$, $a_3 = -50$, $b_3 = 50$, and $s_3 = 10$.

The discrete feature X_7 and mixed feature X_8 are jointly distributed. (X_7, X_8) has value $(\mu_{7,1}, \mu_{8,1})$ with probability s_7, value $(\mu_{7,2}, \mu_{8,2})$ with probability s_8, and value $(\mu_{7,2}, X_{8U})$ with probability $1 - s_7 - s_8$, where X_{8U} has a continuous uniform distribution

with support $[a_8, b_8]$. The joint density function of these features is

$$\begin{aligned} p_{7,8}(x_7, x_8) = s_7\delta(x_7 - \mu_{7,1})\delta(x_8 - \mu_{8,1}) + \delta(x_7 - \mu_{7,2}) \\ \cdot \left[s_8\delta(x_8 - \mu_{8,2}) + (1 - s_7 - s_8)p_U(x_8; a_8, b_8)\right], \end{aligned} \tag{5.11}$$

where $p_U(\cdot; a_8, b_8)$ is the density function of the uniform distribution with support $[a_8, b_8]$. We use $\mu_{7,1} = 0$, $\mu_{7,2} = 1$, $s_7 = 0.25$, $\mu_{8,1} = -1$, $\mu_{8,2} = 0$, $a_8 = 0$, $b_8 = 50$, and $s_8 = 0.25$.

5.3 Calculations

5.3.1 Density

In order to calculate the redundancy in a set of continuous features, we require the differential entropy of that set of features, as well as the differential entropy of each feature in the set. These, in turn, require the joint density of the set of features, as well as the density of each individual feature. For the six continuous features, the joint densities that we calculate are

$$p_{4,1}(x_4, x_1) = p_1(x_1) \int_{-\infty}^{\infty} p_2(x_4 - x_1 - y)p_H(y)dy\,; \tag{5.12}$$

$$p_{4,2}(x_4, x_2) = p_2(x_2) \int_{-\infty}^{\infty} p_1(x_4 - x_2 - y)p_H(y)dy\,; \tag{5.13}$$

$$p_{5,1}(x_5, x_1) = p_1(x_1) \int_{-\infty}^{\infty} p_3(x_5 - x_1 - y)p_H(y)dy\,; \tag{5.14}$$

$$p_{5,3}(x_5, x_3) = p_3(x_3) \int_{-\infty}^{\infty} p_1(x_5 - x_3 - y)p_H(y)dy\,; \tag{5.15}$$

$$p_{6,2}(x_6, x_2) = p_2(x_2) \int_{-\infty}^{\infty} p_3(x_6 - x_2 - y)p_H(y)dy\,; \tag{5.16}$$

$$p_{6,3}(x_6, x_3) = p_3(x_3) \int_{-\infty}^{\infty} p_2(x_6 - x_3 - y)p_H(y)dy\,. \tag{5.17}$$

These joint densities can be used to calculate the densities of X_4, X_5, and X_6:

$$p_4(x_4) = \int_{-\infty}^{\infty} p_{4,1}(x_4, x_1)dx_1 = \int_{-\infty}^{\infty} p_{4,2}(x_4, x_2)dx_2\,; \tag{5.18}$$

$$p_5(x_5) = \int_{-\infty}^{\infty} p_{5,1}(x_5, x_1)dx_1 = \int_{-\infty}^{\infty} p_{5,3}(x_5, x_3)dx_3\,; \tag{5.19}$$

$$p_6(x_6) = \int_{-\infty}^{\infty} p_{6,2}(x_6, x_2)dx_2 = \int_{-\infty}^{\infty} p_{6,3}(x_6, x_3)dx_3\,. \tag{5.20}$$

We calculate all of these density functions by hand.

The expressions for the joint density functions given in Equations 5.12 to 5.17 are easily derived. As an example, consider $p_{4,2}(\cdot,\cdot)$. Since X_1, X_2, and X_H are independent, the joint density of these three variables is given by

$$p_{1,2,H}(x_1, x_2, x_H) = p_1(x_1)p_2(x_2)p_H(x_H)\,. \tag{5.21}$$

The joint density function of X_4, X_2, and X_H is formed by substituting $x_4 - x_2 - x_H$ for x_1, since the sum of x_1, x_2, and x_H is x_4:

$$p_{4,2,H}(x_4, x_2, x_H) = p_1(x_4 - x_2 - x_H)p_2(x_2)p_H(x_H)\,. \tag{5.22}$$

Finally, $p_{4,2}(\cdot,\cdot)$ is obtained by integrating from $-\infty$ to ∞ with respect to x_H.

As an example of the calculations required to evaluate the integrals in Equations 5.12 to 5.17, consider the integral in the expression for $p_{4,2}(\cdot,\cdot)$:

$$\int_{-\infty}^{\infty} p_1(x_4 - x_2 - y)p_H(y)dy = \frac{1}{(b_1 - a_1)(b_H - a_H)} \int_{y_1}^{y_2} dy\,. \tag{5.23}$$

The conditions $a_1 \leq x_4 - x_2 - y \leq b_1$ and $a_H \leq y \leq b_H$ yield two possibilities for the lower limit y_1 and two possibilities for the upper limit y_2:

$$y_1 = \begin{cases} a_H & \text{if } a_H \geq x_4 - x_2 - b_1 \\ x_4 - x_2 - b_1 & \text{otherwise} \end{cases}; \tag{5.24}$$

$$y_2 = \begin{cases} b_H & \text{if } b_H \leq x_4 - x_2 - a_1 \\ x_4 - x_2 - a_1 & \text{otherwise} \end{cases}. \tag{5.25}$$

With the additional constraint $y_1 \leq y_2$, the integral evaluates to

$$\int_{-\infty}^{\infty} p_1(x_4 - x_2 - y) p_H(y) dy$$

$$= \frac{1}{(b_1 - a_1)(b_H - a_H)} \begin{cases} & \text{if } x_4 - x_2 \geq a_1 + a_H \\ x_4 - x_2 - a_1 - a_H & \text{and } x_4 - x_2 \leq a_1 + b_H \\ & \text{and } x_4 - x_2 \leq b_1 + a_H \\ & \text{if } x_4 - x_2 \geq b_1 + a_H \\ b_1 - a_1 & \text{and } x_4 - x_2 \leq a_1 + b_H \\ & \text{if } x_4 - x_2 \geq a_1 + b_H \\ b_H - a_H & \text{and } x_4 - x_2 \leq b_1 + a_H \\ & \text{if } x_4 - x_2 \geq a_1 + b_H \\ -x_4 + x_2 + b_1 + b_H & \text{and } x_4 - x_2 \geq b_1 + a_H \\ & \text{and } x_4 - x_2 \leq b_1 + b_H \\ 0 & \text{otherwise} \end{cases} . \tag{5.26}$$

The density functions of the six continuous features are shown in Figure 5.1.

To calculate the redundancy between X_7 and X_8, we require the density functions of these features. These densities can easily be calculated from the joint density $p_{7,8}(\cdot, \cdot)$:

$$p_7(x_7) = \int_{-\infty}^{\infty} p_{7,8}(x_7, x_8) dx_8 = s_7 \delta(x_7 - \mu_{7,1}) + (1 - s_7) \delta(x_7 - \mu_{7,2}) \,; \tag{5.27}$$

$$\begin{aligned} p_8(x_8) = \int_{-\infty}^{\infty} p_{7,8}(x_7, x_8) dx_7 &= s_7 \delta(x_8 - \mu_{8,1}) + s_8 \delta(x_8 - \mu_{8,2}) \\ &\quad + (1 - s_7 - s_8) p_U(x_8; a_8, b_8) \,. \end{aligned} \tag{5.28}$$

5.3.2 Redundancy

We use `SciPy`'s numerical integration functions to calculate the differential entropy of each continuous feature and the differential entropy of each pair of features for which we calculated the joint density. The calculated entropy values, with estimated errors, are shown in Table 5.1. We use these values to calculate the redundancy between the features shown in Table 5.2.

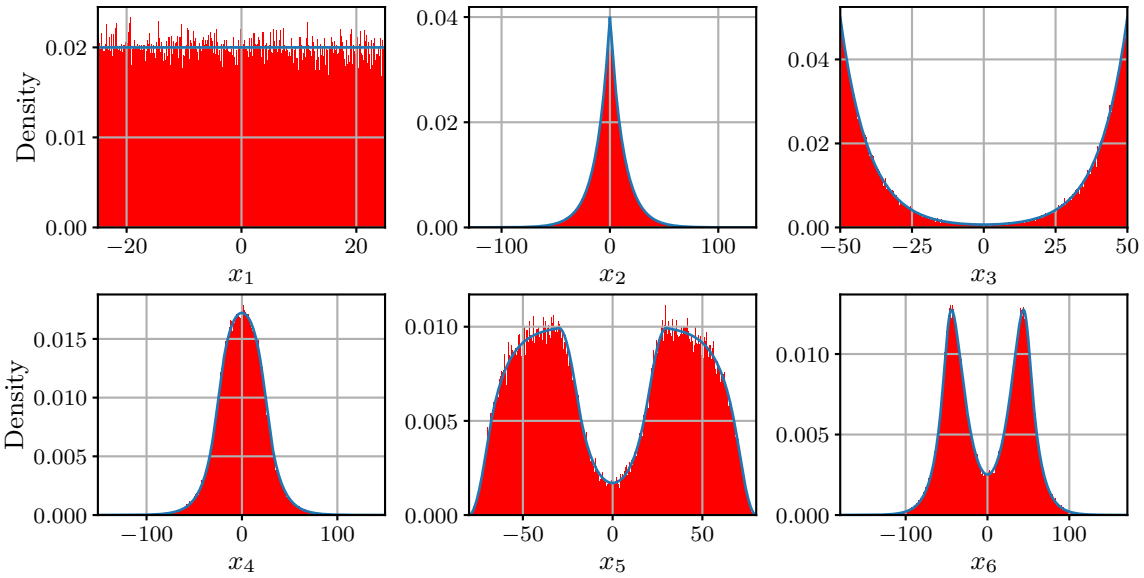

Figure 5.1: Probability density functions (blue) and histograms of sampled data (red) for features $X_1, \ldots, X_6$ of the synthetic data.

Table 5.1: Differential entropy of several features and pairs of features of the synthetic data, calculated using numerical integration.

Entropy	Value	Estimated Error
$\mathbb{H}(X_1)$	3.912023	1.656878×10^{-8}
$\mathbb{H}(X_2)$	4.218876	3.698920×10^{-11}
$\mathbb{H}(X_3)$	3.974654	1.284623×10^{-8}
$\mathbb{H}(X_4)$	4.545364	3.161223×10^{-8}
$\mathbb{H}(X_5)$	4.905168	5.427215×10^{-8}
$\mathbb{H}(X_6)$	4.993611	4.347815×10^{-8}
$\mathbb{H}(X_1, X_4)$	8.153664	3.453910×10^{-8}
$\mathbb{H}(X_1, X_5)$	8.121989	5.350940×10^{-7}
$\mathbb{H}(X_2, X_4)$	8.172874	4.615246×10^{-7}
$\mathbb{H}(X_2, X_6)$	8.419736	3.760388×10^{-7}
$\mathbb{H}(X_3, X_5)$	7.986680	2.577597×10^{-6}
$\mathbb{H}(X_3, X_6)$	8.216295	4.284244×10^{-8}

Table 5.2: Redundancy in several pairs of features of the synthetic data. The redundancy in each of the first six pairs is calculated from the differential-entropy values. The redundancy in each of the last six pairs can be derived from the fact that the features in each of these pairs are independent.

Redundancy	Value	Estimated Error
$\mathbb{I}(X_1, X_4)$	0.3037226	8.272011×10^{-8}
$\mathbb{I}(X_1, X_5)$	0.6952028	6.059349×10^{-7}
$\mathbb{I}(X_2, X_4)$	0.5913658	4.931738×10^{-7}
$\mathbb{I}(X_2, X_6)$	0.7927515	4.195540×10^{-7}
$\mathbb{I}(X_3, X_5)$	0.8931424	2.644715×10^{-6}
$\mathbb{I}(X_3, X_6)$	0.7519704	9.916683×10^{-8}
$\mathbb{I}(X_1, X_2)$	0.0000000	0.000000
$\mathbb{I}(X_1, X_3)$	0.0000000	0.000000
$\mathbb{I}(X_1, X_6)$	0.0000000	0.000000
$\mathbb{I}(X_2, X_3)$	0.0000000	0.000000
$\mathbb{I}(X_2, X_5)$	0.0000000	0.000000
$\mathbb{I}(X_3, X_4)$	0.0000000	0.000000

To calculate the redundancy between X_7 and X_8 without resorting to measure theory, we use a strategy often employed in physics and engineering. Rather than working with the Dirac delta function directly, we approximate the Dirac delta function by

$$\delta_\epsilon(x) = \begin{cases} \frac{1}{2\epsilon} & \text{if } -\epsilon \leq x \leq \epsilon \\ 0 & \text{otherwise} \end{cases}, \tag{5.29}$$

where ϵ is a positive real number. This function has properties similar to that of the Dirac delta function:

$$\delta_\epsilon(x) = 0 \quad \text{for all } x \notin [-\epsilon, \epsilon] \tag{5.30}$$

$$\int_a^b \delta_\epsilon(x)dx = 1 \quad \text{if } [-\epsilon, \epsilon] \subseteq (a, b)\,. \tag{5.31}$$

In some sense, $\delta_\epsilon(\cdot)$ approaches $\delta(\cdot)$ as $\epsilon \to 0^+$. After using this approximation to calculate the approximate redundancy $I_\epsilon(X_7, X_8)$, we can obtain the true redundancy by evaluating the limit of $I_\epsilon(X_7, X_8)$ as $\epsilon \to 0^+$ [41], [42].

Given that $\mu_{7,1} \neq \mu_{7,2}$, $\mu_{8,1} \neq \mu_{8,2}$, $\mu_{8,1} \notin [a_8, b_8]$, and $\mu_{8,2} = a_8$, the approximate

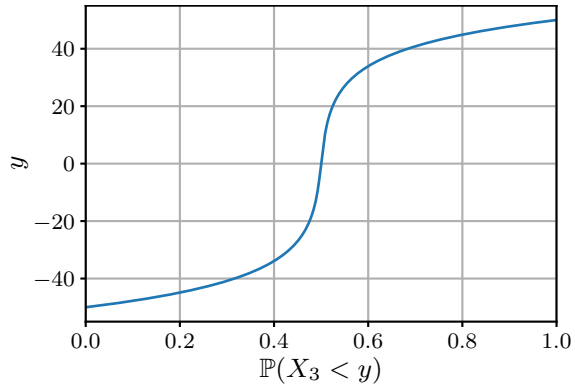

Figure 5.2: Inverse cumulative distribution function of feature X_3 of the synthetic data.

redundancy between X_7 and X_8, for sufficiently small ϵ, is

$$\begin{aligned}\mathbb{I}_\epsilon(X_7, X_8) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left[p_{7,8}(x_7, x_8)\log\left(\frac{p_{7,8}(x_7, x_8)}{p_7(x_7)p_8(x_8)}\right)\right]_{\delta(\cdot)=\delta_\epsilon(\cdot)} dx_7 dx_8 \\ &= -s_7\log(s_7) - (1-s_7)\log(1-s_7)\,.\end{aligned} \tag{5.32}$$

Therefore,

$$\mathbb{I}(X_7, X_8) = \lim_{\epsilon\to 0^+} I_\epsilon(X_7, X_8) = -s_7\log(s_7) - (1-s_7)\log(1-s_7) \approx 0.562335\,, \tag{5.33}$$

which is also, in this case, the discrete entropy of X_7.

5.4 Sampling

We sample X_1, X_2, X_7, and X_8 using `NumPy`'s random number generators; X_3 is sampled using inverse-transform sampling. To employ inverse-transform sampling, we require the inverse of the cumulative distribution function of X_3. The cumulative distribution function of X_3 is

$$c_3(x) = \int_{-\infty}^{x} p_3(y)dy\,. \tag{5.34}$$

If $c_3^{-1}(\cdot)$ is the inverse of $c_3(\cdot)$, and X_U is uniformly distributed between 0 and 1, then $c_3^{-1}(X_U)$ is distributed according to the density function $p_3(\cdot)$ [46]. Thus, we sample X_U and calculate $X_3 = c_3^{-1}(X_U)$. Figure 5.2 shows $c_3^{-1}(\cdot)$.

The synthetic dataset consists of 100 000 sampled points. The distributions of the sampled features are shown in Figure 5.1.

5.5 Discussion

In this chapter, we generated synthetic data and calculated the redundancy between several pairs of features of the synthetic data. We use this data to evaluate the redundancy estimators in the following chapter.

Chapter 6

Estimator Performance

We evaluate and compare the redundancy estimators on the synthetic data. We use these results to select the estimator that is best suited to our task.

6.1 Chapter Overview

In order to select the estimator and parameter values best suited to our task, we evaluate the redundancy estimators described in Chapter 4 on features from the synthetic dataset generated in Chapter 5. The results of this evaluation are also used to verify that the estimators do not show any unexpected behaviour. In Section 6.2, we evaluate the accuracy of the estimators as the parameter k and the number n of sample points vary. We evaluate the computational performance of the LNC estimator in Section 6.3, and we discuss the results of the evaluations in Section 6.4.

6.2 Accuracy

In this section, we evaluate the behaviour of the redundancy estimators on the synthetic dataset as the parameter k varies (Section 6.2.1) and as the number n of sample points varies (Section 6.2.2). The estimators are evaluated for all pairs of continuous features $X_1, \ldots, X_6$, and for the pair consisting of the discrete feature X_7 and the mixed feature X_8. For the pairs with known redundancy values (calculated in Section 5.3), we compare the estimated values to the true values. The estimates for the pairs with un-

known redundancy values are simply used to confirm that the estimators do not behave unexpectedly.

Based on the description of the redundancy estimators in Chapter 4, we expect the KSG2 and KSG2m estimators to behave similarly for the continuous features. We also expect the LNC and LNCm estimators to behave similarly for these features. For the discrete and mixed features, we expect the KSG1 and KSG2 estimators to behave poorly, and we expect the LNC term to encounter numerical problems. Additionally, for these features, we expect the KSG1m, KSG2m, and LNCm estimators to behave similarly.

The parameter choices for the LNC and LNCm terms are as discussed in Section 4.6.

6.2.1 Variation in Number of Neighbours

We first evaluate the stability of the estimators as the parameter k varies. To do this, we randomly select n points from the synthetic dataset, with $n = 1\,000$ and $n = 10\,000$, and, using these points, we estimate the redundancy in each pair of continuous features, as well as the pair (X_7, X_8), with values of k ranging from 2 to 20, inclusive. This is repeated for each of 10 different initialisation seeds of the random number generator (RNG). The estimates for the continuous features are shown in Figures 6.1 and 6.2 for $n = 1\,000$. The estimates for the pair (X_7, X_8) are shown in Figure 6.3.

Figures 6.1 to 6.3 confirm that the estimators behave mostly as expected. For the continuous features, the KSG2, KSG2m, LNC, and LNCm estimators behave as expected. For the discrete and mixed features, the KSG1 and KSG2 estimators are very inaccurate, and the LNC term is poorly defined for these features — divisions by zero and divisions of zero by zero are frequently encountered. For these features, the KSG2m and LNCm estimators also behave as expected, but the large difference between the KSG1m and LNCm estimators for $n = 1\,000$ is not expected.

This poor behaviour of the KSG1m estimator with larger values of k and $n = 1\,000$, shown in Figure 6.3, does not occur when the number of sample points is increased to $n = 10\,000$. The behaviour can also be rectified for $n = 1\,000$ by scaling the discrete feature X_7 with a sufficiently large factor.

