Structural Degeneracy in Neural Networks

Minor Thesis
submitted in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Matthew Farrugia-Roberts*
School of Computing and Information Systems
The University of Melbourne

Supervised by

Daniel Murfet
School of Mathematics and Statistics
The University of Melbourne

Nic Geard
School of Computing and Information Systems
The University of Melbourne

Submitted: October, 2022. Minor revision: December, 2022.

* matthew@far.in.net
0000-0002-9077-109X

Abstract

Neural networks learn to implement input–output functions based on data. Their ability to do so has driven applications in many task domains. However, a solid theoretical understanding of deep learning remains elusive. For example, classical theoretical frameworks fail to account for the performance of large neural networks despite their capacity to implement overly complex functions. There is a need for basic theoretical research clarifying the relationship between structure, function, and complexity in neural networks.

For almost all neural networks, the relationship between structure and function is well understood: all parameters implementing a given function are related by simple operations such as exchanging the weights of units. However, for the remaining *degenerate* neural networks, there are additional parameters implementing the same function. Since some of these additional parameters correspond to smaller, less complex neural networks, degenerate neural networks are positioned to play a crucial role in understanding neural network learning. However, prior work investigating neural network structures has largely emphasised the generic non-degenerate case.

In light of this situation, I present a comprehensive theoretical investigation of structural degeneracy in a simple family of neural networks (single-hidden-layer biased hyperbolic tangent networks). I develop an algorithmic framework for analysing degeneracy, including an ideal measure of degeneracy called the *rank* of a neural network (the minimum number of hidden units required to implement an identical function). Using this framework, I characterise the class of functionally equivalent parameters including in the degenerate case, extending prior work that has only considered the non-degenerate case. I show that in the degenerate case, the functional equivalence class is piecewise linear path connected. Moreover, I study a measure of approximate degeneracy, the *parametric approximate rank,* based on proximity to low-rank parameters. Drawing on computational complexity theory, I show that determining such proximity is an $\mathcal{NP}$-complete problem.

The insights into structural degeneracy developed in this thesis have the potential to clarify topics of current interest in deep learning. More broadly, this thesis lays a foundation for future work developing efficient measures of degeneracy and empirically investigating the role of degeneracy in deep learning.

Declaration

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

I certify that the thesis is between 25,000 and 30,000 words in length, including text and headings from the start of the introduction to the end of the conclusion, while excluding front matter; references; appendices; text in figures, tables, and captions; mathematical symbols; and citations.

I certify that where necessary I have received clearance for this research from the University's ethics committee and have submitted all required data to the School. [N/A]

Acknowledgements

I acknowledge the Wadawurrung people and the Wurundjeri people of the Kulin Nation as the Traditional Custodians of the lands on which I completed this research. I pay my respects to their Elders past, present, and emerging.

My work towards this degree has been partially financially supported by the Melbourne School of Engineering Foundation Scholarship and the Long-Term Future Fund.

I thank my primary supervisor, Dan Murfet, for generous support and thoughtful advice—both technical and personal—during the formulation and completion of my thesis; for modelling excellence in supervision and research; and for pushing me to do the best work I can do. I thank my secondary supervisor, Nic Geard, for much administrative guidance throughout the completion of my project and degree. I thank my teacher and colleague, Harald Søndergaard, for motivation and mentorship throughout my degree, and for always taking an interest in my work.

I thank the students of the Melbourne Deep Learning Group for helpful presentations, discussions, and suggestions during the project—thank you, Spencer Wong, Edmund Lau Tiew Hong, Adrian Xu, Rohan Hitchcock, Liam Carroll, and Tom Waring. I thank my peers from the School of Computing and Information Systems who offered LaTeX advice, joined me for daily online writing sprints, encouraged me, and helped me stick to my deadlines—thank you, Amy Mendelsohn, Stewart Webb, James Barnes, Albert Darmawan, Liam Saliba, and Alan Lewis. For helpful questions, discussions, and encouragement, I thank the team from the Centre for Brain, Mind and Markets, including Carsten Murawski, Nitin Yadav, Harvey Huang, Karlo Doroc, and James Fodor. For feedback on my written work I thank Harald Søndergaard, Alan Lewis, Dan Murfet, Nic Geard, and Athir Saleem.

I thank Annie-Emma Italiano for everything else.

Contents

Chapter 1

Introduction

I felt that he must have some solid grounds for the assured and easy demeanour with which he treated the singular mystery which he had been called upon to fathom.

—Dr. John H. Watson, “A Case of Identity,”
The Adventures of Sherlock Holmes

In machine learning, one leverages data to automatically construct software systems. The subfield of deep learning studies systems called neural networks, comprising interconnected layers of simple computational units. Neural networks learn to implement input–output functions by forming hierarchical representations of patterns in data (Schmidhuber, 2015; LeCun et al., 2015; Goodfellow et al., 2016).

In practice, neural networks are capable of learning complex functions from many kinds of data. Deep learning has thus contributed to several recent milestones in artificial intelligence (e.g., Mnih et al., 2015; Silver et al., 2018; Brown et al., 2020; Jumper et al., 2021). Moreover, deep learning has driven a broadening range of applications in industrial, scientific, medical, and consumer technology (Jordan and Mitchell, 2015)—with correspondingly increasing real-world impacts.

At the same time, the mechanisms behind the learning capabilities of neural networks remain poorly understood. For example, classical theoretical frameworks fail to account for empirically observed capabilities, due in part to the capacity of neural networks to represent patterns of arbitrary complexity (Zhang et al., 2017; 2021). Moreover, the vast scale of practical neural networks raises the financial and environmental costs of deep learning (see, e.g., Strubell et al., 2019; Ahmed and Wahed, 2020; Crawford, 2021), and makes the resulting learned software systems difficult for humans to understand (see, e.g., Lipton, 2018; Rudin, 2019; Molnar, 2022). To address these and other challenges, there is a pressing need for basic theoretical research that aims to enhance our understanding of neural network structure, function, and learning.

Neural network structure and function

A neural network can be understood at one level as a graph data structure, comprising simple computational units connected by weighted edges. In practice, this structure is encoded as a *parameter*—a high-dimensional vector specifying the edge weights and unit properties. At another level, a neural network implements a mathematical *function,* with the units propagating input signals through the graph to produce outputs. Deep learning studies methods for adaptively deriving from data ("learning") a parameter such that the implemented function performs some complex task, such as classifying images (e.g., Cireşan et al., 2011; Krizhevsky et al., 2012) processing natural language (e.g., Brown et al., 2020) or making sequential decisions (e.g., Mnih et al., 2015; Silver et al., 2018).

Of central importance in forming a deeper understanding of neural networks is studying the link between neural network structure and function. This is the topic of *neural network geometry,* on which there is a large existing literature. For example, theorists have shown that neural networks are capable of approximately implementing many functions (see universal approximation results, e.g., Cybenko, 1989; Hornik et al., 1989; Leshno et al., 1993), including highly complex functions given enough computational units (see, e.g., Barron, 1993; Murata, 1996; Mhaskar, 1996; Montúfar et al., 2014).

A further central question in neural network geometry is the *functional equivalence* question: when do two neural network structures implement the same function? Functionally equivalent parameters can be generated by simple operations, such as the unit exchange and negation operations exemplified in Figure 1.1. Moreover, for many classes of neural networks, theorists have shown that there are *no other* functionally equivalent structures—as long as the neural networks satisfy various *non-degeneracy conditions* (e.g., Sussmann, 1992; Chen et al., 1993; Fefferman, 1994; Phuong and Lampert, 2020).

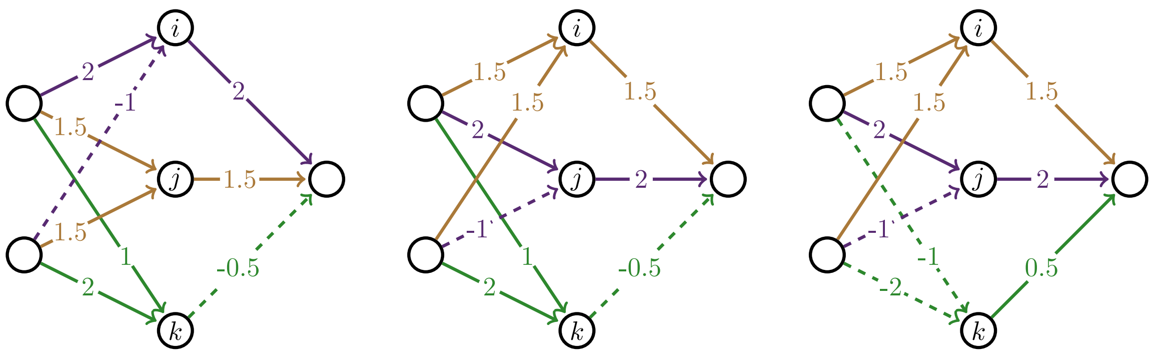


Figure 1.1: Three functionally equivalent neural network structures. See Sections 2.2 and 3.2 for a detailed introduction to neural networks. Left–middle: the weights for units i and j have been exchanged. Middle–right: the weights for unit k have been negated. Assuming the units respond as an odd function of their inputs (such as, for example, with the hyperbolic tangent function) these three structures implement the same function.

Degenerate neural networks

The aforementioned non-degeneracy conditions include almost all neural networks. However, there are certain *degenerate neural networks* for which there is a richer set of equivalent structures (beyond just those reachable by unit exchanges and negations, or similar basic operations), as exemplified in Figure 1.2.

Most existing work on the functional equivalence question has excluded the degenerate case. A commonly cited justification is that degenerate structures are *atypical:* if a neural network parameter is selected at random, there is effectively zero probability that it will be degenerate (formally, the set of degenerate parameters is a measure zero subset of the parameter space, or, roughly, it has zero volume; see, e.g., Phuong and Lampert, 2020).

However, this atypicality does *not* mean degenerate neural networks are irrelevant to deep learning. First, the learning process applies a non-random selection pressure on neural network structures, so one cannot rule out *a priori* atypical structures among *learned* networks. Second, degenerate neural network geometry can have practical consequences for non-degenerate networks that are, nevertheless, *approximately* degenerate.

Degenerate neural networks may indeed play an important role in deep learning. For some classes of neural networks, degeneracy corresponds to *non-minimality,* that is, degenerate structures implement functions that could be implemented with fewer units (e.g., Sussmann, 1992). Degeneracy could therefore help explain the empirical success of learned neural networks despite their apparent complexity.

The occurrence of degenerate structures among learned neural networks is an empirical hypothesis that should be tested, and the role played by degeneracy in deep learning remains to be clarified. The first step in this research direction is to establish a deeper understanding of structural degeneracy than is currently available in the literature.

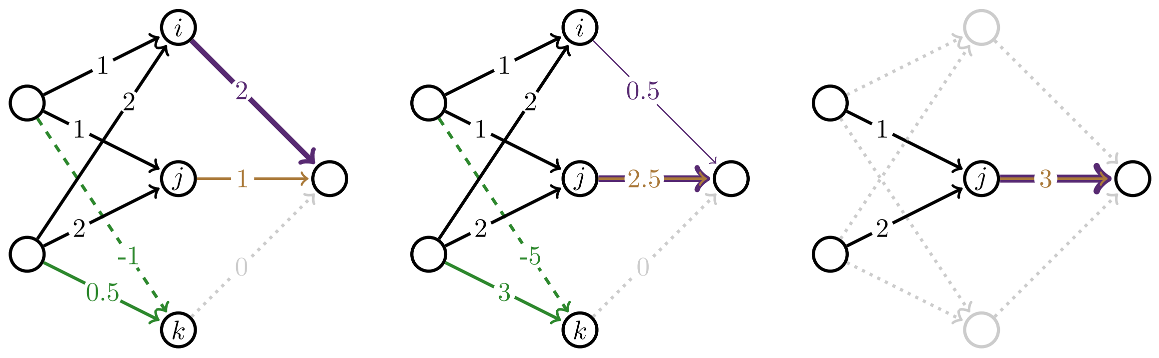


Figure 1.2: Examples of functionally equivalent degenerate neural network structures. See Section 2.4 for a definition of degenerate neural networks. Left–middle: since units i and j have the same incoming weights, their outputs are identical, so their outgoing weights can be traded off. Moreover, since unit k has zero outgoing weight, the function implemented is independent of its incoming weights. Right: accordingly, the same function could be implemented by a neural network with two fewer units.

Algorithmic geometry and topology of degenerate neural networks

In this thesis, I present a comprehensive theoretical investigation of structural degeneracy in a simple family of neural networks (namely, single-hidden-layer biased hyperbolic tangent neural networks). Within this setting, my contributions are as follows.

1. In Chapter 4 ("Neural Network Reduction and Rank"), I develop an algorithmic framework for analysing and measuring degeneracy, including defining the *rank* (the minimum number of hidden units required to implement an identical function). Using this framework, I show that sets of bounded rank parameters are closed and algebraic, and sets of bounded rank functions are highly non-convex.

2. In Chapter 5 ("Degenerate Neural Network Geometry"), I completely answer the functional equivalence question, including in the degenerate case, characterising the functional equivalence class by inverting the algorithms from Chapter 4. I also show that degenerate functional equivalence classes are piecewise linear path connected.

3. In Chapter 6 ("Degenerate Neighbourhoods in Parameter Space"), I introduce the *parametric approximate rank,* a measure of approximate degeneracy based on proximity to low-rank parameters (the minimum rank of nearby parameters).[1]

   Drawing on computational complexity theory, I show that detecting such proximity is an $\mathcal{NP}$-complete problem. This result involves a multi-stage problem reduction from Boolean satisfiability, via a restricted variant of Boolean satisfiability and a novel problem involving partitioning points in the plane into small squares.

The remainder of the thesis is organised as follows. In Chapter 2 ("Background"), I provide an elementary introduction to deep learning and neural networks, I review the existing literature on non-degenerate neural networks, and I review motivations for studying the geometry and topology of degenerate neural networks.

In Chapter 3 ("Formal Preliminaries"), I document important notational conventions,[2] formally introduce the setting for my analysis, and review the foundational results of Sussmann (1992) characterising degenerate networks in this setting. Moreover, I review the computational complexity theory needed to support Chapter 6.

In Chapter 7 ("Discussion"), I outline promising directions for future work revealed by my investigation, including extensions beyond simple neural networks, developing efficient approximation algorithms for measuring approximate degeneracy, and applying a degeneracy-aware perspective to clarify topics of current interest in deep learning.

[1]Appendix A additionally considers the $\mathscr{L}_2$ neighbourhoods of low-rank functions, introducing the *functional approximate rank* and relating it to the parametric approximate rank.

[2]Appendix B, listing all definitions and symbols used throughout the thesis (alongside all figures, tables, problems, algorithms, and major theorems), may serve as a helpful reference for the reader.

Chapter 2

Background

This chapter reviews in more detail the background context and motivations outlined in the introduction. In Section 2.1, I establish the broad context and motivation for my thesis. I introduce the field of *deep learning* and survey some of the applications of the techniques studied within this field, as well as important challenges facing the field motivating theoretical work aiming for a clearer understanding of deep learning. In Section 2.2, I provide an elementary introduction to *neural networks*—the central object of study in deep learning, this thesis, and related work.

Readers already familiar with deep learning and neural networks[1] are invited to advance to Sections 2.3 to 2.5, wherein I turn to the direct motivation and specific context for this thesis:

1. In Section 2.3, I review the existing literature on neural network geometry, illustrating a clear research gap in terms of degenerate neural networks.

2. In Section 2.4, I reframe and synthesise various perspectives from this prior work to arrive at a clear characterisation of degenerate neural networks as neural network structures with redundant units.

3. In Section 2.5, I conclude the chapter by reviewing various motivations for studying degenerate neural networks.

The discussion in this chapter is mainly conceptual. A technical introduction to the family of neural networks studied in this thesis, along with the most relevant existing mathematical results from neural network geometry that apply in this setting, is deferred until Chapter 3 ("Formal Preliminaries").

[1]For example, readers not among the explicitly intended audience for Sections 2.1 and 2.2 include those who already understand the phrases "neural networks display successful performance in a wide range of task domains despite their capacity to learn to memorise essentially arbitrarily complex patterns in data" and "I study neural networks with a fully-connected feed-forward architecture with a single hidden layer of biased units using the hyperbolic tangent activation function."

2.1 Machine learning and deep learning

Over recent decades, the field of *machine learning,* and, in particular, the subfield of *deep learning,* have risen to prominence as paradigms for creating software systems for practical applications. In this section, I introduce machine learning and deep learning and survey some of their applications, impacts, and challenges.

Learning machines, or, data-driven software engineering

Machine learning studies and develops software systems that learn to perform tasks based on data. The field is named by analogy to how humans (and other animals) learn through experience. In this analogy, one casts software systems as intelligent artefacts, or agents, using experience to improve their performance (Mitchell, 2006; Jordan and Mitchell, 2015; Russell and Norvig, 2021, §19). This language and this perspective go back, at least, to Turing (1950, §7), who proposed developing a machine with the intelligence of a human adult through a two-stage process, first building a machine with the relevant cognitive architecture of a human child, and then subjecting the system to experiences mirroring a human child's development.

The above anthropomorphic conceptualisation of machine learning is far from the only available perspective. Machine learning can alternatively be understood in terms of statistics (Breiman, 2001; Hastie et al., 2009), probabilistic inference (Devroye et al., 1996; Murphy, 2012), information theory (MacKay, 2003), pattern recognition (Ripley, 1996; Bishop, 2006), optimisation (Mitchell, 2017), cybernetics (Wiener, 1961; 1964, p. 14), and the philosophy of science (Rathmanner and Hutter, 2011).

Still another perspective views machine learning as a data-driven software engineering methodology (Mitchell, 2006; Norvig, 2016; Karpathy, 2017). According to this perspective, one develops a software system for a given task by, for example:[2]

1. assembling a *data set* comprising many examples of task inputs and outputs;
2. specifying an *architecture* comprising the high-level system structure (while omitting many low-level details); and
3. conducting a *learning process,* automatically deriving ("learning") the missing details such that the system's behaviour comes to match the examples in the data set.

The end result is a *learned software system,* whose behaviour is implicitly directed by examples, as compared to a *programmed software system,* whose behaviour is specified in deliberate and comprehensive detail with a programming language.

[2]This methodology corresponds to *supervised learning,* also called *learning with a teacher,* where the desired inputs and outputs are included in the data set. Machine learning also studies techniques for so-called *unsupervised learning* (learning from inputs only; see, e.g., Hastie et al., 2009, §14) and *reinforcement learning* (learning by interacting with an environment; see, e.g., Sutton and Barto, 2018).

Deep learning and neural networks

Machine learning research includes devising specific architectures and learning processes that can successfully learn behaviours exemplified in certain kinds of data. Among many available approaches, deep learning is an approach in which the architecture is a so-called neural network (Schmidhuber, 2015; LeCun et al., 2015; Goodfellow et al., 2016). The details of neural networks are central to this thesis, and I turn to them in Section 2.2. For now, I offer a conceptual overview of deep learning.

The neural network architecture traces its original inspiration (along with its name) to connectionist models of animal learning and cognition from neuroscience. Such models view the brain as a directed, weighted graph of simple computational *units* (or *neurons*), arranged so as to perform complex distributed computations by the propagation of signals between connected units (McCulloch and Pitts, 1943). Moreover, learning in these models corresponds to the strengthening and weakening of connection weights in response to experience, so as to change the implemented computation (e.g., Hebb, 1949). Analogously, the neural network system architecture is a connection graph of simple computational units, with the connection weights to be found during the learning process.

As a well-known example, consider the animal visual system. This system is understood as comprising layers of neurons, where the neurons in early layers function to detect simple sensory features such as edge patterns (Hubel and Wiesel, 1959; 1962), and those in later layers detect higher-order features from these primitives (Hubel and Wiesel, 1965). This model influenced early deep learning architectures such as the neocognitron (Fukushima, 1980) and the convolutional neural network (LeCun et al., 1989b;a). The latter architecture, and, moreover, the fundamental principle of learning *hierarchical representations* of inputs as a learning and computational strategy, remain central in deep learning today (Schmidhuber, 2015; LeCun et al., 2015; Goodfellow et al., 2016).

Today, neuroscience is just one of many influences drawn upon by researchers in the field of deep learning (Goodfellow et al., 2016, §1.2.1). The primary goal of the field is to create effective software systems, so advances in the understanding of biological learning and cognition serve as a guide, but are not considered constraints (Goodfellow et al., 2016; Hassabis et al., 2017).[3] For example, modern deep learning algorithms draw heavily from the study of numerical optimisation methods, casting the learning process as a high-dimensional optimisation problem in the space of all possible network weights (Goodfellow et al., 2016; Bottou et al., 2018). The resulting computation is not necessarily a plausible model for the adaptation of neural connection strengths in the brain.

[3]The related field of *computational neuroscience* develops neural network systems for the separate purpose of understanding biological learning and cognition. In this field, biological plausibility is an important constraint. While many aspects of modern deep learning techniques stray from strict biological plausibility, insights from deep learning are still considered useful for computational neuroscience (see, e.g., Marblestone et al., 2016; Richards et al., 2019; Lillicrap and Körding, 2019; Saxe et al., 2021).

Applications and impacts of learned software systems

The above outline of machine learning and deep learning makes it clear that the development process for learned software systems, as well as the systems themselves, can be remarkably different from their artisanal counterparts. These differences carve out a niche among possible applications within which machine learning is a more appropriate methodology than developing software systems by programming (Mitchell, 2006; Karpathy, 2017). In particular, there are many task domains for which examples of the desired behaviour—that is, data—are abundant or can be feasibly collected. In these cases, machine learning's reliance on data liberates one from having to specify a system's behaviour in comprehensive detail (as may be infeasible; Mitchell, 2006; Russell and Norvig, 2021, §19).

Deep learning has played an integral part in recent breakthroughs of software capabilities in several task domains meeting this description, including computer vision (Cireşan et al., 2011; Krizhevsky et al., 2012), automatic speech recognition (Deng et al., 2013; Yu and Deng, 2015), natural language processing (Vaswani et al., 2017; Brown et al., 2020), game playing (Mnih et al., 2015; Silver et al., 2016; 2017; 2018; Vinyals et al., 2019; OpenAI et al., 2019), and protein folding (Senior et al., 2020; Jumper et al., 2021).

Beyond these prominent examples, machine learning (especially deep learning) has seen many further applications. For example, many routine digital activities involve interaction with learned software systems. Learned software assists in ranking web search results (Liu, 2011), blocking email spam (Mujtaba et al., 2017), and organising network traffic (Boutaba et al., 2018). Learned software also helps secure digital infrastructure—looking out for cyberattacks, intrusions, malware, and fraud (Joseph et al., 2019).

Machine learning also finds applications in domains outside of computing. Recent literature surveys show example applications spanning domains as diverse as marketing (Brei, 2020, §5) and agriculture (Liakos et al., 2018). Learned software is beginning to be used as an aid for decision-making in healthcare, for diagnosis and treatment of ailments physical (Qayyum et al., 2020) and psychological (Dwyer et al., 2018), and in various aspects of the legal system (Surden, 2021). Moreover, machine learning has applications in addressing grand societal challenges (Rudin and Wagstaff, 2014), including the global fight against the coronavirus pandemic (Lalmuanawma et al., 2020).

The above applications hint at an important trend: learned software systems, especially neural networks, are increasingly positioned to have widespread and profound real-world impacts. Depending on whether the learned systems function as intended,[4] these impacts can range from vastly positive to vastly negative. Both possibilities motivate research that addresses technical challenges facing deep learning.

[4]As with all automation, the impacts of learned software systems depend not only on their technical correctness, but also on their design and use (cf. Parasuraman and Riley, 1997). Beyond the scope of this review is an important and growing literature on the appropriate design and use of learned software

Some challenges facing deep learning

It is well understood that the leading performance of the deep learning approach has been driven not only by advances in learning techniques but also, jointly, by the increased provision of data and computational resources (Jordan and Mitchell, 2015; Goodfellow et al., 2016, §1.2; Amodei and Hernandez, 2018; Hernandez and Brown, 2020; Sevilla et al., 2022). To give a sense of scale, state-of-the-art performance typically requires data sets with millions of examples (Goodfellow et al., 2016, p. 20), and the largest modern architectures have many billions of connection weights to learn (e.g., Brown et al., 2020).

Accordingly, the neural network learning process, and the resulting neural networks themselves, are highly complex. This complexity underpins several important technical challenges facing the field, including the following.

1. *Generalisation:* will a complex learned neural network reliably behave as desired?
2. *Efficiency:* how can one address the problematic data and computational costs involved in the learning process for complex neural networks?
3. *Transparency:* can a complex learned neural network be understood by a human?

I review each of these challenges in more detail below.

Challenge 1: Generalisation

Recall (cf. above) that a reliance on data liberates the machine learning practitioner from having to comprehensively specify the desired behaviour of a software system. However, with this freedom comes a question: will the resulting system respond as desired to novel inputs (that is, those inputs not exemplified in the data set)? This is the question of *generalisation* (see, e.g., Russell and Norvig, 2021, §19.1). Since almost all applications of machine learning expect the learned software systems to handle novel inputs, good generalisation is a fundamental requirement of all machine learning approaches.[5,6]

Guaranteeing generalisation in deep learning presents a particular challenge. Large neural network architectures have the potential to realise essentially arbitrarily complex behaviours. For example, Zhang et al. (2017; 2021) showed that standard deep learning techniques easily learn neural networks that perfectly match outputs for data sets with no underlying structure. This result is disconcerting, since it is unclear why such "memorising" neural networks couldn't arise in practice, rather than neural networks that capture underlying structure and generalise as (typically) desired.

systems (e.g., Russell et al., 2015; Hecht et al., 2018; Whittlestone et al., 2019; Mitchell et al., 2021).

[5]Generalisation requirements are especially stringent in domains such as computer security, in which *any* undesirable behaviour can be exploited by an intelligent adversary (Joseph et al., 2019, §1.1).

[6]What counts as "good generalisation" depends on the desires of the system designers, which may also be at odds with available data (cf. Mitchell et al., 2021, Figure 1; see also Footnote 4, above).

Challenge 2: Efficiency

A second significant challenge facing deep learning stems from the costs associated with each step in the process of producing a large-scale neural network. First, collecting large, high-quality data sets often requires significant manual effort. After data collection, the learning process (which may also be repeated as part of a trial-and-error search for an appropriate architecture) involves significant computational resources. These costs limit the accessibility of machine learning for groups without industrial-scale resources (Ahmed and Wahed, 2020; Schwartz et al., 2020) and create resource-centralising network effects (Bietti and Vatanparast, 2020).

Some of the costs of deep learning are external to the production process and are rather borne by individuals, the environment, and society (Crawford, 2021). For example, efficient data collection may come at the expense of poor working conditions for data collectors (Gray and Suri, 2019), or the privacy and intellectual property rights of users whose data is appropriated (Paullada et al., 2021). Moreover, there are significant environmental as well as financial costs associated with the computational resources involved in learning (Strubell et al., 2019; Schwartz et al., 2020; Patterson et al., 2021).

History has shown significant progress in improving the data- and computational efficiency of learning techniques (Hernandez and Brown, 2020). Nevertheless, more progress is needed to broaden the accessibility of and reduce the external impacts of deep learning.

Challenge 3: Transparency

The computations of the individual units and connections comprising a learned neural network software system are readily understandable. However, due to the vast size of practical architectures, it is often difficult to form a higher-level understanding of the mechanism by which the system produces particular outputs. For this reason, neural networks are often described as *opaque* or *black box* systems, as opposed to *transparent* or *intrinsically interpretable* systems (see, e.g., Lipton, 2018; Rudin, 2019; Molnar, 2022).

The lack of transparency of neural network systems becomes problematic in certain settings. For example, opaque systems are not alone sufficient for applications in high-stakes decision-making, where outputs must often be accompanied by an explanation or justification (cf. Biran and Cotton, 2017; Miller, 2019b;a). Likewise, opaque systems are not amenable to auditing by human experts, who can otherwise detect and correct problematic learned behaviours (Caruana et al., 2015). Moreover, in some contexts, obtaining a human-understandable model of data, for example as a source of scientific hypotheses to be tested, is itself the aim of learning (Lipton, 2018).

There is an ongoing effort to develop techniques by which learned neural networks can be interrogated by practitioners and their outputs explained to non-technical users (see, e.g., Molnar, 2022, for a general overview).

A deep understanding of deep learning

Not only are *learned* neural networks opaque as software systems, but the field also lacks a deep understanding of the learning process producing these systems. For example, there remain numerous empirical phenomena that resist explanation, including the observed generalisation performance itself despite the scale of architectures used in practice (Zhang et al., 2017; 2021); observed relationships between scale and performance (Hestness et al., 2017; Belkin et al., 2019; see also Viering and Loog, 2021); and the phenomena of so-called *mode connectivity* (Freeman and Bruna, 2017; Sagun et al., 2018; Draxler et al., 2018; Garipov et al., 2018), *Hessian singularity* (Sagun et al., 2017; 2018; Chaudhari et al., 2017; 2019; Papyan, 2018; Ghorbani et al., 2019), and *lottery tickets* (Frankle and Carbin, 2019; Frankle et al., 2020).

A deeper understanding of the learning process could help to address the three challenges reviewed above:

1. Understanding the learning process is necessary to clarify the generalisation abilities (and limits) of neural networks, because the learning process plays a crucial part in determining learned neural network behaviour (Zhang et al., 2017).

2. Understanding the learning process could clarify the conditions under which learning will succeed, reducing the need for expensive trial and error in application and research (cf. Strubell et al., 2019).

3. The learning process provides an alternative route to understanding learned neural networks, rather than a direct or "bottom-up" approach of building layers of abstraction atop a vast connection graph (cf. recent proposals in neuroscience, Lillicrap and Körding, 2019; Richards et al., 2019).

One path to a deeper understanding is to seek rigorous mathematical theories describing the learning process.[7] This approach has traditionally yielded powerful mathematical guarantees for learning techniques—consider, for example, the generalisation bounds yielded by Vapnik–Chervonenkis theory (Vapnik, 2000). Unfortunately, traditional statistical learning theory fails to account for observed deep learning performance, due in part to the scale and complexity of neural networks (Zhang et al., 2017).[8]

Accordingly, there is a need for basic theoretical research—particularly that which speaks to the nature of complexity in neural networks—clarifying all aspects of deep learning. This is the broad motivation towards which this thesis is directed.