The effects of the difference in the construction of the vector $\mathbf{C}^{(j)}$, discussed in Section 4.4, seem to appear in Figures 6.1 and 6.2. These figures show that the KSG1m estimates are consistently higher than the KSG1 estimates. Given that $C_i^{(j)}$ for the

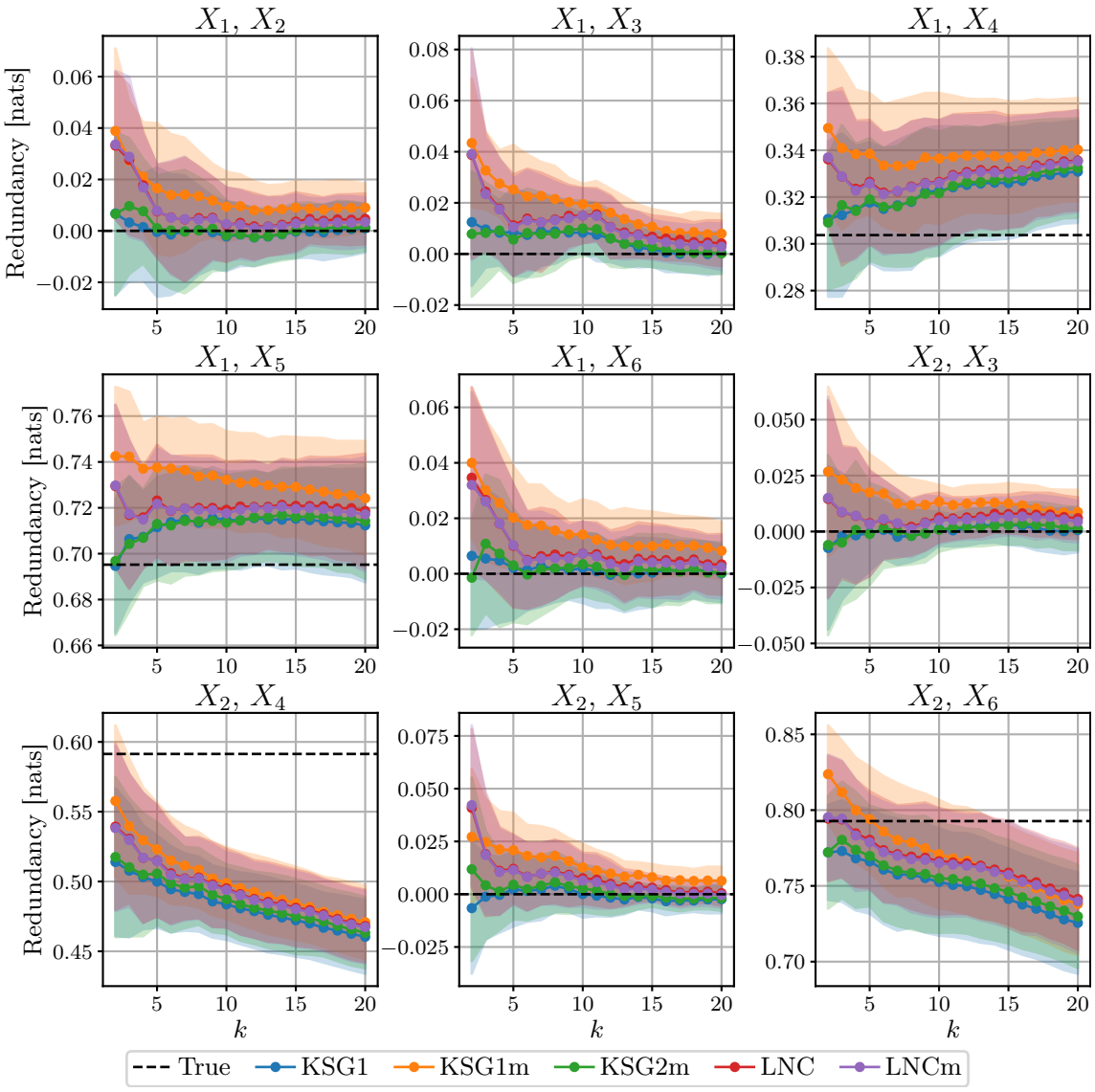

Figure 6.1: Redundancy in pairs of continuous features of the synthetic data, estimated with various values of k on randomly selected subsets of 1 000 points. The true redundancy is shown with dashed lines. Shaded areas indicate the standard deviation across RNG seeds. Results of the KSG2 estimator are identical to those of the KSG2m estimator.

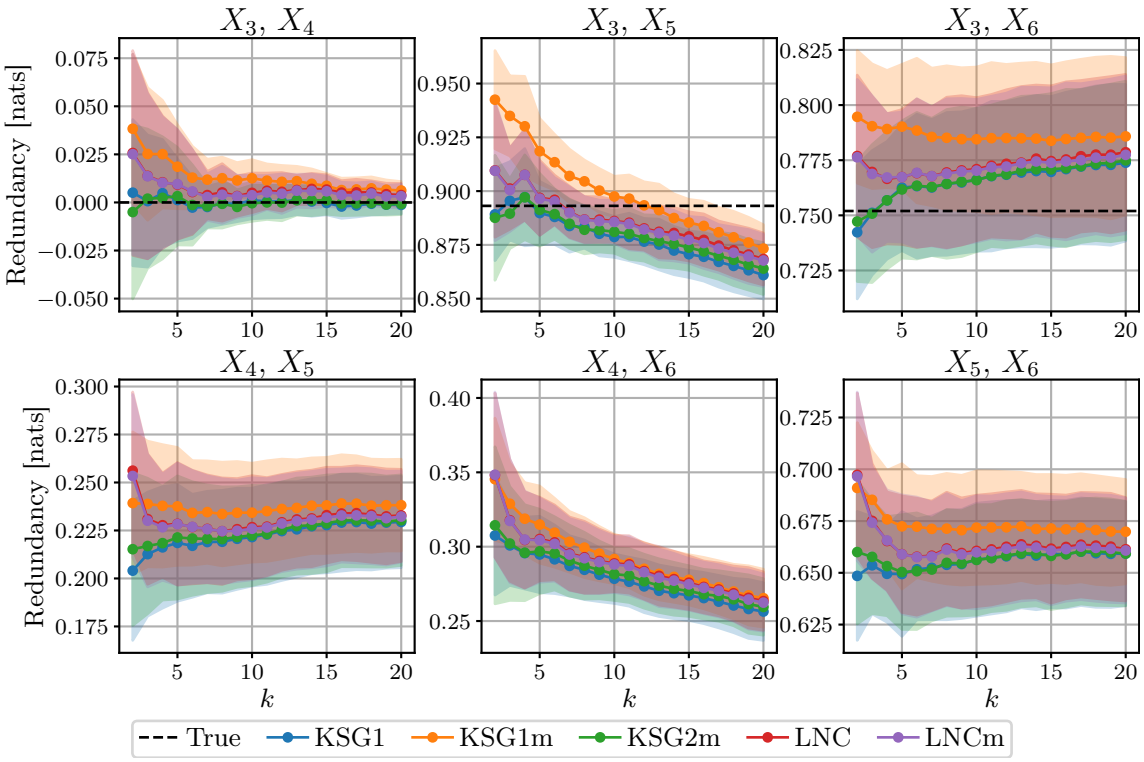

Figure 6.2: Redundancy in pairs of continuous features of the synthetic data, estimated as for Figure 6.1 for additional pairs of features. The true redundancy values for the pairs in the bottom row are not available.

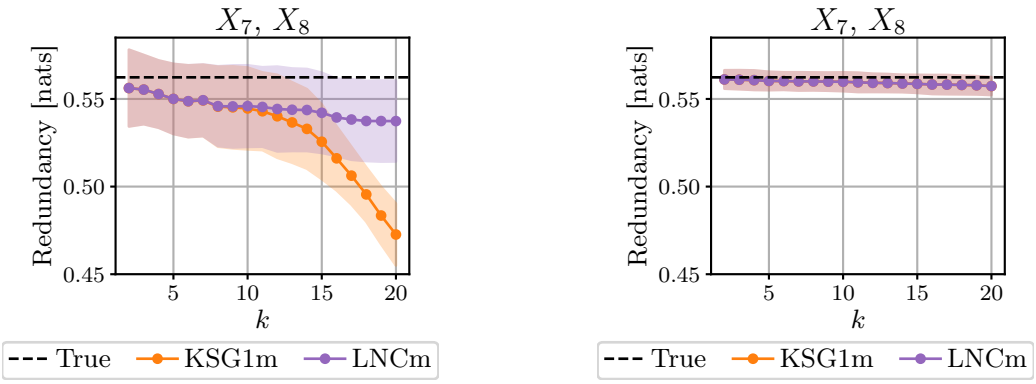

Figure 6.3: Redundancy between the discrete feature and mixed feature of the synthetic data, estimated with various values of k on randomly selected subsets of 1 000 (left) and 10 000 (right) points. The true redundancy is shown with dashed lines. Shaded areas indicate the standard deviation across RNG seeds. Results of the KSG1, KSG2, and LNC estimators are not shown. Results of the KSG2m estimator are identical to those of the LNCm estimator. **Right:** Results of the KSG1m estimator are identical to those of the LNCm estimator.

KSG1m estimator is always equal to or greater than that for the KSG1 estimator, this is expected.

To select the most suitable value of k, we calculate the root-mean-squared (RMS) error of the estimates using the true redundancy values calculated for the synthetic data. The error values for $n = 1\,000$ are shown in Figure 6.4. Those for $n = 10\,000$ are shown in Figure 6.5.

Figures 6.4 and 6.5 show that, for the continuous features, the LNCm estimator at $k = 5$ is most accurate for $n = 1\,000$, while the KSG1m estimator at $k = 6$ is most accurate for $n = 10\,000$, on average. For $k \geq 4$ and $n = 10\,000$, however, the differences in the accuracy of the KSG1m and LNCm estimators are negligible for our purposes.

These figures also show that, for the discrete and mixed features, the KSG1m and LNCm estimators are equally accurate for $k \leq 8$. For $k > 8$ and $n = 1\,000$, however, the LNCm estimator is more accurate.

6.2.2 Variation in Number of Points

Next, we evaluate the stability of the estimators as the number n of data points varies. To do this, we randomly select n points from the synthetic dataset with values of n

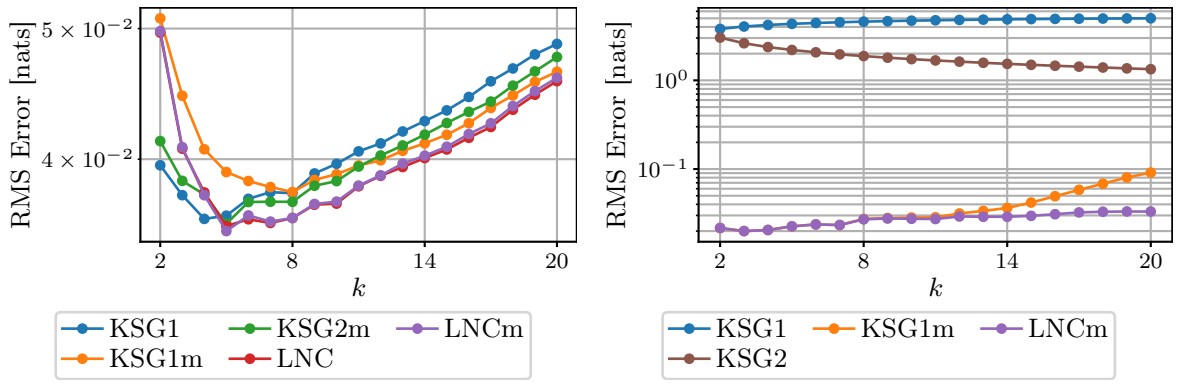

Figure 6.4: Root-mean-squared errors of the redundancy estimates on the synthetic data for $n = 1\,000$. **Left:** RMS errors for the continuous features. Errors of the KSG2 estimator are identical to those of the KSG2m estimator. **Right:** RMS errors for features X_7 and X_8. Errors of the LNC estimator are not shown. Errors of the KSG2m estimator are identical to those of the LNCm estimator.

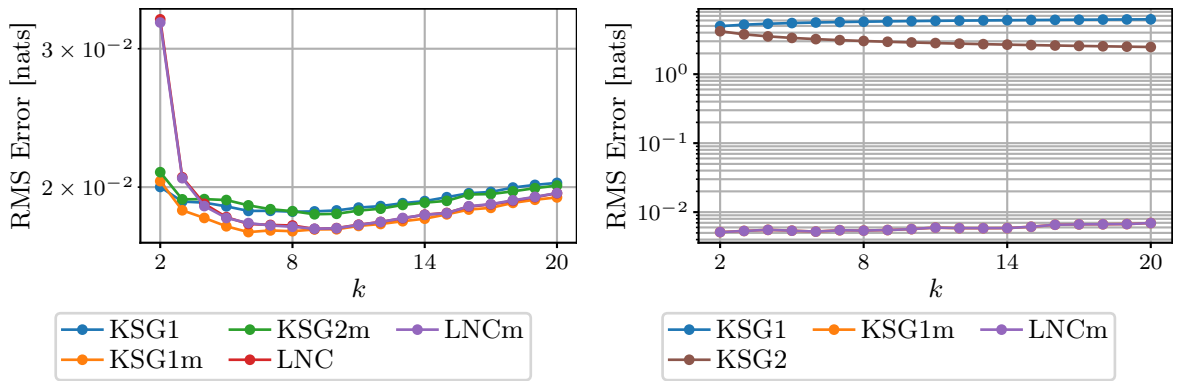

Figure 6.5: Root-mean-squared errors of the redundancy estimates on the synthetic data for $n = 10\,000$. **Left:** RMS errors for the continuous features. Errors of the KSG2 estimator are identical to those of the KSG2m estimator. **Right:** RMS errors for features X_7 and X_8. Errors of the LNC estimator are not shown. Errors of the KSG1m and KSG2m estimators are identical to those of the LNCm estimator.

linearly spaced between 1 000 and 10 000, inclusive. Using these points, we estimate the redundancy in each pair of continuous features, as well as the pair (X_7, X_8), with $k = 5$. This is repeated for each of 10 different initialisation seeds of the RNG. The estimates are shown in Figures 6.6, 6.7, and 6.8.

Figures 6.6 to 6.8 again confirm that the estimators behave as expected.

6.3 Computational Performance

The authors of [25] provide an implementation of the KSG2 estimator and the LNC term. Their implementation primarily uses standard Python features to perform calculations, and most calculations are performed sequentially in a single thread. As mentioned in Section 4.6, our implementation relies on various libraries for efficient computation.

To evaluate the computational performance of our estimator implementation, we measure the time required to calculate a single KSG2 redundancy estimate and the associated LNC term during execution of the experiment described in Section 6.2.2. This is measured for our implementation and for the reference implementation provided by the authors of [25]. These measurements are shown in Figure 6.9.

The measurements show that the various libraries used by our implementation significantly improve the performance of the estimators. The `Numba` library, which is used in our implementation, performs just-in-time (JIT) compilation to improve performance. An effect of this, however, is that the first execution of the estimator requires significantly more time than subsequent executions. This effect manifests as the large standard deviation at $n = 1\,000$ in Figure 6.9.

6.4 Discussion

In this chapter, we evaluated the performance of the redundancy estimators on the synthetic data. Based on the results of this evaluation, we select the LNCm estimator with $k = 5$ for all remaining redundancy estimates. In addition to being supported by our analysis, this choice of k is also the most natural if we aim to minimise systematic errors by selecting a small value [20] while following the recommendation to use $k > 2d$ [25].

To further verify that our LNCm estimator behaves correctly, we use the KSG1m esti-

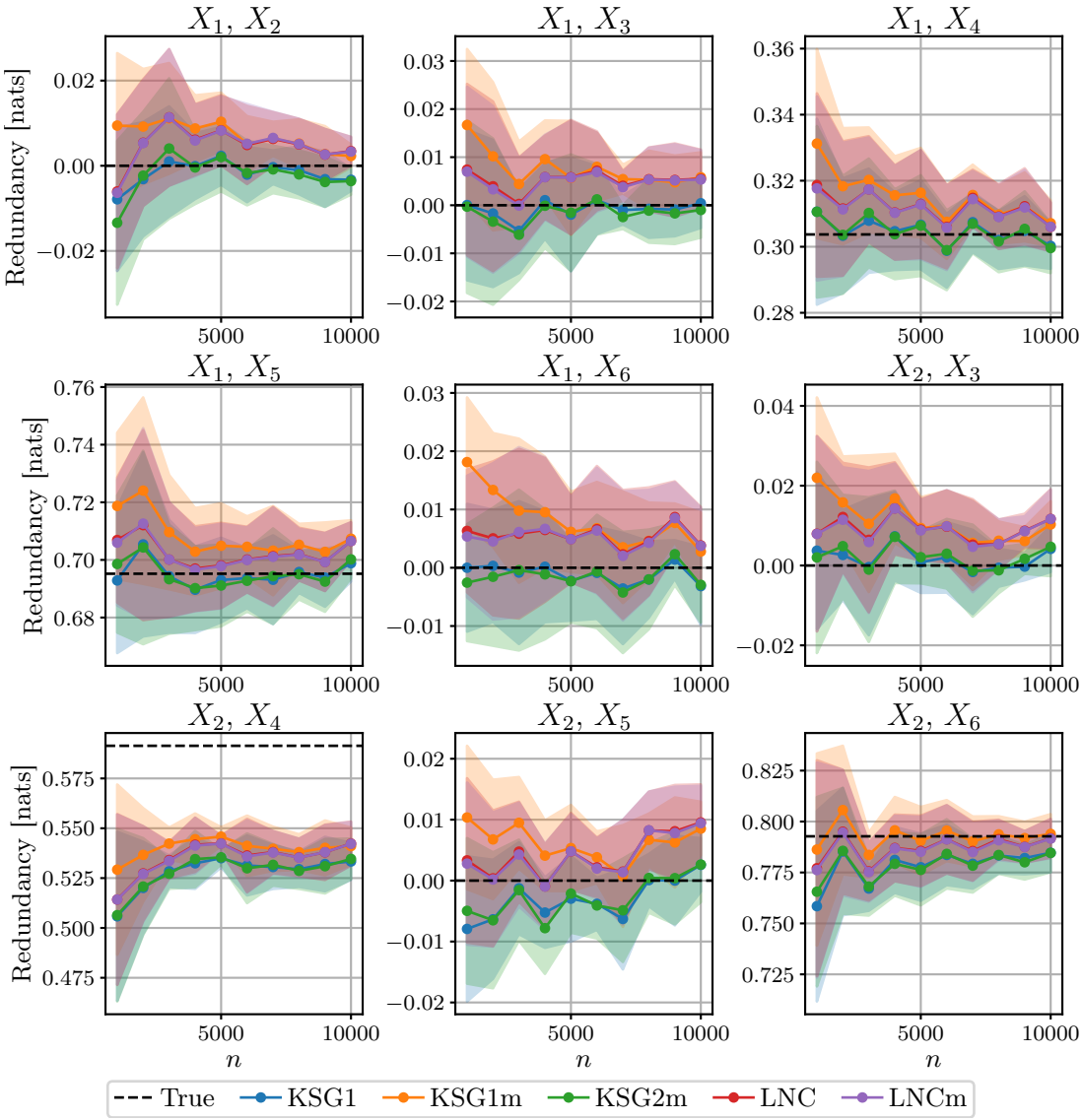

Figure 6.6: Redundancy in pairs of continuous features of the synthetic data, estimated on randomly selected subsets of varying sizes with $k = 5$. The true redundancy is shown with dashed lines. Shaded areas indicate the standard deviation across RNG seeds. Results of the KSG2 estimator are identical to those of the KSG2m estimator.

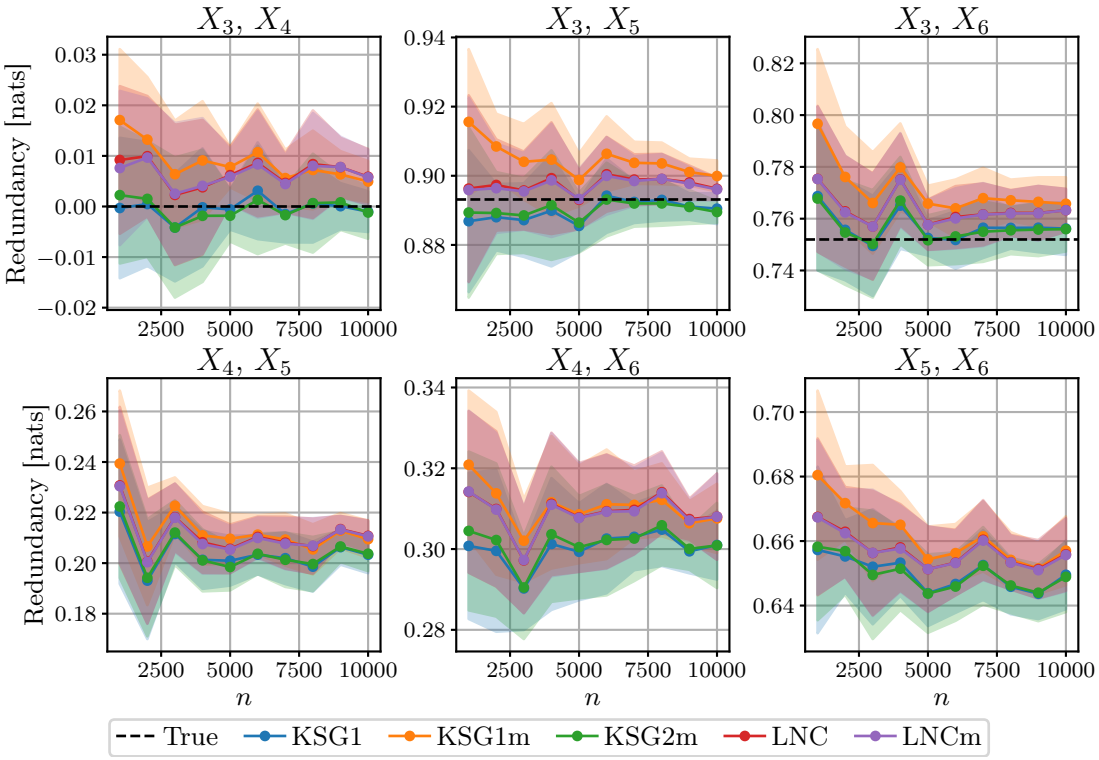

Figure 6.7: Redundancy in pairs of continuous features of the synthetic data, estimated as for Figure 6.6 for additional pairs of features. The true redundancy values for the pairs in the bottom row are not available.

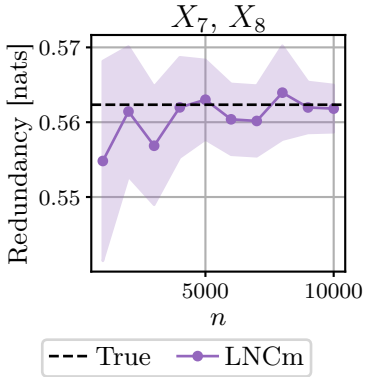

Figure 6.8: Redundancy between the discrete feature and mixed feature of the synthetic data, estimated on randomly selected subsets of varying sizes with $k = 5$. The true redundancy is shown with a dashed line. Shaded areas indicate the standard deviation across RNG seeds. Results of the KSG1, KSG2, and LNC estimators are not shown. Results of the KSG1m and KSG2m estimators are identical to those of the LNCm estimator.

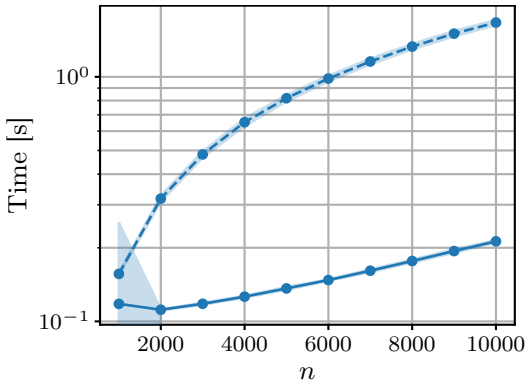

Figure 6.9: Average time required to calculate a single KSG2 redundancy estimate and the associated LNC term for subsets of varying sizes. Solid lines show measurements for our implementation, and dashed lines show measurements for the implementation provided by the authors of [25]. Shaded areas indicate the standard deviation across RNG seeds. Measurements are taken on an AMD Ryzen 3700X CPU (8 cores/16 threads, 3.6 GHz base clock, 4.4 GHz max boost clock).

mator in addition to the LNCm estimator to verify the LNCm estimates. We only report the estimates produced by the LNCm estimator, but we do compare these estimates to those produced by the KSG1m estimator.

Although our evaluation suggests that the LNCm estimator is most accurate for a combination of continuous, discrete, and mixture distributions, our analysis is not extensive enough to allow us to state this as a general claim. We do, however, select the LNCm estimator as the most reliable estimator for the remainder of this study. A more thorough study of our KSG2m and LNCm estimators is left for future work. More thorough analyses of the KSG1, KSG2, LNC, and KSG1m estimators are presented in [20], [21], [25], [47].

In the following chapter, we apply the LNCm estimator to several trained neural networks.

Chapter 7

Redundancy in Deep Neural Networks

We train several neural networks for specific generalisation behaviour. We use these networks to demonstrate the utility of our redundancy estimators.

7.1 Chapter Overview

To demonstrate the use of our redundancy estimators, we train two sets of networks such that networks from different sets differ mainly in generalisation ability. We then estimate the redundancy between several variables in these networks and demonstrate that redundancy estimates can reveal differences in networks with different generalisation characteristics. The estimates that we consider, however, are not sufficient for the prediction of generalisation ability.

In Section 7.2, we describe the process used to train the networks. We explore the redundancy between node activation values and class labels in Section 7.3. We briefly explore the redundancy between different node activation values in Section 7.4, and, in Section 7.5, we draw conclusions from the results of our exploration.

7.2 Training

The main purpose of the network training process is to produce networks that differ mainly in generalisation ability. We measure generalisation ability through the generalisation gap of a network (the difference between the network error on the evaluation set and the error on the training set). To facilitate comparisons between networks, we aim to produce differences in generalisation ability by producing differences in performance on the evaluation set while keeping performance on the training set as similar as possible.

We have found that networks with poor performance on the evaluation set can be produced by inhibiting[1] a fraction p_F of nodes in each hidden layer from receiving weight updates. With these frozen nodes, we train the network until, for n_{F1} training epochs, no increase in the difference between the validation error and training error is observed. Then, we unfreeze all nodes by allowing weight updates, decrease the learning rate by a factor of s_F, and continue training until no increase in the difference between the validation error and training error is observed for n_{F2} epochs. By setting these variables (p_F, s_F, n_{F1}, and n_{F2}) appropriately, we can obtain networks with the required generalisation characteristics.

To train networks with better performance on the evaluation set, we follow a similar process that differs only in the usage of frozen nodes and in the criteria used to stop training. We train the network until, for n_{F1} training epochs, no decrease in the training error, rather than an increase in the difference between the validation error and training error, is observed. Then, we decrease the learning rate by a factor of s_F and continue training until no decrease in the training error is observed for n_{F2} epochs. For this training process, no nodes are frozen.

We perform a single grid search to find both networks with strong generalisation performance and networks with weaker generalisation performance. The grid search involves training 36 networks with the strong-generalisation strategy and 180 networks with the weak-generalisation strategy. Details of the grid search are as follows:

- **Dataset:** We train all networks on the MNIST dataset [48]. The validation set consists of 12 000 points selected from the original training set, and the remaining 48 000 points are used for training. The evaluation set consists of 10 000 points.

[1] We implement this by setting the learning rate for the relevant groups of weights to zero.

- **Architecture:** We use fully connected, feedforward networks with ReLU activation functions in the hidden layers and identity activation functions in the output layer. All networks have 4 hidden layers, and the number of nodes in each hidden layer is either 50 or 100. Only the first hidden layer has a bias node. For this architecture, bias nodes in the other hidden layers are not required [9].

- **Weight Initialisation:** Weights are initialised using the uniform He (Kaiming) initialisation method [49]. We vary the seed of the RNG used for initialisation between three different values.

- **Training:** All networks are trained using minibatches of 64 points each. Networks are trained with either mean-squared-error (MSE) loss or cross-entropy (CE) loss. We use the Adam optimiser with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$ [50]; the initial learning rate is either 5.5×10^{-4}, 7.0×10^{-4}, or 8.5×10^{-4}. The learning rate is decreased by a factor of $s_F = 50$ after a minimum of $n_{F1} = 50$ epochs, and training continues with this smaller learning rate for a minimum of $n_{F2} = 5$ epochs.

- **Frozen Nodes:** With the weak-generalisation strategy, the fraction p_F of frozen nodes in each hidden layer is varied between one of five different values. For those networks with 50 nodes in each hidden layer, this fraction is either 0.6, 0.65, 0.7, 0.75, or 0.8. For the remaining networks with 100 nodes in each hidden layer, this fraction is either 0.8, 0.85, 0.9, 0.925, or 0.95.

We group the trained networks based on architecture and loss function. Each group is then analysed separately to limit the effects of these aspects on the analysis. For each group, we find the network with the strongest generalisation ability, shown in Table 7.1. To find a comparable network with similar training performance but poorer evaluation performance, we consider only those networks with training errors smaller than or equal to 0.5 %. From this set, we select the network with the weakest generalisation ability in each group, shown in Table 7.2. A reference to the strongest network in a group refers to the appropriate network in Table 7.1, while a reference to the weakest network in a group refers to the appropriate network in Table 7.2.

Table 7.1: Performance of networks with the strongest generalisation ability in their group. The group of networks with 50 nodes per hidden layer is indicated with "4×50", while that of networks with 100 nodes per hidden layer is indicated with "4×100". "MSE" and "CE" refer to the loss function used during training.

Group	Training Error	Evaluation Error	Generalisation Gap
4×50 MSE	0.07 %	2.41 %	2.34 %
4×50 CE	0.00 %	2.23 %	2.23 %
4×100 MSE	0.00 %	1.86 %	1.86 %
4×100 CE	0.00 %	1.67 %	1.67 %

Table 7.2: Performance of networks with the weakest generalisation ability among those in their group with a training error smaller than or equal to 0.5 %. The group of networks with 50 nodes per hidden layer is indicated with "4×50", while that of networks with 100 nodes per hidden layer is indicated with "4×100". "MSE" and "CE" refer to the loss function used during training.

Group	Training Error	Evaluation Error	Generalisation Gap
4×50 MSE	0.44 %	4.73 %	4.29 %
4×50 CE	0.04 %	6.22 %	6.18 %
4×100 MSE	0.49 %	4.60 %	4.11 %
4×100 CE	0.00 %	6.17 %	6.17 %

7.3 Node-Class Redundancy

First, we investigate the amount of information that each node activation contains about the class label of the input point. This entire analysis is performed using training data only. Thus, the networks are not exposed to any unseen data during the analysis.

To perform this analysis, the activation value of each node is viewed as a random variable with a mixture distribution. The class label of the input point is viewed as a discrete random variable. We denote the activation value of the nth node in layer l by $Z_{l,n}$, where both indices n and l start at 1. The first layer index is assigned to the shallowest layer, and layer indices increase with depth. Nodes within a layer are not indexed in any particular order. The class label is denoted by C.

For the networks in each group, we estimate the redundancy $\mathbb{I}(Z_{l,n}, C)$ between the activation value $Z_{l,n}$ of each node and the class label C as a measurement of each node's ability to distinguish between all classes. To measure each node's ability to distinguish a single class from the other classes, we construct a binary indicator label $K_c = \mathbb{T}(C = c)$ for each class c such that $K_c = 1$ if $C = c$ and $K_c = 0$ otherwise. We then estimate the redundancy $\mathbb{I}(Z_{l,n}, K_c)$ between the activation value $Z_{l,n}$ of each node and each indicator label K_c. We refer to $\mathbb{I}(Z_{l,n}, C)$ as full node-class redundancy and to $\mathbb{I}(Z_{l,n}, K_c)$ as indicator node-class redundancy.

For computational tractability, these estimates are calculated on a subset of 2 500 random points from the training set. We do not calculate the redundancy for nodes that do not activate for any of the points in the subset. The estimates are calculated for all networks trained in the grid search described in Section 7.2. The calculations are repeated for two additional randomly selected subsets of the same size.

We first analyse some of the redundancy estimates subjectively to identify any potentially interesting patterns (Section 7.3.1). Then, we investigate the distribution of the estimates across the hidden nodes of a few of the networks (Section 7.3.2). Finally, we use all of the available networks to determine the consistency of the observed patterns (Section 7.3.3).

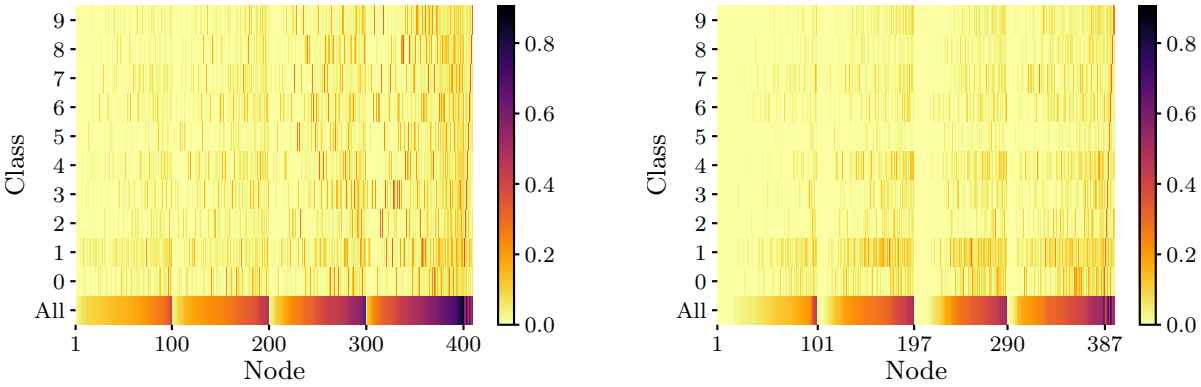

Figure 7.1: Estimated redundancy between node activation values and class labels for the strongest network (left) and the weakest network (right) in the 4×100-CE group. The class "All" refers to the full node-class redundancy, while the remaining classes refer to the indicator node-class redundancy for the corresponding class. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the full node-class redundancy.

7.3.1 Raw Estimates

We identify differences between the strongest and weakest networks (as defined in Section 7.2) in each group by visualising the estimates as in Figure 7.1. This figure shows the estimates for the strongest and weakest networks in the 4×100-CE group (networks with 100 nodes per hidden layer trained with CE loss). Results for the other groups are given in Appendix A.

A subjective analysis of the visualisations seems to reveal two patterns, which are visible in Figure 7.1. Firstly, for each class, the maximum full node-class redundancy seems to be higher for the network with the stronger generalisation ability. This also seems to be the case for the indicator node-class redundancy when only considering hidden nodes. Secondly, for each class, the mean of both the full and indicator node-class redundancy seems to be higher for the network with the stronger generalisation ability.

7.3.2 Redundancy Distribution

To investigate the observed differences further, we plot the full node-class redundancy for the hidden nodes of the strongest and weakest networks in each group. We repeat

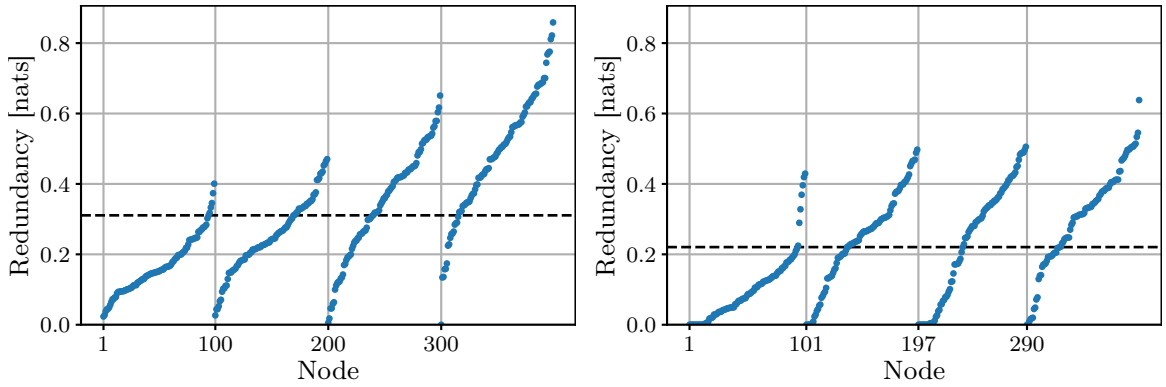

Figure 7.2: Distribution of the full node-class redundancy for the hidden nodes of the strongest network (left) and weakest network (right) in the 4×100-CE group. The mean redundancy value is indicated with a dashed line. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the full node-class redundancy.

this with the indicator node-class redundancy for each class. The plots of the full node-class redundancy for the networks in the 4×100-CE group are shown in Figure 7.2, and those of the indicator node-class redundancy for networks in this group are shown in Figures 7.3 and 7.4. The plots for the networks in the other groups are shown in Appendix A.

The distributions of the node-class redundancy of hidden nodes, like those shown in Figures 7.2 to 7.4, seem to confirm that the maximum of both the full and indicator redundancy is frequently higher for networks with stronger generalisation ability. However, the mean redundancy seems to have a much weaker relationship with generalisation ability. It is interesting to note that the maximum redundancy tends to be higher in deeper layers than in shallower layers. Within each group, the standard deviation of both the maximum and mean indicator redundancy across classes also tends to be higher for networks with weaker generalisation ability.

7.3.3 Redundancy Correlation

Maximum Redundancy

To determine how strongly the maximum node-class redundancy in a network is related to the generalisation ability of the network, we plot the maximum indicator node-class

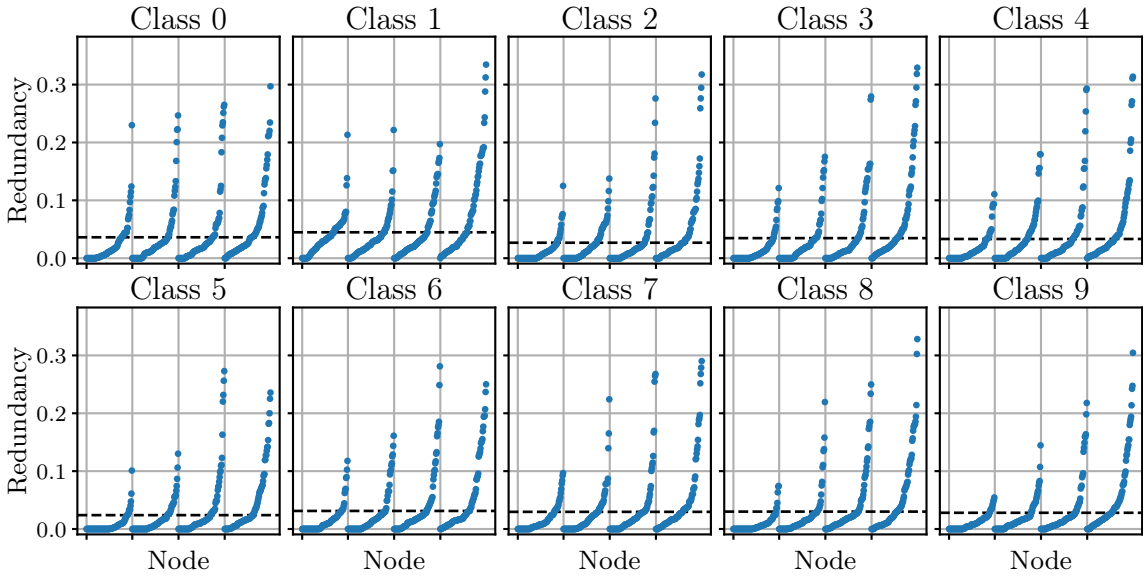

Figure 7.3: Distribution of the indicator node-class redundancy for the hidden nodes of the strongest network in the 4×100-CE group. The mean redundancy value is indicated with a dashed line. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the indicator node-class redundancy.

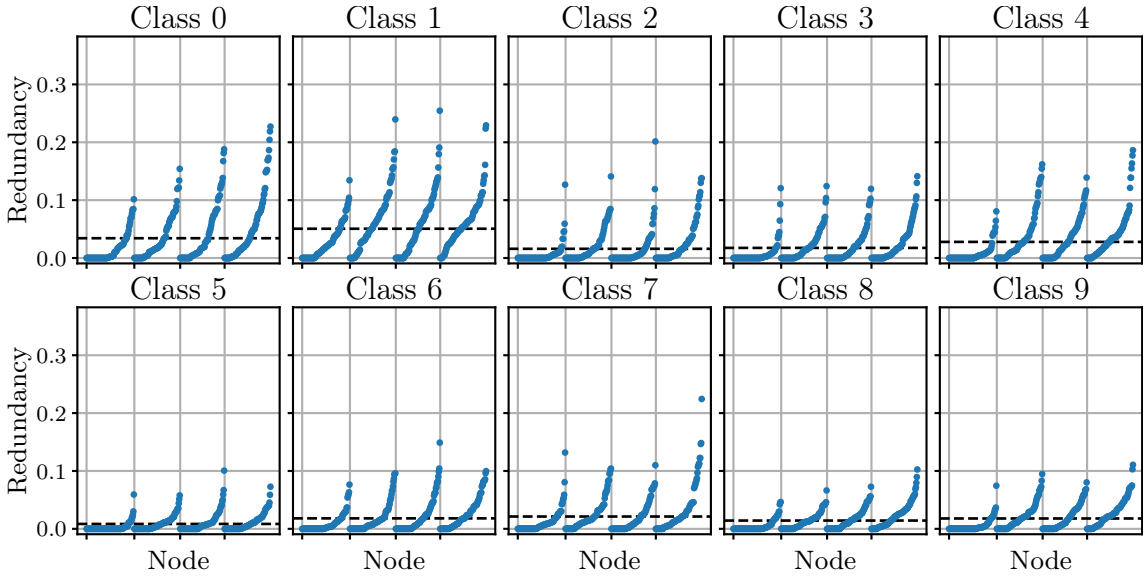

Figure 7.4: Distribution of the indicator node-class redundancy for the hidden nodes of the weakest network in the 4×100-CE group, displayed as in Figure 7.3.

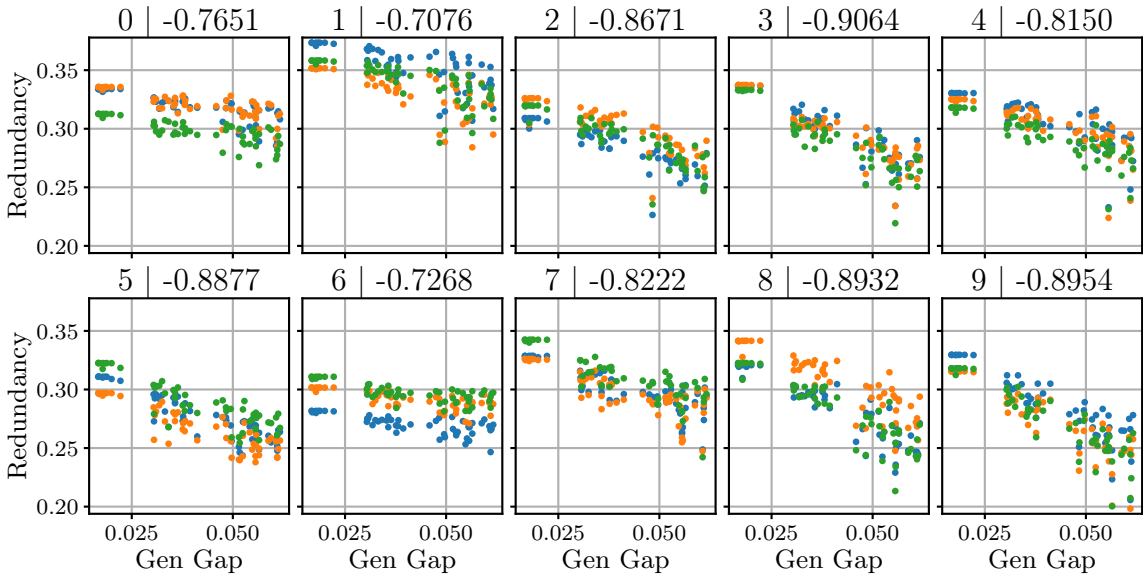

Figure 7.5: Relationship between the maximum indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, for each class and all networks in the 4×100-CE group. Different colours correspond to different subsets used for estimation. The title of each plot shows the corresponding class label and the average Pearson correlation coefficient, across subsets, between the plotted quantities.

redundancy across the hidden nodes in each network against the generalisation gap of the network for each class. We also calculate the average Pearson correlation coefficient, across the three subsets, between the maximum redundancy value and the generalisation gap for each class. The correlation coefficients are calculated for both the full and indicator redundancy. The resulting plots and correlation coefficients are shown in Figure 7.5 for the 4×100-CE group and in Figure 7.6 for the 4×100-MSE group. The correlation coefficients for these two groups are also given in Table 7.3. Results for the other groups are given in Appendix A.