[7]Of course, empirical study of the learning process is a complementary path to a deeper understanding, as has indeed yielded much of the field's current understanding (albeit at great computational expense).

[8]As another example, classical statistical theory (*à la* Cramér, 1946; Akaike, 1973; 1974; see, e.g., Newey and McFadden, 1994) fails to apply to neural network learning due to the presence of so-called *information singularities* (cf., e.g., Hagiwara et al., 1993; Watanabe, 2007; 2009; 2018; Wei et al., 2022). Such singularities are closely linked with degenerate neural networks (Fukumizu, 1996).

2.2 Neural network structure, function, and learning

Neural networks can be understood at several interrelated levels (cf. Figure 2.1).

1. The *macro-structural* level: a neural network is a *graph data structure,* comprising a network of computational *units* of different kinds, connected by directed *edges.*

2. The *micro-structural* level: a neural network is an *instantiation* of a macro-structural template, with particular edge weights and individual unit properties.

3. The *implementation* level: a neural network is an *information-processing system,* implementing a mathematical function by propagating signals through its structure.

4. The *adaptive* level: a neural network is a *learning system,* adaptively modifying its micro-structure to bring its function into alignment with input–output examples.

The term "neural network" may invoke any or all of these levels. As the distinctions are central to neural network geometry, I introduce the following terminology.

1. *Neural network architecture:* a macro-structural template—a unit connection graph.

2. *Neural network parameter:* a vector of weights and unit properties indexing a particular micro-structure.

3. *Neural network function:* the input–output map implemented by some structure.

4. *Neural network learning algorithm:* an algorithm that takes as input a data set and architecture, and then searches for a matching parameter and function.

In this section, I introduce each of the above elements of neural networks in detail.

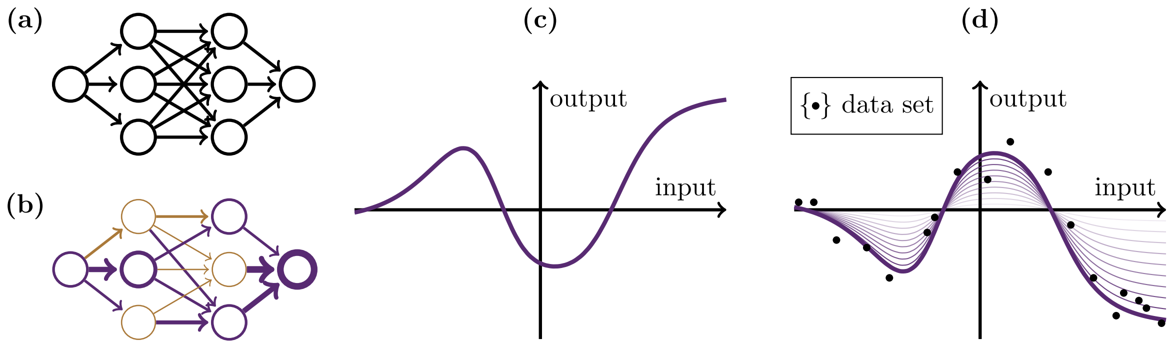


Figure 2.1: Levels of understanding neural networks. **(a)** Macro-structural/architecture (a connection graph). **(b)** Micro-structural/parameter (a particular choice of edge weights and unit properties, indicated here as line widths, with colour indicating sign). **(c)** Implementation/function (input–output map based on structure). **(d)** Adaptive/learning algorithm (process of adapting the parameter to align the function with some data set).

Macro-structure: Neural network architectures

A neural network *architecture* specifies the network's abstract form including the number and bulk properties of the computational units and the edges connecting them.

To clarify this abstract definition, consider a fundamental example—the *fully-connected feed-forward* architecture (also called the *multi-layer perceptron* architecture). Units are arranged in three or more *layers,* with the units of one layer connected to all those in the next (cf. Figure 2.2). The source units in the first layer (the *input layer*) are called *input units.* The sink units in the final layer (the *output layer*) are called *output units.* The remaining layers and units are called *hidden layers* and *hidden units.*

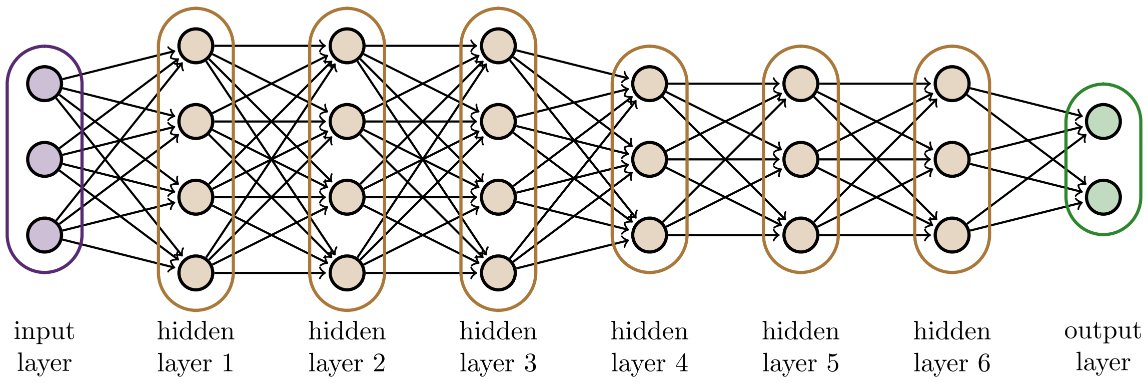


Figure 2.2: Connection graph for an example fully-connected feed-forward neural network architecture. The graph has 8 layers, 26 units, and 80 edges. **Left:** input layer of three input units. **Middle:** six hidden layers each with three or four hidden units. **Right:** output layer of two output units.

As for the bulk computational properties of each unit, a fully-connected feed-forward architecture specifies an *activation function* (also called a *transfer function*) governing the unit's response to incoming signals, and whether the unit has a *bias* (also called *threshold*) property. The most appropriate choices depend on the application (Goodfellow et al., 2016). Several activation functions relevant to related work are as follows (cf. Figure 2.3).

1. *Identity* ($\mathrm{id}(z) = z$). Often used for input and output units. If also used for hidden units, the result is a *linear neural network*—a simple kind of neural network sometimes analysed as a first step in theoretical work (Baldi and Hornik, 1995).

2. *Hyperbolic tangent* ($\tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$). Inspired by saturation in biological neurons. Historically popular for hidden units in practice (Goodfellow et al., 2016), and therefore also in early theoretical analyses (e.g., Sussmann, 1992). A variant is the related *logistic sigmoid* ($\sigma(z) = (1 + e^{-z})^{-1} = \frac{1}{2}\tanh\left(\frac{1}{2}z\right) + 1$).

3. *Rectified linear unit* ($\mathrm{relu}(z) = \max\{0, z\}$). Inspired by thresholding in biological neurons (Nair and Hinton, 2010; Glorot et al., 2011). Currently popular for hidden units in practice (Goodfellow et al., 2016).

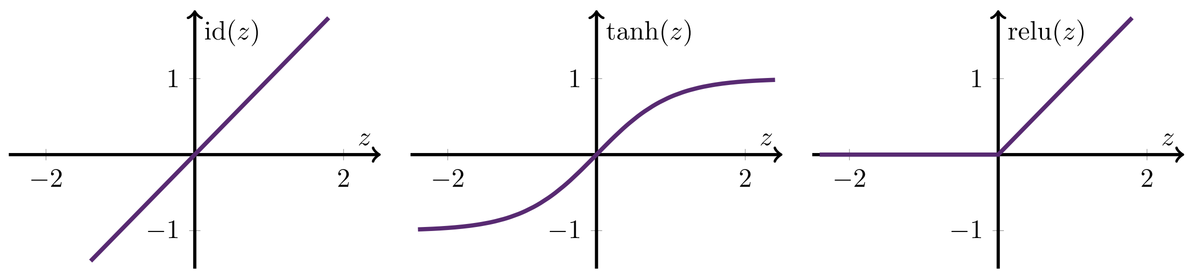


Figure 2.3: Example activation functions. Left: identity (id). Middle: hyperbolic tangent (tanh). Right: rectified linear unit (relu).

In practice, many applications call for more sophisticated architectures, better suited to learning from certain data than is the fully-connected feed-forward architecture. Prominent architectures include convolutional neural networks (LeCun et al., 1989b;a; also reviewed in Rawat and Wang, 2017), recurrent neural networks (Hochreiter and Schmidhuber, 1997b; also reviewed in De Mulder et al., 2015; Lipton et al., 2015), and transformers (Vaswani et al., 2017; see also Phuong and Hutter, 2022; Yu et al., 2022).

Nevertheless, fully-connected feed-forward architectures are fundamental—all of the above-listed architectures use this simpler architecture as a "building block," in the sense that their connection graphs contain fully-connected feed-forward subgraphs. Accordingly, theoretical work often begins in the fully-connected feed-forward setting.

Micro-structure: Neural network parameters

Once a neural network architecture fixes the units and connections, it remains to set the many edge weights and tune any internal unit properties (such as biases) in one of many possible ways, each constituting a micro-structure. A neural network *parameter* is a vector that encodes a particular micro-structure, with the vector's components determining the edge weights and unit properties (cf. Figure 2.4). The space of vectors corresponding to all possible neural network micro-structures for a given architecture is called the *parameter space*.

$$(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13) \in \mathbb{R}^{13}$$

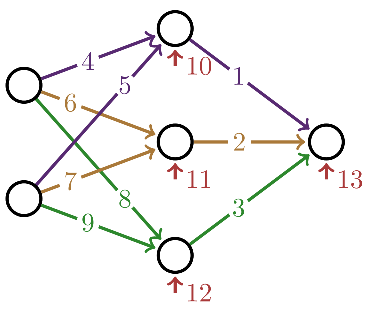


Figure 2.4: Example of a neural network parameter (left) and micro-structure (right). The neural network architecture has 9 edges connecting 6 units (including 4 with biases).

Implementation: Neural network functions

The implementation perspective on neural networks imbues a neural network's structure with semantics—the units and edges constitute an information-processing system mapping input signals to output signals (in a manner depending on the unit properties). The corresponding mathematical function is a *neural network function.*

In the case of a fully-connected feed-forward architecture, a given micro-structure processes input signals as follows (cf. Figure 2.5).[9]

1. The components of an input vector are given to the input units, which send these values along their outgoing edges.
2. Each edge multiplies its value by its edge weight, giving this value to its target unit.
3. Each hidden unit sums the values received along its incoming edges, adds its bias value, feeds the total through its activation function, and sends the result along its own outgoing edges.
4. Steps (2) and (3) repeat for each layer (until signals reach the output layer).
5. Each output unit sums the values received along its incoming edges, and adds its bias value. The resulting values constitute the function output.

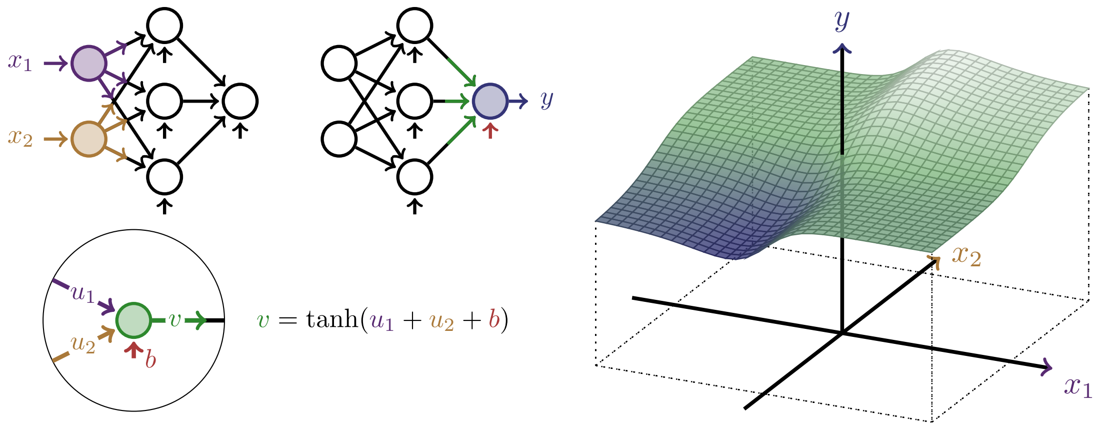


Figure 2.5: Implementation of a mathematical function by a neural network. Top left: Input units encode a numerical representation of the function input vector. This value travels along their edges, multiplied by edge weights. Bottom left: The values arriving at each hidden unit are added together (with the unit's bias). The unit then emits its own signal according to its activation function (shown, hyperbolic tangent, tanh). Top middle: The process repeats, until eventually the output units emit the function output. Right: The result is a mathematical function mapping input signals to output signals.

[9]This description of neural network implementation is mainly expository. In practice, for increased efficiency, the computation is implemented using highly parallelised matrix operations on graphics processing units or other specialised hardware.

Adaptation: Neural network learning algorithms

The final level of understanding neural networks brings together the macro-structural, micro-structural, and implementation levels, and also connects neural networks back to the motivating context of machine learning and deep learning (Section 2.1), as follows. Given a data set and a neural network architecture, a *neural network learning algorithm* is a procedure for finding a neural network parameter such that the implemented neural network function closely matches the input–output examples in the data set.

The concept of "matching" is formalised using a *loss function* (also called a *cost function* or *risk function*). A well-known example loss function is the *mean squared error*—as used in linear regression (Legendre, 1805; Gauss, 1809; 1821)—which measures the average squared distance between the example outputs and the neural network function's outputs for the inputs in the data set. The appropriate loss function depends on the application (Goodfellow et al., 2016).

The prevailing approach to this difficult search problem draws on the numerical optimisation literature—on local iterative search methods in particular. These methods explore the loss landscape by incremental perturbation of the parameter in a direction chosen so as to reduce the loss, searching for a (local) minimum (cf. Figure 2.6). The scale of modern architectures and data sets typically necessitate first-order methods, and, even then, advanced algorithmic techniques and specialised hardware are required for efficiently estimating gradients. Beyond the scope of this review is a wealth of literature on practical learning algorithms (see, e.g., Goodfellow et al., 2016, §8; Bottou et al., 2018).

Suffice it to note that the behaviour of any local search method is determined by the nature of the loss landscape. The nature of the loss landscape is, in turn, determined by the link between neural network structures and neural network functions—the subject of neural network geometry. Understanding neural network geometry is therefore key to understanding neural network learning. To this topic, I now turn.

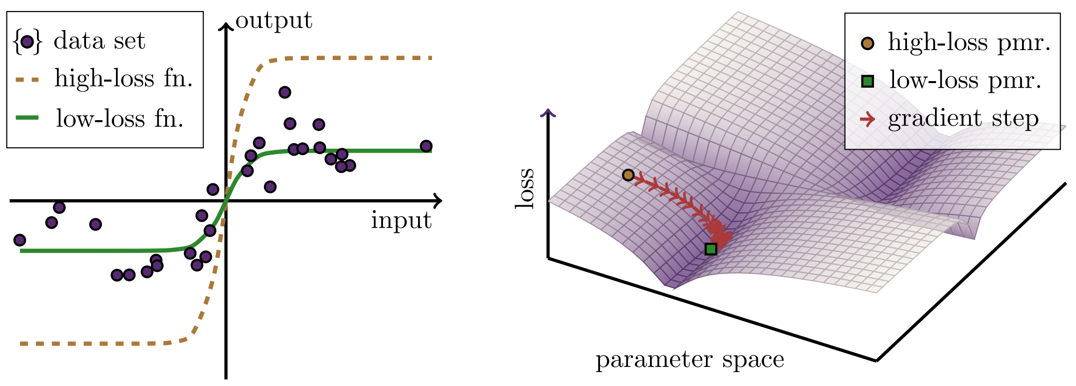


Figure 2.6: Left: A loss function scores how well each neural network function matches a data set. Right: Such scores define a *loss landscape* over parameter space, in which gradient-based methods can iteratively search for a low-loss parameter.

2.3 Non-degenerate neural network geometry

Neural network geometry is the study of the relationship between neural network structure and function. Neural network geometry addresses topics such as the existence and uniqueness of neural network parameters implementing certain mathematical functions.

It turns out that neural networks have the capacity to (approximately) implement many mathematical functions. In fact, given a neural network architecture with a single hidden layer, a suitable activation function, and a large enough number of hidden units, any continuous function on a bounded domain can be approximated to any desired degree of accuracy (see, e.g., Cybenko, 1989; Hecht-Nielsen, 1989; Hornik et al., 1989; Leshno et al., 1993; see also work on the approximation achievable with a given number of hidden units, e.g., Barron, 1993; Murata, 1996; Mhaskar, 1996). This celebrated *universal approximation* result distinguishes single-hidden-layer neural networks from precursor architectures (without hidden layers; Minsky and Papert, 1988), positioning them alongside function classes of traditional interest in analysis, such as trigonometric and algebraic polynomials (Fourier, 1822; Weierstrass, 1885; cf. Rudin, 1976, §7,§8; Pinkus, 2000, §3).

It further turns out that the parameters implementing neural network functions are far from unique—in general, there are many parameters implementing each neural network function. As simple examples, refer back to Figures 1.1 and 1.2. Call two neural network parameters *functionally equivalent* if they implement the same neural network function. In lieu of uniqueness, the question becomes: what are all of the functionally equivalent structures—can one enumerate them or detect them? As I review in this section, these questions have been addressed from several related perspectives, namely:

1. *Structural identifiability:* (partially) recovering the implementing structure from a neural network function (e.g., Sussmann, 1992; Albertini et al., 1993; Kůrková and Kainen, 1994; Fefferman, 1994; Phuong and Lampert, 2020; Bona-Pellissier et al., 2021; Vlačić and Bölcskei, 2021; Stock and Gribonval, 2022).

2. *Structure–function symmetries:* generating functionally equivalent parameters by simple parameter transformations (e.g., Hecht-Nielsen, 1990; Chen et al., 1993; Rüger and Ossen, 1997; DiMattina and Zhang, 2010).

3. *Unique representatives:* finding subsets of the parameter space in which uniqueness holds (e.g., Hecht-Nielsen, 1990; Chen et al., 1993; Kůrková and Kainen, 1994; Rüger and Ossen, 1997).

The main aim of this review is to show that—regardless of perspective—prior work has restricted its attention to certain *non-degenerate* neural network structures, leaving the geometry of the remaining *degenerate* neural networks to be clarified.

Identifiability: (Partially) recovering structure from function

Given a neural network function and a family of architectures (leaving, say, the number of hidden units to be determined), can one uniquely recover the implementing neural network parameter? No: there are usually many possible implementations. However, if one can characterise *all* of these implementations—the so-called *functional equivalence class* of the parameter—then one can claim to have recovered as much information about the parameter as possible. This is the most common framing by which the functional equivalence question has been addressed.[10,11]

Perhaps the original contribution within this framing comes from Sussmann (1992). Sussmann studied the setting of single-hidden-layer neural networks with the hyperbolic tangent activation function, showing that for parameters that satisfy certain non-degeneracy conditions, all functionally equivalent parameters are related by the simple unit exchange and negation operations exemplified in Figure 1.1. Similar results were soon achieved for architectures with certain analytic (Albertini et al., 1993) or asymptotically constant (Kůrková and Kainen, 1994; Kainen et al., 1994) activation functions (see also Fukumizu, 1996). Fefferman and Markel (1993, also Fefferman, 1994) achieved a similar result for multi-layer hyperbolic tangent architectures, as did Albertini and Sontag (1992; 1993a;b;c, also Albertini et al., 1993) for certain recurrent architectures.

More recently, there has been a resurgence of work on this topic. In line with modern architectural fashion, Phuong and Lampert (2020), Bona-Pellissier et al. (2021), and Stock and Gribonval (2022) have studied multi-layer networks with the rectified linear unit activation function, showing that (again, under various non-degeneracy assumptions), functionally equivalent parameters are related by unit exchanges and a positive scaling operation (see also Carroll, 2021, §4). Vlačić and Bölcskei (2021; 2022) have given a more general result in terms of the basic affine symmetries of certain activation functions for a very general class of connection graphs.

All of these results leave the somewhat optimistic impression that, though neural network implementations are not unique, at least they are "almost" unique, in that all but the order of and, say, the sign or scaling of units can be recovered. However, these results rely on the deliberate exclusion of degenerate parameters, acknowledged as counterexamples for which "almost"-uniqueness fails. While there has been some effort to obtain minimal non-degeneracy assumptions (e.g., Sussmann, 1992; Vlačić and Bölcskei, 2021), the richer functional equivalence classes of degenerate parameters remain to be clarified.

[10]Briefly shifting to a statistical perspective on deep learning (e.g., White, 1989; Tishby et al., 1989; Levin et al., 1990), parameter recovery is essentially *statistical identifiability* (Koopmans, 1949; Koopmans and Reiersol, 1950; see also Ran and Hu, 2017; Lewbel, 2019), with the lack of uniqueness corresponding to *partial identifiability* (also called *set identifiability;* Manski, 2003; Lewbel, 2019, §6.2). This statistical perspective does not appear to have been widely adopted, though see DiMattina and Zhang (2010).

[11]I review work on the topic of recover*ability*, which is separate from but related to work devising algorithms for recovering parameters from input–output functions (see, e.g., Rolnick and Körding, 2020).

Symmetries: Transformations generating equivalent parameters

An alternative perspective on neural network geometry sees it as the study of the mathematical properties of the *structure–function map*—the function mapping neural network parameters to the neural network functions they implement.[12] From this perspective, approximation theory studies the map's range, and the functional equivalence question is the study of its *symmetries*.

A structure–function symmetry is a parameter transformation that leaves the implemented function unchanged (these symmetries are also called *equi-output transformations*). This perspective allows one to easily generate functionally equivalent parameters (by applying transformations), and moreover to view the transformations as a mathematical group (with composition as the group operation). Structure–function symmetries also have direct implications for the shape of the loss landscape used in learning.

The question of structure–function symmetries was perhaps first formulated by Hecht-Nielsen (1990), who documented the linear symmetries corresponding to unit exchanges in multi-layered architectures. Hecht-Nielsen (1990) noted that further symmetries exist, for example, based on unit negation for odd activation functions (cf. Figure 1.1). Chen et al. (1993, also Chen and Hecht-Nielsen, 1991) went on to show that, assuming a hyperbolic tangent activation function, there are no other *analytic* symmetries than those expressible as combinations of unit exchanges and negations. Rüger and Ossen (1997) extended this result to architectures using certain sigmoidal activation functions.

Structure–function symmetries for architectures with more general activation functions, including the rectified linear unit, have not been studied directly. Recent work from the identifiability perspective (e.g., Phuong and Lampert, 2020; Stock and Gribonval, 2022, reviewed above) immediately suggests that in the case of the rectified linear unit, unit scaling operations generate linear symmetries for these architectures. With one such operation for each positive constant this gives a continuum of symmetries, allowing smooth variation of a parameter while maintaining functional equivalence.[13] Such local symmetries have also been studied for certain biologically-motivated activation functions by DiMattina and Zhang (2010).

The analyticity condition (or the stronger linearity condition) on the transformations in the above results is crucial. Chen et al. (1993) and Rüger and Ossen (1997) acknowledge that their results do not rule out *discontinuous* symmetries. Such discontinuous symmetries can be created by acting on subsets of degenerate parameters (cf. above) in ways that exploit their richer functional equivalence classes, while transforming non-degenerate parameters according to one of the basic transformations outlined above. Chen et al. (1993) left the study of these discontinuous symmetries as future work.

[12]Note carefully that the output of the structure–function map is itself a function.

[13]This basic property of rectified linear unit architectures has been widely noted outside of direct work on neural network geometry (see, e.g., Dinh et al., 2017).

Unique representatives: Canonical regions and canonicalisation

A third perspective on functional equivalence is the search for certain *representative* implementations for each function, as an attempt to achieve some of the benefits of unique implementations despite the reality of extensive non-uniqueness. This work has pursued two aims: (1) to identify subsets of parameter space within which neural network functions have unique implementations—I term these regions *canonical regions,* and (2) to find equivalent implementations of particular parameters within those regions—a process I term *canonicalisation.*

There have been various results constructing partial canonical regions. Studying multi-layer hyperbolic tangent networks, Hecht-Nielsen (1990) constructed cone-shaped subsets of the parameter space containing at least one parameter implementing each function. Chen et al. (1993, also Chen and Hecht-Nielsen, 1991) improved this result, finding cone- and wedge-shaped regions containing exactly one implementation of each function, except possibly for parameters on the boundaries of the region. Kůrková and Kainen (1994) extended this result for single-hidden-layer architectures with certain asymptotically constant activation functions, and Rüger and Ossen (1997) for multi-layer architectures with certain sigmoidal activation functions.

Rüger and Ossen (1997) also gave a partial algorithm for canonicalisation. The algorithm is designed for parameters with non-zero biases, but is also extensible to any non-degenerate parameters, as it works to counteract the basic symmetries of unit exchange and negation that are the typical source of non-uniqueness in the architecture.

A common pattern in the above partial canonical regions, and the partial canonicalisation algorithm, is that they successfully create and compute unique representations of non-degenerate parameters, but they fail to account for degenerate parameters. Degenerate parameters occupy (non-uniquely) the boundaries of the partial canonical regions, and the algorithm of Rüger and Ossen (1997) will fail to produce a unique output for functionally equivalent degenerate parameters in general. The unique representation of degenerate neural networks remains to be studied.

Towards degenerate neural network geometry

The above works, and, in particular, their collective technical assumptions and limitations, carve out a gap in the parameter space where the extent of functional equivalence, the structure–function symmetries, and the demands of canonicalisation have more complex and richer answers that remain unexplored. A full understanding of neural network geometry requires a dedicated analysis of the degenerate case. Fortunately—as I discuss in the next section—there is much in existing work that can be reframed towards this end, providing a solid foundation for my investigation of degenerate neural network geometry.

2.4 Degenerate neural networks

In this section I synthesise three characterisations of degenerate neural networks:

1. *Richer geometry:* a neural network parameter is degenerate if it has a more complex functional equivalence class than that generated by the simple operations that are typically studied (for example, unit exchange and negation, cf. Section 2.3).

2. *Non-minimality*, or, *global redundancy:* a neural network parameter is degenerate if there exists *some* structure with fewer hidden units implementing the same function.

3. *Reducibility*, or, *local redundancy:* a neural network parameter is degenerate if there are obvious means of removing units while preserving the implemented function.

These characterisations are *equivalent*—that is, under certain assumptions on the architecture, they describe the exact same neural network parameters. This is a central insight and the foundation for the investigation of degeneracy presented in this thesis. The first perspective places degeneracy as the determinant of the answers to geometric questions about neural network structure and function (cf. Section 2.3 and Chapter 5). The second perspective places degeneracy in a position to clarify the role of neural network complexity in deep learning (cf. end of Section 2.1). The third perspective reveals an effective means to analyse and measure degeneracy, which is the basis for my framework (cf. Chapter 4).

Reframing Sussmann's characterisations of non-degeneracy

The clearest formal characterisation of degeneracy comes, indirectly, from Sussmann (1992). In the setting of single-hidden-layer neural networks with the hyperbolic tangent activation function, Sussmann studied the functional equivalence question from the perspective of structural identifiability (cf. Section 2.3), showing that the following three conditions on a neural network parameter are formally equivalent.

1. *Simple geometry:* the functional equivalence class is exhaustively described by the operations of unit exchange and negation (cf. Figure 1.1).

2. *Minimality:* the parameter implements a neural network function that *cannot* be implemented with fewer hidden units.

3. *Irreducibility:* the parameter does *not* display any of a short list of patterns indicating that a unit could be removed by a simple operation (cf. Table 2.7, below).

Sussmann (1992) framed these results around the non-degenerate case. However, negating each condition reveals the equivalence of the characterisations of degenerate neural networks in terms of (1) *richer geometry,* (2) *non-minimality,* and (3) *reducibility* (cf. above).

Sussmann's conditions of reducibility

The "patterns"—or *conditions of reducibility*—in the third of Sussmann's equivalent characterisations of non-degeneracy (Sussmann, 1992) can be distilled as follows.

(i) The *outgoing* weights for some hidden unit are all zero (such units contribute zero to the implemented function).

(ii) The *incoming* weights for some hidden unit are all zero (the contribution of such units to the output is a constant independent of the input).

(iii) The incoming weights and biases for some pair of hidden units are *equal* (the contributions of such pairs of units are proportional).

(iv) The incoming weights and biases for some pair of hidden units are *negated* (these contributions are also proportional, since the hyperbolic tangent is an odd function).

These conditions are exemplified in Table 2.7.

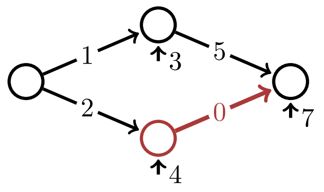
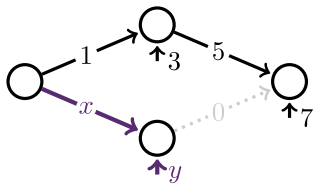
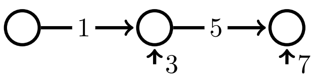
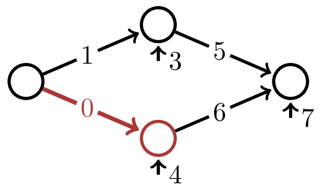
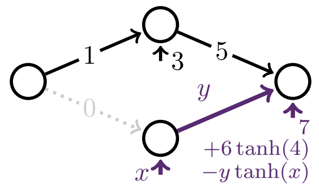
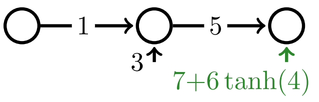
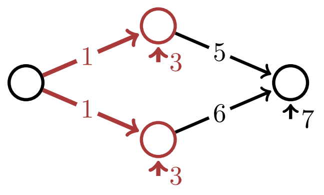
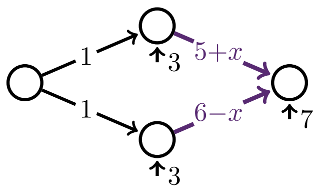
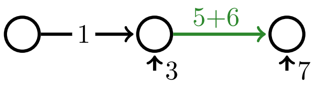
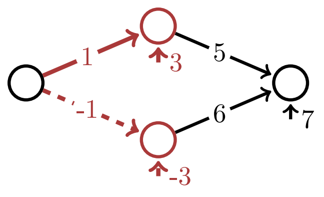
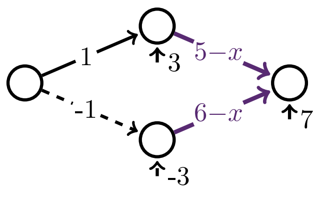
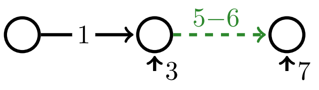
Condition	Example reducible network	Richer geometry	Non-minimality
(i)			
(ii)		 1 0 3 x 5 y 7 +6 tanh(4) $-y$ tanh(x)	
(iii)			
(iv)			

Table 2.7: Left column: Example neural network structures meeting each condition of reducibility (highlighted: offending units, weights, and biases). Middle column: The conditions allow some of the remaining weights and biases to be varied ($x, y \in \mathbb{R}$) without changing the function. Right column: Moreover, with an appropriate variation, units can effectively be removed, implying a smaller, functionally equivalent structure.