Figure 7.5 and Table 7.3 confirm that the maximum node-class redundancy across the hidden nodes of the networks in the 4×100-CE group is strongly correlated with the generalisation ability of the network. The corresponding correlation for the networks in the 4×100-MSE group, shown in Figure 7.6 and Table 7.3, is the weakest compared to the correlations in the other three groups. It seems that the correlation for networks trained with MSE loss is generally weaker than that for networks trained with CE loss,

Table 7.3: Average Pearson correlation coefficient, across subsets, between the generalisation gap of a network and the maximum and mean node-class redundancy across the hidden nodes of the network, for networks with 100 nodes per hidden layer. The class "All" refers to the full node-class redundancy, while the remaining classes refer to the indicator node-class redundancy for the corresponding class.

Group	Class	Correlation with Maximum	Correlation with Mean
4×100 CE	All	-0.7878	-0.7514
	0	-0.7651	-0.3794
	1	-0.7076	0.4294
	2	-0.8671	-0.7625
	3	-0.9064	-0.7695
	4	-0.8150	-0.5602
	5	-0.8877	-0.8030
	6	-0.7268	-0.7658
	7	-0.8222	-0.7406
	8	-0.8932	-0.7314
	9	-0.8954	-0.7440
4×100 MSE	All	-0.5518	0.2707
	0	-0.1891	0.4234
	1	-0.2009	0.2012
	2	-0.2503	0.0532
	3	-0.2663	-0.0769
	4	-0.1696	0.1656
	5	-0.1667	-0.4867
	6	-0.1583	0.2544
	7	-0.2981	0.1528
	8	-0.1695	-0.4026
	9	-0.2665	-0.1387

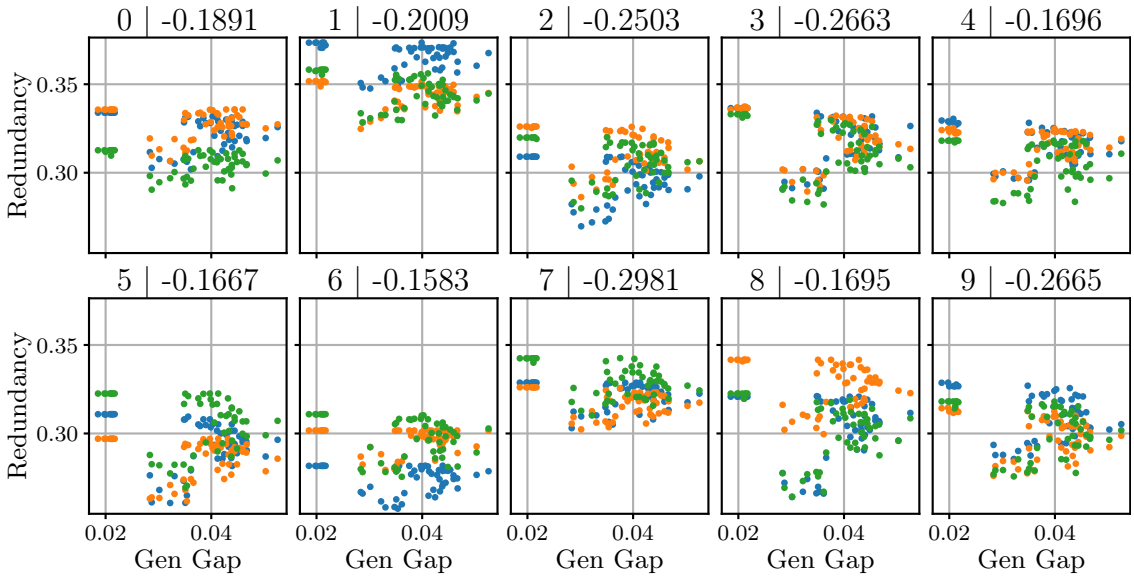

Figure 7.6: Relationship between the maximum indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, displayed as in Figure 7.5 for all networks in the 4×100-MSE group.

indicating that the training process affects the strength of this correlation.

Mean Redundancy

Similar to the analysis of maximum redundancy, we plot the mean node-class redundancy value against the generalisation gap and calculate the corresponding correlation coefficients. The resulting plots and correlation coefficients are shown in Figure 7.7 for the 4×100-CE group and in Figure 7.8 for the 4×100-MSE group. The correlation coefficients for these two groups are also given in Table 7.3. Results for the other groups are given in Appendix A.

Table 7.3 shows that the mean node-class redundancy across the hidden nodes of the networks in the 4×100-CE group is strongly correlated with the generalisation ability of the network for most of the classes. Inspection of Figure 7.7, however, reveals the formation of a low-redundancy cluster and a high-redundancy cluster for most classes. The high-redundancy clusters consist of networks trained with the strong-generalisation strategy, and the low-redundancy clusters consist of networks trained with the weak-generalisation strategy. This suggests that the correlations are significantly influenced

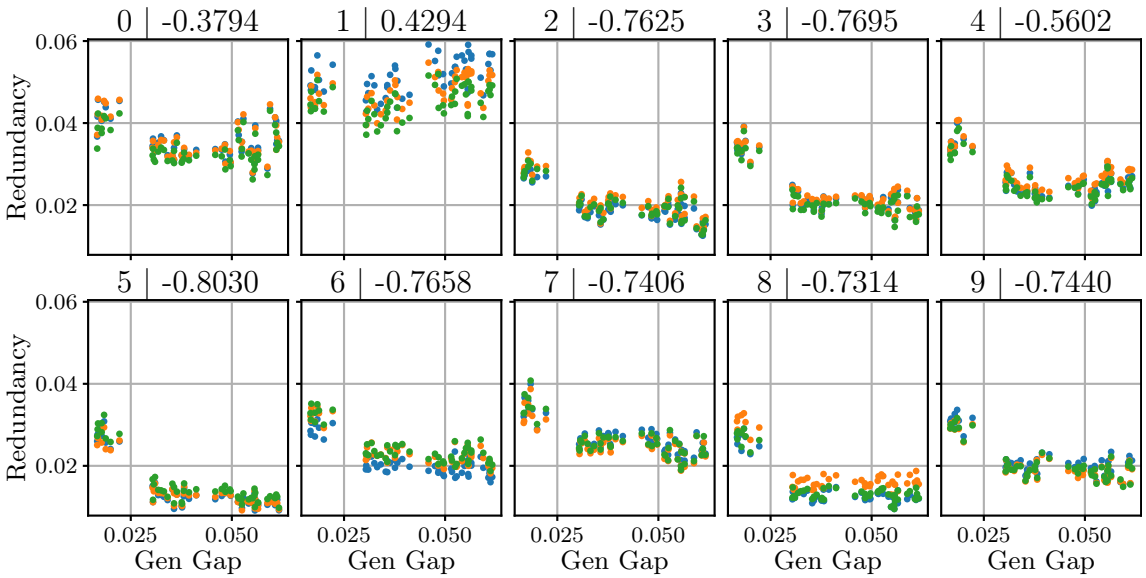

Figure 7.7: Relationship between the mean indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, for each class and all networks in the 4×100-CE group. Different colours correspond to different subsets used for estimation. The title of each plot shows the corresponding class label and the average Pearson correlation coefficient, across subsets, between the plotted quantities.

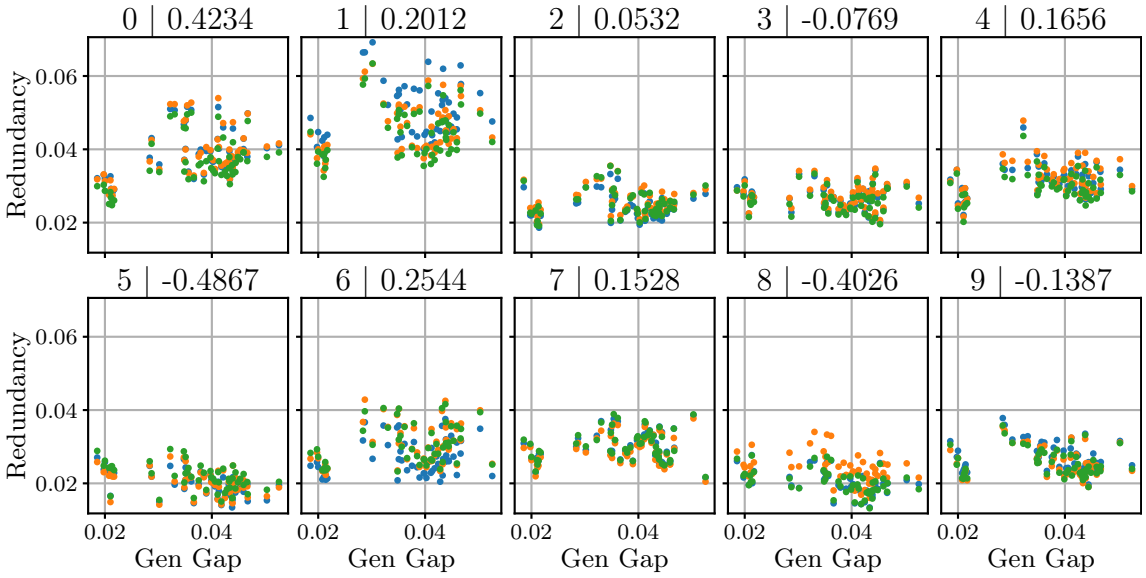

Figure 7.8: Relationship between the mean indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, displayed as in Figure 7.7 for all networks in the 4×100-MSE group.

by the differences in training strategy. Producing the plots and correlation coefficients for only the networks trained with a particular strategy (not shown here) confirms that the majority of the correlation can be attributed to differences in training strategy.

Figure 7.8 and Table 7.3 show that the mean node-class redundancy of the networks in the 4×100-MSE group is very weakly correlated with the generalisation ability of the network.

7.4 Node Redundancy

Now, we investigate the amount of redundant information in the activation values of the nodes. This analysis is again performed using training data only. As is done in Section 7.3, activation values are viewed as random variables, and the same notation is used to refer to these variables.

For the strongest and weakest networks in each group, we estimate the pairwise node redundancy $\mathbb{I}(Z_{l,n}, Z_{k,m})$ in each pair of activation values, where the activation values in each pair correspond to distinct nodes ($l \neq k$ or $n \neq m$).

For computational tractability, these estimates are calculated on a subset of 2 500 random points from the training set. We do not calculate the redundancy for nodes that do not activate for any of the points in the subset. The calculations are repeated for two additional randomly selected subsets of the same size. The estimates for the networks in the 4×100-CE group are shown in Figure 7.9.

A subjective analysis of Figure 7.9 seems to suggest that the mean pairwise redundancy in the first two hidden layers is higher for the network with the weaker generalisation ability. This pattern also appears in the estimates for the other groups. To investigate this, we calculate the mean pairwise redundancy within each hidden layer of the strongest and weakest networks in each group (only pairs where both corresponding nodes are in the same layer are considered). We then calculate the mean of these values across the three subsets used for estimation. The results are shown in Table 7.4.

Table 7.4 shows that, in the second hidden layer only, the mean pairwise redundancy is consistently higher for the weakest network. No consistent ordering relationship seems to exist for the mean redundancy in the other hidden layers.

Although we cannot draw any conclusions from such a brief analysis of the estimated

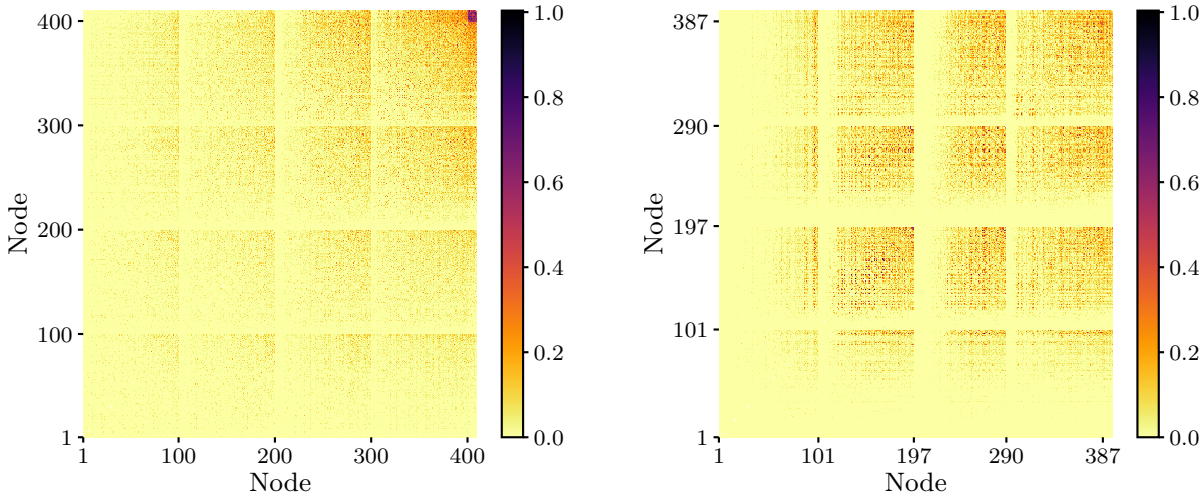

Figure 7.9: Estimated redundancy in pairs of node activation values for the strongest network (left) and the weakest network (right) in the 4×100-CE group. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the full node-class redundancy. (Nodes have the same order as in Figure 7.1.)

Table 7.4: Mean pairwise node redundancy across the nodes in the hidden layers of the strongest and weakest networks in each group.

Group	Layer	Strongest Network	Weakest Network
4×100 CE	1	0.0067	0.0038
	2	0.0115	0.0562
	3	0.0262	0.0379
	4	0.0634	0.0520
4×100 MSE	1	0.0070	0.0040
	2	0.0101	0.0291
	3	0.0285	0.0508
	4	0.1168	0.0397
4×50 CE	1	0.0084	0.0092
	2	0.0155	0.0369
	3	0.0263	0.0408
	4	0.0554	0.0395
4×50 MSE	1	0.0084	0.0095
	2	0.0196	0.0278
	3	0.0955	0.0671
	4	0.0481	0.0514

pairwise node redundancy, the patterns observed in these estimates provide a good starting point for a future in-depth analysis.

7.5 Discussion

In this chapter, we trained several neural networks to have different generalisation characteristics. We then estimated the redundancy between each node activation and several class labels using the training data. An analysis of these estimates showed that, for some of the networks, the maximum node-class redundancy across the hidden nodes of a network is strongly correlated with the generalisation gap of the network. This correlation appeared to be somewhat inconsistent. We performed a similar analysis of the mean node-class redundancy, but this analysis revealed no significant relationship between the generalisation gap of a network and the mean node-class redundancy across the hidden nodes. Finally, we estimated the redundancy in pairs of node activation values and pointed out some basic patterns in these estimates as a starting point for future work.

From these results, we conclude that the estimation of redundancy between the node activation values in a network and several class labels shows promise as a potential part of measuring confirmation and confirmation ability. The apparent relationship between these estimates and generalisation, although not strong enough to allow the prediction of generalisation, is particularly significant, since the redundancy estimates are calculated using training data only.

To reach the larger goal of measuring confirmation ability, additional redundancy values would need to be measured. Potentially useful measurements include the redundancy between node activation values and input features or the redundancy in larger sets of node activation values.

Other interesting observations in this chapter that could form part of an in-depth study of the distribution of node-class redundancy include:

- The maximum node-class redundancy in the hidden layers of a network seems to increase as the depth of the layer increases.
- The distribution of node-class redundancy seems to depend on the training process.

In the following chapter, we conclude the dissertation with a review of our work and a set of topics for future work.

Chapter 8

Conclusion

We review the key findings of our research. We discuss the implications of the findings and propose possible avenues for future work.

8.1 Chapter Overview

In this chapter, we review the key findings of our research in Section 8.2 and discuss the implications of the findings in Section 8.3. In Section 8.4, we propose possible topics for future work.

8.2 Key Findings

The goal of our research is to find a method of measuring redundancy in neural networks and to use this method to explore the measurement of data utilisation. In pursuit of this goal, the following key findings are uncovered:

- Mixture distributions are common in ReLU-activated networks. The properties of these distributions prevent the calculation of differential entropy and complicate the calculation of relative entropy and redundancy.
- Several estimators that can produce redundancy estimates from a finite sample are available. We find that the KSG1, KSG2, and LNC estimators are unsuitable in

cases where the underlying distribution is a mixture distribution. The KSG1m estimator can be used in such cases.

- We derive the KSG2m estimator and the associated LNCm term. The KSG2m estimator can be applied to data generated by a mixture distribution. The LNCm term reduces the number of sample points required to obtain an accurate estimate with the KSG2m estimator.

- By evaluating these redundancy estimators on synthetic data, we confirm that the estimators behave as expected. We also find that, on the synthetic dataset, the LNCm estimator is most accurate on average. This analysis is limited to estimates of redundancy in pairs of variables; further analysis is required to characterise the behaviour of these estimators when applied to larger sets of variables.

- Using the LNCm estimator, we estimate several redundancy values in fully connected, feedforward networks and find interesting regularities and differences between networks with different generalisation characteristics. This analysis, however, is only preliminary, and the uncovered patterns require further investigation.

- The redundancy between node activation values and several types of class labels, which include binary indicator labels, has proven to be particularly interesting, since initial estimates of this redundancy using training data appear to be correlated with generalisation in certain contexts.

8.3 Implications

The key findings of our research have the following implications:

- The KSG1m, KSG2m, and LNCm estimators allow redundancy to be estimated without requiring a detailed analysis of the properties of the underlying probability distributions. Our implementation of these estimators, which we plan to release, should aid others in calculating such redundancy estimates.

- Redundancy estimates can be used to measure task-relevant information in individual node activations, and these estimates can be used to analyse the generalisation ability of fully connected, feedforward networks.

8.4 Future Work

In future work, we aim to address the following topics:

- The causes of the patterns observed in this work can be investigated. This could provide additional insight into the operation of neural networks and the relationship between the observed patterns and generalisation.
- The initial observations made in this work can be confirmed for additional network architectures and experimental setups. Experimenting with different methods of obtaining several networks with similar training performance but different evaluation performance would form a particularly interesting part of such a study.
- The evolution of several redundancy values throughout training can be investigated. Such a study could provide additional insight into the operation of neural networks and the role of the training process in the generalisation ability of networks.
- The redundancy between other variables relevant to network analysis can be studied. This could provide additional values that can be used in the measurement of confirmation and data utilisation.
- The use of redundancy estimates in the measurement of confirmation and data utilisation can be explored. If a suitable measurement method is found, the relationship between generalisation and data utilisation can also be explored.
- Network analysis using other information-theoretic quantities, in combination with redundancy, can be explored. Quantities that might prove useful include relative entropy and code-length measures similar to those in [39].

8.5 Final Remarks

The generalisation ability of DNNs is not completely understood. A wide variety of approaches have been used to study this generalisation ability, and these approaches have yielded interesting insights. In this work, we developed tools that allow us to investigate the use of redundancy measurements for explaining the generalisation ability

of neural networks. Our results seem promising, and we hope that they can be used in future work to explore generalisation further.

References

[1] C. M. Bishop, "Chapter 1: Introduction," in *Pattern Recognition and Machine Learning*. Springer New York, 2006, pp. 1–66.

[2] I. Goodfellow, Y. Bengio, and A. Courville, "Section 5.2: Capacity, overfitting and underfitting," in *Deep Learning*. MIT Press, 2016, pp. 110–120.

[3] D. J. MacKay, "Chapter 28: Model comparison and Occam's razor," in *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005, pp. 343–355.

[4] P. L. Bartlett and S. Mendelson, "Rademacher and Gaussian complexities: Risk bounds and structural results," *Journal of Machine Learning Research*, vol. 3, pp. 463–482, 2002.

[5] T. Hastie, R. Tibshirani, and J. Friedman, "Chapter 7: Model assessment and selection," in *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2017, pp. 219–259.

[6] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires re-thinking generalization," in *International Conference on Learning Representations*, (Toulon, France), Apr. 2017.

[7] T. Hastie, R. Tibshirani, and J. Friedman, "Chapter 2: Overview of supervised learning," in *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2017, pp. 9–41.

[8] M. W. Theunissen, M. H. Davel, and E. Barnard, "Benign interpolation of noise in deep learning," *South African Computer Journal*, vol. 32, no. 2, 2020.

[9] M. H. Davel, M. W. Theunissen, A. Pretorius, and E. Barnard, "DNNs as layers of cooperating classifiers," in *AAAI Conference on Artificial Intelligence*, vol. 34, Apr. 2020, pp. 3725–3732.