Table 2.7 illustrates how each reducibility condition implies a richer functional equivalence class—accessible by continuously modifying the affected weights and biases in certain ways. These modifications lead to parameters that are not related to the original parameter by unit exchanges and negations. For this reason, these parameters have been excluded from previous work studying the functional equivalence question.

Table 2.7 also illustrates how, with a simple operation (involving varying the free parameters so as to bring a unit's contribution to the implemented function to zero), each reducibility condition implies the existence of a smaller equivalent structure. This can be interpreted as a kind of "local redundancy", in the sense that it is possible to eliminate some unit from the network with a simple operation.

Conversely, consider the case where it is possible to implement the same function with *some* smaller structure. This situation could be described as a kind of "global redundancy", since, *a priori*, there is no reason to think that the smaller structure has much in common with the initial structure. However, Sussmann's equivalence result implies that any non-minimal structure is also reducible (Sussmann, 1992). In other words, *there is no global redundancy without local redundancy.* This property of single-hidden-layer hyperbolic tangent networks makes it possible to create algorithms that "minimise" neural network structures, eliminating global redundancy simply by repeatedly removing instances of local redundancy (cf. Section 4.2).

Beyond single-hidden-layer hyperbolic tangent networks

In the broader neural network geometry literature, there has been some work finding similar equivalence results in other architectures. Like Sussmann (1992), these authors emphasise the implications for the non-degenerate case. However, with a similar reframing, it is possible to learn about the degenerate case from this work.

Albertini et al. (1993) observed that the proofs of Sussmann's equivalence results rely crucially on a property of the hyperbolic tangent activation function they called the *independence property,* which concerns the existence of non-trivial linear combinations of affine transformations of the activation function. They gave some sufficient conditions under which certain analytic activation functions satisfy this property. Kůrková and Kainen (1994) generalised Sussmann's results in a different direction, studying a certain class of asymptotically constant odd or even activation functions.

More recently, Vlačić and Bölcskei (2021; 2022) investigated structural redundancy in a much broader class of architectures based on arbitrary connection graphs. The results of Vlačić and Bölcskei (2021) essentially imply that a similar set of equivalences will hold for a given family of neural network structures if the only irreducible implementation of the zero function is an empty neural network.

2.5 The relevance of degenerate neural networks

The conditions characterising degenerate neural network parameters describe a vanishing proportion of the parameter space.[14] If the components of a neural network parameter are chosen at random (for example, independently sampled from a standard normal distribution), then there is effectively zero probability that the resulting parameter is degenerate. In this sense, degenerate neural networks are highly *atypical.*

However, the neural networks that are relevant to deep learning are not (only) those that are typical according to chance, but rather:

1. those neural networks that typically arise as the outcome of the learning process using real data, that is, *learned neural networks;* and, in turn,

2. those neural networks that are typically encountered during the search process that constitutes learning, since these influence the selection of the former.

Moreover, non-degenerate neural networks that are, nevertheless, *approximately* degenerate can share some of the properties of degenerate neural networks. For example, such parameters inherit from their degenerate neighbours a larger set of *approximately* functionally equivalent parameters. Since neural network learning algorithms operate by local search it follows that degenerate neural networks can influence the course of learning even if they are merely approached (rather than encountered exactly).

In this section, I briefly review empirical and theoretical evidence for the role of (approximately) degenerate neural networks in these contexts, despite their *a priori* atypicality.[15] Further work is needed to establish the prevalence of (approximate) degeneracy among typical learned neural network structures and in the course of learning. An important precursor to this work is to develop methods for measuring (approximate) degeneracy, such as the measures I study in this thesis.

[14]Compared to a non-degenerate neural network parameter, any individual degenerate neural network parameter has a larger functional equivalence class (characteristically—cf. Section 2.4). However, there are vastly more distinct non-degenerate parameters than degenerate neural network parameters.

[15]Before turning to examples within the context of deep learning, I note mathematics and machine learning are replete with well-studied examples of atypical but important structures—look no further than the coordinate axes (cf. regularised regression techniques such as the lasso, Tibshirani, 1996), or consider the set of all singular matrices (cf. low-rank matrix approximation techniques such as truncated singular value decomposition, see, e.g., Trefethen and Bau, 1997, §5; Murphy, 2012, §12.2.3). In the latter case, consider moreover that ill-conditioned (approximately singular) matrices behave, for the purposes of numerical linear algebra, somewhat like their troublesome neighbours (see, e.g., Trefethen and Bau, 1997, §12). Finally, the course of a non-linear dynamical system is fundamentally determined by the characteristics of stationary points, which govern the dynamics of nearby trajectories even if the points themselves are never reached (cf., e.g., discussion of *phase portraits* in Strogatz, 1994, §§6.1–6.3). This last example has an immediate resemblance to gradient-based learning—see also discussion below.

Degeneracy among learned neural networks

Learned neural networks are chosen not at random, but for their ability to represent the input–output behaviour exemplified in a data set. Degenerate neural network parameters are candidate solutions to the learning problem, to be evaluated by computing their loss, not by chance. Put another way, by the strict characterisation of non-degenerate neural networks as minimal implementations of their functions (cf. Section 2.4), the one neural network function goes from non-degenerate to degenerate by the addition of a single unit to the architecture used in learning, even if the data are unchanged.

There is some empirical evidence that, in practice, learned neural networks are often (approximately) non-minimal implementations of their neural network functions. It is often possible to *compress,* or *distil,* a learned neural network into an architecture with significantly fewer units, without a large change in performance (see, e.g., Buciluǎ et al., 2006; Hinton et al., 2014; and in particular Sanh et al., 2019 for a large-scale example).

Similarly, work on *pruning* studies the effects of removing weights or units from the connection graph (e.g., Casper et al., 2021). In particular, the recent investigation of learned neural network structures by Casper et al. (2021) found many instances of units with weak or correlated outputs. Casper et al. (2021) found that such units could be eliminated without having a large effect on performance by simple operations bearing a striking resemblance to the reducibility operations discussed in Section 2.4.

Degeneracy and the learning process

Recall (cf. Section 2.2) that neural network learning algorithms typically operate by local search in the loss landscape as directed by gradient information. Of fundamental importance to such search processes is therefore the occurrence of *critical points,* such as local minima or saddle points. Apart from acting as fixed points in the learning process, the properties of these points govern the structure of the nearby loss landscape.

Fukumizu and Amari (2000), Fukumizu et al. (2019), and Şimşek et al. (2021) have constructed critical points by various methods of *embedding* the parameter space of one neural network architecture into that of another architecture with additional units (where the embedding is designed to preserve functional equivalence). Since the resulting critical points satisfy the non-minimality characterisation of degeneracy, they are degenerate neural network parameters.

Similarly, Amari et al. (2006, also Wei et al., 2008; Cousseau et al., 2008) have shown that gradient-based learning dynamics are distorted in the neighbourhood of degenerate networks, exhibiting so-called Milnor attractor dynamics, leading to slow learning. This particular issue disappears in deeper networks (Amari et al., 2017), but the earlier analysis establishes the broader principle that the dynamics of learning can be influenced by the local presence of degenerate neural networks.

Chapter 3

Formal Preliminaries

Before embarking upon my analysis I use this opportunity to establish some fundamental technical definitions and results of use throughout this thesis.

1. In Section 3.1, I document my conventions for denoting vectors and their components. I recall the uniform (L_∞) metric used for measuring the size of and distance between vectors. I recall the lexicographic total order on vectors, and use it to extend the notions of sorting, sign, and absolute value from scalars to vectors.

2. In Section 3.2, I formally introduce the family of *simple neural networks* and associated notational conventions. These neural networks are the setting for the formal analysis presented in this thesis.

3. In Section 3.3, I review the technical results of Sussmann (1992) equivalently characterising degenerate simple neural networks in terms of *reducibility,*[1] *non-minimality,* and *richer geometry.* I give a slight generalisation of Sussmann's results, extending the conditions to the setting of neural networks with multiple outputs.

   The conditions of reducibility in Definition 3.22 serve as the operational definition of degeneracy used throughout this thesis. Later in this section, I characterise, visualise, and study the properties of the subset of parameter space containing reducible neural network parameters.

4. In Section 3.4, I review the requisite computational complexity theory to support the statement and proof of the hardness results in Chapter 6, including $\mathcal{NP}$-completeness and problem reductions,[1] and I prove the hardness of a restricted variant of the Boolean satisfiability problem.

Many additional mathematical definitions and notational conventions are introduced throughout the remaining chapters of this thesis—see Appendix B for a comprehensive list of all definitions, which may serve as a useful reference.

[1]Note carefully the double use of the term "reducibility" exemplified by Sections 3.3 and 3.4. Context should suffice to disambiguate the intended meaning throughout the remaining chapters.

3.1 Packing, measuring, sorting, and signing vectors

Vectors are used throughout this thesis to represent neural network inputs, parameters, and outputs. In particular the vector space of neural network parameters plays a key role in my analysis. In this section, I document notational conventions and foundational concepts relating to vectors.

1. I introduce my notational conventions for referring to vectors and their parameters.
2. I recall the normed vector space structure of $\mathbb{R}^p$ given by the *uniform norm* and the associated metric space structure given by the *uniform metric.*
3. I recall the total order structure of $\mathbb{R}^p$ given by the *lexicographic order.*
4. I use the lexicographic order to extend the notions of *sign* and *absolute value* from scalars to vectors.

I assume that the concepts denoted are familiar to my readers, or refer them to an introductory text on linear algebra (e.g., Halmos, 1958, §§1–3; Axler, 2015) analysis (e.g., Kreyszig, 1978, §1; Ó Searcóid, 2006; Lindstrøm, 2017, §3), or order theory (e.g., Halmos, 1960, §14; Harzheim, 2005).

Vectors and their components

Throughout this thesis I make extensive use of the following conventions for flexibly referring to vectors and their components.

1. *Unpacking components:* given a vector denoted with some letter, for example $v \in \mathbb{R}^p$, I denote the p ordered components of v as $v_1, \ldots, v_p$. If the vector already bears a subscript, for example $v_i \in \mathbb{R}^p$, I denote the components as $v_{i,1}, \ldots, v_{i,p}$.
2. *Packing subvectors:* given two vectors $u \in \mathbb{R}^p$ and $u \in \mathbb{R}^q$, I denote by (u, v) the $(p+q)$-dimensional vector $(u_1, \ldots, u_p, v_1, \ldots, v_q) \in \mathbb{R}^{p+q}$. I extend this convention to arbitrary lists of vectors and singleton components.
3. *Vector functions:* given a function f on the domain $\mathbb{R}^p$, I drop the second layer of parentheses in application to a vector specified by its components, for example writing $f(v_1, \ldots, v_p)$ rather than $f((v_1, \ldots, v_p))$.

Moreover, given two vectors $u, v \in \mathbb{R}^p$, I usually denote the dot product implicitly as in

$$uv = u \cdot v = \sum_{i=1}^{p} u_i v_i.$$

The uniform metric

An important notion of the size of a p-dimensional vector is the uniform norm (also called the infinity norm, max norm, or the Chebyshev norm of a vector). The uniform norm gives rise to the uniform metric (also L_∞ or Chebyshev metric), a distance metric on $\mathbb{R}^p$.

Definition 3.1 (Uniform norm)**.** Consider a vector $v \in \mathbb{R}^p$. The *uniform norm* of v, denoted $\|v\|_\infty$, is defined as the largest absolute component of v:

$$\|v\|_\infty = \max_{i=1,\dots,p} |v_i|.$$

Definition 3.2 (Uniform distance)**.** Consider a pair of vectors $u, v \in \mathbb{R}^p$. The *uniform distance* between u and v, denoted $\|u - v\|_\infty$, is defined as the largest component-wise absolute difference between u and v:

$$\|u - v\|_\infty = \max_{i=1,\dots,p} |u_i - v_i|.$$

I use the following definition and notation for closed uniform neighbourhoods (also called L_∞ balls) of a vector.

Definition 3.3 (Closed uniform neighbourhood)**.** Consider a vector $v \in \mathbb{R}^p$. Given a positive scalar $\varepsilon \in \mathbb{R}^+$, the *closed uniform neighbourhood of v with radius ε,* denoted $\bar{B}_\infty(v;\varepsilon)$, is the set of vectors with uniform distance *at most* ε from v:

$$\bar{B}_\infty(v;\varepsilon) = \{\, u \in \mathbb{R}^p \mid \|u - v\|_\infty \leq \varepsilon \,\}.$$

The lexicographic order of vectors

I introduce a total order (also called a linear order) on $\mathbb{R}^p$. I use this order throughout my analysis to sort vectors, as a means to canonicalise a list of vectors or to detect duplicate vectors in a computationally efficient manner.

Definition 3.4 (Lexicographic order of vectors)**.** Given $p \in \mathbb{N}$, the *lexicographic order* is a relation on $\mathbb{R}^p$ denoted $\preceq$, defined such that for $u, v \in \mathbb{R}^p$,

$$u \preceq v \quad \Leftrightarrow \quad (u = v) \vee \bigvee_{i=1}^{p} \big((u_1 = v_1) \wedge \cdots \wedge (u_{i-1} = v_{i-1}) \wedge (u_i < v_i)\big).$$

Moreover, define the *strict lexicographic order* relation $u \prec v \Leftrightarrow (u \preceq v \wedge u \neq v)$, and the usual reversed relations $u \succ v \Leftrightarrow v \prec u$, and $u \succeq v \Leftrightarrow v \preceq u$.

Remark 3.5. The above definition essentially says that $u \preceq v$ when, if u and v differ in at least one component, in the first component in which they differ, $u_i < v_i$.

Remark 3.6. The lexicographic order is a total order on $\mathbb{R}^p$ (see, e.g., Harzheim, 2005, Theorem 4.1.11). Therefore a list of vectors can be sorted into non-decreasing lexicographic order using a comparison-based sorting algorithm. Standard sorting algorithms such as MERGESORT (Goldstine and von Neumann, 1948, pp. 49–68; Cormen et al., 2009, §2), HEAPSORT (Williams, 1964; Cormen et al., 2009, §6), and QUICKSORT (Hoare, 1962; Cormen et al., 2009, §7) use $\mathcal{O}(n \log n)$ comparisons to sort a list of n items.[2] Evaluating the lexicographic order relation on $\mathbb{R}^p$ takes $\mathcal{O}(p)$ time (assuming that the components use a standard, bounded-precision representation).

Lexicographic sign and absolute value

Aside from governing the sorting of vectors, the lexicographic order also allows the extension of the concepts of sign (as in, positive or negative) and absolute value from real scalars to vectors in $\mathbb{R}^p$, streamlining several algorithms and proofs in this thesis.

Definition 3.7 (Lexicographic sign)**.** Consider a vector $v \in \mathbb{R}^p$. Define the *lexicographic sign* of v, denoted $\operatorname{sign}_{\text{lex}}(v)$, as

$$\operatorname{sign}_{\text{lex}}(v) = \begin{cases} +1 & (v \succ 0), \\ 0 & (v = 0), \\ -1 & (v \prec 0). \end{cases}$$

Say that v is *lexicographically positive (or negative)* if $\operatorname{sign}_{\text{lex}}(v) = +1$ (or -1).

Remark 3.8. Lexicographically positive vectors are non-zero vectors whose first non-zero component is positive. Similarly, lexicographically negative vectors are non-zero vectors whose first non-zero component is negative. The zero vector $0 \in \mathbb{R}^p$ is neither lexicographically positive nor lexicographically negative (and is the only such vector).

Definition 3.9 (Lexicographic absolute value)**.** Consider a vector $v \in \mathbb{R}^p$. Define the *lexicographic absolute value* of v, denoted $\operatorname{abs}_{\text{lex}}(v) \in \mathbb{R}^p$, as

$$\operatorname{abs}_{\text{lex}}(v) = \operatorname{sign}_{\text{lex}}(v) \cdot v = \begin{cases} +v & (v \succ 0), \\ 0 & (v = 0), \\ -v & (v \prec 0). \end{cases}$$

Remark 3.10. One can think of the uniform norm and the lexicographic absolute value as alternative generalisations of the absolute value of a scalar. Note that, unlike the uniform norm of a vector—a scalar—the lexicographic absolute value $\operatorname{abs}_{\text{lex}}(v) \in \mathbb{R}^p$.

[2] QUICKSORT uses $\mathcal{O}(n^2)$ comparisons in the worst case, but can be made to require only $\mathcal{O}(n \log n)$ in expectation (Cormen et al., 2009, §7).

3.2 Simple neural networks

In this thesis, I study a family of fully-connected, feed-forward neural network architectures with an arbitrary number of input and output units and a single hidden layer of biased hidden units with the hyperbolic tangent activation function.[3] I refer to such neural networks as *simple neural networks.*

In this section, I clarify my notational conventions for simple neural networks. These conventions are summarised in Table 3.1. Each row of the table is explained in more detail throughout this section.

Concept	Definition	Notation	Description
Architecture	3.11	$\mathcal{A}_h^{n,m}$	n input units, m output units, and h hidden units.
Parameter space	3.12	$\mathcal{W}_h^{n,m}$	$\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$.
Parameter	3.13	$w \in \mathcal{W}_h^{n,m}$	$w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$ (see Figure 3.2).
Function	3.16	$f_w : \mathbb{R}^n \to \mathbb{R}^m$	$f_w(x) = d + \sum_{i=1}^{h} a_i \tanh(b_i x + c_i)$.
Function family	3.18	$\mathcal{F}_h^{n,m}$	$\mathcal{F}_h^{n,m} = \{ f_w \mid w \in \mathcal{W}_h^{n,m} \}$.
Extended family	3.20	$\mathcal{F}_\infty^{n,m}$	$\mathcal{F}_\infty^{n,m} = \bigcup_{h=0}^{\infty} \mathcal{F}_h^{n,m}$.

Table 3.1: Summary of simple neural network notation.

Simple neural network architecture

I formalise the family of simple neural network architectures (macro-structures) as follows.

Definition 3.11 (Simple neural network architecture)**.** Consider an input dimension $n \in \mathbb{N}^+$, an output dimension $m \in \mathbb{N}^+$, and a number of hidden units $h \in \mathbb{N}$.[4] The *simple neural network architecture* $\mathcal{A}_h^{n,m}$ is a fully-connected feed-forward neural network architecture with n input units, m biased output units, and a single hidden layer of h biased hidden units with the hyperbolic tangent activation function

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$

[3]The hyperbolic tangent function is not the most commonly used activation function in modern deep learning practice, but is chosen following Sussmann (1992), whose foundational results in neural network geometry serve as the clearest available starting point for a thorough analysis of degeneracy (cf. Sections 2.4 and 3.3). In Section 7.1 I discuss extensions of the analysis to additional settings.

[4]$\mathbb{N} = \{0, 1, 2, \ldots\}$, $\mathbb{N}^+ = \{1, 2, \ldots\}$. See Remark 3.14.

Simple neural network parameters

Each simple neural network architecture gives rise to a space of possible neural network parameters (micro-structures), called the parameter space. I formalise this concept below.

Definition 3.12 (Simple neural network parameter space)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$. Note that the number of edges (requiring weights) is $nh+mh$ and the number of hidden/output units (requiring biases) is $h+m$, for a total of $(n+m+1)h+m$ components. Define the *simple neural network parameter space*

$$\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}.$$

Definition 3.13 (Simple neural network parameter)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. A *simple neural network parameter* $w \in \mathcal{W}_h^{n,m}$ has the form

$$w = (a_1, \ldots, a_h, \quad b_1, \ldots, b_h, \quad c_1, \ldots, c_h, \quad d)$$

where for each hidden unit $i = 1, \ldots, h$, $a_i \in \mathbb{R}^m$ is the hidden unit's outgoing weight vector, $b_i \in \mathbb{R}^n$ is the hidden unit's incoming weight vector, and $c_i \in \mathbb{R}$ is the hidden unit's bias; and $d \in \mathbb{R}^m$ contains for each output unit $i = 1, \ldots, m$, the output unit's bias $d_i \in \mathbb{R}$.

Definition 3.13 exemplifies the typical way in which I unpack the components of the $((n+m+1)h+m)$-dimensional parameter vector into vectors of components (cf. Section 3.1). There are other ways of ordering and grouping the components of a neural network parameter, but this convention is the most appropriate for my analysis. I clarify the meaning of each of the component vectors in Figure 3.2.

$$w = \begin{pmatrix} a_1, \ldots, a_i, \ldots, a_h, \\ b_1, \ldots, b_i, \ldots, b_h, \\ c_1, \ldots, c_i, \ldots, c_h, \\ d_1, \ldots, d_m \end{pmatrix} \in \mathcal{W}_h^{n,m}$$

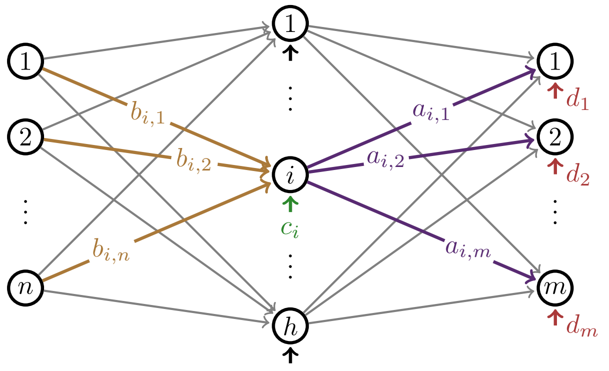


Figure 3.2: Illustration of the components of a simple neural network. I often refer to particular groups of components from within a parameter vector $w \in \mathcal{W}_h^{n,m}$. For each hidden unit $i = 1, \ldots, n$, there is an *outgoing weight vector* $a_i \in \mathbb{R}^m$, an *incoming weight vector* $b_i \in \mathbb{R}^n$, and a *bias* $c_i \in \mathbb{R}$. There is also a vector of output unit biases $d \in \mathbb{R}^m$.

Remark 3.14. The above definitions are designed to include the case $h = 0$. In this case, the parameter space $\mathcal{W}_0^{n,m} = \mathbb{R}^m$, with parameters simply consisting of output biases (thus implementing constant functions $f_d(x) = d$, cf. Definition 3.16). Similarly, I define $\mathcal{W}_{-1}^{n,m} = \emptyset$, and likewise for the function families defined below. Though considering neural networks without hidden units may seem unnatural, these boundary conventions allow my analysis to smoothly extend to constant functions.

Remark 3.15. In practice, neural network parameters are implemented with bounded-precision representations of components (for example, with single-precision floating-point numbers). The distinction does not appear to matter for deep learning practice, since similar performance is achievable with even less precision (e.g., Gupta et al., 2015). However, strictly speaking, the decision affects the computational complexity analysis of some of the algorithms studied in this thesis.

Simple neural network functions

A simple neural network parameter implements a mathematical function, called the neural network function.

Definition 3.16 (Simple neural network function)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. A parameter $w \in \mathcal{W}_h^{n,m}$ implements a *simple neural network function* $f_w : \mathbb{R}^n \to \mathbb{R}^m$, where for $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$,

$$f_w(x) = d + \sum_{i=1}^{h} a_i \tanh(b_i \cdot x + c_i).$$

Given a neural network function, one can formalise the geometric concept of functional equivalence of neural network parameters.

Definition 3.17 (Functional equivalence)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Two parameters $w, w' \in \mathcal{W}_h^{n,m}$ are *functionally equivalent* if and only if $f_w = f_{w'}$ (that is, $\forall x \in \mathbb{R}^n, f_w(x) = f_{w'}(x)$).

Moreover, I define the family of functions implemented by all of the neural network parameters in a given architecture's parameter space. An elementary observation is that there is substantial overlap between these function families for varying numbers of hidden units h. In particular, they form a containment hierarchy.

Definition 3.18 (Simple neural network function family)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$, define the *family of simple neural network functions* on h units,

$$\mathcal{F}_h^{n,m} = \{\, f_w \mid w \in \mathcal{W}_h^{n,m} \,\}.$$

Proposition 3.19. *Let $n, m \in \mathbb{N}^+$. Then, with $\subsetneq$ denoting proper inclusion,*

$$\mathcal{F}_0^{n,m} \subsetneq \mathcal{F}_1^{n,m} \subsetneq \cdots \subsetneq \mathcal{F}_{h-1}^{n,m} \subsetneq \mathcal{F}_h^{n,m} \subsetneq \mathcal{F}_{h+1}^{n,m} \subsetneq \cdots$$

Proof. ($\mathcal{F}_h^{n,m} \subset \mathcal{F}_{h+1}^{n,m}$): Let $f_w \in \mathcal{F}_h^{n,m}$ where $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$. One way to implement f_w with a larger network is to augment the h-unit network with an additional hidden unit with outgoing weight zero. Construct a parameter

$$w' = (a_1, \ldots, a_h, 0, b_1, \ldots, b_h, 0, c_1, \ldots, c_h, 0, d) \in \mathcal{W}_{h+1}^{n,m}.$$

Then $f_w = f_{w'} \in \mathcal{F}_{h+1}^{n,m}$.

($\mathcal{F}_{h-1}^{n,m} \neq \mathcal{F}_h^{n,m}$): For $i = 1, \ldots, h$ put $a_i = (1, 0, \ldots, 0) \in \mathbb{R}^m$, $b_i = (i, 0, \ldots, 0) \in \mathbb{R}^n$, and $c_i = 0 \in \mathbb{R}$; and put $d = 0 \in \mathbb{R}^m$. Then it is a corollary of Theorem 3.25 (Section 3.3) that the parameter $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$ implements a function $f_w \in \mathcal{F}_h^{n,m} \setminus \mathcal{F}_{h-1}^{n,m}$. □

One can also speak collectively of the family of functions implementable within any simple neural network architecture (with a given input and output dimension).

Definition 3.20 (Extended simple neural network function family)**.** Given $n, m \in \mathbb{N}^+$, define the *extended family of simple neural network functions,*

$$\mathcal{F}_\infty^{n,m} = \bigcup_{h=0}^{\infty} \mathcal{F}_h^{n,m}.$$

This leads to the conceptual picture of simple neural network function families illustrated in Figure 3.3. Note that this visualisation accurately depicts the containment structure, but is a possibly misleading picture of the properties of the function families. For example, I show in Section 4.4 that $\mathcal{F}_h^{n,m}$ is highly non-convex.

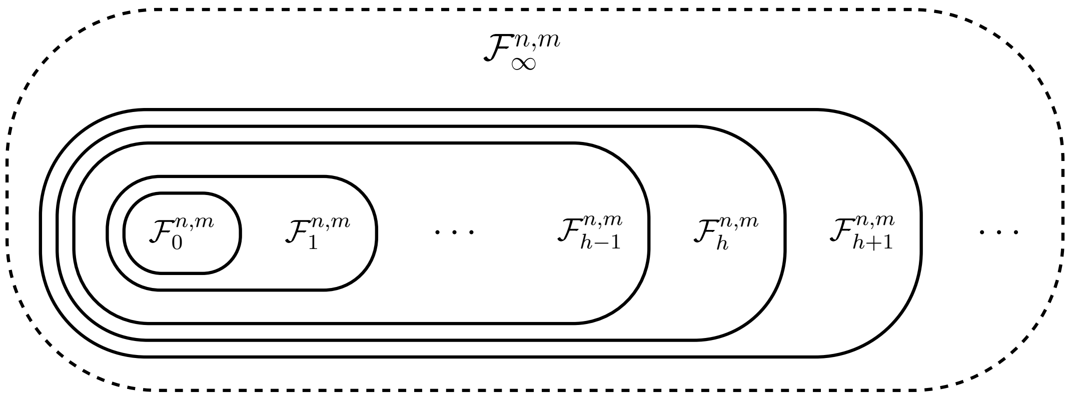


Figure 3.3: Conceptual illustration of the containment hierarchy of simple neural network function families $\mathcal{F}_0^{n,m} \subsetneq \mathcal{F}_1^{n,m} \subsetneq \cdots$, existing inside the extended family $\mathcal{F}_\infty^{n,m}$.

The extended family of simple neural network functions has its own (abstract, infinite-dimensional) vector space structure, since it is closed under the vector operations.

Proposition 3.21. *The extended family of simple neural network functions is closed under pointwise scalar multiplication and pointwise vector addition. That is, given a scalar $\alpha \in \mathbb{R}$ and two simple neural network functions $f, g \in \mathcal{F}_\infty^{n,m}$,*

(i) $\alpha f \in \mathcal{F}_\infty^{n,m}$, and

(ii) $f + g \in \mathcal{F}_\infty^{n,m}$.

Proof. (i): Let $h \in \mathbb{N}$ such that $f \in \mathcal{F}_h^{n,m}$. Write $f = f_w$ where

$$w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}.$$

Construct a new parameter $w' \in \mathcal{W}_h^{n,m}$ where

$$w' = (\alpha a_1, \ldots, \alpha a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, \alpha d)\,.$$

Then, for all x,

$$f_{w'}(x) = (\alpha d) + \sum_{i=1}^{h} (\alpha a_i) \tanh(b_i x + c_i) = \alpha \left(d + \sum_{i=1}^{h} a_i \tanh(b_i x + c_i) \right) = \alpha f_w(x)$$

Thus, $\alpha f = \alpha f_w = f_{w'} \in \mathcal{F}_h^{n,m} \subset \mathcal{F}_\infty^{n,m}$.

(ii): Let $h, k \in \mathbb{N}$ such that $f \in \mathcal{F}_h^{n,m}$ and $g \in \mathcal{F}_k^{n,m}$ write $f = f_u$ and $g = f_v$ where

$$\begin{aligned} u &= (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}, \qquad \text{and} \\ v &= (a'_1, \ldots, a'_k, b'_1, \ldots, b'_k, c'_1, \ldots, c'_k, d') \in \mathcal{W}_k^{n,m}. \end{aligned}$$

Construct a new parameter $w \in \mathcal{W}_{h+k}^{n,m}$ where

$$w = (a_1, \ldots, a_h, a'_1, \ldots, a'_k, b_1, \ldots, b_h, b'_1, \ldots, b'_k, c_1, \ldots, c_h, c'_1, \ldots, c'_k, d + d')\,.$$

Then, for all x,

$$\begin{aligned} f_w(x) &= (d + d') + \sum_{i=1}^{h} a_i \tanh(b_i x + c_i) + \sum_{i=1}^{k} a'_i \tanh(b'_i x + c'_i) \\ &= \left(d + \sum_{i=1}^{h} a_i \tanh(b_i x + c_i) \right) + \left(d' + \sum_{i=1}^{k} a'_i \tanh(b'_i x + c'_i) \right) \\ &= f_u(x) + f_v(x) \end{aligned}$$

Thus, $f + g = f_u + f_v = f_w \in \mathcal{F}_{h+k}^{n,m} \subset \mathcal{F}_\infty^{n,m}$. □

3.3 Reducibility: Characterising degeneracy

The determining factor in a simple neural network's geometry and complexity is whether the neural network is degenerate or non-degenerate (cf. Section 2.3). Establishing a clear understanding of the boundary of degeneracy is therefore the first step of my analysis. As reviewed in Section 2.4, the clearest delineation of degenerate simple neural networks comes (indirectly) from Sussmann (1992), who gave an effective characterisation of degeneracy in terms of *reducibility*—the presence of simple relationships between a parameter's components (cf. Table 2.7). Sussmann's conditions can be formalised as follows.

Definition 3.22 (Reducible simple neural network parameter)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a parameter

$$w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m},$$

call w *reducible* (respectively, *irreducible*) if it satisfies any (respectively, none) of the following conditions:

(i) $a_i = 0$ for some i,

(ii) $b_i = 0$ for some i,

(iii) $(b_i, c_i) = (b_j, c_j)$ for some $i \neq j$, or

(iv) $(b_i, c_i) = -(b_j, c_j)$ for some $i \neq j$.

In this thesis, reducibility serves as my operational definition of degeneracy. In this section, I review the equivalence between reducibility and the alternative characterisations of *non-minimality* and *richer geometry*. Further, I offer a characterisation and visualisation of the reducible and irreducible subsets of the parameter space, and document some of their basic properties.

Reducibility and non-minimality

Sussmann (1992) showed that degeneracy can be equivalently characterised in terms of whether a simple neural network's function could be implemented with fewer hidden units. I extend this result to simple neural networks with multiple output units.

Definition 3.23 (Non-minimal simple neural network parameter)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. A neural network parameter $w \in \mathcal{W}_h^{n,m}$ is *non-minimal* if there exists some $h' < h$ and $w' \in \mathcal{W}_{h'}^{n,m}$ such that $f_w = f_{w'}$. Otherwise, w is *minimal*.

Example 3.24. Consider $(0,1,0,0) \in \mathcal{W}_1^{1,1}$. This parameter implements the function $f_{0,1,0,0}(x) = 0 + 0\tanh(1x+0) = 0$. The same function is implemented by the parameter $0 \in \mathcal{W}_0^{1,1}$, so $(0,1,0,0)$ is non-minimal. On the other hand, the parameter $(1,1,0,0) \in \mathcal{W}_1^{1,1}$ implements the function $f_{1,1,0,0}(x) = 0 + 1\tanh(1x+0) = \tanh(x)$, which is not implemented by any parameter in $\mathcal{W}_0^{1,1}$ (such parameters implement only constant functions, cf. Remark 3.14), so $(1,1,0,0)$ is minimal.

Theorem 3.25 (Equivalence of reducibility and non-minimality)**.** *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, and a parameter $w \in \mathcal{W}_h^{n,m}$. Then w is non-minimal if and only if w is reducible.*

Proof. This theorem is a slight generalisation (and reframing) of the result given by Sussmann (1992), who proved that the (contrapositive) condition holds in the case of simple neural networks with a single output unit. I generalise the result as follows.[5]

($\Leftarrow$): First, if any of the conditions hold, then a smaller functionally equivalent parameter can be constructed as follows.

(i) If $a_i = 0$ for some i, then hidden unit i fails to contribute to the neural network function output. Construct a new parameter $w' \in \mathcal{W}_{h-1}^{n,m}$ with hidden unit i omitted:

$$w' = (a_1,\dots,a_{i-1},a_{i+1},\dots,a_h,b_1,\dots,b_{i-1},b_{i+1},\dots,b_h,c_1,\dots,c_{i-1},c_{i+1},\dots,c_h,d).$$

Then $f_w(x) - f_{w'}(x) = a_i\tanh(b_ix+c_i) = 0$.

(ii) If $b_i = 0$ for some i, then hidden unit i contributes only a constant to the function. Define $d' = d + a_i\tanh(c_i) \in \mathbb{R}^m$, and construct a new parameter $w' \in \mathcal{W}_{h-1}^{n,m}$ with hidden unit i omitted and output layer bias vector changed to d' to compensate:

$$w' = (a_1,\dots,a_{i-1},a_{i+1},\dots,a_h,b_1,\dots,b_{i-1},b_{i+1},\dots,b_h,c_1,\dots,c_{i-1},c_{i+1},\dots,c_h,d').$$

Then $f_w(x) - f_{w'}(x) = d - d' + a_i\tanh(b_ix+c_i) = 0$.

(iii) If $(b_i,c_i) = (b_j,c_j)$ for some $i \neq j$, then hidden units i and j contribute in proportion. They can be combined into a single unit (say i) with the same incoming weights and bias, and the combined outgoing weight vector $a' = a_i + a_j \in \mathbb{R}^m$. Construct a new parameter $w' \in \mathcal{W}_{h-1}^{n,m}$ accordingly:

$$\begin{aligned} w' = (&a_1,\dots,a_{i-1},a',a_{i+1},\dots,a_{j-1},a_{j+1},\dots,a_h,\\ &b_1,\dots,b_{j-1},b_{j+1},\dots,b_h,c_1,\dots,c_{j-1},c_{j+1},\dots,c_h,d).\end{aligned}$$

Then $f_w(x) - f_{w'}(x) = a_i\tanh(b_ix+c_i) + a_j\tanh(b_jx+c_j) - (a_i+a_j)\tanh(b_ix+c_i) = 0$.

[5]Fukumizu (1996) offers a similar proof.

(iv) If $(b_i, c_i) = -(b_j, c_j)$ for some $i \neq j$, then hidden units i and j contribute in negative proportion. Due to the odd property of tanh they can be combined into a single unit (say i) with incoming weight and bias vectors (b_i, c_i) and the combined outgoing weight vector $a' = a_i - a_j \in \mathbb{R}^m$. Construct a new parameter $w' \in \mathcal{W}^{n,m}_{h-1}$ accordingly:

$$\begin{aligned} w' = (&a_1, \ldots, a_{i-1}, a', a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_h, \\ &b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_h, c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_h, d). \end{aligned}$$

Then, since $\tanh(z) = -\tanh(-z)$,

$$\begin{aligned} f_w(x) - f_{w'}(x) &= a_i \tanh(b_i x + c_i) + a_j \tanh(b_j x + c_j) - (a_i - a_j) \tanh(b_i x + c_i) \\ &= a_i \tanh(b_i x + c_i) - a_j \tanh(-b_j x - c_j) - (a_i - a_j) \tanh(b_i x + c_i) \\ &= 0. \end{aligned}$$

In all cases, the new parameter $w' \in \mathcal{W}^{n,m}_{h-1}$ has $f_{w'} = f_w$, so w is reducible.

($\Rightarrow$): Let h' be the smallest number of hidden units required to implement f_w, and let $w' \in \mathcal{W}^{n,m}_{h'}$ such that $f_{w'} = f_w$. Suppose $h' < h$. It remains to show that w satisfies at least one of the reducibility conditions. To achieve this, I reduce to the single-output case and apply the result of Sussmann (1992).

To reduce to the single-output case, I introduce some notation. Write the vector function $f_w : \mathbb{R}^n \to \mathbb{R}^m$ as a vector of component functions $f_w^{(1)}, f_w^{(2)}, \ldots, f_w^{(m)} : \mathbb{R}^n \to \mathbb{R}$ such that for $x \in \mathbb{R}^n$,

$$f_w(x) = \left(f_w^{(1)}(x), f_w^{(2)}(x), \ldots, f_w^{(m)}(x)\right).$$

Each of these component functions is a simple neural network function in $\mathcal{F}^{n,1}_h$ with a structure corresponding to a subgraph of the connection graph of the original neural network, as illustrated in Figure 3.4.

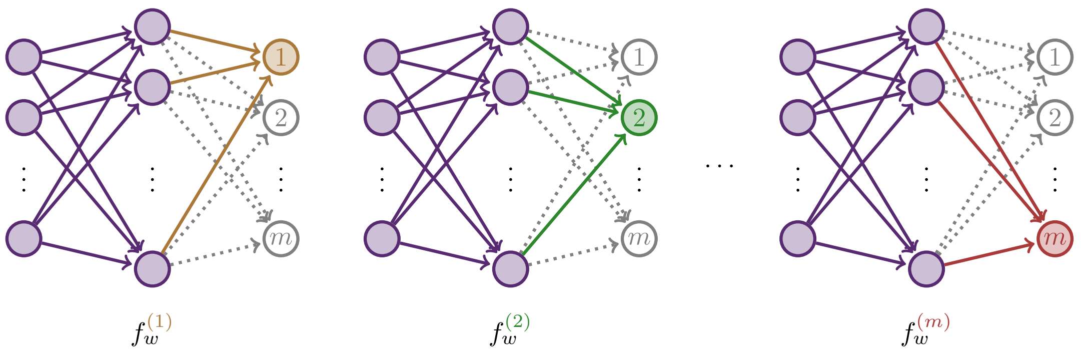


Figure 3.4: The connection graphs of the component functions of f_w. The included units and weights are coloured. Note that the hidden units of each network share the same incoming weights (and biases, not shown).

Denote the corresponding (overlapping) subvectors of the parameter $w \in \mathcal{W}_h^{n,m}$ as $w_{(1)}, \ldots, w_{(m)} \in \mathcal{W}_h^{n,1}$. That is, for $\mu = 1, \ldots, m$,

$$w_{(\mu)} = (a_{1,\mu}, \ldots, a_{h,\mu}, b_1, \ldots, b_h, c_1, \ldots, c_h, d_\mu) \in \mathcal{W}_h^{n,1}.$$

Apply the same conventions to $f_{w'}$ to produce $f_{w'}^{(1)}, \ldots, f_{w'}^{(m)} \in \mathcal{F}_{h'}^{n,1}$ and $w'_{(1)}, \ldots, w'_{(m)} \in \mathcal{W}_{h'}^{n,1}$. Note that since $f_w = f_{w'}$ (by assumption), it follows that $f_w^{(\mu)} = f_{w'}^{(\mu)}$ for $\mu = 1, \ldots, m$.

Apply the results of Sussmann (1992) as follows. For each $w_{(\mu)}$ note that $w'_{(\mu)}$ is a functionally equivalent parameter using fewer units. Therefore, the reducibility conditions (in the special case of $m = 1$) must hold for each $w_{(\mu)}$. Since conditions (ii–iv) only apply to the incoming weights and biases, if any of these conditions hold for any $w_{(\mu)}$, then they must also hold for w itself and the proof is complete. It remains only to consider the case in which conditions (ii–iv) fail to hold for any $w_{(\mu)}$, and to show that condition (i) holds for w itself.

To do so, first introduce some further notation. For $i = 1, \ldots, h$ denote by $\varphi_i : \mathbb{R}^n \to \mathbb{R}$ the function $\varphi_i(x) = \tanh(b_i x + c_i)$. Similarly for $j = 1, \ldots, h'$ denote by $\psi_j : \mathbb{R}^n \to \mathbb{R}$ the function $\psi_j(x) = \tanh(b'_j x + c'_j)$. Then by assumption, no φ_i is constant (ii) and no two are proportional (iii, iv). The same holds for the ψ_j—conditions (i–iv) do not hold for $w'_{(\mu)}$, since h' was assumed to be minimal from the start. Yet, for $\mu = 1, \ldots, m$, the linear combination of functions

$$d_\mu + \sum_{i=1}^{h} a_{i,\mu} \varphi_i - d'_\mu - \sum_{j=1}^{h'} a'_{j,\mu} \psi_j = f_w^{(\mu)} - f_{w'}^{(\mu)} = 0$$

yields the zero function. This linear combination remains when excluding those terms with $a_{i,\mu} = 0$ or $a'_{j,\mu} = 0$. Applying the same reasoning as that in Sussmann (1992), due to the independence property of the hyperbolic tangent function (Sussmann, 1992, Lemma 3.1) the remaining terms must be in bijection, such that

$$\varphi_i = \pm \psi_j \tag{†}$$

for some j with $a'_{j,\mu} \neq 0$ for each i with $a_{i,\mu} \neq 0$.

To complete the proof, note that these relationships (†) between the units of w and w' are independent of μ. However, the relationships are "exclusive" in the sense that no two φ_i can be proportional to the same ψ_j, else they would also be proportional to each other—a possibility handled already above. Since there are only h' units $\psi_1, \ldots, \psi_{h'}$, it follows that there must be one hidden unit i (actually at least $h - h'$ many units) for which $a_{i,\mu} = 0$ for all $\mu = 1, \ldots, m$, allowing φ_i to avoid any such relationship. That is, $a_i = (a_{i,1}, \ldots, a_{i,m}) = 0$, satisfying condition (i) for w as required. □

Remark 3.26. The equivalence relies crucially on the fact that the parameter space is unbounded. In practice, it is common to consider a bounded parameter space. Such a restriction falsifies the converse direction (reducibility implies non-minimality), as exemplified below (Example 3.27). Conversely, the forward direction (non-minimality implies reducibility) holds despite any restriction of the parameter space.

Example 3.27. Consider the following setting, like the simple setting, but in which the parameter space for $\mathcal{A}_h^{n,m}$ is constrained with some maximum absolute weight/bias $M > 0$ (that is, $\mathcal{W}_h^{n,m} = [-M, M]^{(n+m+1)h+m} \subset \mathbb{R}^{(n+m+1)h+m}$). Then the parameter $(M, M, 1, 1, 0, 0, 0) \in \mathcal{W}_2^{1,1}$, realising the function $M \tanh(x) + M \tanh(x) = 2M \tanh(x)$, meets condition (iii) of Definition 3.22 and is formally reducible. However, $2M \tanh(x)$ is not representable as a single unit with bounded weights, so the parameter is minimal.

Remark 3.28. Reducibility is defined as a property of a neural network parameter. However, non-minimality depends only on the implemented neural network function. It follows that if two parameters in $\mathcal{W}_h^{n,m}$ are functionally equivalent, then they are either both reducible or both irreducible.

Remark 3.29. Moreover, it is possible to define the reducibility of neural network functions (with respect to a given architecture). Given a simple neural network architecture $\mathcal{A}_h^{n,m}$, the *reducible neural network functions* are those appearing in the function family $\mathcal{F}_{h-1}^{n,m}$ (which includes all smaller families by Proposition 3.19). See Figure 3.5.

Remark 3.30. Note carefully that the definition *is* sensitive to the choice of architecture. For example if $w' \in \mathcal{W}_{h'}^{n,m}$ is irreducible, then, with respect to $\mathcal{A}_h^{n,m}$ for $h > h'$, all functionally equivalent parameters, and f_w itself, are reducible (with w' as witness).

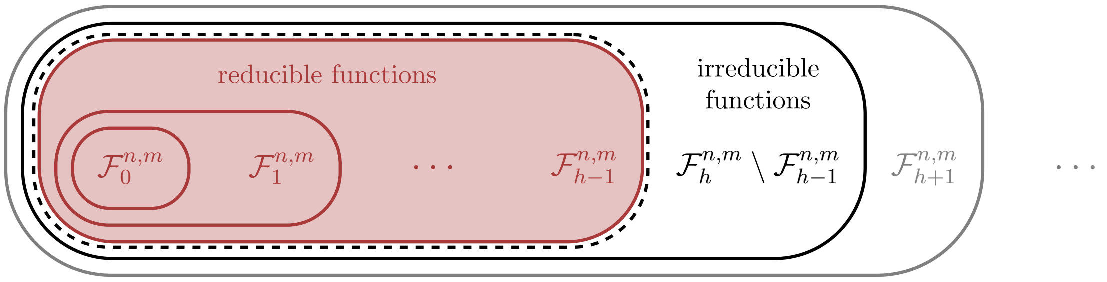

reducible functions
$\mathcal{F}_0^{n,m}$
$\mathcal{F}_1^{n,m}$
$\cdots$
$\mathcal{F}_{h-1}^{n,m}$
irreducible functions
$\mathcal{F}_h^{n,m} \setminus \mathcal{F}_{h-1}^{n,m}$
$\mathcal{F}_{h+1}^{n,m}$
$\cdots$

Figure 3.5: Conceptual illustration of reducibility and irreducibility of neural networks in terms of function families. The reducible neural network functions are those implementable within a smaller architecture, that is, those neural network functions from those smaller architectures' function families. Note however that this definition is relative to a choice of architecture.

Permutation and negation transformations

In order to connect reducibility to the third characterisation of degeneracy, I need to review the basic operations of *unit exchange* and *unit negation* characterising the functional equivalence class in the non-degenerate case (cf. Section 2.3). I begin by formalising these operations with the following definitions.

Definition 3.31 (Permutation)**.** Let $h \in \mathbb{N}$. A *permutation* (on $\{1, \ldots, h\}$) is a bijective function $\pi : \{1, \ldots, h\} \to \{1, \ldots, h\}$. The set of all permutations on $\{1, \ldots, h\}$ is the *symmetric group on* $\{1, \ldots, h\}$, denoted S_h.

Definition 3.32 (Simple neural network permutation)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $\pi \in S_h$ be a permutation. The *simple neural network permutation* based on π, denoted $T_\pi : \mathcal{W}_h^{n,m} \to \mathcal{W}_h^{n,m}$, is a parameter transformation function mapping $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$ to

$$T_\pi(w) = (a_{\pi(1)}, \ldots, a_{\pi(h)}, b_{\pi(1)}, \ldots, b_{\pi(h)}, c_{\pi(1)}, \ldots, c_{\pi(h)}, d).$$

Remark 3.33. Conceptually, T_π applies the permutation π to the hidden units of a simple neural network, yielding an isomorphic weighted connection graph. Note that this affects a series of individual unit exchange operations.

Definition 3.34 (Sign vector)**.** Let $h \in \mathbb{N}$. A *sign vector* (of dimension h) is a vector $\sigma \in \{-1, +1\}^h$.

Definition 3.35 (Simple neural network negation)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $\sigma \in \{-1, +1\}^h$ be a sign vector. The *simple neural network negation* based on σ, denoted $T_\sigma : \mathcal{W}_h^{n,m} \to \mathcal{W}_h^{n,m}$, is a parameter transformation function mapping $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$ to

$$T_\sigma(w) = (\sigma_1 a_1, \ldots, \sigma_h a_h, \sigma_1 b_1, \ldots, \sigma_h b_h, \sigma_1 c_1, \ldots, \sigma_h c_h, d).$$

Remark 3.36. Conceptually, T_σ negates the incoming weights, biases, and outgoing weights of a subset of the hidden units of a simple neural network.

Proposition 3.37. *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, a permutation $\pi \in S_h$, and a sign vector $\sigma \in \{-1, +1\}^h$. Then T_π and T_σ are isometries with respect to the uniform distance. That is, for $u, v \in \mathcal{W}_h^{n,m}$,*

$$\|T_\pi(u) - T_\pi(v)\|_\infty = \|u - v\|_\infty = \|T_\sigma(u) - T_\sigma(v)\|_\infty .$$

Proof. Follows since T_π and T_σ are linear and the uniform norm is defined as an (unordered) maximum of the absolute value of a vector's components. $\square$

Functional equivalence class for irreducible parameters

The third characterisation of degeneracy is as the parameters whose functional equivalence class is characterised by more than just the operations defined above. A full characterisation of the richer functional equivalence classes of reducible parameters is deferred to Chapter 5, but here I review (a slight extension of) the result of Sussmann (1992) that these transformations exhaustively describe the functional equivalence class for irreducible parameters.

Definition 3.38 (Functional equivalence class)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a parameter $w \in \mathcal{W}_h^{n,m}$, the *functional equivalence class* of w, denoted $\mathfrak{F}[w]$, is the set of functionally equivalent parameters

$$\mathfrak{F}[w] = \{\, v \in \mathcal{W}_h^{n,m} \mid f_w = f_v \,\} \subset \mathcal{W}_h^{n,m}.$$

Theorem 3.39 (Functional equivalence class for irreducible parameters)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, and a parameter $w \in \mathcal{W}_h^{n,m}$. If w is irreducible, then the functional equivalence class $\mathfrak{F}[w]$ is the following discrete set of cardinality $h! \cdot 2^h$:*

$$\mathfrak{F}[w] = \bigcup_{\pi \in S_h} \bigcup_{\sigma \in \{-1,+1\}^h} \{T_\sigma(T_\pi(w))\}\,.$$

Proof. (Cardinality): Since w is irreducible, the incoming weight and bias vectors are non-zero and have distinct lexicographic absolute values (Definition 3.22). It follows that each permutation and negation transformation produces a distinct parameter vector. There are exactly $h! \cdot 2^h$ such transformations since $|S_h| = h!$ and $\left|\{-1,+1\}^h\right| = 2^h$.

($\supset$): Let $\sigma \in \{-1,+1\}^h$ and $\pi \in S_h$. Write $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$ and $v = T_\sigma(T_\pi(w))$. For $x \in \mathbb{R}^n$, since $\tanh(z) = \pm\tanh(\pm z)$ and the sum is commutative, $f_v(x) = d + \sum_{i=1}^h \sigma_i a_{\pi(i)} \tanh\left(\sigma_i b_{\pi(i)} x + \sigma_i c_{\pi(i)}\right) = d + \sum_{i=1}^h a_i \tanh(b_i x + c_i) = f_w(x)$. Thus $T_\sigma(T_\pi(w)) \in \mathfrak{F}[w]$.

($\subset$): Proven by Sussmann (1992, Theorem 2.1) if $m = 1$. I reduce the $m > 1$ case to Sussmann's result as follows. Suppose $w' \in \mathfrak{F}[w]$. Introduce the same decomposition of the two neural networks as in the proof of Theorem 3.25, namely, the component functions $f_w^{(1)}, \ldots, f_w^{(m)}, f_{w'}^{(1)}, \ldots, f_{w'}^{(m)} \in \mathcal{F}_h^{n,1}$ implemented by the parameter subvectors $w_{(1)}, \ldots, w_{(m)}, w'_{(1)}, \ldots, w'_{(m)} \in \mathcal{W}_h^{n,1}$ (cf. Figure 3.4). For $\mu = 1, \ldots, m$, note that $w'_{(\mu)} \in \mathfrak{F}\left[w_{(\mu)}\right] \subset \mathcal{W}_h^{n,1}$, so by Sussmann (1992, Theorem 2.1), there exists $\sigma^{(\mu)} \in \{-1,+1\}^h$ and $\pi^{(\mu)} \in S_h$ such that $w'_{(\mu)} = T_{\sigma^{(\mu)}}(T_{\pi^{(\mu)}}(w_{(\mu)}))$. Now since the component parameters share incoming weight and bias vectors, by the cardinality argument above, $\sigma^{(1)} = \cdots = \sigma^{(m)}$ and $\pi^{(1)} = \cdots = \pi^{(m)}$. Dispensing with superscripts, it follows that $w' = T_\sigma(T_\pi(w))$. □

Reducible regions of parameter space

I turn from individual reducible parameters to the corresponding subset of the parameter space. Proposition 3.42 characterises this subset as a union of linear subspaces.

Definition 3.40 (Reducible region)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. The *reducible region* of $\mathcal{W}_h^{n,m}$, denoted $\mathfrak{R}[\mathcal{W}_h^{n,m}]$, is the subset of reducible parameters:

$$\mathfrak{R}[\mathcal{W}_h^{n,m}] = \{\, w \in \mathcal{W}_h^{n,m} \mid w \text{ is reducible} \,\}.$$

Definition 3.41 (Irreducible region)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. The *irreducible region* of $\mathcal{W}_h^{n,m}$, denoted $\mathfrak{I}[\mathcal{W}_h^{n,m}]$, is the subset of irreducible parameters:

$$\mathfrak{I}[\mathcal{W}_h^{n,m}] = \{\, w \in \mathcal{W}_h^{n,m} \mid w \text{ is irreducible} \,\}.$$

Equivalently, the irreducible region is the complement of the reducible region:

$$\mathfrak{I}[\mathcal{W}_h^{n,m}] = \mathcal{W}_h^{n,m} \setminus \mathfrak{R}[\mathcal{W}_h^{n,m}]\,.$$

Proposition 3.42. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$. The reducible region $\mathfrak{R}[\mathcal{W}_h^{n,m}] \subset \mathcal{W}_h^{n,m}$ is the union of h^2+h distinct linear subspaces:*

$$\mathfrak{R}[\mathcal{W}_h^{n,m}] = \bigcup_{i=1}^{h} (A_i \cup B_i) \cup \bigcup_{i=1}^{h} \bigcup_{j=i+1}^{h} (C_{i,j} \cup D_{i,j})\,,$$

where, for $i = 1, \ldots, h$ and $j = i+1, \ldots, h$,

$$\begin{aligned}
A_i &= \{\, (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \mid a_i = 0 \,\},\\
B_i &= \{\, (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \mid b_i = 0 \,\},\\
C_{i,j} &= \{\, (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \mid b_i = b_j, c_i = c_j \,\}, \qquad \textit{and}\\
D_{i,j} &= \{\, (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \mid b_i = -b_j, c_i = -c_j \,\}.
\end{aligned}$$

Proof. By definition a parameter $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$ is reducible if and only if satisfies at least one of the equations $a_i = 0$, $b_i = 0$, $(b_i, c_i) = (b_j, c_j)$, or $(b_i, c_i) = -(b_j, c_j)$ for some i and some $j > i$. Satisfying these equations corresponds to membership in the subspaces A_i, B_i, $C_{i,j}$, or $D_{i,j}$, respectively. □

Visualising reducible regions

For suitably small architectures it is possible to effectively visualise the reducible region. Consider the simple neural network architecture $\mathcal{A}_1^{1,1}$. With just one hidden unit, only reducibility conditions (i) and (ii) are in play, and the first two dimensions of $\mathcal{W}_1^{1,1} = \mathbb{R}^4$ determine reducibility. This allows one to visualise the reducible region in the plane. Similarly, one can visualise the reducible region of $\mathcal{W}_1^{2,1} = \mathbb{R}^5$ and $\mathcal{W}_1^{1,2} = \mathbb{R}^6$ in three dimensions. Figure 3.6 illustrates $\mathfrak{R}\left[\mathcal{W}_1^{1,1}\right]$, $\mathfrak{R}\left[\mathcal{W}_1^{2,1}\right]$, and $\mathfrak{R}\left[\mathcal{W}_1^{1,2}\right]$ in this way.

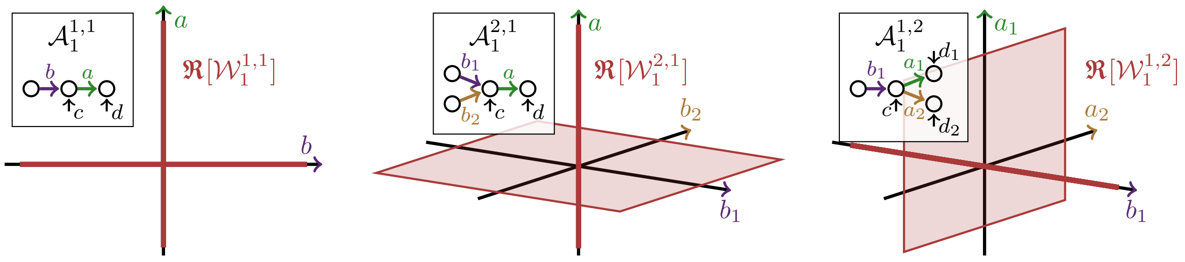


Figure 3.6: Visualisation of the reducible regions (**red**) for three single-hidden-unit architectures, ($\mathcal{A}_1^{1,1}$, $\mathcal{A}_1^{2,1}$, and $\mathcal{A}_1^{1,2}$). A slice of the parameter space (respectively $\mathcal{W}_1^{1,1} = \mathbb{R}^4$, $\mathcal{W}_1^{2,1} = \mathbb{R}^5$, and $\mathcal{W}_1^{1,2} = \mathbb{R}^6$) is shown with the active dimensions indicated in the legend (the shape is constant in the other dimensions). The regions generally have one component where a or (a_1, a_2) is zero (reducibility condition (i)) and another where b or (b_1, b_2) is zero (reducibility condition (ii)).

Consider the simple neural network architecture $\mathcal{A}_2^{1,1}$ with parameter space $\mathcal{W}_2^{1,1} = \mathbb{R}^7$ With the introduction of a second hidden unit, conditions (iii) and (iv) become relevant. To emphasise these conditions, fix the outgoing weights a_1, a_2 non-zero, and one incoming weight b_2 positive (and fix d arbitrarily). There remain three dimensions within which the region of reducibility can be visualised, as in Figure 3.7.