[10] D. G. Haasbroek and M. H. Davel, "Exploring neural network training dynamics through binary node activations," in *Southern African Conference for Artificial Intelligence Research*, Dec. 2020, pp. 304–320.

[11] C. M. Bishop, "Section 1.6: Information theory," in *Pattern Recognition and Machine Learning*. Springer New York, 2006, pp. 48–58.

[12] D. J. MacKay, "Chapter 1: Introduction to information theory," in *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005, pp. 3–21.

[13] D. J. MacKay, "Chapter 2: Probability, entropy, and inference," in *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005, pp. 22–46.

[14] D. J. MacKay, "Chapter 8: Dependent random variables," in *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005, pp. 138–144.

[15] E. T. Jaynes, "Information theory and statistical mechanics," in *Statistical Physics*. W.A. Benjamin, 1963, pp. 181–218.

[16] I. Goodfellow, Y. Bengio, and A. Courville, "Section 3.13: Information theory," in *Deep Learning*. MIT Press, 2016, pp. 73–75.

[17] W. F. Trench, "Chapter 3: Integral calculus of functions of one variable," in *Introduction to Real Analysis*, Free Edition 2.03. 2003, pp. 113–177.

[18] E. T. Jaynes, "Prior probabilities," *IEEE Transactions On Systems Science and Cybernetics*, vol. 4, no. 3, pp. 227–241, 1968.

[19] T. S. Han, "Nonnegative entropy measures of multivariate symmetric correlations," *Information and Control*, vol. 36, no. 2, pp. 133–156, 1978.

[20] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical Review E*, vol. 69, no. 6, 066138, 2004.

[21] W. Gao, S. Kannan, S. Oh, and P. Viswanath, "Estimating mutual information for discrete-continuous mixtures," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[22] A. M. Saxe, Y. Bansal, J. Dapello, *et al.*, "On the information bottleneck theory of deep learning," in *International Conference on Learning Representations*, (Vancouver, BC, Canada), Apr. 2018.

[23] M. Noshad, Y. Zeng, and A. O. Hero III, "Scalable mutual information estimation using dependence graphs," 2018. arXiv: 1801.09125v2 [cs.IT].

[24] A. Kolchinsky and B. Tracey, "Estimating mixture entropy with pairwise distances," *Entropy*, vol. 19, no. 7, 361, 2017.

[25] S. Gao, G. Ver Steeg, and A. Galstyan, "Efficient estimation of mutual information for strongly dependent variables," in *International Conference on Artificial Intelligence and Statistics*, (San Diego, California, USA), vol. 38, May 2015, pp. 277–286.

[26] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in deep learning," 2020. arXiv: 1710.05468v6 [stat.ML].

[27] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of Machine Learning Research*, vol. 2, pp. 499–526, 2002.

[28] H. Xu and S. Mannor, "Robustness and generalization," *Machine Learning*, vol. 86, no. 3, pp. 391–423, 2012.

[29] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," in *International Conference on Learning Representations*, (Toulon, France), Apr. 2017.

[30] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *International Conference on Learning Representations*, (Toulon, France), Apr. 2017.

[31] G. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, "Large margin deep networks for classification," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 842–852.

[32] Y. Jiang, D. Krishnan, H. Mobahi, and S. Bengio, "Predicting the generalization gap in deep networks with margin distributions," in *International Conference on Learning Representations*, (New Orleans, Louisiana, United States), May 2019.

[33] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," 1999. arXiv: physics/0004057v1 [physics.data-an].

[34] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *IEEE Information Theory Workshop*, 2015, pp. 1–5.

[35] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," 2017. arXiv: 1703.00810v3 [cs.LG].

[36] Z. Goldfeld, E. van den Berg, K. Greenewald, *et al.*, "Estimating information flow in deep neural networks," in *International Conference on Machine Learning*, (Long Beach, California, USA), vol. 97, Jun. 2019, pp. 2299–2308.

[37] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention*, 2015, pp. 234–241.

[38] S. Lee and I. V. Bajić, "Analysis of information flow through U-nets," 2021. arXiv: 2101.08427v2 [cs.LG].

[39] X. Zhang, X. Li, D. Dou, and J. Wu, "Measuring information transfer in neural networks," 2020. arXiv: 2009.07624v2 [cs.LG].

[40] M. H. Davel, "Using summary layers to probe neural network behaviour," *South African Computer Journal*, vol. 32, no. 2, 2020.

[41] Y. T. Li and R. Wong, "Integral and series representations of the Dirac delta function," *Communications on Pure & Applied Analysis*, vol. 7, no. 2, pp. 229–247, 2008.

[42] A. I. Saichev and W. A. Woyczynski, "Chapter 1: Basic definitions and operations," in *Distributions in the Physical and Engineering Sciences: Distributional and Fractal Calculus, Integral Transforms and Wavelets*. Birkhäuser, 1997, vol. 1, pp. 1–36.

[43] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[44] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLOS ONE*, vol. 9, no. 2, 2014.

[45] D. J. MacKay, "Chapter 23: Useful probability distributions," in *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005, pp. 311–318.

[46] L. Devroye, "Section 2.2: The inversion method," in *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986, pp. 27–39.

[47] W. Gao, S. Oh, and P. Viswanath, "Demystifying fixed k-nearest neighbor information estimators," 2016. arXiv: 1604.03006v2 [cs.LG].

[48] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, (San Diego, California, USA), May 2015.

Appendix A

Additional Results

A.1 Raw Redundancy Estimates

Estimates of the node-class redundancy, calculated as described in Section 7.3, are shown in Figures A.1 to A.3 for the strongest and weakest networks in the 4×100-MSE, 4×50-CE, and 4×50-MSE groups, respectively.

A.2 Redundancy Distribution

The distributions of the node-class redundancy of hidden nodes in the strongest and weakest networks in the 4×100-MSE, 4×50-CE, and 4×50-MSE group are shown in Figures A.4 to A.12.

A.3 Redundancy Correlation

The relationship between the maximum indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network is shown in Figures A.13 and A.14 for the 4×50-CE and 4×50-MSE groups, respectively. The correlation coefficients for these groups are shown in Table A.1.

The relationship between the mean indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network is shown in Figures A.15 and A.16 for the 4×50-CE and 4×50-MSE groups, respectively. The correlation coefficients for these groups are shown in Table A.1.

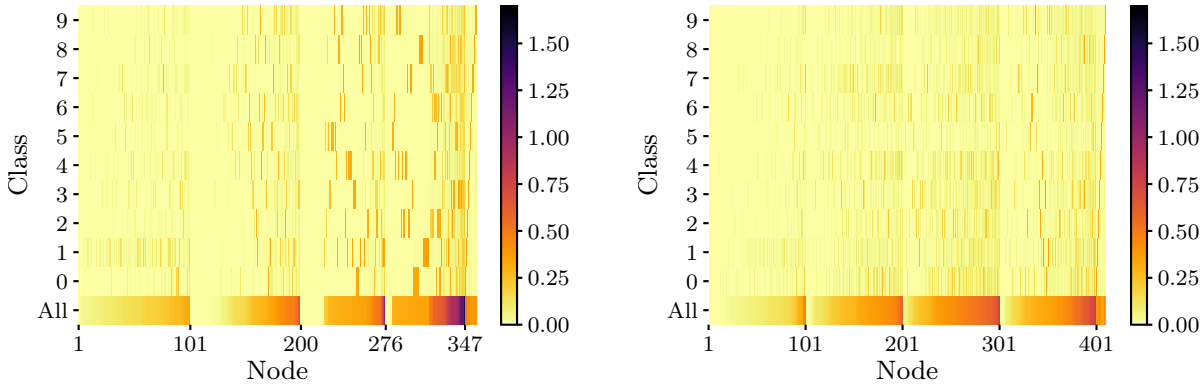

Figure A.1: Estimated redundancy between node activation values and class labels for the strongest network (left) and the weakest network (right) in the 4×100-MSE group. The class "All" refers to the full node-class redundancy, while the remaining classes refer to the indicator node-class redundancy for the corresponding class. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the full node-class redundancy.

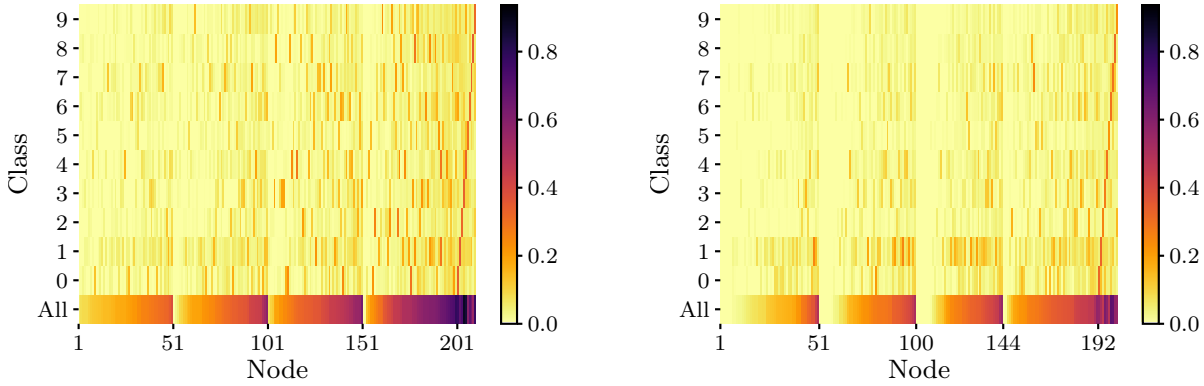

Figure A.2: Estimated redundancy between node activation values and class labels, displayed as in Figure A.1 for networks in the 4×50-CE group.

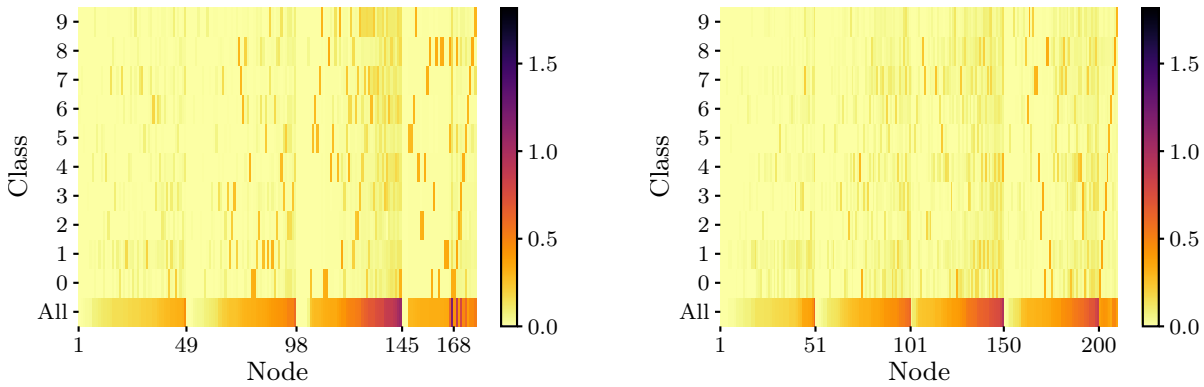

Figure A.3: Estimated redundancy between node activation values and class labels, displayed as in Figure A.1 for networks in the 4×50-MSE group.

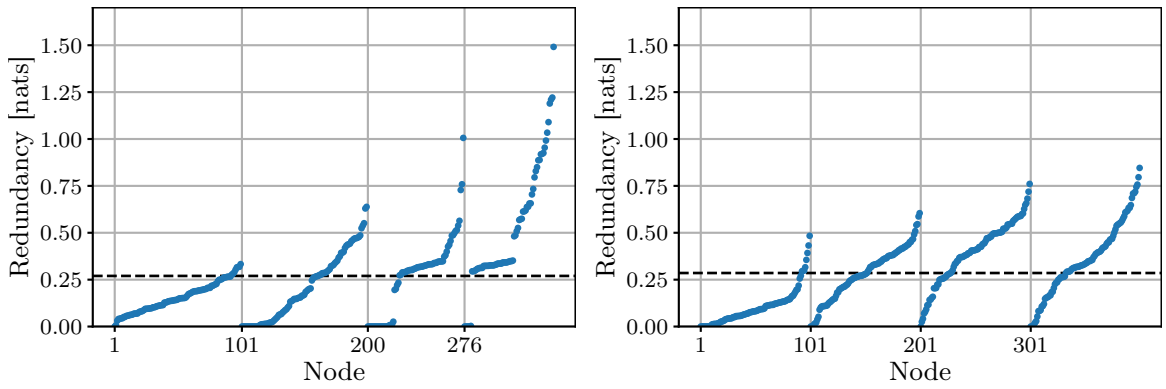

Figure A.4: Distribution of the full node-class redundancy for the hidden nodes of the strongest network (left) and weakest network (right) in the 4×100-MSE group. The mean redundancy value is indicated with a dashed line. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the full node-class redundancy.

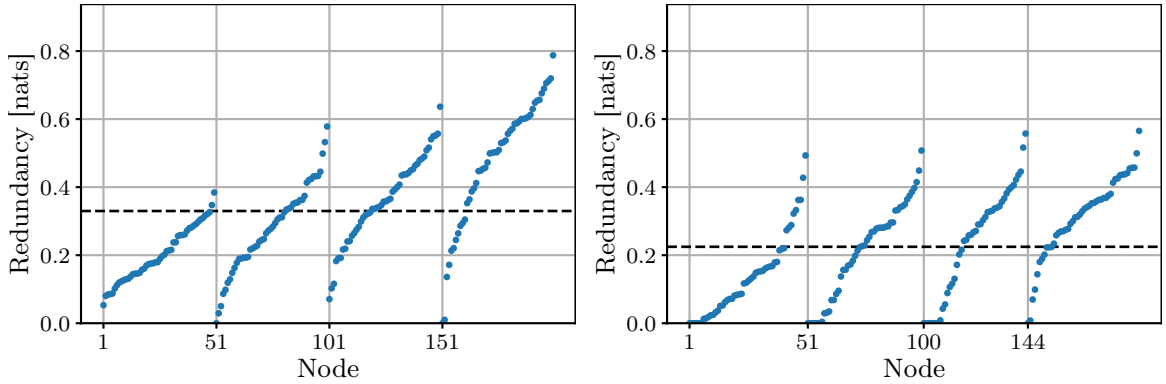

Figure A.5: Distribution of the full node-class redundancy, displayed as in Figure A.4 for networks in the 4×50-CE group.

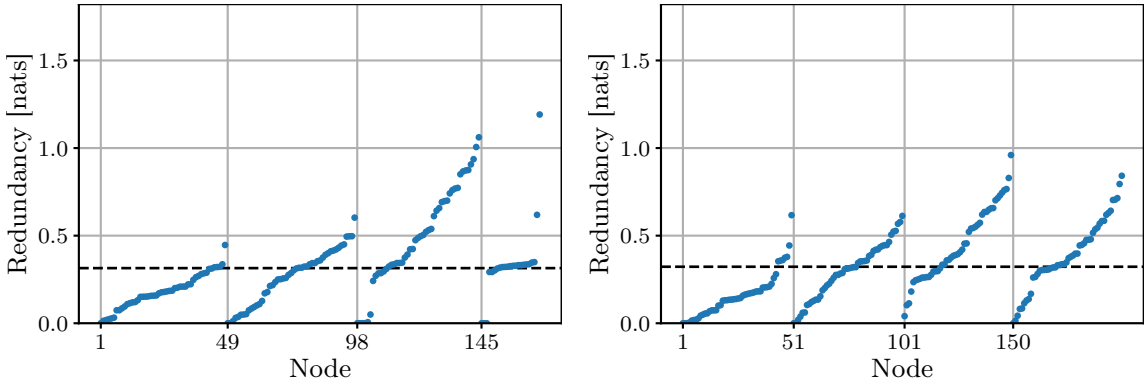

Figure A.6: Distribution of the full node-class redundancy, displayed as in Figure A.4 for networks in the 4×50-MSE group.

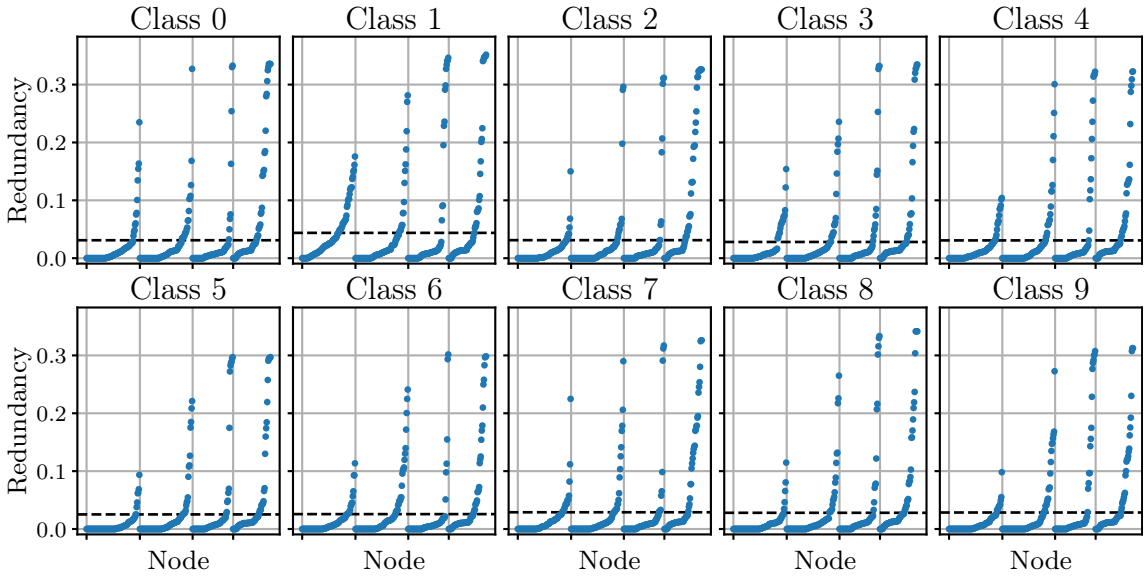

Figure A.7: Distribution of the indicator node-class redundancy for the hidden nodes of the strongest network in the 4×100-MSE group. The mean redundancy value is indicated with a dashed line. Nodes are numbered from shallowest to deepest, with node labels indicating the first node in a layer. Within a layer, nodes are ordered based on the indicator node-class redundancy.

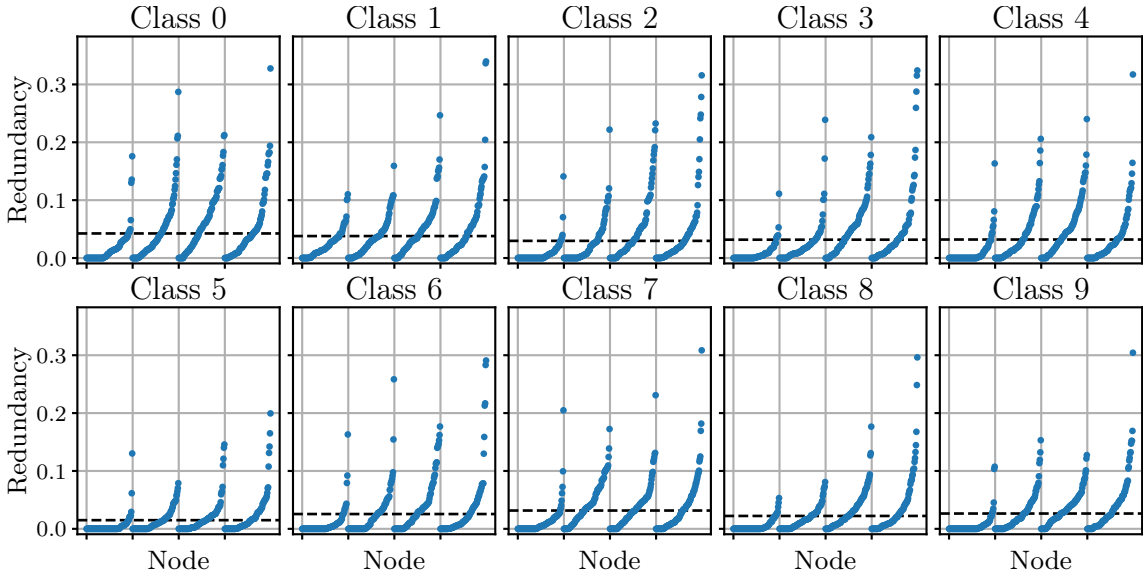

Figure A.8: Distribution of the indicator node-class redundancy for the hidden nodes of the weakest network in the 4×100-MSE group, displayed as in Figure A.7.

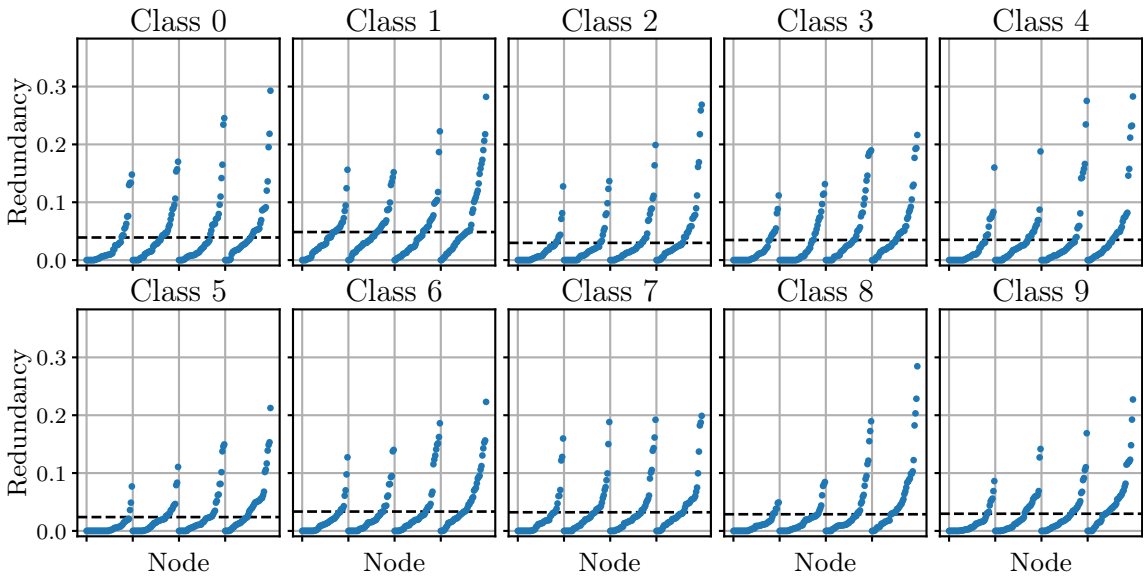

Figure A.9: Distribution of the indicator node-class redundancy for the hidden nodes of the strongest network in the 4×50-CE group, displayed as in Figure A.7.

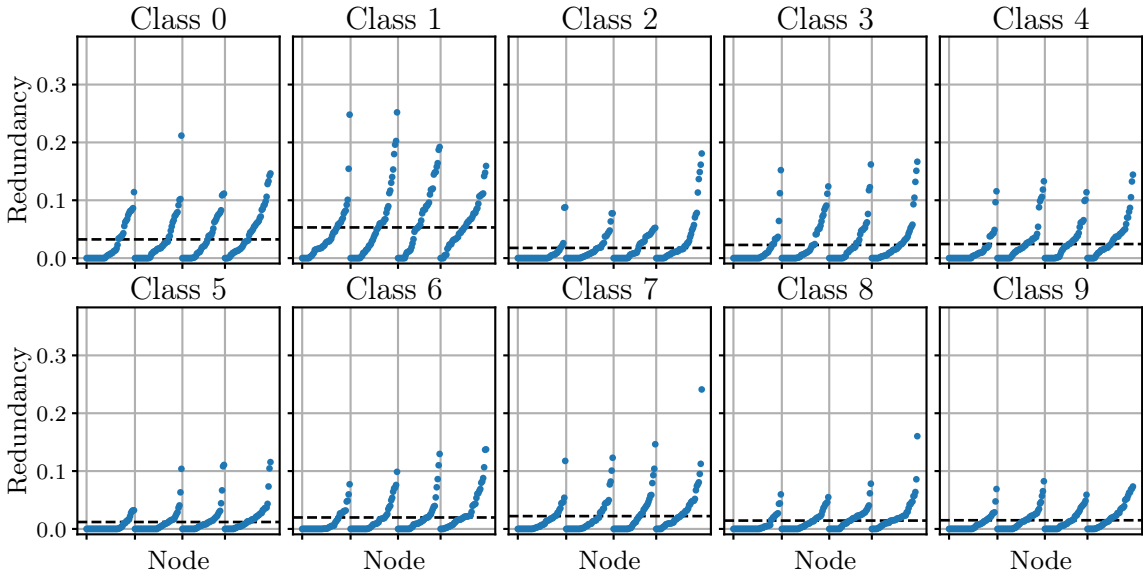

Figure A.10: Distribution of the indicator node-class redundancy for the hidden nodes of the weakest network in the 4×50-CE group, displayed as in Figure A.7.

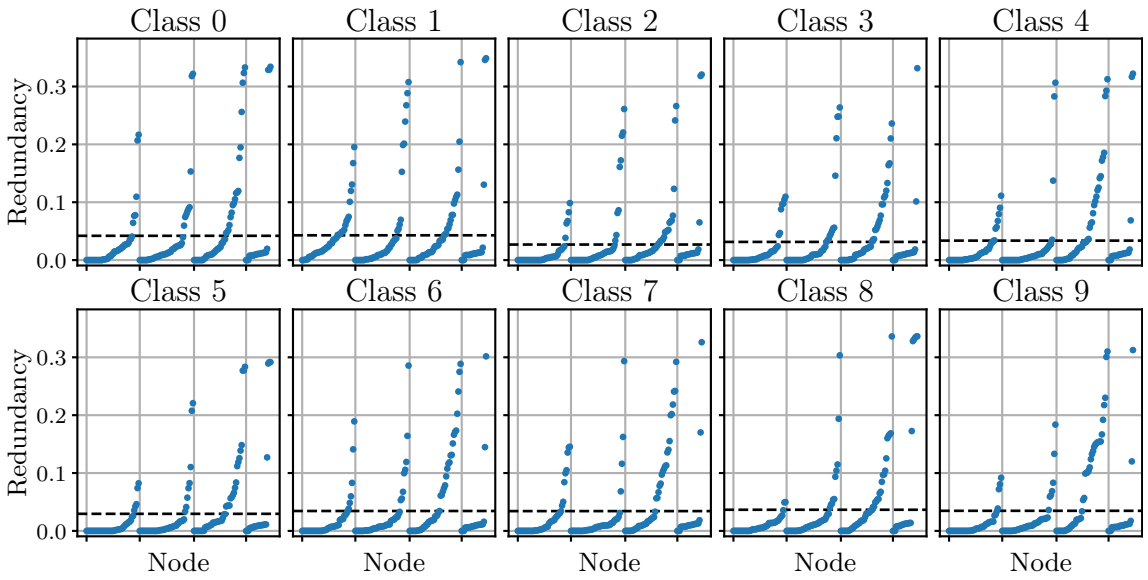

Figure A.11: Distribution of the indicator node-class redundancy for the hidden nodes of the strongest network in the 4×50-MSE group, displayed as in Figure A.7.

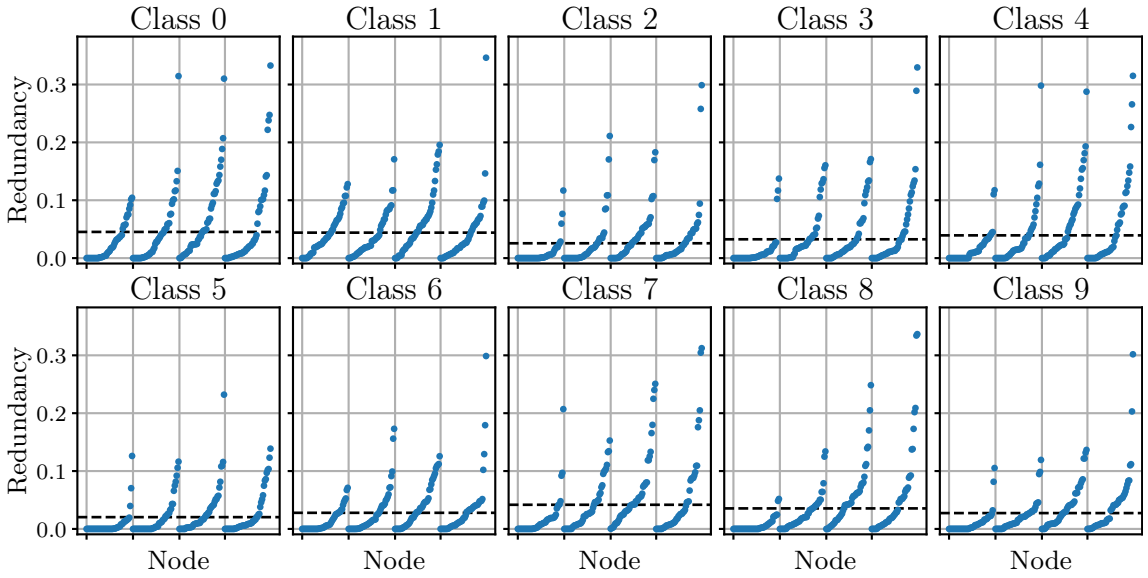

Figure A.12: Distribution of the indicator node-class redundancy for the hidden nodes of the weakest network in the 4×50-MSE group, displayed as in Figure A.7.

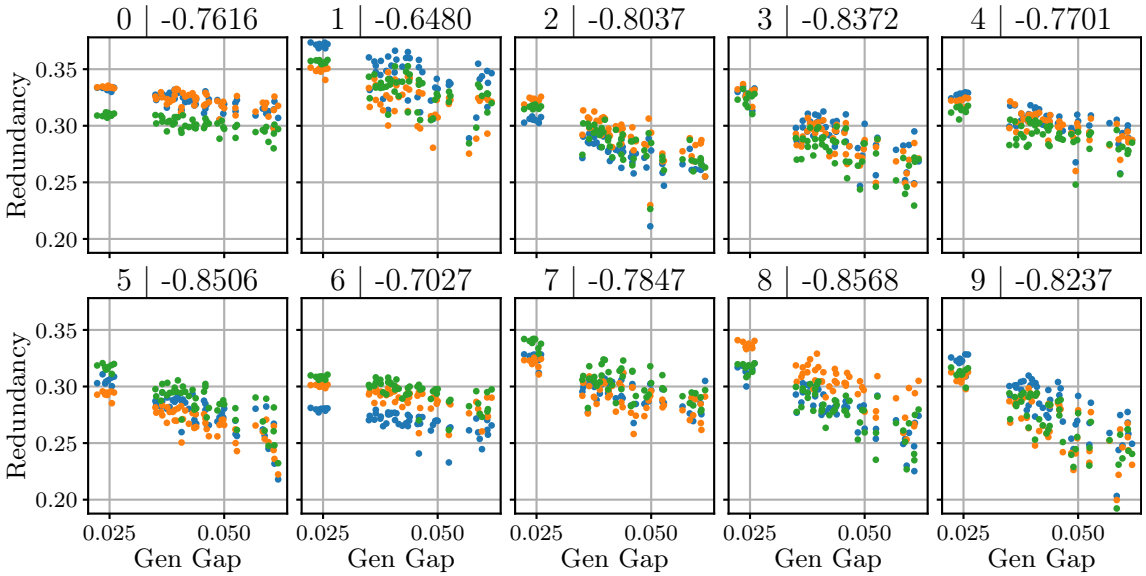

Figure A.13: Relationship between the maximum indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, for each class and all networks in the 4×50-CE group. Different colours correspond to different subsets used for estimation. The title of each plot shows the corresponding class label and the average Pearson correlation coefficient, across subsets, between the plotted quantities.

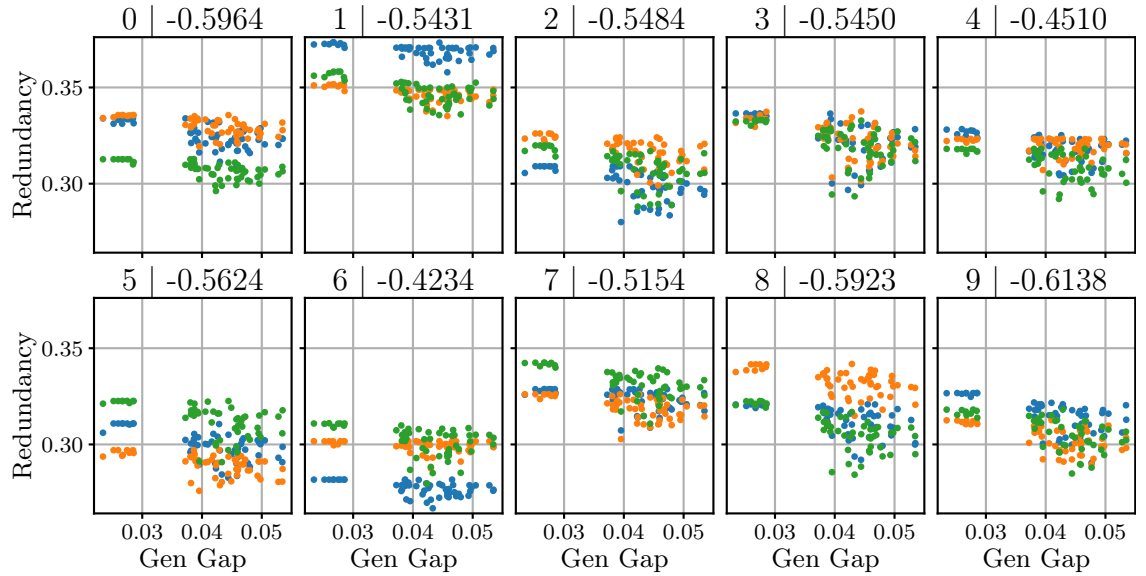

Figure A.14: Relationship between the maximum indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, displayed as in Figure A.13 for all networks in the 4×50-MSE group.

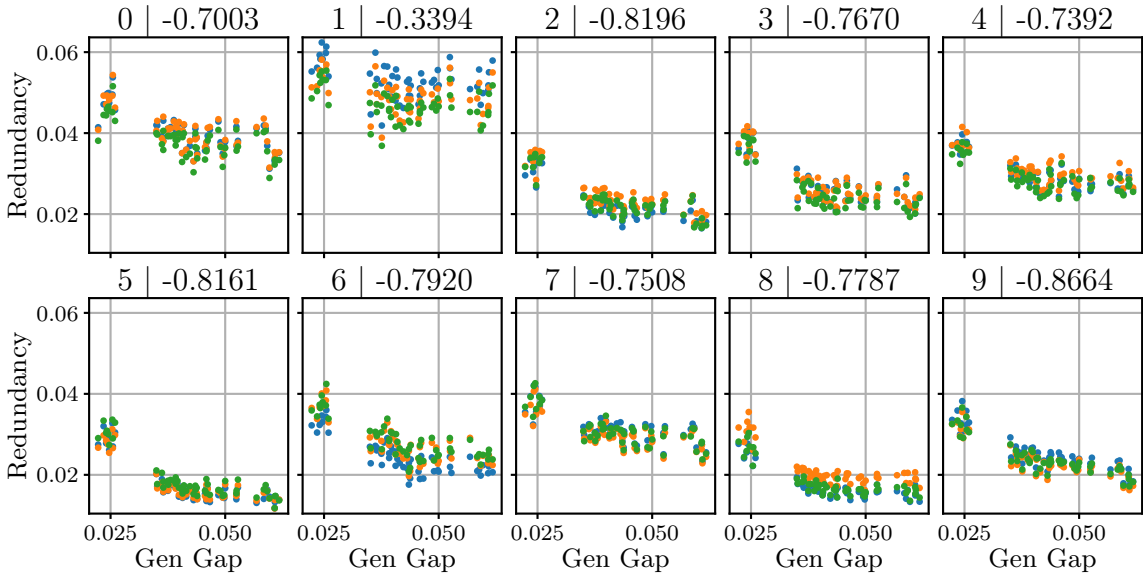

Figure A.15: Relationship between the mean indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, for each class and all networks in the 4×50-CE group. Different colours correspond to different subsets used for estimation. The title of each plot shows the corresponding class label and the average Pearson correlation coefficient, across subsets, between the plotted quantities.

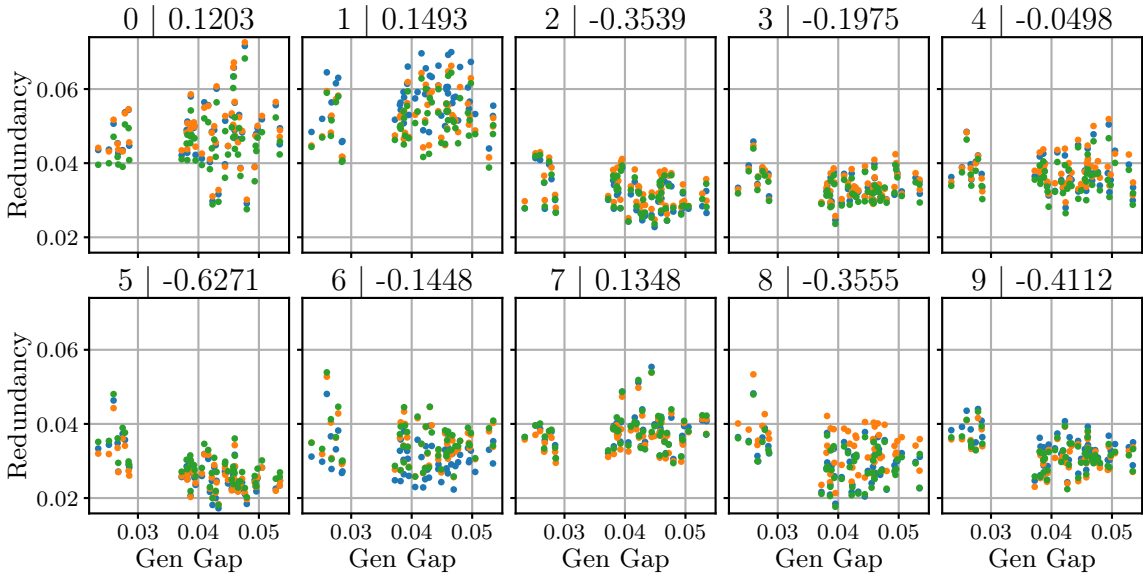

Figure A.16: Relationship between the mean indicator node-class redundancy across the hidden nodes of a network and the generalisation gap of the network, displayed as in Figure A.15 for all networks in the 4×50-MSE group.

Table A.1: Average Pearson correlation coefficient, across subsets, between the generalisation gap of a network and the maximum and mean node-class redundancy across the hidden nodes of the network, for networks with 50 nodes per hidden layer. The class "All" refers to the full node-class redundancy, while the remaining classes refer to the indicator node-class redundancy for the corresponding class.

Group	Class	Correlation with Maximum	Correlation with Mean
4×50 CE	All	-0.7161	-0.8376
	0	-0.7616	-0.7003
	1	-0.6480	-0.3394
	2	-0.8037	-0.8196
	3	-0.8372	-0.7670
	4	-0.7701	-0.7392
	5	-0.8506	-0.8161
	6	-0.7027	-0.7920
	7	-0.7847	-0.7508
	8	-0.8568	-0.7787
	9	-0.8237	-0.8664
4×50 MSE	All	-0.4149	-0.0771
	0	-0.5964	0.1203
	1	-0.5431	0.1493
	2	-0.5484	-0.3539
	3	-0.5450	-0.1975
	4	-0.4510	-0.0498
	5	-0.5624	-0.6271
	6	-0.4234	-0.1448
	7	-0.5154	0.1348
	8	-0.5923	-0.3555
	9	-0.6138	-0.4112

Appendix B

Publications

A pre-study performed as part of this degree resulted in a conference paper [10] co-authored by prof Marelie H. Davel and presented at the SACAIR 2020 conference. This paper is reproduced on the following page and forms the remainder of this document.

Exploring neural network training dynamics through binary node activations

Daniël G. Haasbroek[0000−0002−9974−3626] and
Marelie H. Davel[0000−0003−3103−5858]

Multilingual Speech Technologies (MuST), North-West University, South Africa; and
CAIR, South Africa.

Abstract. Each node in a neural network is trained to activate for a specific region in the input domain. Any training samples that fall within this domain are therefore implicitly clustered together. Recent work has highlighted the importance of these clusters during the training process but has not yet investigated their evolution during training. Towards this goal, we train several ReLU-activated MLPs on a simple classification task (MNIST) and show that a consistent training process emerges: (1) sample clusters initially increase in size and then decrease as training progresses, (2) the size of sample clusters in the first layer decreases more rapidly than in deeper layers, (3) binary node activations, especially of nodes in deeper layers, become more sensitive to class membership as training progresses, (4) individual nodes remain poor predictors of class membership, even if accurate when applied as a group. We report on the detail of these findings and interpret them from the perspective of a high-dimensional clustering process.

Keywords: Neural networks · Generalisation · Clustering

1 Introduction

Deep neural networks (DNNs) have been used to solve increasingly difficult tasks with increasingly high accuracy, and are particularly successful when modelling complex relationships from large quantities of high-dimensional data [8]. While DNN models perform extremely well given sufficient training data, the DNN training process itself is computationally inefficient, with model optimisation requiring expensive searches across a large number of interacting hyperparameters. This search process is mainly guided by heuristics, and by tracking performance on training and held-out validation sets, as no comprehensive theoretical framework yet exists with which to reason about the training process or the expected ability of the optimised models to generalise to out-of-sample data.

The generalisation ability of DNNs has been the topic of much controversy and has been studied from a variety of perspectives. Studies that aim to characterise and predict the generalisation ability of DNNs include approaches that consider the complexity of the hypothesis space, the geometry of the loss surface, characteristics of the classification margins, and statistical measures of uniform

stability and robustness. (See Section 2.1.) While each approach provides additional insight, a general analysis framework of DNN behaviour remains elusive. The 'apparent paradox' that DNNs are able to generalise well despite extremely large capacity remains largely unresolved [14].

With 'DNN behaviour' we refer to the performance of a DNN during and after training, as measured on different subsets of the data, both seen during training and not. Characterising this behaviour should allow us to reason about the training process and differences among networks, and to predict characteristics that lead to better performance.