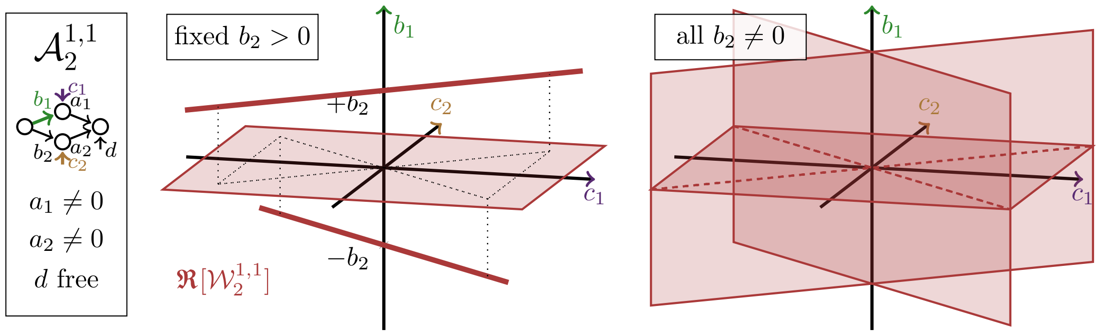


Figure 3.7: Visualisation of the reducible region (**red**) for $\mathcal{A}_2^{1,1}$, an architecture with two hidden units. Left: slice of $\mathcal{W}_2^{1,1} = \mathbb{R}^7$. The horizontal plane corresponds to condition (ii). The diagonal horizontal lines correspond to conditions (iii) and (iv). Right: a projection of the region into three dimensions. Shows the effect of varying $b_2 \neq 0$, raising or lowering the diagonal lines ($b_2 = 0$ is excluded because that entire hyperplane is reducible).

Properties of reducible and irreducible regions

It is difficult to visualise reducible and irreducible regions in higher-dimensional parameter spaces. Corollary 3.43 provides some scalable intuition in terms of topology (see, e.g., Munkres, 1974), algebraic geometry (see, e.g., Cox et al., 2015), and measure theory (see, e.g., Cohn, 2013).

Corollary 3.43. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$.*

(i) In the standard topology on $\mathcal{W}_h^{n,m}$, the reducible region $\mathfrak{R}[\mathcal{W}_h^{n,m}]$ is a closed set with empty interior, and the irreducible region $\mathfrak{I}[\mathcal{W}_h^{n,m}]$ is a dense, open set.

(ii) The reducible region $\mathfrak{R}[\mathcal{W}_h^{n,m}] \subset \mathcal{W}_h^{n,m}$ is a real affine algebraic set.

(iii) The reducible region $\mathfrak{R}[\mathcal{W}_h^{n,m}] \subset \mathcal{W}_h^{n,m}$ has Lebesgue measure zero.

Proof. (i): Each of the (proper) linear subspaces A_i, B_i, $C_{i,j}$, and $D_{i,j}$ is closed and has empty interior. It follows that their (finite) union, $\mathfrak{R}[\mathcal{W}_h^{n,m}]$, is closed and has empty interior. Then the complement, $\mathfrak{I}[\mathcal{W}_h^{n,m}]$, is open and dense.

(ii): Each of the linear subspaces from Proposition 3.42 is a real affine algebraic set: $A_i = \mathbb{V}(a_{i,1}, \ldots, a_{i,m})$, $B_i = \mathbb{V}(b_{i,1}, \ldots, b_{i,n})$, $C_{i,j} = \mathbb{V}(b_{i,1} - b_{j,1}, \ldots, b_{i,n} - b_{j,n}, c_i - c_j)$, and $D_{i,j} = \mathbb{V}(b_{i,1} + b_{j,1}, \ldots, b_{i,n} + b_{j,n}, c_i + c_j)$. As the union of a finite collection of real affine algebraic sets, the reducible region $\mathfrak{R}[\mathcal{W}_h^{n,m}]$ is real affine algebraic.

(iii): Each of the (proper) linear subspaces A_i, B_i, $C_{i,j}$, and $D_{i,j}$ has Lebesgue measure zero. It follows that their (finite) union, $\mathfrak{R}[\mathcal{W}_h^{n,m}]$, has Lebesgue measure zero. □

Remark 3.44. Two intuitive consequences of Corollary 3.43(i) are:

1. There is an open neighbourhood around every irreducible parameter containing only irreducible parameters.
2. In every open neighbourhood of a reducible parameter, there is an irreducible parameter.

Remark 3.45. Corollary 3.43(iii) forms part of the technical justification for the emphasis on the irreducible (non-degenerate) case in prior work. The main chapters of this thesis amount to an exploration of the structure of the inside of this measure zero subset (Chapters 4 and 5) and of its neighbourhood (Chapter 6).

3.4 Computational complexity theory

In Chapter 6 I draw on computational complexity theory to show that determining the proximity of a parameter to highly degenerate subsets of the parameter space, and exactly measuring the degree of approximate degeneracy of a parameter, is an intractable problem. In this section I review the requisite preliminaries.

1. I recall the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard, and the concept of polynomial-time problem reduction.
2. I recall classical results on the hardness of the Boolean satisfiability problem and some restricted variants (including a minor extension of these results).

For a more detailed and rigorous introduction to these topics (for example, in terms of encodings, languages, and Turing machines), consult Garey and Johnson (1979).

Complex computational problems

Computational complexity theory begins with *computational problems.* Computational problems are formal specifications of the desired behaviour of a computer system performing some task. Specifying a computational problem requires outlining the set of possible *problem instances,* or inputs to the system, and the *requirements,* or desired properties of the system outputs. The *solution* to a computational problem is an algorithm that, given any particular problem instance, computes an output satisfying the requirements.

Problems can be loosely classified based on the nature of their requirements. Two examples of such classes are as follows.

1. *Decision problems,* in which the requirement is for the system to output an answer to a binary "yes" or "no" question about the instance (consider, for example, the Boolean satisfiability problem, reviewed below).
2. *Optimisation problems,* in which the system must produce an optimal object with respect to some criterion (consider, for example, the learning problem).

A fundamental question arises when solving computational problems: what is the most efficient solution for a given problem, in terms of computational resource usage? The asymptotic performance of the most efficient algorithms for a computational problem is a measure of the difficulty—or *computational complexity*—of the problem. Computational complexity theory classifies computational problems into *complexity classes* based on their difficulty. Well-known examples of complexity classes include the classes $\mathcal{P}$ and $\mathcal{NP}$, including, roughly, decision problems for which outputs can be computed (respectively, the correctness of "yes" outputs verified given a certificate) in a number of steps polynomial in the size of the instance, using a deterministic model of computation.

Boolean satisfiability

Boolean satisfiability is an important decision problem, in which the problem instances are Boolean logic formulae (typically represented in conjunctive normal form), and the requirement is to output "yes" if there is a satisfying assignment of truth values to the Boolean variables that makes the whole formula evaluate to "true", or else to output "no." The Boolean satisfiability problem can be formalised as follows.[6]

Definition 3.46 (Boolean formula)**.** Given n *variables* $v_1, \ldots, v_n$, a *Boolean formula* (in *conjunctive normal form*) is a finite conjunction of *clauses,* where each clause is a finite disjunction of *literals,* and each literal is either a variable v_i or its negation $\bar{v}_i$ (called, respectively, a *positive occurrence* or *negative occurrence* of the variable v_i).

Definition 3.47 (Satisfiable Boolean formula)**.** Consider a Boolean formula ϕ. A *truth assignment* is a mapping assigning each of the variables of ϕ to the values "true" or "false." The formula ϕ is *satisfiable* if there exists a truth assignment such that each clause contains at least one positive occurrence of a variable assigned "true," or at least one negative occurrence of a variable assigned "false" (such that the entire formula logically evaluates to "true").

Example 3.48. Consider six variables denoted $v_1, \ldots, v_6$. Using $\wedge$ to denote conjunction and $\vee$ to denote disjunction, three example Boolean formulae are as follows.

$$\begin{aligned}
\phi_1 &= (v_1 \vee v_2) \wedge (\bar{v}_1 \vee v_2) \wedge (v_3 \vee v_4) \wedge (\bar{v}_3 \vee v_4) \wedge (\bar{v}_2 \vee \bar{v}_4) \\
\phi_2 &= (\bar{v}_1 \vee v_2) \wedge (v_1 \vee v_2 \vee \bar{v}_3) \wedge (\bar{v}_2 \vee v_3) \\
\phi_3 &= (\bar{v}_1 \vee \bar{v}_3 \vee v_4) \wedge (v_1 \vee \bar{v}_2 \vee v_5) \wedge (v_3 \vee \bar{v}_4 \vee v_6) \wedge (v_2 \vee \bar{v}_5) \wedge (v_5 \vee v_6) \wedge (v_4 \vee \bar{v}_6)
\end{aligned}$$

Enumerating possible truth assignments shows that ϕ_1 is unsatisfiable whereas ϕ_2 and ϕ_3 are each satisfiable.

Problem 3.49 (Boolean satisfiability, `SAT`)**.** Given a Boolean formula ϕ, decide whether the formula ϕ is satisfiable.

A simple algorithm solving the Boolean satisfiability problem involves enumerating and evaluating the 2^n possible truth assignments. While various improvements can be made to the above (see, e.g., Zhang and Malik, 2002), the precise computational complexity of Boolean satisfiability is a long-standing open problem. Since a satisfying truth assignment can be verified in polynomial time, $\texttt{SAT} \in \mathcal{NP}$. However, no deterministic polynomial-time solution is known to exist.

[6] For a more rigorous formalisation, consult, for example, Wolf (2005, §1.2).

$\mathcal{NP}$-complete problems and problem reductions

If one had a polynomial-time solution for Boolean satisfiability, then it could be transformed into a polynomial-time solution for any problem in $\mathcal{NP}$. This is a property of the complexity class $\mathcal{NP}$-complete, defined as follows.[7]

Definition 3.50 (Problem reduction)**.** Let X and Y be two decision problems. A *(polynomial-time) problem reduction* from X to Y is a deterministic polynomial-time algorithm that takes an instance of the problem X as input and returns an *equivalent* instance of the problem Y as output, where an instance of Y is *equivalent* to an instance of X if both instances require the same decision under their respective problems.

Definition 3.51 (Reducibility between computational problems)**.** Let X and Y be decision problems. If there exists a problem reduction from X to Y, say X *is (polynomial-time) reducible to* Y and write

$$X \xrightarrow{\mathcal{P}} Y.$$

Definition 3.52 ($\mathcal{NP}$-complete problem)**.** A decision problem $Y \in \mathcal{NP}$ is $\mathcal{NP}$*-complete* if all problems in $\mathcal{NP}$ are reducible to Y (that is, $\forall X \in \mathcal{NP}, X \xrightarrow{\mathcal{P}} Y$).

The Cook–Levin theorem says that `SAT` is $\mathcal{NP}$-complete (Cook, 1971; Levin, 1973). The proof of this theorem involves constructing a Boolean formula encoding the computation of a verification algorithm for an arbitrary problem in $\mathcal{NP}$ (see, e.g., Cook, 1971; Garey and Johnson, 1979, §2.6; Sipser, 2013, §7.4). This result is important because reducibility is a transitive relation. Therefore, having established `SAT` as an $\mathcal{NP}$-complete problem, additional decision problems can be shown to be $\mathcal{NP}$-complete by reduction from `SAT`, or from those problems in turn (Cook, 1971; Karp, 1972; Garey and Johnson, 1979). In general, this methodology for proving a new problem to be $\mathcal{NP}$-complete proceeds as follows (Garey and Johnson, 1979, §3):

1. show that the new problem is itself in $\mathcal{NP}$ (by constructing a verification algorithm that runs in polynomial time); and

2. reduce a known $\mathcal{NP}$-complete problem to the new problem (including showing that the reduction runs in polynomial time).

Remark 3.53. By definition, the complexity class $\mathcal{NP}$-complete contains only problems in $\mathcal{NP}$, which in turn contains only decision problems. A related complexity class, $\mathcal{NP}$-hard, contains problems that are not necessarily in $\mathcal{NP}$, and are not necessarily decision problems, but to which all problems in $\mathcal{NP}$ are still reducible (with a suitably modified definition of reducibility for non-decision problems). In this thesis, I state my main results in terms of $\mathcal{NP}$-completeness and, where appropriate, comment on $\mathcal{NP}$-hardness of related non-decision variants, but I omit a formal treatment of $\mathcal{NP}$-hardness.

[7] For a more rigorous definition, consult Garey and Johnson (1979, §2.5).

Restricted Boolean satisfiability

In Chapter 6, I reduce several computational problems to SAT. This involves translating arbitrary instances of SAT into instances of the new problems. To substantially simplify the reductions in Chapter 6, I introduce a SAT-variant called *restricted Boolean satisfiability* (denoted xSAT), which has simpler instances, but is still $\mathcal{NP}$-complete (by reduction from SAT itself; Theorem 3.58). It therefore suffices to consider only instances of xSAT in Chapter 6.

Definition 3.54 (Bipartite variable–clause incidence graph)**.** Given a Boolean formula ϕ with variables $v_1, \ldots, v_n$ and clauses $c_1 \wedge \cdots \wedge c_m$, the *bipartite variable–clause incidence graph* is an undirected graph (V, E) with vertices $V = \{v_1, \ldots, v_n, c_1, \ldots, c_m\}$ and edges

$$E = \{\, \{v_i, c_j\} \mid \text{ variable } v_i \text{ occurs (as a positive or negative literal) in clause } c_j \,\}.$$

Definition 3.55 (Restricted Boolean formula)**.** A *restricted Boolean formula* is a Boolean formula ϕ with variables $v_1, \ldots, v_n$ and clauses $c_1 \wedge \cdots \wedge c_m$, meeting the following additional requirements:

1. Each variable v_i occurs in either two clauses or three clauses, including exactly one negative occurrence (and one or two positive occurrences).
2. Each clause c_j contains either two or three literals (these may be any combination of positive and negative).
3. The bipartite variable–clause incidence graph of ϕ is planar.

Problem 3.56 (Restricted Boolean satisfiability, xSAT)**.** Given a restricted Boolean formula ϕ, decide whether the formula is satisfiable.

Example 3.57. Observe that ϕ_1, ϕ_2, and ϕ_3 of Example 3.48 meet the restrictions in Definition 3.55 (for planarity, see Figure 3.8).

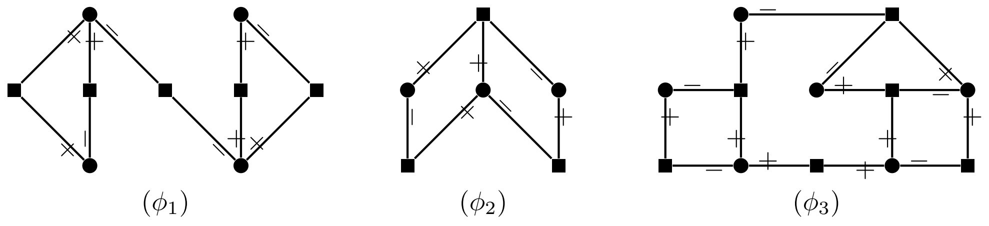


Figure 3.8: Examples of planar bipartite variable clause incidence graphs for the formulae in Examples 3.48 and 3.57. Circles indicate variable vertices. Squares indicate clause vertices. Positive and negative occurrences (edges) are marked accordingly.

Theorem 3.58 (xSAT is $\mathcal{NP}$-complete)**.**

Proof. xSAT $\in \mathcal{NP}$ as a restricted variant of SAT $\in \mathcal{NP}$. To show SAT $\xrightarrow{\mathcal{P}}$ xSAT much of the work is already done:

1. Cook (1971) reduced SAT to 3-SAT, a variant with at most three literals per clause.
2. Lichtenstein (1982) extended this reduction to planar 3-SAT, a variant with at most three literals per clause and a planar bipartite clause–variable incidence graph (in fact the planarity condition studied by Lichtenstein is even stronger).
3. Cerioli et al. (2004; 2011) extended the reduction to planar 3-SAT$_{\overline{3}}$—a variant of planar 3-SAT with at most three occurrences per variable.[8]

It remains to (efficiently) construct from an instance ϕ of planar 3-SAT$_{\overline{3}}$ an equisatisfiable formula ϕ' having *additionally* (a) at least two occurrences per variable, (b) at least two variables per clause, and (c) exactly one negative occurrence per variable. This can be achieved by removing variables, literals, and clauses and negating occurrences in ϕ (noting that such operations do not affect the conditions on ϕ) as follows. First, establish (a) and (b)[9] by exhaustively applying the following (polynomial-time) operations.

(i) If a variable always occurs with one sign (including never or once), remove the variable and all incident clauses. The resulting (sub)formula is equisatisfiable: extend a satisfying assignment by satisfying the removed clauses with the removed variable.

(ii) If a clause contains a single literal, this variable is determined in a satisfying assignment. Remove the variable and clause along with any other clauses in which the variable occurs with that sign. For other occurrences, retain the clause but remove the literal.[9] If the resulting formula is unsatisfiable, then the additional variable won't help, and if the resulting formula is satisfiable, then so is the original with the appropriate setting of the variable to satisfy the singleton clause.

Only a polynomial number of operations are possible as each removes one variable. Moreover, thanks to (i), each variable with two occurrences now has one negative occurrence (as required). For variables with three occurrences, one or two are negative. Establish (c) by negating all three occurrences for those that have two negative occurrences (so that the two become positive and the one becomes negative as required). The result is equisatisfiable because satisfying assignments can be translated by negating the truth value assigned to this variable. Carrying out this negation operation for the necessary variables takes polynomial time and completes the reduction. □

[8]Cerioli et al. (2004; 2011) used similar techniques to Tovey (1984), Jansen and Müller (1995), and Berman et al. (2003), who studied variants of SAT with bounded occurrences per variable.

[9]If a clause contains no literals, whether this is allowed initially or caused by the removal of a literal through operation (ii), then the formula is unsatisfiable. Short circuit the entire reduction, returning a trivial unsatisfiable instance of xSAT, such as ϕ_1 of Examples 3.48 and 3.57.

Chapter 4

Neural Network Reduction and Rank

The notion of reducibility (Sussmann, 1992; cf. Section 3.3) delineates the boundary between degenerate and non-degenerate neural networks. Further analysis of the properties and geometry of individual degenerate neural networks depends on the exact nature and extent of their degeneracy. In this chapter, I develop an algorithmic framework for analysing and measuring degeneracy in simple neural networks (neural networks with a single hidden layer of biased units and the hyperbolic tangent activation function, cf. Section 3.2). In particular, my analysis proceeds as follows:

1. In Section 4.1, I define the *rank,* an idealised measure of the degeneracy of a simple neural network parameter or function based on the minimum number of hidden units required to implement the same function. I discuss the rank from several alternative perspectives.

2. In Section 4.2, I give two algorithms for *reducing* a simple neural network parameter into a minimal implementation of the same neural network function, based on the repeated application of the reducibility conditions (cf. Section 3.3). I also give an efficient algorithm for computing the rank of a simple neural network parameter.

3. In Section 4.3, I characterise and study the properties of subsets of a given parameter space with a given maximum rank, showing that these subsets have many properties in common with the reducible regions studied in Section 3.3.

4. In Section 4.4, I study the properties of families of simple neural network functions with a given maximum rank. In particular, I show that such function families are highly non-convex (as a set of functions).

The framework developed in this chapter, especially in Sections 4.1 and 4.2, serves as the basis for my investigation of the geometry of degenerate neural networks in Chapter 5 and the measure of approximate degeneracy studied in Chapter 6 (which is based on the neighbourhoods of bounded rank regions of parameter space).

4.1 The rank of a simple neural network

In this section, I define the *rank* of a simple neural network parameter as the minimum number of hidden units required to implement the same neural network function. I interpret the rank from several additional perspectives:

1. Rank generalises the concept of reducibility of simple neural networks (Section 3.3).
2. Rank stratifies the containment hierarchy of function families (Section 3.2).
3. Rank measures the optimal lossless compressibility of a neural network function, in terms of the number of hidden units that can be removed without changing the function implemented.
4. Rank can be seen as a generalisation of the homonymous measure familiar from linear algebra, the rank of a matrix or linear transformation, which corresponds to the minimum number of hidden units required to implement a function with an unbiased *linear* neural network.

Defining rank

I formalise the definition sketched above as follows.

Definition 4.1 (Rank of a simple neural network parameter)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $w \in \mathcal{W}_h^{n,m}$. Define the *rank* of w, denoted $\operatorname{rank}(w)$, as the smallest r for which $f_w \in \mathcal{F}_r^{n,m}$:

$$\operatorname{rank}(w) = \min \{ r \in \mathbb{N} \mid f_w \in \mathcal{F}_r^{n,m} \}.$$

That is, the rank of w is the minimum number of hidden units required to implement f_w.

Example 4.2. Consider $w = (1, 1, 1, 1, 1, -1, 0, -1, 1, 0) \in \mathcal{W}_3^{1,1}$. Then

$$f_w(x) = \tanh(x) + \tanh(x-1) + \tanh(1-x) = \tanh(x).$$

Thus $f_w \in \mathcal{F}_1^{1,1} \subset \mathcal{F}_2^{1,1} \subset \cdots$. Moreover, $f_w \notin \mathcal{F}_0^{1,1}$ (= constant functions). Thus

$$\operatorname{rank}(w) = \min \{1, 2, \ldots\} = 1.$$

Remark 4.3. The rank is defined as a property of a neural network parameter. However, the definition depends on the implemented neural network function, rather than the parameter itself. If two parameters in $\mathcal{W}_h^{n,m}$ are functionally equivalent, then they have the same rank.

Remark 4.4. The minimum number of units required to implement a neural network function does not depend on the original number of units used to specify the function. It follows that if two neural network parameters in two *different* parameter spaces are functionally equivalent, then they still have the same rank.

Remarks 4.3 and 4.4 indicate that rank is also a well-defined property of a simple neural network function. Unlike for reducibility (cf. Remark 3.30), the definition is even independent of the choice of architecture in which to represent the functions. This leads to the following definition of the rank of a simple neural network function.

Definition 4.5 (Rank of a simple neural network function)**.** Consider the extended family of simple neural network functions $\mathcal{F}_{\infty}^{n,m}$. Let $f \in \mathcal{F}_{\infty}^{n,m}$. Define the *rank* of f, denoted $\operatorname{rank}(f)$, as the smallest number of hidden units r such that f is in the function family $\mathcal{F}_{r}^{n,m}$. That is,

$$\operatorname{rank}(f) = \min \{\, r \in \mathbb{N} \,|\, f \in \mathcal{F}_{r}^{n,m} \,\}.$$

Interpretation 1: Rank as a generalisation of reducibility

Recall that, as illustrated in Figure 3.5, reducible neural networks are those whose neural network functions arise at some earlier layer of the containment hierarchy than the layer corresponding to the present architecture. This suggests asking, *how much* earlier does a given neural network function arise? This question is answered by the rank, which corresponds to the smallest h' available for the definition of non-minimality.

Unlike reducibility, the rank of a neural network is independent of a choice of architecture. However, given any particular architecture, rank and reducibility are closely linked, as captured in the following result.

Proposition 4.6. *Consider a simple neural network architecture $\mathcal{A}_{h}^{n,m}$ with parameter space $\mathcal{W}_{h}^{n,m}$. For a neural network parameter $w \in \mathcal{W}_{h}^{n,m}$,*

(i) $\operatorname{rank}(w) \le h$*;*

(ii) w is reducible if and only if $\operatorname{rank}(w) < h$*; and*

(iii) w is irreducible if and only if $\operatorname{rank}(w) = h$*.*

Proof. (i): Since $f_w \in \mathcal{F}_{h}^{n,m}$, h is in the set whose minimum is the rank of w (Definition 4.1). It follows that h is an upper bound on the rank.

(ii): If w is reducible then by Theorem 3.25 it is also non-minimal, that is, there is a functionally equivalent $w' \in \mathcal{W}_{h'}^{n,m}$ where $h' < h$. It follows that $f_w = f_{w'} \in \mathcal{F}_{h'}^{n,m}$. Then, by (i), $\operatorname{rank}(w) \le h' < h$. Conversely, if $\operatorname{rank}(w) = h' < h$ then by definition $f_w \in \mathcal{F}_{h'}^{n,m}$. It follows that there is a functionally equivalent $w' \in \mathcal{W}_{h'}^{n,m}$, and so w is non-minimal and reducible.

(iii): Follows from (i) and (ii). □

Interpretation 2: Rank stratifies the containment hierarchy

The rank of a function allows one to define a stratified version of the containment hierarchy of simple neural network function families. This notion is formalised in Definition 4.7 and Proposition 4.8 and illustrated in Figure 4.1.

Definition 4.7 (Family of fixed-rank simple neural network functions)**.** Let $n, m \in \mathbb{N}^+$ and $r \in \mathbb{N}$. Define the *family of fixed-rank simple neural network functions* with rank r, denoted $\mathcal{R}_r^{n,m}$, as

$$\mathcal{R}_r^{n,m} = \{\, f \in \mathcal{F}_\infty^{n,m} \mid \operatorname{rank}(f) = r \,\}.$$

Proposition 4.8. *Let $n, m \in \mathbb{N}^+$. The sequence $\mathcal{R}_0^{n,m}, \mathcal{R}_1^{n,m}, \mathcal{R}_2^{n,m}, \ldots$ forms a stratified version of the containment hierarchy $\mathcal{F}_0^{n,m} \subsetneq \mathcal{F}_1^{n,m} \subsetneq \mathcal{F}_2^{n,m} \subsetneq \cdots$. That is,*

(i) $\mathcal{R}_0^{n,m} = \mathcal{F}_0^{n,m}$ *and, for* $r \geq 1$, $\mathcal{R}_r^{n,m} = \mathcal{F}_r^{n,m} \setminus \mathcal{F}_{r-1}^{n,m}$*;*

(ii) $\displaystyle\bigcup_{r=0}^{h} \mathcal{R}_r^{n,m} = \mathcal{F}_h^{n,m}$ *for* $h \geq 0$*; and*

(iii) $\mathcal{R}_r^{n,m} \cap \mathcal{R}_{r'}^{n,m} = \emptyset$ *for* $r \neq r'$.

Proof. (i): $f \in \mathcal{R}_r^{n,m}$ if and only if $r = \min\{\, r' \mid f \in \mathcal{F}_{r'}^{n,m} \,\}$ if and only if $f \in \mathcal{F}_r^{n,m}$ and $f \notin \mathcal{F}_{r'}^{n,m}$ for $r' < r$ if and only if $f \in \mathcal{F}_r^{n,m} \setminus \mathcal{F}_{r-1}^{n,m}$ (including for $r = 0$, cf. Remark 3.14).

(ii): By (i) and induction on h, with base case $\mathcal{R}_0^{n,m} = \mathcal{F}_0^{n,m}$, and inductive case

$$\bigcup_{r=0}^{h} \mathcal{R}_r^{n,m} = \left(\bigcup_{r=0}^{h-1} \mathcal{R}_r^{n,m} \right) \cup \mathcal{R}_h^{n,m} = \mathcal{F}_{h-1}^{n,m} \cup \left(\mathcal{F}_h^{n,m} \setminus \mathcal{F}_{h-1}^{n,m} \right) = \mathcal{F}_h^{n,m}.$$

(iii): Assume $r' < r$ (without losing generality). Then by (i) and Proposition 3.19, $\mathcal{R}_r^{n,m} = \mathcal{F}_r^{n,m} \setminus \mathcal{F}_{r-1}^{n,m}$ and $\mathcal{R}_{r'}^{n,m} \subset \mathcal{F}_{r'}^{n,m} \subset \mathcal{F}_{r-1}^{n,m}$. The result follows. □

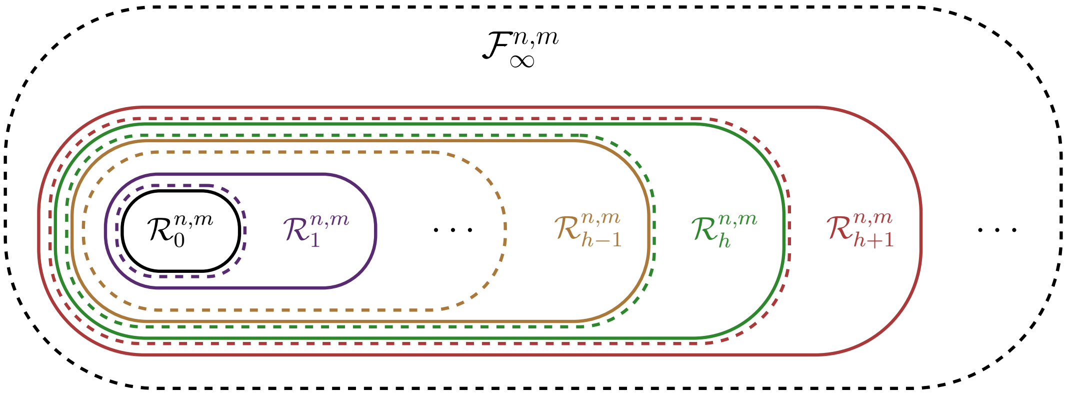


Figure 4.1: Conceptual illustration of the stratified hierarchy of rank-r simple neural network function families $\mathcal{R}_0^{n,m}, \mathcal{R}_1^{n,m}, \ldots, \mathcal{R}_{h-1}^{n,m}, \mathcal{R}_h^{n,m}, \mathcal{R}_{h+1}^{n,m}, \ldots \subsetneq \mathcal{F}_\infty^{n,m}$. Compared with Proposition 3.19 (see Figure 3.3), the elements of the stratified hierarchy are disjoint.

Interpretation 3: Rank as neural network compression

The *compressibility* of a neural network—how much its description length can be reduced with various coding and compression techniques—has been proposed as a learning objective (see, e.g., Hinton and van Camp, 1993; Aytekin et al., 2019) and used as a basis for compression-based generalisation bounds (see, e.g., Suzuki et al., 2020a;b). Compression also has applications in optimising neural networks for low-resource computing environments (see, e.g., Cheng et al., 2018; 2020; Choudhary et al., 2020).