One approach towards probing DNN behaviour is to consider nodes as individual classifiers, collaborating in solving a network-wide task [4]. A consequence of this analysis is that each individual node is implicitly associated with the specific cluster of samples for which it activates. These sample clusters then become useful elements in analysing network behaviour. Specifically, as any node delineates a region in input space for which it activates, any samples that fall within this region are in effect clustered together. These clusters are then used to refine the weights linked to the specific node, improving cluster boundaries.

While the potential importance of these clusters during the training process has been highlighted [4,27], their evolution during training has not yet been explored. Towards this goal, we train several ReLU-activated MLPs on a simple classification task (MNIST) and track the process whereby these sample sets are formed during training. The **main contributions** of this paper are the following:

1. We provide additional motivation for the potential importance of 'sample sets' (the set of samples that activates an individual node) and their corresponding sample-feature clusters when analysing DNN behaviour.
2. We report on the evolution of these clusters during the training process of different fully-connected feedforward networks, and demonstrate that a consistent training process emerges.
3. We interpret these findings in terms of a high-dimensional clustering process, which we conjecture to be a useful perspective when analysing the generalisation ability of neural networks.

We first present relevant background (Section 2), before discussing our motivation for studying sample clusters (Section 3). A description of the analysis approach follows in Section 4, with Section 5 measuring and reporting on the the process whereby sample sets evolve during training. Findings are discussed in Section 6, and summarised in Section 7.

2 Background

We briefly discuss the generalisation ability of DNNs and approaches towards studying this. We then review earlier work related to the role and analysis of sample sets, specifically focusing on sample sets as an element in probing the generalisation ability of DNNs.

2.1 Generalisation in DNNs

DNNs are well understood from the perspective of expressivity: it is known that even a shallow network with sufficient nodes and non-linear activations is able to approximate any function, given some caveats that typically do not apply to real-world data [24]. Similarly, gradient-based optimisation procedures are theoretically well-grounded, with the conditions for finding minima known. Being expressive and trainable, however, does not imply the ability to generalise well; this requires a model to accurately capture the 'true' underlying data distribution, identifying relevant features and their interaction throughout the input domain.

Statistical learning theory (SLT) suggests that the generalisation error of a trained model is bounded above by the complexity of the hypothesis space of the modelling method [2,11]. This bound explains the generalisation ability of many model types, in the sense that an increase in the complexity of the hypothesis space, beyond the amount necessary to capture important relationships, leads to poorer generalisation of trained models [11]. When the complexity of the hypothesis space is higher than needed, good generalisation can still be obtained by introducing a preference for certain functions in the hypothesis space [10]. The use of such regularisation techniques also explains the generalisation ability of many model types [10,28].

For DNNs, neither the complexity of the hypothesis space of a particular network nor the use of regularisation techniques during training can adequately explain generalisation [28]. This was demonstrated in [28] and [26], among others, where the authors obtained good generalisation with networks that could, without any modification, fit random data easily, regardless of the use of regularisation techniques.

A significant body of work has studied generalisation in DNNs. In addition to a host of empirical studies [1,17,22,23], we highlight four approaches:

- **Geometry of the loss landscape.** It has been argued that the smoothness of the loss landscape and, specifically, the flatness of minima can lead to better generalisation [12,15]. This follows the intuition that, under these conditions, an applicable minimum should be more easily accessible during gradient descent, and small perturbations in either input or parameter space should not influence model behaviour. In high-dimensional space, however, it is extremely difficult to obtain a consistent perspective of the error surface, and it has been shown that this error surface can fairly easily be manipulated with little effect on generalisation [5].
- **Statistical measures.** Both the stability of the training process (stability when trained on different datasets) and its robustness (expected behaviour when trained on all possible datasets) have produced insights into the generalisation behaviour of DNNs [9,25,18]. Kawaguchi et al. [14] argue convincingly that there is no paradox when applying such measures from SLT to DNNs; rather, their direct applicability is restricted.
- **Complexity of the hypothesis space.** Complexity can be reduced through regularisation (as discussed above) or through inducing sparsity. With a

smaller set of trainable parameters, prediction accuracy is expected to be more stable, as justified by SLT [21]. To date, sparsity measures have been more useful in improving the computational cost and interpretability of networks than in predicting generalisation ability [7,19].
- **Margin distributions.** These approaches study the decision margin in order to explain generalisation ability, as has been successfully done with linear models such as support vector machines (SVMs). Results related to DNNs [6,13] are promising, with [13], specifically, demonstrating that a linear model, trained on the margin distributions of numerous DNNs, can predict generalisation error.

These approaches tend to consider the network as a whole, with less attention paid to the role of the individual subcomponents — an approach taken in [4], and discussed in more detail below.

2.2 Sample sets

Simultaneously introduced in [3,4,26], the term 'sample set' refers to the node-specific set of samples that activate a given node. The initial definition arose in the context of a *ReLU-activated*, fully-connected feedforward network applied to a classification task; this is also the context we use for the current discussion.

In [4], a DNN is viewed as consisting of layers of local classifiers, collaborating to solve the overall classification task. During gradient descent, weights linked to a node are only updated based on those samples that activate the node. Gradient descent can then be viewed as a two-step process: (1) during the forward pass, sample sets are created; (2) during the backward pass and parameter-update step, the sample sets are refined [4]. This refinement process is both locally and globally aware: at the local level, only selected samples (those in the sample set) are utilised to update local parameters; globally, the loss value associated with each sample takes all network parameters into account [4].

In a related study [26], extended in [27], sample-set composition is analysed in the presence of different types of noise. Interestingly, if the features of training samples are corrupted (in contrast with corrupting labels) high levels of noise — up to 90% for the tasks studied — can be absorbed by the models, without causing any detrimental effect on classification performance. Probing this behaviour revealed that nodes tend to have sample sets that contain either true or corrupted samples, rather than both. Features attuned to the noisy samples, therefore, have almost no effect on noiseless test results. This provides a simple mechanism for preventing additional parameters from hurting performance: additional parameters create an increased number of sub-components with which to isolate detrimental samples from the rest of the training data, without fundamentally changing the way in which the uncorrupted samples are modelled [27].

In both [26] and [3] it is noted that the sample set of any node is fully described by that node's fan-in weight vector; if the activation vector in the prior layer is aligned with this weight vector (if their dot product is positive), the node will activate, otherwise not. The weight vector, therefore, creates a

boundary that separates samples included in the current sample set from those excluded. Finally, in [4] it is shown that, in deeper layers, sample sets become class-sensitive, containing either almost all or almost none of the samples of any class. This behaviour is consistent across a range of architectures [4].

In summary, the above findings indicate that the training process creates clusters of samples with very specific characteristics. This points towards sample sets as a useful element in understanding network behaviour, a view that we explore further in Section 3.

3 Motivation for exploring sample-feature clusters

As discussed in Section 2.2, sample sets identify regions in the input space where specific features produce coherent behaviour. This can also be understood by viewing the process whereby samples are included or excluded from these sets. As in [4], we focus on ReLU-activated feedforward networks and define[1] the sample set $\hat{S}_{b,l,j}$ at node j of layer l as those samples in batch b that produce a positive activation value at node j. For any sample in the sample cluster, the node j in layer l can be connected to an arbitrary number of active nodes in layer $l+1$. By selecting one active node per layer, the weights connecting these active nodes can be used to define an active path $p = \{w_{p_1}, w_{p_2}, \ldots, w_{p_{N-l}}\}$ associated with a specific sample, starting at layer l and ending at a node in the output layer N.

During standard SGD training, the sample set (and only the sample set) influences the weight update. If we initially limit our analysis to networks with no bias values beyond the first layer, the weight update equation simplifies considerably. As derived in [4], the SGD weight update $\delta w_{l,j,i}$ for the weight connecting node i to node j at layer l is then given by

$$\delta w_{l,j,i} = -\eta \sum_{\mathbf{s}\in\hat{S}_{b,l,j}} \left[z^{\mathbf{s}}_{l-1,i} \sum_{p\in P^{\mathbf{s}}_j} \left(\lambda^{\mathbf{s}}_p \prod_{k=1}^{N-l} w_{p_k} \right) \right] , \tag{1}$$

where the superscript $\mathbf{s}$ indicates sample-specific values, η indicates the learning rate, $z^{\mathbf{s}}_{l-1,i}$ indicates the post-activation value at node i in layer $l-1$, $P^{\mathbf{s}}_j$ indicates the set of all active paths linking node j to the output layer, and $\lambda^{\mathbf{s}}_p$ indicates the derivative of the loss function with respect to the network output.

This sample-specific weight update can be expressed in terms of the *node-supported cost*, a scalar value that represents the portion of the final cost that can be attributed to all active paths emanating from node j, when processing sample $\mathbf{s}$ [3]. Specifically, the sample-specific node-supported cost at layer l, node j can be defined as

$$\phi^{\mathbf{s}}_{l,j} = \sum_{p\in P^{\mathbf{s}}_j} \left(\lambda^{\mathbf{s}}_p \prod_{k=1}^{N-l} w_{p_k} \right) . \tag{2}$$

[1] Note that we follow the derivations from [4] but use a different notation, for better clarity.

The update to the weight vector $\mathbf{w}_{l,j}$ feeding into node j at layer l is then given by

$$\delta \mathbf{w}_{l,j} = -\eta \sum_{\mathbf{s} \in \hat{S}_{b,l,j}} \mathbf{z}^{\mathbf{s}}_{l-1} \phi^{\mathbf{s}}_{l,j} \,. \tag{3}$$

We first consider a setup where all layers prior to l are frozen, i.e. earlier weights and consequently earlier activation values are not allowed to change. Note that this is the true situation at the first hidden layer only. Let all symbols (including sample sets) reflect the values *prior to* the weight update. Then it can be shown that any single sample $\mathbf{t}$ will be included in the sample cluster if

$$\mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{w}_{l,j} > \eta \sum_{\mathbf{s} \in \hat{S}_{b,l,j}} \left(\mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{z}^{\mathbf{s}}_{l-1} \right) \phi^{\mathbf{s}}_{l,j} \tag{4}$$

and removed otherwise. If we now define $\Sigma^{\mathbf{t}}_{l,j}$ as the net cost of the sample set at node j (layer l) as aligned with vector $\mathbf{z}^{\mathbf{t}}_{l-1}$, that is

$$\Sigma^{\mathbf{t}}_{l,j} = \sum_{\mathbf{s} \in \hat{S}_{b,l,j}} \left(\mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{z}^{\mathbf{s}}_{l-1} \right) \phi^{\mathbf{s}}_{l,j} \,, \tag{5}$$

we can derive the precise conditions for which a sample $\mathbf{t}$ not previously in the cluster will be added:

$$\mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{w}_{l,j} \leq 0; \qquad \Sigma^{\mathbf{t}}_{l,j} < 0; \qquad \left| \mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{w}_{l,j} \right| < \eta \left| \Sigma^{\mathbf{t}}_{l,j} \right| \,, \tag{6}$$

or a member sample $\mathbf{t}$ (previously in the cluster) removed:

$$\mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{w}_{l,j} > 0; \qquad \Sigma^{\mathbf{t}}_{l,j} > 0; \qquad \left| \mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{w}_{l,j} \right| \leq \eta \left| \Sigma^{\mathbf{t}}_{l,j} \right| \,. \tag{7}$$

The absolute value signs are used here to emphasise that it is the magnitude of the values that is important. In effect, a margin is created around the decision boundary (where $\mathbf{z}^{\mathbf{t}}_{l-1} \cdot \mathbf{w}_{l,j} = 0$), with the size of this boundary directly specified by the learning rate and the summed loss of all samples that activate the specific node. Only samples falling within this boundary will either be drawn in or excluded from the sample set during the update. (This concept is illustrated in Figure 8 in the Appendix.) Note that this boundary *estimates* the net win of including or excluding additional samples by measuring their alignment with all other loss-generating samples in the set. If the effect is not as anticipated, the boundary will be shifted again in the following update. In this process, the boundary forms by separating samples that do not have coherent loss behaviour. Also note that a larger loss value implies that Equation 4 will be less strict, and, thus, samples that are less aligned with the weight vector might be accepted in cases where this would not have happened with a lower node-specific loss.

This process creates natural boundaries in the input space, separating areas that will benefit from being modelled separately. We illustrate this with an example: synthetic 1-dimensional data is generated, matching an underlying distribution as illustrated in Figure 1. A small network (1 hidden layer of 30 nodes)

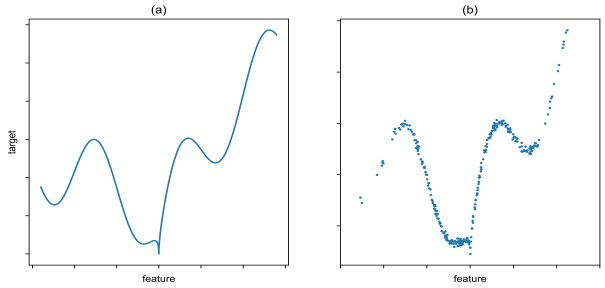

Fig. 1. 1-dim. regression example: (a) underlying distribution, (b) generated data.

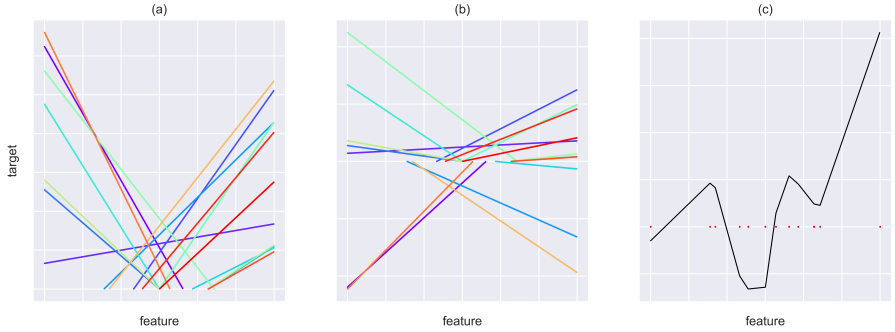

Fig. 2. 1-dim. regression example: (a) fan-in weight vectors, (b) vectors after activation and scaling, (c) weighted sum at target. Red dots indicate sample-set boundaries.

is trained to solve the regression task, and the resulting weight vectors are plotted as shown in Figure 2. Each fan-in vector at a node can be depicted as a line, based on the weight and bias value at that node. After ReLU-activation, all negative values are suppressed, and in the next layer, values are scaled based on fan-out weights. Finally, all contributions are summed in order to estimate the target value.

Once the sample-set boundaries have been drawn, scaling intermediary results to solve the overall task becomes a straightforward process; finding these boundaries is not. We propose that the heart of the training process can be studied through this clustering process of grouping relevant samples in the context of relevant features.

4 Experimental approach

Having provided our motivation for studying sample sets, we now outline our approach to investigating the dynamics of sample sets during training. We are interested in the size of the sample sets, the fraction of samples of a given class

that is included in a sample set, how predictable sample-set membership is given class identity, and how informative sample-set membership is of class identity. These are intuitive concepts that we formalise below.

4.1 Setup

We train several networks, with the number of layers (excluding the input layer) being either 2, 5, or 9, and the number of nodes per hidden layer being either 20, 80, or 320. Bias parameters are added to the first hidden layer of each network. We use ReLU activation functions for the hidden layers. All networks are trained for 200 epochs on MNIST using the Adam optimisation algorithm [16] with a minibatch size of 60. Since we do not aim to maximise the performance of any of the networks, we train all the networks with the default optimiser hyperparameters suggested in [16] ($\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$). We train the networks with identity activation functions at the output layer and mean-squared-error loss. We repeat the training with cross-entropy loss (softmax activation functions followed by negative log-likelihood loss). The training is performed for 3 different initialisation seeds. For all networks, the training converges.

4.2 Measurements

We measure different characteristics of sample sets during training. The per-class *sample set* of node i in hidden layer l for class c, and the entire sample set of the same node are, respectively,

$$\hat{S}_{l,i}^{(c)} = \left\{ \mathbf{s} : z_{l,i}^{\mathbf{s}} > 0, \mathbf{s} \in S_c \right\} ; \qquad \hat{S}_{l,i} = \bigcup_{c \in C} \hat{S}_{l,i}^{(c)} , \tag{8}$$

where S_c is the set of samples belonging to class c, and $z_{l,i}^{\mathbf{s}}$ is the activation of node i in hidden layer l for sample $\mathbf{s}$ [3,4,26]. At several points in the training process, we calculate the per-class *sample-set size* $|\hat{S}_{l,i}^{(c)}|$ for every node i in every hidden layer l for each class c.

We refer to the fraction of samples of a given class that is included in a sample set as the class-specific *activation fraction*, and define it for node i in hidden layer l for class c as

$$f_{l,i}^{(c)} = \frac{\left|\hat{S}_{l,i}^{(c)}\right|}{|S_c|} . \tag{9}$$

We can view the activation of node i in hidden layer l as a random variable $Z_{l,i}$, defining $\hat{Z}_{l,i} = 0$ if $Z_{l,i} \leq 0$ and $\hat{Z}_{l,i} = 1$ if $Z_{l,i} > 0$. If we also view the class of the input sample as a random variable Y, then we can approximate

$$P\left(\hat{Z}_{l,i} = 0, Y = c\right) \approx e_{l,i}(0,c) = \frac{|S_c| - \left|\hat{S}_{l,i}^{(c)}\right|}{|S|} ; \tag{10}$$

$$P\left(\hat{Z}_{l,i} = 1, Y = c\right) \approx e_{l,i}(1,c) = \frac{\left|\hat{S}_{l,i}^{(c)}\right|}{|S|} , \tag{11}$$

where S is the set of all samples. We approximate the conditional entropy of $\hat{Z}_{l,i}$ given Y and that of Y given $\hat{Z}_{l,i}$, for nodes with $|\hat{S}_{l,i}| \neq 0$ and $|\hat{S}_{l,i}| \neq |S|$, as

$$H\left(\hat{Z}_{l,i} \,|\, Y\right) \approx -\sum_{c \in C} \sum_{\hat{z} \in \{0,1\}} e_{l,i}(\hat{z}, c) \log \left(\frac{e_{l,i}(\hat{z}, c)}{\sum\limits_{x \in \{0,1\}} e_{l,i}(x, c)} \right); \tag{12}$$

$$H\left(Y \,|\, \hat{Z}_{l,i}\right) \approx -\sum_{\hat{z} \in \{0,1\}} \sum_{c \in C} e_{l,i}(\hat{z}, c) \log \left(\frac{e_{l,i}(\hat{z}, c)}{\sum\limits_{x \in C} e_{l,i}(\hat{z}, x)} \right), \tag{13}$$

where we set $0 \log(0) = 0$ [20]. Based on this, we define the *predictability* of node i in hidden layer l as

$$p_{l,i} = 1 + \frac{1}{\log(2)} \sum_{c \in C} \sum_{\hat{z} \in \{0,1\}} e_{l,i}(\hat{z}, c) \log \left(\frac{e_{l,i}(\hat{z}, c)}{\sum\limits_{x \in \{0,1\}} e_{l,i}(x, c)} \right), \tag{14}$$

and the *informativeness* as

$$u_{l,i} = 1 + \frac{1}{\log(|C|)} \sum_{\hat{z} \in \{0,1\}} \sum_{c \in C} e_{l,i}(\hat{z}, c) \log \left(\frac{e_{l,i}(\hat{z}, c)}{\sum\limits_{x \in C} e_{l,i}(\hat{z}, x)} \right), \tag{15}$$

where we again set $0 \log(0) = 0$ [20]. We do not define $p_{l,i}$ or $u_{l,i}$ for nodes with $|\hat{S}_{l,i}| = 0$ or $|\hat{S}_{l,i}| = |S|$, that is, for nodes that are always inactive (dead nodes) or always active (bias nodes). The predictability of a node is the difference between the maximum possible entropy of $\hat{Z}_{l,i}$ (based on the number of possible values of $\hat{Z}_{l,i}$) and the entropy of $\hat{Z}_{l,i}$ given Y, expressed as a fraction of the maximum possible entropy of $\hat{Z}_{l,i}$. Informally, this is proportional to the average amount of information known about $\hat{Z}_{l,i}$ given only an observation of Y. The informativeness indicates similar information about Y given $\hat{Z}_{l,i}$.

We calculate $f_{l,i}^{(c)}$, $p_{l,i}$, and $u_{l,i}$ at several logarithmically spaced points in the training process on a validation set of 12000 samples. We then aggregate these values as follows:

$$f_{l,i} = \frac{1}{|C|} \sum_{c \in C} f_{l,i}^{(c)}; \qquad f_l = \frac{1}{\left|N_a^{(l)}\right|} \sum_{i \in N_a^{(l)}} f_{l,i}; \tag{16}$$

$$p_l = \frac{1}{\left|N_b^{(l)}\right|} \sum_{i \in N_b^{(l)}} p_{l,i}; \qquad u_l = \frac{1}{\left|N_b^{(l)}\right|} \sum_{i \in N_b^{(l)}} u_{l,i}, \tag{17}$$

where $N_a^{(l)}$ is the set of nodes in hidden layer l for which $|\hat{S}_{l,i}| \neq 0$, and $N_b^{(l)}$ is the set of nodes in hidden layer l for which $|\hat{S}_{l,i}| \neq 0$ and $|\hat{S}_{l,i}| \neq |S|$.

5 Results

Here, we highlight interesting patterns observed by presenting results that display typical behaviour. Any other results that do not follow these patterns are pointed out. The performance of all networks during training is similar to that shown in Figure 3. All networks with 80 or more nodes per hidden layer achieve error rates smaller than 5% on the validation set. Those with 20 nodes per hidden layer achieve error rates smaller than 7% on the validation set.