Since one means of compressing a neural network is by removing hidden units, and the rank measures the minimum number of units required to implement a given neural network function, the rank represents an idealised measure of lossless compressibility.[1]

Interpretation 4: Rank as compared to that of linear algebra

In naming the measure of degeneracy described in this chapter the "rank" I evoke an analogy to the familiar notion from linear algebra of the *rank* of a linear transformation or matrix (see, e.g., Halmos, 1958; Axler, 2015). The rank of a linear transformation is the dimension of its image (e.g., Halmos, 1958, §50). Equivalently, the rank of a matrix is the dimension of its row (or column) space (e.g., Axler, 2015).

The analogy proceeds as follows. Consider an $m \times n$ matrix F. Then the rank of F is exactly the minimum number of hidden units h required to implement the corresponding linear transformation $f : \mathbb{R}^n \to \mathbb{R}^m$ as a single-hidden-layer unbiased linear neural network function, that is, as a function of the form

$$f(x) = \sum_{i=1}^{h} a_i \operatorname{id}(b_i x) = ABx$$

where A is an $m \times h$ matrix with columns $a_1, \ldots, a_h \in \mathbb{R}^m$ and B is an $h \times n$ matrix with rows $b_1, \ldots, b_h \in \mathbb{R}^n$. Such a minimal AB corresponds to a full-rank factorisation of F (see, e.g., Piziak and Odell, 1999).

Of course, the analogy is not complete. For example, one important difference between these notions of rank is that the rank of a linear transformation is bounded by its input and output dimension, while the rank of a simple neural network, even one implementing functions from $\mathbb{R}$ to $\mathbb{R}$, is unbounded.

[1]This notion of compressibility is idealised for at least three reasons. First, units are not the only measure of a network's description length. For example, the sparsity and precision of the weights may play a role. Second, in practice, one need not preserve the function exactly—it suffices to approximately preserve the function (*lossy* vs. lossless compressibility), and moreover the degree of approximation may be allowed to deteriorate for unlikely inputs. In Appendix A, I develop a variant of rank that corresponds more closely to approximate equality of functions on relevant inputs. Finally, the question ignores the task of *finding* a compressed parameter. In Section 4.2, I give efficient algorithms for computing a minimal implementation of a neural network function. Minimising a lossy compressed implementation should be substantially more difficult in general (cf. Chapter 6 and Appendix A).

4.2 Reducing parameters and computing rank

In this section, I derive an efficient algorithm for computing the rank of a simple neural network parameter. The idea behind the algorithm is to eliminate the local redundancy between units in the parameter (cf. Section 2.4) to find an irreducible, functionally equivalent parameter in an architecture with (possibly) fewer hidden units. Call this a *reduced* parameter. The rank is the number of hidden units in the reduced parameter (cf. Proposition 4.6).

In total, I present three algorithms in this section.

1. I present Algorithm 4.9, SlowReduce, a simple algorithm for computing a reduced parameter. The algorithm works by repeated application of the constructions in the proof of Theorem 3.25, removing hidden units one by one until an irreducible parameter is obtained. The algorithm runs time cubic in the initial number of hidden units.

2. I present Algorithm 4.12, FastReduce, an improved algorithm for computing a reduced parameter. The algorithm works by detecting eliminable units in a small number of passes, running in linear time after pre-sorting the hidden units.

3. Finally, I present Algorithm 4.15, Rank, an algorithm for computing the rank, by following Algorithm 4.12 but only performing those computations necessary for obtaining the final count of hidden units (not the reduced parameter itself).

Before introducing the algorithms, one customary qualification regarding complexity analysis. All times quoted above (and given in more detail throughout this section) assume a standard random-access model of computation and, moreover, assume that all parameter vectors are represented as arrays of bounded-precision components, such as using constant-width floating point numbers.

Computing reduced parameters by iteration

The following parameter reduction algorithm, SlowReduce, proceeds by searching for units meeting the reducibility conditions and then applying the constructions from the proof of Theorem 3.25 to replace the parameter with a smaller one. This process is repeated until an irreducible parameter is reduced.

Algorithm 4.9 formalises the algorithm. Some details, such as the order in which conditions are selected, are left unspecified. The result is an intentionally simplistic algorithm that serves as a stepping stone to the faster algorithm described below, and also as a basis for proofs involving parameter reduction (see, e.g., Section 4.3). For brevity, the details of the conditions and constructions are not restated within the algorithm—see Theorem 3.25 and its proof in Section 3.3.

Algorithm 4.9 (Parameter reduction by iteration)**.** Given an initial simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, proceed:

```
1: procedure SlowReduce($w \in \mathcal{W}_h^{n,m}$)
2:     while $w$ is reducible do        ▷ check using conditions from Definition 3.22
3:         $w \leftarrow$ new parameter $w' \in \mathcal{W}_{h-1}^{n,m}$ constructed as in Theorem 3.25 proof
4:         $h \leftarrow h - 1$
5:     end while
6:     return $w$
7: end procedure
```

Correctness Theorem 4.10 (Algorithm 4.9)**.** *Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, and a simple neural network parameter $w \in \mathcal{W}_h^{n,m}$, compute $w' = \text{SlowReduce}(w) \in \mathcal{W}_{h'}^{n,m}$. Then:*

(i) the computation terminates;

(ii) $f_{w'} = f_w$; and

(iii) $w' \in \mathcal{W}_{h'}^{n,m}$ is irreducible.

Proof. For (i), note that each iteration reduces the number of hidden units by exactly one, which can happen at most h times. For (ii), each parameter update preserves the neural network function (see Theorem 3.25 proof), so the final parameter is functionally equivalent to the input parameter, as required. For (iii), note that this is precisely the condition that allows termination of the loop. □

Remark 4.11. Algorithm 4.9 is intentionally simple. A coarse runtime analysis puts its complexity at cubic in h, in particular, at $\mathcal{O}(nh^3 + mh^2)$. The main operation is checking the reducibility conditions, of which there are $\mathcal{O}(h)$ involving vectors in $\mathbb{R}^m$ and $\mathcal{O}(h^2)$ involving vectors in $\mathbb{R}^n$, and which is performed no more than h times.

Computing reduced parameters by partitioning

Algorithm 4.9 might be improved by applying the reducibility conditions in a more effective order. Note that the constructions in the proof of Theorem 3.25 involve either

1. *unit elimination:* removing a unit, possibly with an adjustment to the outgoing unit bias vector, if the unit meets reducibility conditions (i) or (ii); or

2. *unit merging:* coalescing a pair of units meeting reducibility conditions (iii) or (iv) into a single unit with a combined outgoing weight vector (possibly with an associated sign change).

A key insight towards a more efficient algorithm is that such opportunities for reduction can mostly be detected based on the original parameter, in the following way.

1. Unit elimination applies to all units with zero incoming or outgoing weight vector. These can be identified and removed in one pass (or two separate passes, see below).

2. Unit merging applies to all groups of units with the same lexicographic absolute incoming weight and bias vector (cf. Definition 3.9). Such units can also be identified (efficiently using lexicographic sorting) and merged *en masse* in a single pass.

Moreover, the constructions have limited interaction. Unit elimination cannot produce additional units susceptible to reduction. Unit merging may produce units with zero outgoing weight, but this possibility can be handled by performing the merging before the elimination. This strategy for producing a reduced parameter is formalised in Algorithm 4.12.

Algorithm 4.12 (Parameter reduction by partitioning)**.** Given an initial simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, proceed:

```
1:  procedure FastReduce($w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$)
2:      ▷ Pass 1: Identify and remove units with zero incoming weight ◁
3:      $I \leftarrow \{\, i \in \{1, \ldots, h\} \mid b_i \neq 0 \,\}$
4:      $\delta \leftarrow d + \sum_{i \notin I} \tanh(c_i) \cdot a_i$
5:      ▷ Pass 2: Identify and merge units with shared incoming weights and biases ◁
6:      $\Pi_1, \ldots, \Pi_k \leftarrow$ partition $I$ by the relation $i \sim j \Leftrightarrow \mathrm{abs}_{\mathrm{lex}}(b_i, c_i) = \mathrm{abs}_{\mathrm{lex}}(b_j, c_j)$
7:      for $i \leftarrow 1, \ldots, k$ do
8:          $\alpha_i \leftarrow \sum_{j \in \Pi_i} \mathrm{sign}_{\mathrm{lex}}(b_j)\, a_j$
9:          $\beta_i \leftarrow \mathrm{sign}_{\mathrm{lex}}(b_j)\, b_j$                ▷ arbitrary $j \in \Pi_i$
10:         $\gamma_i \leftarrow \mathrm{sign}_{\mathrm{lex}}(b_j)\, c_j$
11:     end for
12:     ▷ Pass 3: Identify and remove merged units with zero outgoing weight ◁
13:     $i_1, \ldots, i_r \leftarrow \{\, i \in \{1, \ldots, k\} \mid \alpha_i \neq 0 \,\}$
14:     return $(\alpha_{i_1}, \ldots, \alpha_{i_r}, \beta_{i_1}, \ldots, \beta_{i_r}, \gamma_{i_1}, \ldots, \gamma_{i_r}, \delta) \in \mathcal{W}_r^{n,m}$
15: end procedure
```

Correctness Theorem 4.13 (Algorithm 4.12)**.** *Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, and a simple neural network parameter $w \in \mathcal{W}_h^{n,m}$, compute $w' = $* FastReduce$(w) \in \mathcal{W}_r^{n,m}$*. Then*

(i) $f_{w'} = f_w$*; and*

(ii) w' *is irreducible.*

Proof. (i): The idea is simpler than notation permits the demonstration to be. Basically, I show that the summation defining f_w can be rearranged so as to have the form of $f_{w'}$. Formally, for each $x \in \mathbb{R}^n$,

$$
\begin{aligned}
f_w(x) &= d + \sum_{i=1}^{h} a_i \tanh(b_i x + c_i) \\
&= d + \sum_{i \notin I} a_i \tanh(b_i x + c_i) + \sum_{i \in I} a_i \tanh(b_i x + c_i) && \text{(cf. line 3)} \\
&= d + \sum_{i \notin I} a_i \tanh(c_i) + \sum_{i \in I} a_i \tanh(b_i x + c_i) && (i \notin I \Rightarrow b_i = 0) \\
&= \delta + \sum_{i \in I} a_i \tanh(b_i x + c_i) && \text{(cf. line 4)} \\
&= \delta + \sum_{i=1}^{k} \sum_{j \in \Pi_i} a_j \tanh(b_j x + c_j) && \text{(cf. line 6)} \\
&= \delta + \sum_{i=1}^{k} \sum_{j \in \Pi_i} \operatorname{sign}_{\mathrm{lex}}(b_j)\, a_j \tanh(\operatorname{sign}_{\mathrm{lex}}(b_j)\, b_j x + \operatorname{sign}_{\mathrm{lex}}(b_j)\, c_j) && \text{(tanh odd)} \\
&= \delta + \sum_{i=1}^{k} \left(\sum_{j \in \Pi_i} \operatorname{sign}_{\mathrm{lex}}(b_j)\, a_j \right) \tanh(\beta_i x + \gamma_i) && \text{(cf. line 9; †)} \\
&= \delta + \sum_{i=1}^{k} \alpha_i \tanh(\beta_i x + \gamma_i) && \text{(cf. line 8)} \\
&= \delta + \sum_{j=1}^{r} \alpha_{i_j} \tanh(\beta_{i_j} x + \gamma_{i_j}) && \text{(cf. line 13)} \\
&= f_{w'}(x). && \text{(cf. line 14)}
\end{aligned}
$$

Step (†) requires clarification: in this step the incoming weights and biases of the units within each group of the partition are replaced with β_i and γ_i. This preserves those units: for $j \in \Pi_i$, $b_j \neq 0$ (by an earlier step), and it follows that $\operatorname{sign}_{\mathrm{lex}}(b_j)\,(b_j, c_j) = \operatorname{sign}_{\mathrm{lex}}(b_j, c_j)\,(b_j, c_j) = \operatorname{abs}_{\mathrm{lex}}(b_j, c_j)$, a constant within Π_i that also equals (β_i, γ_i).

(ii): Each of the reducibility conditions (Definition 3.22) fails to hold for $w' \in \mathcal{W}_r^{n,m}$: no α_{i_j} or β_{i_j} is zero, chiefly because of lines 13 and 3, and no $(\beta_{i_j}, \gamma_{i_j}) = \pm(\beta_{i_{j'}}, \gamma_{i_{j'}})$, chiefly because of line 6. $\square$

Remark 4.14. The first and third passes run in $\mathcal{O}(nh + mh)$ time. The second pass runs in $\mathcal{O}(nh \log(h) + mh)$ time if the partitioning step is performed by first sorting the lexicographic absolute values of the incoming weight and bias vectors and then grouping vectors with the same lexicographic absolute value. The total runtime is therefore $\mathcal{O}(nh \log(h) + mh)$.

Computing rank

Computing the rank itself is trivial given a reduction algorithm: simply run the reduction algorithm and count the units that remain. The following is a slightly streamlined algorithm, following Algorithm 4.12 but including only those steps that influence the final count.

Algorithm 4.15 (Rank)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, proceed:

```
1: procedure RANK($w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$)
2:     ▷ Identify units with non-zero incoming weight ◁
3:     $I \leftarrow \{\, i \in \{1, \ldots, h\} \mid b_i \neq 0 \,\}$
4:     ▷ Compute outgoing weights for merged units ◁
5:     $\Pi_1, \ldots, \Pi_k \leftarrow$ partition $I$ by the relation $i \sim j \Leftrightarrow \mathrm{abs}_{\mathrm{lex}}(b_i, c_i) = \mathrm{abs}_{\mathrm{lex}}(b_j, c_j)$
6:     for $j \leftarrow 1, \ldots, k$ do
7:         $\alpha_j \leftarrow \sum_{i \in \Pi_j} \mathrm{sign}_{\mathrm{lex}}(b_i)\, a_i$
8:     end for
9:     ▷ Count merged units with non-zero outgoing weights ◁
10:    return $|\{\, j \in \{1, \ldots, k\} \mid \alpha_j \neq 0 \,\}|$        ▷ $|S|$ denotes set cardinality
11: end procedure
```

Correctness Theorem 4.16 (Algorithm 4.15)**.** *Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, and a simple neural network parameter $w \in \mathcal{W}_h^{n,m}$,*

$$\mathrm{rank}(w) = \textsc{Rank}(w).$$

Proof. Comparing Algorithm 4.15 with Algorithm 4.12 reveals that $\textsc{Rank}(w) = r$, the number of units in the reduced parameter $\textsc{FastReduce}(w) \in \mathcal{W}_r^{n,m}$, an irreducible parameter functionally equivalent to w (by Correctness Theorem 4.13). The result follows by Proposition 4.6 and Remark 4.4. □

Remark 4.17. Though some steps are removed compared to Algorithm 4.12, the asymptotic time complexity of Algorithm 4.15 remains $\mathcal{O}(nh\log(h)+mh)$ (following the analysis in Remark 4.14).

4.3 Bounded rank regions of parameter space

The concept of rank allows one to define and study regions within a given parameter space comprising parameters of a given maximum rank. For maximum rank less than the number of available hidden units, these regions are subsets of the reducible region studied in Section 3.3, with which they share several important basic properties. In this section, I define such regions, characterise them as a union of linear subspaces of the parameter space, and document some of their properties.

Defining bounded rank regions

The following definition formalises the regions studied in this section.

Definition 4.18 (Bounded rank region)**.** Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a maximum rank $r \in \mathbb{N}$, the *bounded rank region* of rank r, denoted $\mathfrak{B}_r[\mathcal{W}_h^{n,m}]$, is the subset of parameters of rank at most r:

$$\mathfrak{B}_r[\mathcal{W}_h^{n,m}] = \{\, w \in \mathcal{W}_h^{n,m} \,|\, \operatorname{rank}(w) \leq r \,\}.$$

The following proposition notes some basic properties of the bounded rank regions for varying maximum rank r, and their relation to the reducible region of Section 3.3.

Proposition 4.19. *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Then (with $\subsetneq$ denoting strict inclusion):*

(i) If $r \geq h$, then $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] = \mathcal{W}_h^{n,m}$.

(ii) $\mathfrak{B}_{h-1}[\mathcal{W}_h^{n,m}] = \mathfrak{R}[\mathcal{W}_h^{n,m}]$.

(iii) If $r < r' \leq h$, then $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] \subsetneq \mathfrak{B}_{r'}[\mathcal{W}_h^{n,m}]$.

(iv) If $r < h - 1$, then $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] \subsetneq \mathfrak{R}[\mathcal{W}_h^{n,m}]$.

Proof. (i): If $w \in \mathcal{W}_h^{n,m}$ then, by Proposition 4.6, $\operatorname{rank}(w) \leq h \leq r$, so $w \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$. Thus $\mathcal{W}_h^{n,m} \subset \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$. Since $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] \subset \mathcal{W}_h^{n,m}$ it follows that $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] = \mathcal{W}_h^{n,m}$.

(ii): $w \in \mathfrak{R}[\mathcal{W}_h^{n,m}]$ if and only if $\operatorname{rank}(w) < h$ (by Proposition 4.6) if and only if $\operatorname{rank}(w) \leq h - 1$ (rank is an integer) if and only if $\mathfrak{B}_{h-1}[\mathcal{W}_h^{n,m}]$.

(iii): If $w \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ then $\operatorname{rank}(w) \leq r < r'$ so $w \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$. An irreducible parameter in $\mathcal{W}_{r'}^{n,m}$ embedded into $\mathcal{W}_h^{n,m}$ (cf. Proposition 3.19) has rank r', showing that the inclusion is strict.

(iv): Follows from (ii) and (iii).

□

Characterising bounded rank regions

Bounded rank regions generally have more complex shapes compared to reducible regions. The key to a clearer understanding of bounded rank regions is that for each parameter in $\mathfrak{B}_r[\mathcal{W}_h^{n,m}]$, at least $h - r$ units would be removed by the simple reduction algorithm (Algorithm 4.9). Considering the various possible sequences of reduction operations used in the first $h - r$ iterations of the simple reduction algorithm leads to a characterisation of the bounded rank region as a union of linear subspaces.

To this end, Definition 4.20 captures the notion of a "sequence of $h - r$ reduction operations" as a *reduction trace* of length $h - r$. The various reduction traces of length $h - r$ lead to the various components of my characterisation of the bounded rank region in Theorem 4.21.

Definition 4.20 (Reduction trace). Given $h \in \mathbb{N}$, let $H = \{1, \dots, h\}$. A *reduction trace* on h units is a 5-tuple $(R_{\mathrm{i}}, R_{\mathrm{ii}}, R_{\mathrm{iii}}, R_{\mathrm{iv}}, \mu)$ where

1. $R_{\mathrm{i}}, R_{\mathrm{ii}}, R_{\mathrm{iii}}, R_{\mathrm{iv}}$ are non-overlapping, possibly empty subsets of H (interpreted as sets of units removed by reducibility conditions (i) through (iv) respectively); and
2. $\mu : R_{\mathrm{iii}} \cup R_{\mathrm{iv}} \to H \setminus (R_{\mathrm{iii}} \cup R_{\mathrm{iv}})$ (interpreted as mapping from units removed under reducibility conditions (iii) and (iv) to the units with which they are merged).

Moreover, define the *length* of the reduction trace as $|R_{\mathrm{i}}| + |R_{\mathrm{ii}}| + |R_{\mathrm{iii}}| + |R_{\mathrm{iv}}|$, and, for $k \in \mathbb{N}$, denote by $\Xi_{\mathrm{R}}(h, k)$ the set of all reduction traces of length k on h units.

Theorem 4.21 (Characterisation of bounded rank regions). *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$. If $r < h$, the bounded rank region $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] \subset \mathcal{W}_h^{n,m}$ is a union of linear subspaces*

$$\mathfrak{B}_r[\mathcal{W}_h^{n,m}] = \bigcup_{\xi \in \Xi_{\mathrm{R}}(h, h-r)} S_\xi$$

where $\Xi_{\mathrm{R}}(h, h - r)$ denotes all reduction traces of length $h - r$ on h units and

$$S_{R_{\mathrm{i}},R_{\mathrm{ii}},R_{\mathrm{iii}},R_{\mathrm{iv}},\mu} = \bigcap_{i \in R_{\mathrm{i}}} A_{i, R_{\mathrm{iii}} \cap \mu^{-1}[i], R_{\mathrm{iv}} \cap \mu^{-1}[i]} \cap \bigcap_{i \in R_{\mathrm{ii}}} B_i \cap \bigcap_{j \in R_{\mathrm{iii}}} C_{\mu(j),j} \cap \bigcap_{j \in R_{\mathrm{iv}}} D_{\mu(j),j};$$

$$A_{i,J,K} = \left\{ (a_1, \dots, a_h, b_1, \dots, b_h, c_1, \dots, c_h, d) \in \mathcal{W}_h^{n,m} \;\middle|\; a_i + \sum_{j \in J} a_j - \sum_{k \in K} a_k = 0 \right\},$$

$$B_i = \{ (a_1, \dots, a_h, b_1, \dots, b_h, c_1, \dots, c_h, d) \in \mathcal{W}_h^{n,m} \mid b_i = 0 \},$$

$$C_{i,j} = \{ (a_1, \dots, a_h, b_1, \dots, b_h, c_1, \dots, c_h, d) \in \mathcal{W}_h^{n,m} \mid b_i = b_j, c_i = c_j \}, \qquad \textit{and}$$

$$D_{i,j} = \{ (a_1, \dots, a_h, b_1, \dots, b_h, c_1, \dots, c_h, d) \in \mathcal{W}_h^{n,m} \mid b_i = -b_j, c_i = -c_j \}.$$

Proof. ($\bigcup_\xi S_\xi \subset \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$): Suppose $w \in \bigcup_\xi S_\xi$. Then there is some reduction trace $\xi = (R_\mathrm{i}, R_\mathrm{ii}, R_\mathrm{iii}, R_\mathrm{iv}, \mu) \in \Xi_\mathrm{R}(h, h-r)$ with

$$w \in S_\xi = \bigcap_{i \in R_\mathrm{i}} A_{i, R_\mathrm{iii} \cap \mu^{-1}[i], R_\mathrm{iv} \cap \mu^{-1}[i]} \cap \bigcap_{i \in R_\mathrm{ii}} B_i \cap \bigcap_{j \in R_\mathrm{iii}} C_{\mu(j),j} \cap \bigcap_{j \in R_\mathrm{iv}} D_{\mu(j),j}.$$

Construct a functionally equivalent parameter in $\mathcal{W}_r^{n,m}$ by removing each of the $h - r$ units in $R_\mathrm{i}, R_\mathrm{ii}, R_\mathrm{iii}, R_\mathrm{iv}$ as follows.

1. For each unit $j \in R_\mathrm{iii}$, since $w \in C_{\mu(j),j}$ the unit shares incoming weight and bias vector with unit $\mu(j)$. Merge the two units as per the construction for reducibility condition (iii) in Theorem 3.25.

2. Likewise, for each unit $j \in R_\mathrm{iv}$, since $w \in D_{\mu(j),j}$, the unit shares (negated) incoming weight and bias vector with unit $\mu(j)$. Merge the two units as per the construction for reducibility condition (iv) in Theorem 3.25.

3. For each unit $i \in R_\mathrm{ii}$, since $w \in B_i$ the unit has zero incoming weight vector. Eliminate this unit as per the construction for reducibility condition (ii) in Theorem 3.25.

4. Finally, for each unit $i \in R_\mathrm{i}$, since $w \in A_{i, R_\mathrm{iii} \cap \mu^{-1}[i], R_\mathrm{iv} \cap \mu^{-1}[i]}$, after the above merging steps have merged other units into unit i (if any), the total outgoing weight will be zero. Eliminate this unit as per the construction for reducibility condition (i) in Theorem 3.25.

This construction shows that $\mathrm{rank}(w) \leq r$, thus $w \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$.

($\mathfrak{B}_r[\mathcal{W}_h^{n,m}] \subset \bigcup_\xi S_\xi$): Suppose $w \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$. Since the rank of w is then at most r the execution of Algorithm 4.9 on w will last for at least $h - r$ iterations. Run the algorithm for $h - r$ iterations and construct a reduction trace as follows. Starting with an initially empty $R_\mathrm{i}, R_\mathrm{ii}, R_\mathrm{iii}, R_\mathrm{iv}$, each time a unit is removed by a reducibility condition, add the unit's index (in the original parameter) to the corresponding subset. Moreover, each time a unit j is merged into unit i by condition (iii) or (iv), update μ as follows:

1. If the unit was merged by condition (iii), remap $\mu(k)$ to i for any units $k \in \mu^{-1}[j]$ previously merged with unit j—these units also could have been merged with unit i by the same reducibility condition.

2. If the unit was merged by condition (iv), again remap $\mu(k)$ to i for $k \in \mu^{-1}[j]$, but this time also move the corresponding units from R_iii to R_iv and vice versa, since their relationship with i is negated compared to their relationship with j.

3. Either way, set $\mu(j) = i$.

By the applied reducibility conditions, $w \in S_{R_\mathrm{i}, R_\mathrm{ii}, R_\mathrm{iii}, R_\mathrm{iv}, \mu}$. Thus, $w \in \bigcup_{\xi \in \Xi_\mathrm{R}(h, h-r)} S_\xi$. □

A detailed example characterisation

For small architectures, it is possible to explicitly construct the bounded rank regions. Consider the simple neural network architecture $\mathcal{A}_2^{1,1}$ with parameter space $\mathcal{W}_2^{1,1} = \mathbb{R}^7$. The corresponding bounded rank region of rank 0, $\mathfrak{B}_0\left[\mathcal{W}_2^{1,1}\right]$, contains all two-unit neural network parameters implementing constant functions.[2] By Theorem 4.21, $\mathfrak{B}_0\left[\mathcal{W}_2^{1,1}\right]$ can be understood in terms of all possible reduction traces of length two for two units. There are twelve reduction traces in $\Xi_\mathrm{R}(2,2)$, namely those listed in Table 4.2. The corresponding subspaces of $\mathcal{W}_2^{1,1} = \mathbb{R}^7$ are computed in Table 4.3.

ξ	R_i	R_ii	R_iii	R_iv	μ	ξ	R_i	R_ii	R_iii	R_iv	μ
ξ_1	$\{1,2\}$	$\emptyset$	$\emptyset$	$\emptyset$	–	ξ_2	$\emptyset$	$\{1,2\}$	$\emptyset$	$\emptyset$	–
ξ_3	$\{1\}$	$\{2\}$	$\emptyset$	$\emptyset$	–	ξ_4	$\{2\}$	$\{1\}$	$\emptyset$	$\emptyset$	–
ξ_5	$\{1\}$	$\emptyset$	$\{2\}$	$\emptyset$	$2 \mapsto 1$	ξ_6	$\{2\}$	$\emptyset$	$\{1\}$	$\emptyset$	$1 \mapsto 2$
ξ_7	$\{1\}$	$\emptyset$	$\emptyset$	$\{2\}$	$2 \mapsto 1$	ξ_8	$\{2\}$	$\emptyset$	$\emptyset$	$\{1\}$	$1 \mapsto 2$
ξ_9	$\emptyset$	$\{1\}$	$\{2\}$	$\emptyset$	$2 \mapsto 1$	ξ_{10}	$\emptyset$	$\{2\}$	$\{1\}$	$\emptyset$	$1 \mapsto 2$
ξ_{11}	$\emptyset$	$\{1\}$	$\emptyset$	$\{2\}$	$2 \mapsto 1$	ξ_{12}	$\emptyset$	$\{2\}$	$\emptyset$	$\{1\}$	$1 \mapsto 2$

Table 4.2: The reduction traces of length two given two units: $\Xi_\mathrm{R}(2,2) = \{\xi_1, \ldots, \xi_{12}\}$.

Subspace	Intersection form	Constraints on $w = (a_1, a_2, b_1, b_2, c_1, c_2, d) \in \mathcal{W}_2^{1,1}$
S_{ξ_1}	$A_{1,\emptyset,\emptyset} \cap A_{2,\emptyset,\emptyset}$	$a_1 = 0, a_2 = 0$
S_{ξ_2}	$B_1 \cap B_2$	$b_1 = 0, b_2 = 0$
S_{ξ_3}	$A_{1,\emptyset,\emptyset} \cap B_2$	$a_1 = 0, b_2 = 0$
S_{ξ_4}	$A_{2,\emptyset,\emptyset} \cap B_1$	$a_2 = 0, b_1 = 0$
S_{ξ_5}	$A_{1,\{2\},\emptyset} \cap C_{1,2}$	$a_1 + a_2 = 0, b_1 = b_2, c_1 = c_2$
S_{ξ_6}	$A_{2,\{1\},\emptyset} \cap C_{2,1}$	$a_2 + a_1 = 0, b_2 = b_1, c_2 = c_1$
S_{ξ_7}	$A_{1,\emptyset,\{2\}} \cap D_{1,2}$	$a_1 - a_2 = 0, b_1 = -b_2, c_1 = -c_2$
S_{ξ_8}	$A_{2,\emptyset,\{1\}} \cap D_{2,1}$	$a_2 - a_1 = 0, b_2 = -b_1, c_2 = -c_1$
S_{ξ_9}	$B_1 \cap C_{1,2}$	$b_1 = 0, b_1 = b_2, c_1 = c_2$
$S_{\xi_{10}}$	$B_2 \cap C_{2,1}$	$b_2 = 0, b_2 = b_1, c_2 = c_1$
$S_{\xi_{11}}$	$B_1 \cap D_{1,2}$	$b_1 = 0, b_1 = -b_2, c_1 = -c_2$
$S_{\xi_{12}}$	$B_2 \cap D_{2,1}$	$b_2 = 0, b_2 = -b_1, c_2 = -c_1$

Table 4.3: Component subspaces of bounded rank region $\mathfrak{B}_0\left[\mathcal{W}_2^{1,1}\right] = \bigcup_{i=1}^{12} S_{\xi_i}$. Note that the subspaces are not necessarily distinct. In particular, $S_{\xi_5} = S_{\xi_6}$, $S_{\xi_7} = S_{\xi_8}$, $S_{\xi_9} = S_{\xi_{10}}$, and $S_{\xi_{11}} = S_{\xi_{12}}$. Moreover, $S_{\xi_9}, \ldots, S_{\xi_{12}} \subset S_{\xi_2}$.

[2] $\mathfrak{B}_0\left[\mathcal{W}_2^{1,1}\right]$ is the simplest non-trivial bounded rank region. In any architecture with a single hidden unit, the rank 0 region is the reducible region, as visualised in Figure 3.6 for several small architectures (and the rank 1 region is the entire space). Likewise, $\mathfrak{B}_1\left[\mathcal{W}_2^{1,1}\right] = \mathfrak{R}\left[\mathcal{W}_2^{1,1}\right]$, as visualised in Figure 3.7.

Properties of bounded rank regions

While it is difficult to compute and visualise bounded rank regions in higher-dimensional parameter spaces, the characterisations in Proposition 4.19 and Theorem 4.21 provide some scalable intuition. From these results it follows that the bounded rank regions share many properties with the reducible regions analysed in Section 3.3, as follows.

Corollary 4.22. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$. Let $r < h$ be a maximum rank.*

(i) In the standard topology on $\mathcal{W}_h^{n,m}$, the bounded rank region $\mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ is a closed set with empty interior, and the complement $\mathcal{W}_h^{n,m} \setminus \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ is a dense, open set.

(ii) The bounded rank region $\mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ is a real affine algebraic set.

(iii) Let $p = (n + m + 1)h + m$. The bounded rank region $\mathfrak{B}_0[\mathcal{W}_h^{n,m}]$ has p-dimensional Lebesgue measure zero.

Proof. Proposition 4.19 shows that $\mathfrak{B}_r[\mathcal{W}_h^{n,m}] \subset \mathfrak{R}[\mathcal{W}_h^{n,m}]$. From this, property (iii) follows by Corollary 3.43(iii), and the emptiness of the interior and denseness of the complement follow by Corollary 3.43(i). For property (i) it remains to show that the bounded rank region is closed (the complement is therefore open). Closedness follows from Theorem 4.21, showing that $\mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ is a finite union of linear subspaces of $\mathcal{W}_h^{n,m}$, which are themselves closed. In fact the same linear subspaces are real affine algebraic sets (cf. Corollary 3.43(ii) for a similar proof) and so their finite union is real affine algebraic, as required for (ii). $\square$

Refer to Remark 3.44 and other remarks in Section 3.3 for some implications of these properties. I use the closedness property in the proof of Proposition 6.3 in Chapter 6.

4.4 Bounded rank function families

The sets of functions implemented by neural networks of a given architecture are of interest in deep learning theory (e.g., Rannen Triki, 2020; Petersen et al., 2021). In the case of simple neural network architecture $\mathcal{A}_h^{n,m}$, the set of functions is $\mathcal{F}_h^{n,m}$, the family of simple neural networks on h units (Definition 3.18). Given the framework of this chapter, the function family $\mathcal{F}_h^{n,m}$ can also be understood as the subset of the extended family of simple neural network functions $\mathcal{F}_\infty^{n,m}$ containing functions of rank at most h (cf. Proposition 4.8(ii)).

In this section, I study the properties of these bounded rank function families as subsets of $\mathcal{F}_\infty^{n,m}$. I show that, though $\mathcal{F}_\infty^{n,m}$ has a vector space structure (Proposition 3.21), the bounded rank subsets $\mathcal{F}_h^{n,m}$ are not subspaces (they are not closed under pointwise vector addition). Moreover, I show that the bounded rank function families are highly non-convex as sets of functions.[3]

Vector operations and rank

The following result shows how the vector operations interact with rank.

Proposition 4.23. *Consider an extended family of simple neural network functions $\mathcal{F}_\infty^{n,m}$. Given two simple neural network functions $f, g \in \mathcal{F}_\infty^{n,m}$ and a non-zero scalar $\alpha \in \mathbb{R} \setminus \{0\}$:*

(i) $\operatorname{rank}(0f) = 0$*;*

(ii) $\operatorname{rank}(\alpha f) = \operatorname{rank}(f)$*; and*

(iii) $|\operatorname{rank}(f) - \operatorname{rank}(g)| \leq \operatorname{rank}(f+g) \leq \operatorname{rank}(f) + \operatorname{rank}(g)$.

Proof. (i): Of course, $\operatorname{rank}(0f) = \operatorname{rank}(0) = 0$.

(ii): Let $r = \operatorname{rank}(f)$. Since $f \in \mathcal{F}_r^{n,m}$, $\alpha f \in \mathcal{F}_r^{n,m}$ by the construction in Proposition 3.21(i). Thus $\operatorname{rank}(\alpha f) \leq \operatorname{rank}(f)$. The same argument applied to α^{-1} and αf shows that $\operatorname{rank}(\alpha f) \geq \operatorname{rank}(\alpha^{-1}(\alpha f)) = \operatorname{rank}(f)$.

(iii): Let $r = \operatorname{rank}(f)$ and $s = \operatorname{rank}(g)$. Then since $f \in \mathcal{F}_r^{n,m}$ and $g \in \mathcal{F}_s^{n,m}$, $f + g \in \mathcal{F}_{r+s}^{n,m}$ by the construction in Proposition 3.21(ii). The upper bound follows immediately. For the lower bound, apply the same argument to $f+g$ and $-g$, and use that $\operatorname{rank}(g) = \operatorname{rank}(-g)$ (by (ii), above), to show that

$$\operatorname{rank}(f) = \operatorname{rank}(f+g-g) \leq \operatorname{rank}(f+g) + \operatorname{rank}(-g) = \operatorname{rank}(f+g) + \operatorname{rank}(g).$$

That is, $\operatorname{rank}(f) - \operatorname{rank}(g) \leq \operatorname{rank}(f+g)$. The desired bound follows by symmetry. □

[3]The (non-)convexity of the bounded rank function family is not to be confused with the distinct but similarly-named property of the convexity of a loss landscape. The former (the topic of this section) concerns *convexity of a set* within the vector space of functions. The latter (not discussed here) concerns *convexity of a function* as a surface on the parameter space.

Remark 4.24. Without further assumptions on the relationship between f and g, the bounds in Proposition 4.23(iii) are optimal. More precisely, for any given f and any given rank $r \in \mathbb{N}$ there exists $g, g' \in \mathcal{R}_r^{n,m}$ such that $\operatorname{rank}(f+g) = \operatorname{rank}(f) + \operatorname{rank}(g)$ and $\operatorname{rank}(f+g') = |\operatorname{rank}(f) - \operatorname{rank}(g')|$. Such g, g' can be implemented to share (lexicographic absolute) incoming weight and bias vectors with as few or as many as possible of the units of an irreducible implementation of f.

Remark 4.25. Remark 4.24 indicates that the bounded rank function families are not closed under vector addition. This also follows as a corollary of the non-convexity results given later in this section.

By repeated use of Proposition 4.23 one can bound the rank of an arbitrary finite linear combination of simple neural network functions.

Corollary 4.26. *Given a list of non-zero scalars $\alpha_1, \alpha_2, \ldots, \alpha_s \in \mathbb{R} \setminus \{0\}$, and a corresponding list of simple neural network functions $f_1, f_2, \ldots, f_s \in \mathcal{F}_\infty^{n,m}$,*

$$\operatorname{rank}(f_1) - \sum_{i=2}^{s} \operatorname{rank}(f_i) \leq \operatorname{rank}\left(\sum_{i=1}^{s} \alpha_i f_i\right) \leq \sum_{i=1}^{s} \operatorname{rank}(f_i)\,.$$

Proof (by induction). Base case ($s = 1$): $\operatorname{rank}(f_1) = \operatorname{rank}(\alpha_1 f_1)$ by Proposition 4.23(ii).

Inductive case ($s > 1$): For the upper bound,

$$\begin{aligned}
\operatorname{rank}\left(\sum_{i=1}^{s} \alpha_i f_i\right) &\leq \operatorname{rank}(\alpha_1 f_1) + \operatorname{rank}\left(\sum_{i=2}^{s} \alpha_i f_i\right) && \text{(Proposition 4.23(iii))} \\
&\leq \operatorname{rank}(\alpha_1 f_1) + \sum_{i=2}^{s} \operatorname{rank}(f_i) && \text{(inductive hypothesis)} \\
&= \sum_{i=1}^{s} \operatorname{rank}(f_i)\,. && \text{(Proposition 4.23(ii))}
\end{aligned}$$

For the lower bound,

$$\begin{aligned}
\operatorname{rank}(f_1) - \sum_{i=2}^{s} \operatorname{rank}(f_i) &\leq \operatorname{rank}\left(\sum_{i=1}^{s-1} \alpha_i f_i\right) - \operatorname{rank}(f_s) && \text{(inductive hypothesis)} \\
&= \operatorname{rank}\left(\sum_{i=1}^{s-1} \alpha_i f_i\right) - \operatorname{rank}(\alpha_s f_s) && \text{(Proposition 4.23(ii))} \\
&\leq \operatorname{rank}\left(\sum_{i=1}^{s} \alpha_i f_i\right). && \text{(Proposition 4.23(iii))}
\end{aligned}$$

□

Remark 4.27. The lower bound is non-trivial only when $\operatorname{rank}(f_1) > \sum_{i=2}^{s} \operatorname{rank}(f_i)$, since the rank is a natural number. It might help to choose as f_1 the highest-rank function.

Finite function families are strongly locally non-convex

Bounded rank function families fail to possess vector subspace structure, because, while they are closed under pointwise scalar multiplication, they are not closed under pointwise vector addition: there exist $f, g \in \mathcal{F}_r^{n,m}$ such that $f + g \notin \mathcal{F}_r^{n,m}$.

Moreover, bounded rank function families are not closed under *convex combination:* there exist $f, g \in \mathcal{F}_r^{n,m}$ such that, for some $\tau \in (0,1)$, $(1-\tau)f + \tau g \notin \mathcal{F}_r^{n,m}$. This property is called *non-convexity* of $\mathcal{F}_r^{n,m}$.

These results are immediate corollaries of the property I prove below, in Theorem 4.28. I call this property *strong everywhere non-convexity.* This property strengthens non-convexity in two ways. First, there are pairs of functions for which the interpolating line is entirely outside of the function family (rather than, merely, not entirely inside). Second, parameters implementing such pairs of functions can be found inside every open neighbourhood in parameter space (this implies the same for neighbourhoods in function space, see Remark 4.29).

Theorem 4.28 (Strong everywhere non-convexity of bounded rank function families)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$ and function family $\mathcal{F}_h^{n,m}$. If $h > 0$, then given a parameter $w \in \mathcal{W}_h^{n,m}$ and a positive uniform radius $\varepsilon \in \mathbb{R}^+$, there exists a* nearby *pair of parameters $u, v \in \bar{B}_\infty(w; \varepsilon)$ in the closed uniform neighbourhood of w with radius ε, such that* for all $\tau \in (0,1)$*, the convex combination of the functions is outside the function family:*

$$(1-\tau)f_u + \tau f_v \notin \mathcal{F}_h^{n,m}.$$

Proof. Since the irreducible region of $\mathcal{W}_h^{n,m}$ is dense in $\mathcal{W}_h^{n,m}$ (by Corollary 3.43(i)), there exists an irreducible parameter u in the interior of $\bar{B}_\infty(w; \varepsilon)$ (such that $\|u - w\|_\infty < \varepsilon$). Writing

$$u = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d),$$

construct a second parameter

$$v = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1 + \gamma, \ldots, c_h + \gamma, d) \in \mathcal{W}_h^{n,m}$$

where

$$\gamma = \frac{1}{3} \min \left\{ \varepsilon - \|w - u\|_\infty, \min_{1 \le i < j \le h} \|(b_i, c_i) - (b_j, c_j)\|_\infty, \min_{1 \le i < j \le h} \|(b_i, c_i) + (b_j, c_j)\|_\infty \right\}.$$

Note that $\gamma > 0$, since u is in the interior of $\bar{B}_\infty(w; \varepsilon)$ and is irreducible. In particular, γ is chosen to imbue v with three important properties:

1. $v \in \bar{B}_\infty(w; \varepsilon)$. Since $\gamma < \varepsilon - \|w - u\|_\infty$,

$$\begin{aligned} \|w - v\|_\infty &\leq \|w - u\|_\infty + \|u - v\|_\infty && \text{(triangle inequality)} \\ &= \|w - u\|_\infty + \gamma \\ &< \|w - u\|_\infty + (\varepsilon - \|w - u\|_\infty) \\ &= \varepsilon. \end{aligned}$$

2. v is irreducible. By Definition 3.22. For (i,ii), $a_i, b_i \neq 0$ since u is irreducible. For (iii), the irreducibility of u also implies that for $i < j$, $(b_i, c_i) \neq (b_j, c_j)$ and therefore $(b_i, c_i + \gamma) \neq (b_j, c_j + \gamma)$. For (iv), suppose $(b_i, c_i + \gamma) = -(b_j, c_j + \gamma)$ for some $i < j$. But the definition of γ prevents units i, j being so close, yielding a contradiction:

$$\gamma \leq \frac{1}{3} \|(b_i, c_i) + (b_j, c_j)\|_\infty = \frac{1}{3} \|(0, -2\gamma)\|_\infty = \frac{2}{3}\gamma < \gamma.$$

3. For $i, j \in \{1, \ldots, h\}$, $(b_i, c_i) \neq \pm(b_j, c_j + \gamma)$ (that is, the hidden unit incoming weight and bias vectors of v have no lexicographic absolute overlap with those of u). Suppose $(b_i, c_i) = \pm(b_j, c_j + \gamma)$. Again, this yields a contradiction:

$$\gamma \leq \frac{1}{3} \|(b_i, c_i) \mp (b_j, c_j)\|_\infty = \frac{1}{3} \|(0, \pm\gamma)\|_\infty = \frac{1}{3}\gamma < \gamma.$$

Now, let $\tau \in (0, 1)$, and consider the convex combination function $f_\tau = (1-\tau) f_u + \tau f_v$. Following Proposition 4.23, f_τ is implemented by the parameter

$$\begin{aligned} w_\tau = \big((1-\tau)a_1, \ldots, (1-\tau)a_h, \tau a_1, \ldots, \tau a_h, b_1, \ldots, b_h, b_1, \ldots, b_h, \\ c_1, \ldots, c_h, c_1 + \gamma, \ldots, c_h + \gamma, (1-\tau)d + \tau d\big) \in \mathcal{W}_{2h}^{n,m}. \end{aligned}$$

This parameter w_τ is irreducible: for (i), $(1 - \tau)$, τ are non-zero by definition, and each a_i is non-zero since u is irreducible. For (ii), each b_i is non-zero since u is irreducible. For (iii,iv), each hidden unit incoming weight and bias vector has a unique lexicographic absolute value since u and v are irreducible and by property (3) verified above. It follows by Proposition 4.6 that $(1-\tau) f_u + \tau f_v \notin \mathcal{F}_h^{n,m}$. □

Remark 4.29. Theorem 4.28 shows that parameters implementing strongly non-convex pairs of functions can be found in every open neighbourhood in the parameter space. An alternative condition can be formulated using neighbourhoods in the space of functions. While it is beyond the scope of this chapter to make this function space condition precise, Appendix A.1 reviews the necessary preliminaries to define neighbourhoods in function space, and a consequence of Lemma A.17 in Appendix A.2 is that the parameter space condition in Theorem 4.28 is sufficient to imply the function space condition.

Remark 4.30. Theorem 4.28 is related to existing work on the properties of neural network function families by Petersen et al. (2021). Petersen et al. (2021) showed that for multi-layer neural network architectures with a broad class of activation functions, the corresponding function families are non-convex. Within the special case of simple neural network architectures, Theorem 4.28 is a stronger result. Moreover, I conjecture that this stronger property generalises to additional architectures.

Remark 4.31. The above proof shows that there are, densely distributed within the bounded rank function family, high-rank functions with non-overlapping hidden unit incoming weight and bias vectors. Implementing a convex combination of these functions requires representing all of their units, which easily exceeds the rank bound. However, for *relatively low-rank* functions within the function family, convex combinations may not exceed a given rank bound. This observation takes on special relevance in the setting of *overparameterised* learning, where the number of units available is very high, such that "relatively low-rank" functions still includes some functions with significant complexity.

Chapter 5

Degenerate Neural Network Geometry

In this chapter, I offer a thorough analysis of the geometry of simple neural networks (neural networks with a single hidden layer of biased units and the hyperbolic tangent activation function, cf. Section 3.2).

Crucially, rather than dismissing the degenerate (reducible) parameters, I leverage the rank-based perspective on degeneracy from Chapter 4 to extend my analysis to this case. In particular, I offer the following:

1. In Section 5.1, I derive a canonicalisation algorithm that works for both non-degenerate *and* degenerate simple neural network parameters. This algorithm efficiently determines if two simple neural network parameters (optionally from different architectures) are functionally equivalent.

2. In Section 5.2, I characterise the full functional equivalence class of an arbitrary degenerate simple neural network parameter as a union of simple subsets. Such classes are much richer than those of non-degenerate parameters, and I achieve this characterisation by tracing the execution of the canonicalisation algorithm of Section 5.1. I study basic properties of these functional equivalence classes.

3. In Section 5.3, I show that the functional equivalence class for degenerate neural network parameters is piecewise linear path connected. I discuss how the extent of path connectedness depends on the degree of degeneracy (the rank) of the corresponding neural network parameter.

5.1 Canonicalisation algorithm for all parameters

As reviewed in Section 2.3, a canonicalisation algorithm maps each parameter to a *canonical representative* parameter within its functional equivalence class (such that all functionally equivalent parameters map to the same representative parameter). A canonicalisation algorithm therefore serves as a computational test of membership within a given functional equivalence class.

In this section, I develop a canonicalisation algorithm that works for simple neural network parameters—both irreducible *and* reducible. This extends existing algorithms that only canonicalise irreducible parameters (see, e.g., Rüger and Ossen, 1997).

The canonicalisation algorithm developed in this section combines (1) the parameter reduction algorithm FastReduce (Algorithm 4.12, Section 4.2), which converts a parameter to a functionally equivalent and irreducible parameter in another architecture, and (2) a canonicalisation algorithm for irreducible parameters similar to the algorithm given by Rüger and Ossen (1997). I begin by introducing the latter.

Canonicalisation algorithm for irreducible parameters

Recall (cf. Theorem 3.39) that functionally equivalent irreducible parameters differ only by permutation and negation transformations. Given an irreducible parameter, a canonical representative can be found by (1) lexicographically positivising each hidden unit using a negation transformation, and then (2) lexicographically sorting the hidden units using a permutation transformation. Algorithm 5.1 formalises this canonicalisation method.

Algorithm 5.1 (Absolute parameter sorting)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$ and irreducible region $\mathfrak{I}[\mathcal{W}_h^{n,m}]$, proceed:

```
1: procedure AbsoluteSort($w = (a_1, \dots, a_h, b_1, \dots, b_h, c_1, \dots, c_h, d) \in \mathfrak{I}[\mathcal{W}_h^{n,m}]$)
2:     $\sigma \leftarrow (\mathrm{sign}_{\mathrm{lex}}(b_1, c_1), \dots, \mathrm{sign}_{\mathrm{lex}}(b_h, c_h)) \in \{-1, +1\}^h$
3:     $\pi \leftarrow$ permutation lexicographically sorting $\mathrm{abs}_{\mathrm{lex}}(b_1, c_1), \dots, \mathrm{abs}_{\mathrm{lex}}(b_h, c_h)$
4:     return $T_\pi(T_\sigma(w)) \in \mathcal{W}_h^{n,m}$
5: end procedure
```

The signs on line 2 are non-zero and the permutation on line 3 is unique because w is irreducible. The permutation can be computed using a sorting algorithm (cf. Remark 3.6).

Correctness Theorem 5.2 (Algorithm 5.1)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$ and irreducible region $\mathfrak{I}[\mathcal{W}_h^{n,m}]$. Given $w, w' \in \mathfrak{I}[\mathcal{W}_h^{n,m}]$, let $v =$ AbsoluteSort(w) and $v' =$ AbsoluteSort(w'). Then*

(i) $f_w = f_v$; and

(ii) if $f_w = f_{w'}$ then $v = v'$.

Proof. For (i), note that $v = T_\pi(T_\sigma(w))$, so $f_v = f_w$ by Theorem 3.39. For (ii), suppose $f_w = f_{w'}$. By Theorem 3.39, there exists $\sigma'' \in \{-1,+1\}^h$ and $\pi'' \in S_h$ such that $w' = T_{\sigma''}(T_{\pi''}(w))$. Of course, Algorithm 5.1 is designed to be invariant to just such transformations. To verify this, consider executing the algorithm on w and w', denoting with x' each variable x in the latter execution:

1. First, the computation of the sign vectors on line 2 proceeds:

$$\begin{aligned}
\sigma &= (\mathrm{sign}_{\mathrm{lex}}(b_1, c_1), \ldots, \mathrm{sign}_{\mathrm{lex}}(b_h, c_h)) \\
\sigma' &= (\mathrm{sign}_{\mathrm{lex}}(b'_1, c'_1), \ldots, \mathrm{sign}_{\mathrm{lex}}(b'_h, c'_h)) \\
&= \left(\mathrm{sign}_{\mathrm{lex}}\left(\sigma''_1 b_{\pi''(1)}, \sigma''_1 c_{\pi''(1)}\right), \ldots, \mathrm{sign}_{\mathrm{lex}}\left(\sigma''_h b_{\pi''(h)}, \sigma''_h c_{\pi''(h)}\right)\right) \\
&= \left(\sigma''_1 \sigma_{\pi''(1)}, \ldots, \sigma''_h \sigma_{\pi''(h)}\right).
\end{aligned}$$

   It follows that

$$\begin{aligned}
T_{\sigma'}(w') &= (\sigma'_1 a'_1, \ldots, \sigma'_h a'_h, \sigma'_1 b'_1, \ldots, \sigma'_h b'_h, \sigma'_1 c'_1, \ldots, \sigma'_h c'_h, d') \\
&= (\sigma''_1 \sigma_{\pi''(1)} \sigma''_1 a_{\pi''(1)}, \ldots, \sigma''_h \sigma_{\pi''(h)} \sigma''_h a_{\pi''(h)}, && (\sigma'_i = \sigma''_i \sigma_{\pi''(i)}, \\
&\quad\ \sigma''_1 \sigma_{\pi''(1)} \sigma''_1 b_{\pi''(1)}, \ldots, \sigma''_h \sigma_{\pi''(h)} \sigma''_h b_{\pi''(h)}, && a'_i = \sigma''_i a_{\pi''(i)}, \\
&\quad\ \sigma''_1 \sigma_{\pi''(1)} \sigma''_1 c_{\pi''(1)}, \ldots, \sigma''_h \sigma_{\pi''(h)} \sigma''_h c_{\pi''(h)}, d) && \text{likewise for } b'_i,\ c'_i, \\
&= (\sigma_{\pi''(1)} a_{\pi''(1)}, \ldots, \sigma_{\pi''(h)} a_{\pi''(h)}, && \text{and also, } d' = d) \\
&\quad\ \sigma_{\pi''(1)} b_{\pi''(1)}, \ldots, \sigma_{\pi''(h)} b_{\pi''(h)}, \\
&\quad\ \sigma_{\pi''(1)} c_{\pi''(1)}, \ldots, \sigma_{\pi''(h)} c_{\pi''(h)}, d) && (\sigma''_i \sigma''_i = 1) \\
&= T_{\pi''}(T_\sigma(w)). && (\dagger)
\end{aligned}$$

2. Next, denote the lists sorted on line 3 as L (and L'). Then

$$\begin{aligned}
L &= \mathrm{abs}_{\mathrm{lex}}(b_1, c_1), \ldots, \mathrm{abs}_{\mathrm{lex}}(b_h, c_h), \text{ and} \\
L' &= \mathrm{abs}_{\mathrm{lex}}(b'_1, c'_1), \ldots, \mathrm{abs}_{\mathrm{lex}}(b'_h, c'_h) \\
&= \mathrm{abs}_{\mathrm{lex}}\left(\sigma''_1 b_{\pi''(1)}, \sigma''_1 c_{\pi''(1)}\right), \ldots, \mathrm{abs}_{\mathrm{lex}}\left(\sigma''_h b_{\pi''(h)}, \sigma''_h c_{\pi''(h)}\right) \\
&= \mathrm{abs}_{\mathrm{lex}}\left(b_{\pi''(1)}, c_{\pi''(1)}\right), \ldots, \mathrm{abs}_{\mathrm{lex}}\left(b_{\pi''(h)}, c_{\pi''(h)}\right)
\end{aligned}$$

   It follows that the permutations π sorting L and π' sorting L' are such that

$$\pi' = \pi \cdot (\pi'')^{-1} \qquad (\ddagger)$$

3. By ($\dagger$) and ($\ddagger$), $v' = T_{\pi'}(T_{\sigma'}(w')) = T_\pi(T_{\pi''}^{-1}(T_{\pi''}(T_\sigma(w)))) = T_\pi(T_\sigma(w)) = v$. □

Remark 5.3. Algorithm 5.1 is similar to the algorithm sketched by Rüger and Ossen (1997) for canonicalising neural network parameters with non-zero biases.

Remark 5.4. Computing the sign vector takes $\mathcal{O}(nh)$ time, computing the permutation takes $\mathcal{O}(nh \log(h))$ time, and executing the transformations takes $\mathcal{O}(nh + mh)$ time; so the total asymptotic runtime of Algorithm 5.1 is $\mathcal{O}(nh \log(h) + mh)$ (all assuming a standard model of computation and bounded-precision parameters, cf. Section 4.2).

Remark 5.5. Algorithm 5.1 can be applied to reducible parameters. However, Correctness Theorem 5.2(ii) does not generalise in this case: if two functionally equivalent reducible parameters are not related by permutation and negation transformations then the algorithm will not typically produce the same representative for these parameters.

Canonicalisation across architectures

Algorithm 5.1 only works for irreducible parameters, not reducible parameters. But Algorithm 4.12 can convert reducible parameters into irreducible parameters (in an architecture with fewer hidden units). I combine these algorithms into a canonicalisation algorithm as follows.

Algorithm 5.6 (Cross-architecture parameter canonicalisation)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, proceed:

```
1: procedure XCanonicalise($w \in \mathcal{W}_h^{n,m}$)          ▷ read "cross canonicalise"
2:     $u \leftarrow$ FastReduce($w$) $\in \mathcal{W}_r^{n,m}$          ▷ Algorithm 4.12
3:     $v \leftarrow$ AbsoluteSort($u$) $\in \mathcal{W}_r^{n,m}$        ▷ Algorithm 5.1
4:     return $v \in \mathcal{W}_r^{n,m}$
5: end procedure
```

The output dimension of this algorithm depends on the input: $r = \operatorname{rank}(w) \leq h$.

Correctness Theorem 5.7 (Algorithm 5.6)**.** *Consider two simple neural network architectures $\mathcal{A}_h^{n,m}, \mathcal{A}_{h'}^{n,m}$ with parameter spaces $\mathcal{W}_h^{n,m}, \mathcal{W}_{h'}^{n,m}$. Given $w \in \mathcal{W}_h^{n,m}$ and $w' \in \mathcal{W}_{h'}^{n,m}$, let $v = \text{XCanonicalise}(w)$ and $v' = \text{XCanonicalise}(w')$. Then*

(i) $f_w = f_v$; and

(ii) if $f_w = f_{w'}$ then $v = v'$.

Proof. For (i), $f_w = f_u$ by Correctness Theorem 4.13, and $f_u = f_v$ by Correctness Theorem 5.2(i). For (ii), suppose $f_w = f_{w'}$. While u and u' (line 2 given input w and w' respectively) are not necessarily equal, they are functionally equivalent irreducible parameters in $\mathcal{W}_{\operatorname{rank}(w)}^{n,m} = \mathcal{W}_{\operatorname{rank}(w')}^{n,m}$ (this follows by Correctness Theorem 4.13 and Remark 4.4). Therefore $v = v'$ by Correctness Theorem 5.2(ii). □

Remark 5.8. The runtime of Algorithm 5.6 is $\mathcal{O}(nh \log(h) + mh)$ by Remarks 5.4 and 4.14.

Canonicalisation within one architecture

Algorithm 5.9 is a powerful canonicalisation algorithm that finds canonical representative parameters for all functionally equivalent parameters across all simple neural network architectures. The representative parameter will not be from the same parameter space as the input parameter unless the input is already irreducible. To complete this section I offer the following simple modification to the canonicalisation algorithm that finds representatives within a given parameter space.

Algorithm 5.9 (Within-architecture parameter canonicalisation)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, proceed:

```
1: procedure CANONICALISE($w \in \mathcal{W}_h^{n,m}$)
2:     ▷ Find representative in $\mathcal{W}_r^{n,m}$ using Algorithm 5.6 ◁
3:     $(a_1, \ldots, a_r, b_1, \ldots, b_r, c_1, \ldots, c_r, d) \leftarrow$ XCANONICALISE($w$) $\in \mathcal{W}_r^{n,m}$
4:     ▷ Embed into $\mathcal{W}_h^{n,m}$ by adding $h - r$ blank units ◁
5:     $0_a, 0_b, 0_c \leftarrow 0 \in \mathbb{R}^m, 0 \in \mathbb{R}^n, 0 \in \mathbb{R}$
6:     return $(a_1, \ldots, a_r, 0_a, \ldots, 0_a, b_1, \ldots, b_r, 0_b, \ldots, 0_b, c_1, \ldots, c_r, 0_c, \ldots, 0_c, d) \in \mathcal{W}_h^{n,m}$
7: end procedure
```

Correctness Theorem 5.10 (Algorithm 5.9)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter spaces $\mathcal{W}_h^{n,m}$. Given $w, w' \in \mathcal{W}_h^{n,m}$, let $v =$ CANONICALISE(w) and $v' =$ CANONICALISE(w'). Then*

(i) $f_w = f_v$; and

(ii) if $f_w = f_{w'}$ then $v = v'$.

Proof. For (i), the canonicalisation step (line 3) preserves functional equivalence by Correctness Theorem 5.7(i), and the embedding step (lines 5 and 6) clearly preserves functional equivalence. For (ii), it follows from Correctness Theorem 5.7(ii) that after line 3 the execution on w and w' proceeds identically. □

Remark 5.11. Computing the embedded parameter takes only $\mathcal{O}(nh+mh)$ time, so the total runtime complexity of Algorithm 5.9 remains $\mathcal{O}(nh\log(h) + mh)$ (cf. Remark 5.8).