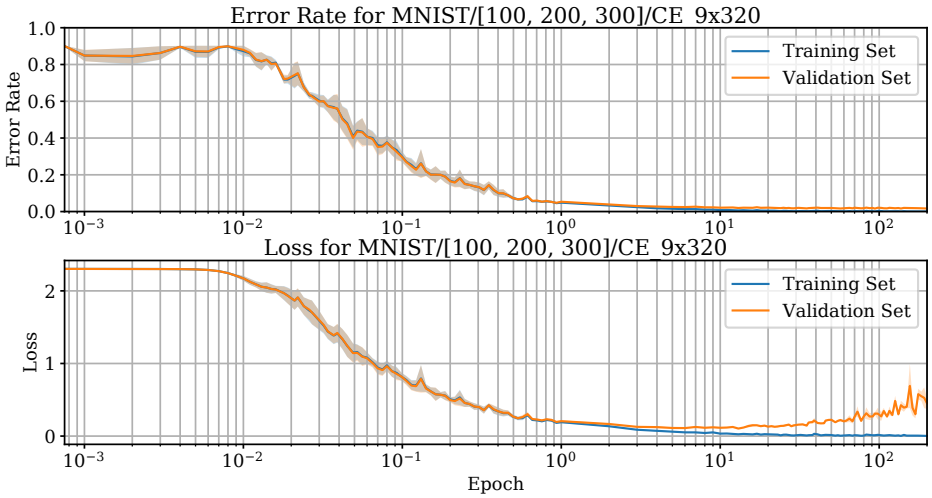

Fig. 3. Performance of 9×320 networks trained with cross-entropy loss, averaged across initialisation seeds. Shaded areas indicate the standard error.

5.1 Activation fraction

The activation fraction $f_{l,i}$ of the hidden nodes of a 9-layer network with 320 nodes per layer is shown in Figure 4. The average activation fraction f_l for the same architecture is shown in Figure 5. It is worth noting that the number of samples per class in the validation set is approximately constant across classes. As a result, $f_{l,i}$ is approximately equal to the fraction of *all* samples for which node i in layer l activates.

For most nodes, an increase in the sample-set size is observed very early in training. This indicates that the training process initially exposes most nodes to most samples. The nodes for which this does not hold activate for almost no samples during the entire training process, and, therefore, these nodes contribute very little to the network. The initial increase in sample-set size is followed by a gradual decrease for the rest of training. For networks trained with cross-entropy loss, this decrease is more rapid in shallower layers than in deeper layers. For

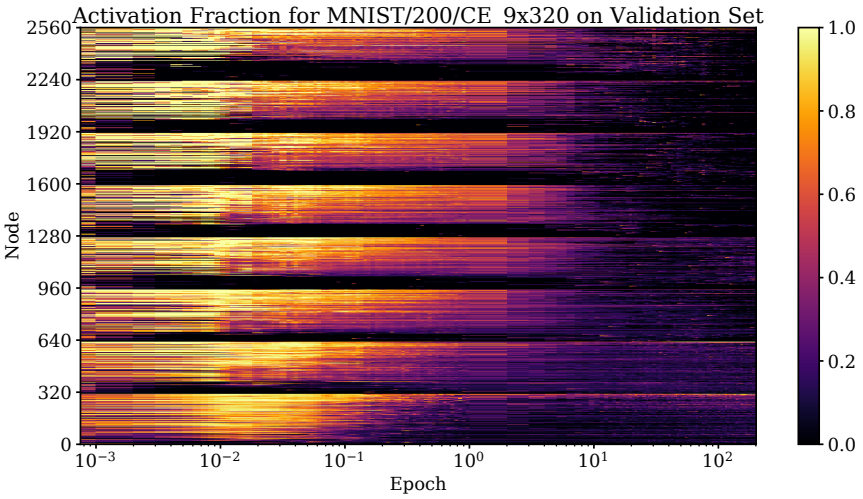

Fig. 4. Activation fraction for all hidden nodes in a 9×320 network trained with cross-entropy loss, calculated on the validation set. Nodes are numbered from shallowest to deepest.

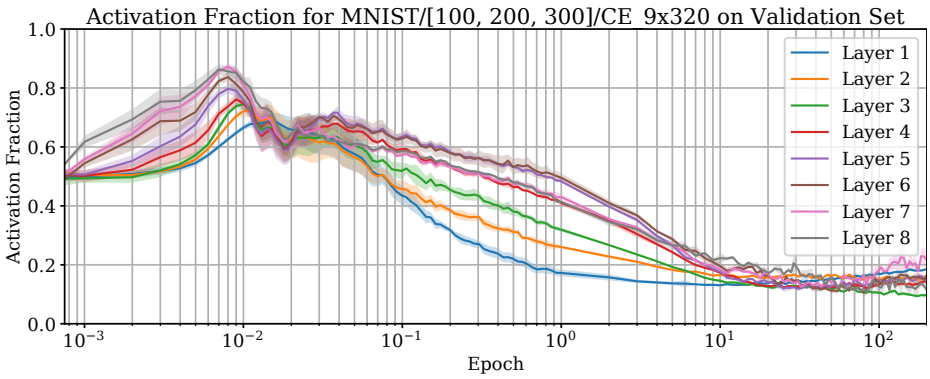

Fig. 5. Activation fraction averaged across nodes and initialisation seeds of 9×320 networks trained with cross-entropy loss, calculated on the validation set. Shaded areas indicate the standard error of the average across seeds. Layers are numbered from shallowest to deepest.

networks trained with MSE loss, this decrease is more rapid in the first hidden layer than in deeper layers.

Some results of individual networks are worth pointing out. For 9-layer networks trained with MSE loss, the sample sets of nodes in deeper layers contain almost all of the samples at the point where the sample sets are largest (see Figure 9 in the Appendix). For the 2-layer networks with 320 nodes per layer trained with MSE loss, the average activation fraction decreases during the entire training process (see Figure 10 in the Appendix). All other networks follow the same trends as in Figures 4 and 5.

5.2 Predictability

The average predictability p_l for a 9-layer network with 320 nodes per layer is shown in Figure 6. The average predictability of all layers increases at the start

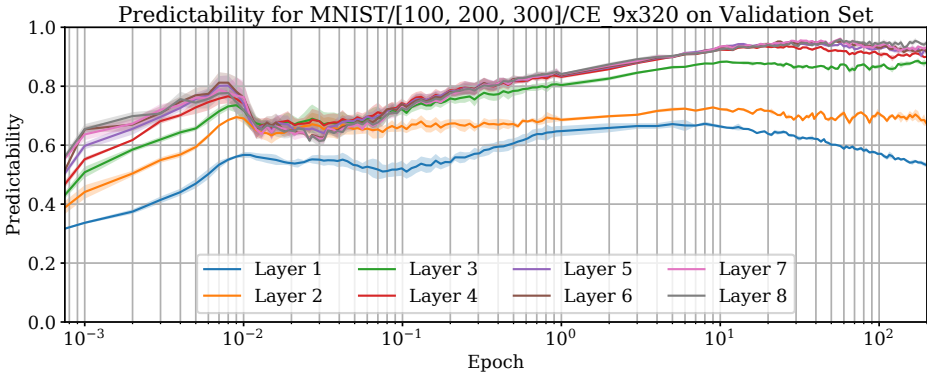

Fig. 6. Predictability averaged across nodes and initialisation seeds of 9×320 networks trained with cross-entropy loss, calculated on the validation set. Shaded areas indicate the standard error of the average across seeds. Layers are numbered from shallowest to deepest.

of training. For a few of the shallower layers, the predictability remains constant or decreases towards the end of training; for the rest of the layers, it continues to increase. The predictability of deeper layers is consistently higher than that of shallower layers. Not all results show the peaks that appear in Figure 6, and, therefore, we refrain from assigning meaning to them. These trends indicate that nodes, especially those in deeper layers, become increasingly more sensitive to class membership as training progresses. It also shows that nodes in deeper layers are more sensitive to class membership — a result that is confirmed by [4].

5.3 Informativeness

The average informativeness u_l for a 9-layer network with 320 nodes per layers is shown in Figure 7. For all layers, the informativeness increases and subsequently

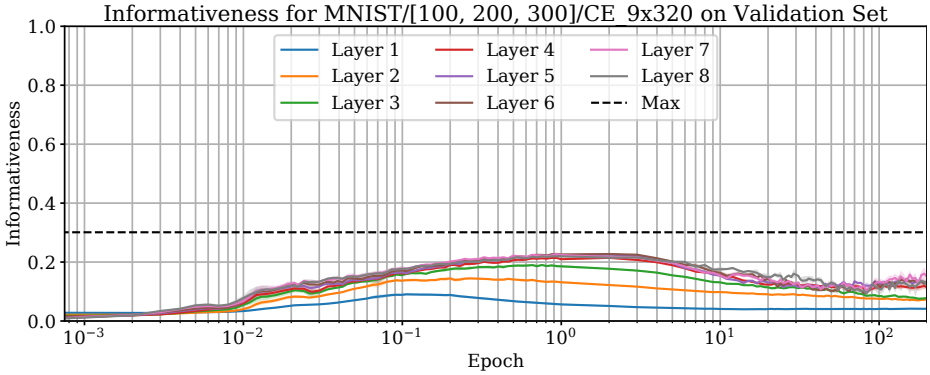

Fig. 7. Informativeness averaged across nodes and initialisation seeds of 9×320 networks trained with cross-entropy loss, calculated on the validation set. Shaded areas indicate the standard error of the average across seeds. Layers are numbered from shallowest to deepest. 'Max' indicates the maximum informativeness that can be achieved.

decreases slightly during training. Assuming that the 10 MNIST classes are perfectly balanced in the validation set, the maximum informativeness that can be achieved is $^{\log(2)}/_{\log(10)} \approx 0.3$, since the binary activation $\hat{Z}_{l,i}$ of a node can only have one of 2 values, but the class Y can have one of 10 values. The increase in informativeness shows that binary node activations become more indicative of class membership as training progresses. However, the amount of information required to establish the class with certainty means that the binary activation of any single node remains a poor predictor of class membership.

6 Discussion

To summarise our empirical findings: (1) Sample clusters initially increase in size and then decrease as training progresses. (2) Most nodes are exposed to most samples very early in the training process. (3) The point where the activation fraction starts decreasing overlaps with the point where network loss starts decreasing. (4) The size of sample clusters in the first layer decreases more rapidly than in deeper layers. (5) Binary node activations, especially of nodes in deeper layers, become more sensitive to class membership as training progresses. (6) Nodes in shallower layers tend to be less sensitive to class membership than nodes in deeper layers. (7) Binary node activations become slightly more indicative of class membership as training progresses, but remain poor predictors of class membership, even if accurate when applied as a group.

How do the above findings shed light on the creation of the sample clusters described in Section 3? When considering the training process, specific phases become evident: Upon initialisation, weights tend to point in arbitrary directions, and nodes in the network activate for samples found somewhere in the vicinity of their fan-in weight vector, creating the initial clusters. Since the loss is initially

very large, additional samples are quickly drawn into the sample set of each of the nodes, according to Equation 4. It is during this time that the activation fraction grows. (See Figure 5.) This process continues up to the point where the activation fraction reaches its maximum. At this point, most nodes are being exposed to most samples, and also to most features.

It is only as the loss starts decreasing (Figure 3) that the activation fraction also starts decreasing, with nodes becoming increasingly specific. As nodes become more specific, weights become attuned to solving smaller subtasks, specifically trying to address any unresolved samples in its own sample set. At this point, nodes actively start selecting features (and directions in feature space) that are the most appropriate for solving its own subtask.

This process continues up to convergence, occasionally displaying a ripple effect where we conjecture that clusters are being re-shuffled. During training, only samples that have not been fully resolved contribute to weight updates. Samples with zero loss are simply ignored, unless a change in the network increases the loss value for such a sample as a side effect, which may cause clusters to break apart or re-combine.

Taken together, this process describes a practical approach to solving a high-dimensional clustering task, and specifically, to finding combinations of samples and features that show coherent behaviour and can therefore be modelled together effectively. It is only in the first layer that the raw input features are used to form clusters; in later layers, this clustering occurs in the transformed space produced by the previous layer.

While the findings presented at the start of this section were empirically confirmed across different architectures, the above interpretation is currently to be considered conjecture, rather than fact. In our current work we are analysing each of these statements in more detail.

7 Conclusion

Motivated by the role that sample sets play in the SGD training process, we studied the evolution of sample sets throughout training for several ReLU-activated networks. Our experiments reveal a consistent training process, as summarised at the start of Section 6. We provide some insight into the SGD training process by interpreting these findings using the conditions under which samples are included or excluded from sample sets (Section 3), and by discussing how this could relate to a high-dimensional clustering process (Section 6).

The current analysis is restricted to ReLU-activated networks. As the analysis in [4], which also studied the binary behaviour of individual nodes, was successfully extended to sigmoid-activated networks, we expect to be able to extend this study to a more diverse set of architectures and datasets as well. Although we do not directly address the apparent generalisation 'paradox' or improve the training process, the presented analyses and interpretations shed light on the training process from an interesting perspective.

References

1. Arpit, D., Jastrzebski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M.S., Maharaj, T., Fischer, A., Courville, A.C., Bengio, Y., Lacoste-Julien, S.: A closer look at memorization in deep networks. In: Proceedings of the 34th International Conference on Machine Learning. pp. 233–242 (2017)
2. Bartlett, P.L., Mendelson, S.: Rademacher and gaussian complexities: Risk bounds and structural results. Journal of Machine Learning Research **3**, 463–482 (2002)
3. Davel, M.H.: Activation gap generators in neural networks. In: Proc. of the South African Forum for Artificial Intelligence Research FAIR. pp. 64–76. Cape Town, South Africa (12 2019)
4. Davel, M.H., Theunissen, M.W., Pretorius, A.M., Barnard, E.: DNNs as layers of cooperating classifiers. In: Thirty-Fourth AAAI Conference on Artificial Intelligence (2020)
5. Dinh, L., Pascanu, R., Bengio, S., Bengio, Y.: Sharp minima can generalize for deep nets. In: Proceedings of the 34th International Conference on Machine Learning. pp. 1019–1028 (2017)
6. Elsayed, G.F., Krishnan, D., Mobahi, H., Regan, K., Bengio, S.: Large margin deep networks for classification. In: Conference on Neural Information Processing Systems (2018)
7. Gale, T., Elsen, E., Hooker, S.: The state of sparsity in deep neural networks. arXiv Preprint **arXiv:1902.09574** (2019)
8. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, pp. 22–25. MIT Press (2016)
9. Hardt, M., Recht, B., Singer, Y.: Train faster, generalize better: Stability of stochastic gradient descent. In: Proceedings of The 33rd International Conference on Machine Learning. pp. 1225–1234 (2016)
10. Hastie, T., Tibshirani, R., Friedman, J.: Chapter 2: Overview of supervised learning. In: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, pp. 9–41. Springer, 2 edn. (2017)
11. Hastie, T., Tibshirani, R., Friedman, J.: Chapter 7: Model assessment and selection. In: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, pp. 219–259. Springer, 2 edn. (2017)
12. Hochreiter, S., Schmidhuber, J.: Flat minima. Neural Computation **9**(1), 1–42 (1997)
13. Jiang, Y., Krishnan, D., Mobahi, H., Bengio, S.: Predicting the generalization gap in deep networks with margin distributions. In: International Conference on Learning Representations (2019)
14. Kawaguchi, K., Kaelbling, L.P., Bengio, Y.: Generalization in deep learning. In: Mathematics of Deep Learning. Cambridge University Press, to be published
15. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. In: International Conference on Learning Representations (2017)
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015)
17. Krueger, D., Ballas, N., Jastrzebski, S., Arpit, D., Kanwal, M.S., Maharaj, T., Bengio, E., Fischer, A., Courville, A.C.: Deep nets don't learn via memorization. In: International Conference on Learning Representations (2017)
18. Kuzborskij, I., Lampert, C.H.: Data-dependent stability of stochastic gradient descent. In: Proceedings of the 35th International Conference on Machine Learning. pp. 2815–2824 (2018)

19. Loroch, D.M., Pfreundt, F.J., Wehn, N., Keuper, J.: Sparsity in deep neural networks – An empirical investigation with TensorQuant. In: ECML PKDD 2018 Workshops. pp. 5–20 (2019)
20. MacKay, D.J.: Chapter 2: Probability, entropy, and inference. In: Information Theory, Inference, and Learning Algorithms, pp. 22–46. Cambridge University Press (2005)
21. Maurer, A., Pontil, M.: Structured sparsity and generalization. Journal of Machine Learning Research **13**, 671–690 (2012)
22. Neyshabur, B., Bhojanapalli, S., McAllester, D., Srebro, N.: Exploring generalization in deep learning. In: Conference on Neural Information Processing Systems (2017)
23. Novak, R., Bahri, Y., Abolafia, D.A., Pennington, J., Sohl-Dickstein, J.: Sensitivity and generalization in neural networks: an empirical study. In: International Conference on Learning Representations (2018)
24. Pinkus, A.: Approximation theory of the mlp model in neural networks. Acta Numerica **8**, 143–195 (1999)
25. Sokolić, J., Giryes, R., Sapiro, G., Rodrigues, M.R.D.: Robust large margin deep neural networks. IEEE Transactions on Signal Processing **65**(16), 4265–4280 (2017)
26. Theunissen, M.W., Davel, M.H., Barnard, E.: Insights regarding overfitting on noise in deep learning. In: Proc. of the South African Forum for Artificial Intelligence Research FAIR. pp. 49–63. Cape Town, South Africa (12 2019)
27. Theunissen, M.W., Davel, M.H., Barnard, E.: Benign interpolation of noise in deep learning. South African Computer Journal (12 2020)
28. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires re-thinking generalization. In: International Conference on Learning Representations (2017)

A Illustration of Equation 6 and 7

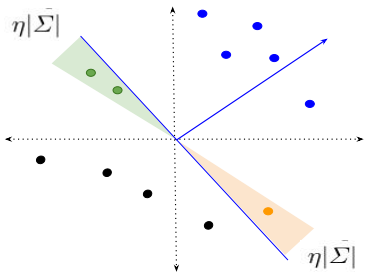

Fig. 8. The size of $\eta|\Sigma|$ determines the area where additional samples will be included in (green) or excluded from (orange) the sample set, after the next weight update. Weight vector is shown in blue, with current boundary line indicated, also in blue. The sample-set membership of other samples already in the sample set (blue) and outside the sample set (black) are not affected.

B Additional results

Figure 9 shows the activation fraction for networks in which the nodes in deeper layers have sample sets that contain almost all of the samples at the point where the sample sets are largest. This pattern holds for all 9-layer networks trained with MSE loss.

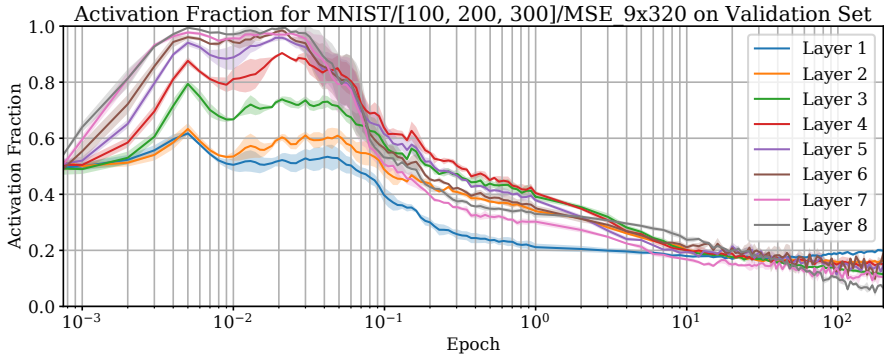

Fig. 9. Activation fraction averaged across nodes and initialisation seeds of 9×320 networks trained with mean-squared-error loss, calculated on the validation set. Shaded areas indicate the standard error of the average across seeds. Layers are numbered from shallowest to deepest.

Figure 10 shows the activation fraction for the networks in which an increase in average activation fraction is not observed. This only holds for 2-layer networks with 320 nodes per layer trained with MSE loss.

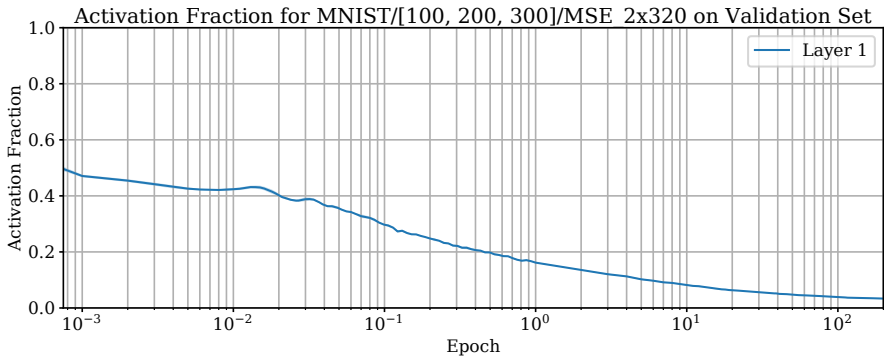

Fig. 10. Activation fraction averaged across nodes and initialisation seeds of 2×320 networks trained with mean-squared-error loss, calculated on the validation set. Shaded areas indicate the standard error of the average across seeds.