5.2 Functional equivalence classes for all parameters

Given an irreducible simple neural network parameter, the functional equivalence class has a simple characterisation in terms of the permutation and negation transformations, as reviewed in Section 3.3. Based on this characterisation, the canonicalisation algorithm for irreducible parameters (Algorithm 5.1) was derived.

To characterise the much richer functional equivalence class of a reducible parameter, I take an essentially reversed approach. Having established a canonicalisation algorithm for all parameters in the previous section (Algorithm 5.9), I work in this section to invert this algorithm as a means of characterising the functional equivalence class (noting that the canonicalisation algorithm computes the same output for all functionally equivalent parameters, by design). After deriving a characterisation of general functional equivalence classes, I document some of their basic properties.

Characterising the functional equivalence class

While the functional equivalence class for a reducible neural network parameter is, in general, a highly complicated subset of the parameter space, it is possible to characterise the subset as a union of simple sets.

Much like for the characterisation of the bounded rank regions in Section 4.3, the key to a simple characterisation is an algorithm. What an entire functional equivalence class has in common is the corresponding output from a canonicalisation algorithm. In particular, there are only so many ways for a parameter's hidden units to be merged, eliminated, negated, and permuted in the course of Algorithm 5.6. Enumerating all such possible *canonicalisation traces* leads to an effective characterisation of the functional equivalence class, as follows.

Definition 5.12 (Canonicalisation trace)**.** Given $r, h \in \mathbb{N}$ with $r \leq h$, a *canonicalisation trace* of order r on h units is a tuple (σ, τ) where

- $\sigma \in \{-1, +1\}^h$ is a sign vector (cf. Definition 3.34, interpreted as tracking unit negations throughout canonicalisation); and
- $\tau : \{1, \dots, h\} \to \{0, 1, \dots, h\}$ is a function with range including $\{1, \dots, r\}$ (interpreted as tracking the merging, elimination, and permutation of units throughout canonicalisation—$\tau(i) = 0$ for units eliminated for having zero incoming weight incorporated into output unit biases, $\tau(i) > r$ for units eliminated as part of a group with total outgoing weight zero, or otherwise $\tau(i) = j$ for units partially implementing unit j of the canonicalised network).

Moreover, denote by $\Xi_{\mathrm{C}}(h, r)$ the set of all canonicalisation traces of order r on h units.

Theorem 5.13 (Characterisation of functional equivalence class)**.** *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$. Given a parameter $w \in \mathcal{W}_h^{n,m}$, let $r = \operatorname{rank}(w)$, and let $(\alpha_1, \ldots, \alpha_r, \beta_1, \ldots, \beta_r, \gamma_1, \ldots, \gamma_r, \delta) = \text{XCanonicalise}(w) \in \mathcal{W}_r^{n,m}$ (Algorithm 5.6). Then the functional equivalence class $\mathfrak{F}[w] \subset \mathcal{W}_h^{n,m}$ is a union of subsets*

$$\mathfrak{F}[w] = \bigcup_{\xi \in \Xi_{\mathrm{C}}(h,r)} S_\xi$$

where $\Xi_{\mathrm{C}}(h,r)$ denotes all canonicalisation traces of order r on h units and

$$S_{\sigma,\tau} = X^{\delta}_{\tau^{-1}[0]} \cap \left(\bigcap_{i=1}^{r} Y^{\alpha_i,\beta_i,\gamma_i}_{\sigma,\tau^{-1}[i]} \right) \cap \left(\bigcap_{i=r+1}^{h} Z_{\sigma,\tau^{-1}[i]} \right);$$

$$X_I^\delta = \left\{ (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \;\middle|\; \begin{array}{c} \forall i \in I, b_i = 0 \text{ and} \\ d + \sum_{i \in I} a_i \tanh(c_i) = \delta \end{array} \right\};$$

$$Y_{\sigma,I}^{\alpha,\beta,\gamma} = \left\{ (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \;\middle|\; \begin{array}{c} \forall i \in I, \sigma_i \cdot (b_i, c_i) = (\beta, \gamma) \\ \text{and } \sum_{i \in I} \sigma_i a_i = \alpha \end{array} \right\}; \text{ and}$$

$$Z_{\sigma,I} = \left\{ (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m} \;\middle|\; \begin{array}{c} \forall i,j \in I, \sigma_i \cdot (b_i, c_i) = \sigma_j \cdot (b_j, c_j) \\ \text{and } \sum_{i \in I} \sigma_i a_i = 0 \end{array} \right\}.$$

Proof. ($\bigcup_\xi S_\xi \subset \mathfrak{F}[w]$): Let $u = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \bigcup_\xi S_\xi$. Then there is some canonicalisation trace $(\sigma, \tau) \in \Xi_{\mathrm{C}}(h,r)$ such that

$$u \in S_{\sigma,\tau} = X^{\delta}_{\tau^{-1}[0]} \cap \left(\bigcap_{i=1}^{r} Y^{\alpha_i,\beta_i,\gamma_i}_{\sigma,\tau^{-1}[i]} \right) \cap \left(\bigcap_{i=r+1}^{h} Z_{\sigma,\tau^{-1}[i]} \right).$$

Writing $v = \text{XCanonicalise}(w)$, it remains to show that $f_u = f_v$, since $f_v = f_w$ by Correctness Theorem 5.7. Let $x \in \mathbb{R}^n$. Decompose

$$\begin{aligned} f_u(x) &= d + \sum_{i=1}^{h} a_i \tanh(b_i x + c_i) \\ &= \underbrace{d + \sum_{i \in \tau^{-1}[0]} a_i \tanh(b_i x + c_i)}_{(1)} + \sum_{j=1}^{r} \underbrace{\sum_{i \in \tau^{-1}[j]} a_i \tanh(b_i x + c_i)}_{(2j)} + \sum_{j=r+1}^{h} \underbrace{\sum_{i \in \tau^{-1}[j]} a_i \tanh(b_i x + c_i)}_{(3j)} \end{aligned}$$

Then (1) equals the constant term of f_v, the (2j) equal the hidden units of f_v, and the (3j) vanish, as can be verified:

1. Since $u \in X^{\delta}_{\tau^{-1}[0]}$, $(1) = d + \sum_{i\in\tau^{-1}[0]} a_i \tanh(b_i x + c_i) = d + \sum_{i\in\tau^{-1}[0]} a_i \tanh(c_i) = \delta$.

2. Since $u \in Y^{\alpha_j,\beta_j,\gamma_j}_{\sigma,\tau^{-1}[j]}$,

$$\begin{aligned}
(2j) &= \sum_{i\in\tau^{-1}[j]} a_i \tanh(b_i x + c_i) \\
&= \sum_{i\in\tau^{-1}[j]} a_i \tanh(\sigma_i\beta_j x + \sigma_i\gamma_j) && (\sigma_i \cdot (b_i, c_i) = (\beta_j, \gamma_j)) \\
&= \sum_{i\in\tau^{-1}[j]} \sigma_i a_i \tanh(\beta_j x + \gamma_j) && (\tanh(\pm z) = \pm\tanh(z)) \\
&= \alpha_j \tanh(\beta_j x + \gamma_j). && (\textstyle\sum_{i\in\tau^{-1}[j]} \sigma_i a_i = \alpha_j)
\end{aligned}$$

3. Since $u \in Z_{\sigma,\tau^{-1}[j]}$,

$$\begin{aligned}
(3j) &= \sum_{i\in\tau^{-1}[j]} a_i \tanh(b_i x + c_i) \\
&= \sum_{i\in\tau^{-1}[j]} \sigma_i a_i \tanh(\sigma_i b_i x + \sigma_i c_i) && (\tanh(z) = \pm\tanh(\pm z)) \\
&= \sum_{i\in\tau^{-1}[j]} \sigma_i a_i \cdot (\text{a constant in } i) && (\sigma_i b_i = \sigma_j b_j, \sigma_i c_i = \sigma_j c_j) \\
&= 0. && (\textstyle\sum_{i\in\tau^{-1}[j]} \sigma_i a_i = 0)
\end{aligned}$$

Thus $f_u(x) = f_v(x)$, and therefore $u \in \mathfrak{F}[w]$.

($\mathfrak{F}[w] \subset \bigcup_\xi S_\xi$): Suppose $w' \in \mathfrak{F}[w]$, so XCanonicalise(w) = XCanonicalise(w') by Correctness Theorem 5.7. Construct a canonicalisation trace $\xi \in \Xi_{\mathrm{C}}(h, r)$ from the execution of the canonicalisation algorithm on w', and show that $w' \in S_\xi$. Proceed to construct the trace $\xi = (\sigma, \tau)$ as follows:

1. From the execution of Algorithm 4.12 (FastReduce) on w', identify the units that have zero incoming weight and are excluded from I in the first pass, and have τ map these units to 0:
$$\forall i \notin I, \tau(i) = 0.$$
Note that it follows immediately that $w' \in X^{\delta}_{\tau^{-1}[0]}$ and this is not jeopardised by the remaining steps.

2. Construct a sign vector σ based on the lexicographic sign of the incoming weight vector of each unit. Use arbitrary signs for the units in $\tau^{-1}[0]$.

3. From the second pass, for $i = 1, \ldots, k$ have τ map each of the units in partition group Π_i to i. It follows that $w' \in Y^{\alpha'_i,\beta'_i,\gamma'_i}_{\sigma,\tau^{-1}[i]}$, where α'_i, β'_i, and γ'_i denote the variables α_i, β_i, and γ_i in the execution of Algorithm 4.12 on w' (cf. theorem statement).

4. From the third pass, remap τ so that those partition groups previously mapping to i such that $\alpha'_i = 0$ map instead onto $\{r+1, \ldots, k\}$ (and remap the remaining groups onto $\{1, \ldots, r\}$, see also the next step). It follows that for $i = r+1, \ldots, k$, $w' \in Y^{0,\beta'_i,\gamma'_i}_{\sigma,\tau^{-1}[i]} \subset Z_{\sigma,\tau^{-1}[i]}$.

5. Also from the third pass, remap the r partition groups whose units remain instead onto $\{1, \ldots, r\}$ in order of increasing lexicographic absolute value of the incoming weight and bias vector. Noting that the sign vector computed in Algorithm 5.1 (AbsoluteSort) is $(1, \ldots, 1)$ (since Algorithm 4.12 already normalises units to have lexicographically positive incoming weight and bias vectors), and the permutation is computed exactly to bring the units into increasing order, it now holds that for $i = 1, \ldots, r$, $w' \in Y^{\alpha_i,\beta_i,\gamma_i}_{\sigma,\tau^{-1}[i]}$.

By construction, $\xi = (\sigma, \tau)$ is a canonicalisation trace of order r, and $w' \in S_{\sigma,\tau}$. Therefore, $w' \in \bigcup_\xi S_\xi$. $\square$

Remark 5.14. It is instructive to consider how the above result contains Theorem 3.39 as a special case. If $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}^{n,m}_h$ is irreducible then $\operatorname{rank}(w) = h$ (Proposition 4.6). So the union is over $\Xi_{\mathrm{C}}(h, h)$, canonicalisation traces of order h on h units. By Definition 5.12, such canonicalisation traces comprise a sign vector $\sigma \in \{-1, +1\}^h$ and a map $\tau : \{1, \ldots, h\} \to \{0, \ldots, h\}$ with range including $\{1, \ldots, h\}$. The latter constraint implies that τ is effectively a permutation of $\{1, \ldots, h\}$, so write $\tau = \pi \in S_h$. The subset corresponding to this canonicalisation trace is the intersection of $X^d_\emptyset$ (parameters with output bias d) and, for each $i = 1, \ldots, h$, $Y^{a_i,b_i,c_i}_{\sigma,\{\pi^{-1}(i)\}}$ (parameters for which unit $j = \pi^{-1}(i)$ has outgoing weight $\sigma_j a_i$, incoming weight $\sigma_j b_i$, and bias $\sigma_j c_i$). Such constraints uniquely identify the parameter $T_\sigma(T_\pi(w))$.

Remark 5.15. Another special case of this result arises when $\operatorname{rank}(w) = h-1$. There are, not counting sign vectors and permutations, essentially three possible canonicalisation traces of order $h-1$ on h units, corresponding to the different ways of mapping the "spare" unit to $\{0, \ldots, h\}$ after the necessary range $\{1, \ldots, h-1\}$ has been covered: (1) to map the spare unit to one of $1, \ldots, h-1$ again; (2) to map the spare unit to 0; or (3) to map the spare unit to h. From the perspective of embedding the reduced parameter into the original parameter space by adding a redundant unit, these three canonicalisation traces correspond to the three embedding methods catalogued by Fukumizu and Amari (2000) and Fukumizu et al. (2019), namely, (1) to split one of the existing units in two; (2) to introduce a new constant unit with zero incoming weight; or, (3) to introduce a new unit with zero outgoing weights.

Remark 5.16. Şimşek et al. (2021) have catalogued methods of adding multiple hidden units. Though they study a setting with simpler reducibility conditions it is instructive to consider the similarities between Theorem 5.13 and Şimşek et al. (2021, Defs. 3.2 & 3.3).

Properties of the functional equivalence class

While it is difficult to compute functional equivalence classes for low-rank parameters in high-dimensional parameter spaces, the characterisation in Theorem 5.13 provides some somewhat-scalable intuition. From this result several basic properties of the functional equivalence class quickly follow.

First, the functional equivalence class for a reducible parameter is a closed subset of the reducible region. It therefore shares several properties with the reducible region (Section 3.3) and bounded rank regions (Section 4.3). Namely, it has open interior, its complement is dense, and it has measure zero (see proof of Corollary 4.22).

Corollary 5.17. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m} = \mathbb{R}^{(n+m+1)h+m}$. Given a reducible parameter $w \in \mathfrak{R}[\mathcal{W}_h^{n,m}]$ (that is, $\operatorname{rank}(w) < h$), the functional equivalence class $\mathfrak{F}[w] \subset \mathcal{W}_h^{n,m}$ is a closed subset of $\mathfrak{R}[\mathcal{W}_h^{n,m}]$.*

Proof. That $\mathfrak{F}[w]$ is a subset of $\mathfrak{R}[\mathcal{W}_h^{n,m}]$ follows by Remark 3.28. That $\mathfrak{F}[w]$ is closed follows as it is a finite union of closed subsets of $\mathcal{W}_h^{n,m}$ (Theorem 5.13). □

Notably, the functional equivalence class for a reducible parameter is *not* algebraic, unlike the bounded rank regions or the reducible region.

Remark 5.18. The functional equivalence class for a reducible parameter is not real affine algebraic. This is chiefly due to the X_I^δ components in the characterisation, which are always present for reducible parameters and crucially involve a non-polynomial (analytic) constraint

$$d + \sum_{i \in I} a_i \tanh(c_i) = \delta.$$

Note that this is the only non-polynomial constraint, and the set can be made algebraic through an analytic reparameterisation of the architecture that replaces the hidden unit bias parameters c_i with $c_i' = \tanh(c_i) \in (0, 1)$. Then the analytic constraint above becomes polynomial:

$$d + \sum_{i \in I} a_i c_i' = \delta.$$

The other constraints involving c_i remain algebraic since tanh is odd and monotonic:

$$\begin{array}{lclclcl} \sigma_i c_i = \gamma & \Leftrightarrow & \tanh(\sigma_i c_i) = \tanh(\gamma) & \Leftrightarrow & \sigma_i c_i' = \gamma' & & (\text{in } Y_{\sigma,I}^{\alpha,\beta,\gamma}) \\ \sigma_i c_i = \sigma_j c_j & \Leftrightarrow & \tanh(\sigma_i c_i) = \tanh(\sigma_j c_j) & \Leftrightarrow & \sigma_i c_i' = \sigma_i c_j'. & & (\text{in } Z_{\sigma,I}) \end{array}$$

This reparameterisation is an important precursor to forming an understanding of the functional equivalence class from the perspective of algebraic geometry (cf. Wong, 2022).

5.3 Connectivity of the functional equivalence class

In this section, I establish one further property of the functional equivalence class of reducible parameters: all points within such a functional equivalence class are connected by a piecewise linear path comprising parameters in the functional equivalence class. This emphasises the contrast between the functional equivalence class of reducible and irreducible parameters (the latter being discrete and disconnected).

Piecewise linear path connectivity

I begin by formally defining the property of piecewise linear path connectivity.

Definition 5.19 (Piecewise linear path)**.** Given a subset $\mathfrak{W} \subset \mathbb{R}^p$, a *piecewise linear path in* $\mathfrak{W}$ is a continuous function $\rho : [0, 1] \to \mathfrak{W}$ comprising a finite number of linear segments. The point $\rho(0) \in \mathfrak{W}$ is the *starting point* of the path and the point $\rho(1) \in \mathfrak{W}$ is the *ending point* of the path. The number of (maximal) segments in the path is the *length* of the path.

Definition 5.20 (Piecewise linear path connectivity relation)**.** Given a subset $\mathfrak{W} \subset \mathbb{R}^p$, define a relation $\leftrightsquigarrow$ on $\mathfrak{W}$, called *piecewise linear path connectivity in* $\mathfrak{W}$*,* such that for $w, w' \in \mathfrak{W}$,

$$w \leftrightsquigarrow w' \qquad \Leftrightarrow \qquad \begin{array}{l} \exists \rho : [0,1] \to \mathfrak{W}, \text{ a piecewise linear path in } \mathfrak{W} \text{ with} \\ \text{starting point } \rho(0) = w \text{ and ending point } \rho(1) = w'. \end{array}$$

Proposition 5.21. *Given a subset* $\mathfrak{W} \subset \mathbb{R}^p$*, the relation of piecewise linear path connectivity in* $\mathfrak{W}$ *is an equivalence relation.*

Proof. (Reflexivity): For $w \in \mathfrak{W}$, $w \leftrightsquigarrow w$ by the "empty" path $\rho(t) = w$.

(Symmetry): For $u, v \in \mathfrak{W}$, if $u \leftrightsquigarrow v$ via the path $\rho_{u\to v}$, then $v \leftrightsquigarrow u$ via the "reverse" path $\rho_{v\to u}(t) = \rho_{u\to v}(1 - t)$.

(Transitivity): For $u, v, w \in \mathfrak{W}$, if $u \leftrightsquigarrow v$ via the path $\rho_{u\to v}$, and $v \leftrightsquigarrow w$ via the path $\rho_{v\to w}$, then $u \leftrightsquigarrow w$ via the "catenated" path

$$\rho_{u\to w}(t) = \begin{cases} \rho_{u\to v}(t/2) & (t \leq 1/2) \\ \rho_{v\to w}(t/2 - 1/2) & (t > 1/2). \end{cases}$$

Note that the length of the catenated path is the sum of the lengths of these two paths, or, if the connected segments are co-linear, the sum less by one. Either way, it is finite as required. □

Definition 5.22 (Piecewise linear path connectivity of a set)**.** Consider a subset $\mathfrak{W} \subset \mathbb{R}^p$. $\mathfrak{W}$ is a *piecewise linear path connected set* if, for every pair $w, w' \in \mathfrak{W}$, $w \leftrightsquigarrow w'$.

Connectivity of the functional equivalence class

The main result in this section is that for reducible neural network parameters, the functional equivalence class is a piecewise linear path connected set.[1] To prove this result I need the following definition.

Definition 5.23 (Pseudo-reduced simple neural network parameter)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. A parameter $w \in \mathcal{W}_h^{n,m}$ is *pseudo-reduced* if, for all but $\operatorname{rank}(w)$ of its hidden units, the incoming weight vector, outgoing weight vector, and bias are zero (call such units *blank*).

Remark 5.24. A pseudo-reduced parameter can be understood as the result of carrying out a reduction algorithm such as Algorithm 4.9 (SlowReduce), modified such that instead of removing a unit from the network in each iteration, it retains the unit but sets the unit's weights and bias to zero. (The unit should be marked as "pseudo-removed" and excluded from future iterations, so that the algorithm will still terminate.)

Pseudo-reduced parameters play a key role in the proof of the following main result, which proceeds by constructing a path between any two functionally equivalent parameters via their pseudo-reduced forms as computed by the modified algorithm outlined above. The proof is supported by two lemmas, given in the remainder of this section, that construct piecewise linear paths connecting each parameter to its pseudo-reduced form, and connecting two such pseudo-reduced parameters.

Theorem 5.25 (Piecewise linear path connectivity of reducible functional equivalence class)**.** *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a reducible parameter $w \in \mathfrak{R}[\mathcal{W}_h^{n,m}]$, the functional equivalence class $\mathfrak{F}[w]$ is piecewise linear path connected.*

Proof. Let $u, v \in \mathfrak{F}[w]$. Let u', v' be the pseudo-reduced versions of u, v, computed via Algorithm 4.9 (SlowReduce) modified as described in Remark 5.24. Then, by Lemma 5.28 (below), $u \leftrightsquigarrow u'$ and $v \leftrightsquigarrow v'$. Moreover, by Lemma 5.30 (below), $u' \leftrightsquigarrow v'$. The theorem follows since $\leftrightsquigarrow$ is an equivalence relation (Proposition 5.21). □

Remark 5.26. If $h > 0$, an irreducible functional equivalence class is disconnected.

Remark 5.27. There is a sense in which the functional equivalence class of a parameter with low rank is "more" connected than the functional equivalence class of a parameter with high rank, in terms of being connected with piecewise linear paths with shorter length (number of maximal linear segments). The constructions given in Lemmas 5.28 and 5.30 show the *existence* of paths, without attempting to economise on length. Remarks 5.29 and 5.31 to 5.33 discuss how shorter paths can be constructed for low-rank parameters.

[1]Shortly before submission, I discovered that Şimşek et al. (2021) published a similar path construction, having approached the problem from a dual perspective of embedding irreducible parameters into higher parameter spaces (cf. Section 2.5), and in a setting with somewhat simpler reducibility conditions.

Piecewise linear connectivity and pseudo-reduction

I show how a piecewise linear path of functionally equivalent parameters can be constructed between a parameter and its pseudo-reduced form.

Lemma 5.28. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a parameter $w \in \mathcal{W}_h^{n,m}$, let w' be computed by the algorithm described in Remark 5.24. Then in the functional equivalence class $\mathfrak{F}[w]$,*

$$w \leftrightsquigarrow w'.$$

Proof. Let $r = \operatorname{rank}(w)$. Then the modified reduction algorithm runs for $h-r$ iterations, producing a chain of intermediate functionally equivalent parameters $w_{(0)}, w_{(1)}, \ldots, w_{(h-r)} \in \mathcal{W}_h^{n,m}$ with $w = w_{(0)}$ and $w_{(h-r)} = w'$. For iteration $i = 1, \ldots, h-r$, write $w_{(i-1)} = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$ and $w_{(i)} = (a'_1, \ldots, a'_h, b'_1, \ldots, b'_h, c'_1, \ldots, c'_h, d')$, and construct a piecewise linear path connecting $w_{(i-1)} \leftrightsquigarrow w_{(i)}$ as follows.

1. If hidden unit j is to be pseudo-removed under reducibility condition (i), then $a_j = a'_j = 0$ and $b'_j = c'_j = 0$. Construct a direct path to $w_{(i)}$.
2. If hidden unit j is to be pseudo-removed under reducibility condition (ii), then $b_j = b'_j = 0$, $a'_j = c'_j = 0$, and $d' = d + a_j \tanh(c_j)$. Construct a two-stage path. First, proceed directly to the point with output bias d' and where unit j has zero outgoing weight, holding unit j's bias constant. From there, proceed as in case (1).
3. If hidden unit k is to be pseudo-merged into hidden unit j under reducibility condition (iii), then $(b'_j, c'_j) = (b_j, c_j) = (b_k, c_k)$, $a'_j = a_j + a_k$, and $a'_k = b'_k = c'_k = 0$. Construct a two-stage path. First, proceed directly to the point where unit j has outgoing weight a'_j and unit k has zero outgoing weight, holding the incoming weight and bias vectors constant. From there, proceed as in case (1).
4. If hidden unit k is to be pseudo-merged into hidden unit j under reducibility condition (iv), then $(b'_j, c'_j) = (b_j, c_j) = -(b_k, c_k)$, $a'_j = a_j - a_k$, and $a'_k = b'_k = c'_k = 0$. Proceed essentially as in case (3).

Each of these paths remains in the functional equivalence class by construction. Thus, $w = w_{(0)} \leftrightsquigarrow w_{(1)} \leftrightsquigarrow \cdots \leftrightsquigarrow w_{(h-r)} = w'$. □

Remark 5.29. The constructed path can be compressed into a path with just two segments. First, execute in parallel all of the shifts of outgoing weight to the eventual units they end up at, and from constant units to the output unit bias vector. Then, for all units with zero outgoing weight, send the incoming weight and bias vectors to zero in a single second segment.

Piecewise linear connectivity and absolute sorting

I show how a piecewise linear path of functionally equivalent parameters can be constructed between two functionally equivalent pseudo-reduced parameters.

Lemma 5.30. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Consider two reducible, functionally equivalent, pseudo-reduced parameters $w, w' \in \mathfrak{R}[\mathcal{W}_h^{n,m}]$. Then in the functional equivalence class $\mathfrak{F}[w]$,*

$$w \leftrightsquigarrow w'.$$

Proof. First, observe that any two functionally equivalent, pseudo-reduced parameters are related by a permutation and negation transformation. This follows from Theorem 3.39, because pseudo-reduced parameters are just irreducible parameters with additional blank units: a permutation transformation can take care of the positioning of the blank units within the network, and the remaining differences are exactly those of the corresponding irreducible parameters. Therefore, for some permutation $\pi \in S_h$ and sign vector $\sigma \in \{-1,+1\}^h$, write $w' = T_\sigma(T_\pi(w))$. To show that $w \leftrightsquigarrow w'$, consider the following cases:

1. If $\sigma = (1, \ldots, 1)$ and π is a transposition between i and j, where i or j is a blank unit: without loss of generality assume j is blank, and construct a path with three linear segments as follows.
   (a) Interpolate the incoming weight and bias vector of the blank unit j to match that of i, keeping the outgoing weight zero.
   (b) Shift the outgoing weight from unit i to unit j.
   (c) Interpolate the incoming weight and bias vector of unit i to zero, keeping the outgoing weight zero.

2. If $\sigma = (1, \ldots, 1)$ and π is a general transposition between i and j (neither of which is a blank unit): since w is reducible and pseudo-reduced, there must be a blank unit k. Using thrice the three-segment construction in case (1), proceed to swap units i and j via k as follows.
   (a) Transpose unit j into blank unit k.
   (b) Transpose unit i into (now) blank unit j.
   (c) Transpose unit k (containing original unit j) into (now) blank unit i.

3. If $\pi = \mathrm{id}$ and σ is an *individual negation* having exactly one negative component, $\sigma_i = -1$: if i is blank, use an empty path. If not, then since w is reducible and pseudo-reduced, there is a blank unit j. Construct a path with three segments as follows.

(a) Interpolate the incoming weight and bias vector of the blank unit j to be negated with respect to that of i, keeping the outgoing weight zero.

(b) Shift (while negating) the outgoing weight from unit i to unit j.

(c) Interpolate the incoming weight and bias vector of unit i to zero, keeping the outgoing weight zero.

This connects w to a parameter with unit i negated, but also transposed with blank unit j. From here use the three-segment construction from case (1) to reach w'.

The general case follows from these special cases, since π can be written as a product of transpositions, and σ as a product of individual negations (each of which preserves the pseudo-reduced property of the parameter). □

Remark 5.31. Between each application of the case (1) construction it is possible to reduce the number of segments by "cutting corners," avoiding the intermediate point restoring the blank unit.

Remark 5.32. The length of the path otherwise depends on the number of individual negations and transpositions separating the two pseudo-reduced parameters. However, if there are multiple blank units (that is, for low-rank parameters), then it is possible to compress the path further by transposing and negating multiple units in parallel.

Remark 5.32 introduces an interesting relationship between the sign vector, the permutation, the rank (determining the number of blank units), and the length of the shortest piecewise linear path implementing the corresponding transformation. If blank units are regarded as "space" and linear segments as "time" this relationship is reminiscent of the kind of problems studied in computational complexity theory. Aside from the below remark, which can be seen as a single point on the "time–space trade-off" curve, I leave it to future work to more thoroughly explore this topic.

Remark 5.33. If at least half of the units are blank ($\text{rank}(w) \leq h/2$), then any two equivalent pseudo-reduced parameters can be connected with a path of just five linear segments. First, with the construction in (1) from the above proof, exchange (at once) all units of w that are occupying some non-blank unit of w' with some units that are blank in both parameters. Then, at once exchange all units into the appropriate positions in w' according to the transformation, using the constructions in (1) or the first part of (3) as appropriate. The resulting six-segment path can be cut to five using Remark 5.31. Note also that combined with Remarks 5.29 and 5.31, this construction implies a path of length seven between any two functionally equivalent parameters with rank at most $h/2$.

Chapter 6

Degenerate Neighbourhoods in Parameter Space

Chapters 4 and 5 investigated (exact) degeneracy in simple neural networks, and its implications for (exact) functional equivalence. However, degenerate neural networks also have important implications for their *neighbourhoods* (cf. Section 2.5).

In this chapter, I study a measure of *approximate degeneracy* for simple neural network parameters based on proximity to degenerate regions. In Section 6.1, I formally define the *parametric*[1] *approximate*[2] *rank*—the rank of the most degenerate neural network parameter within a uniform neighbourhood. I study some of its basic properties, and derive a polynomial-time greedy algorithm computing an upper bound.

In the remainder of this chapter, I show that tightly bounding the parametric approximate rank (or, detecting proximity to bounded rank regions) is $\mathcal{NP}$-complete.

1. The reduction proceeds via a novel combinatorial problem, called *uniform point partition,* which I explore in detail from several equivalent perspectives in Section 6.2.
2. In Section 6.3, I show that uniform point partition is $\mathcal{NP}$-complete, by reduction from the restricted variant of Boolean satisfiability introduced in Section 3.4.
3. In Section 6.4, I show that a decision problem involving the parametric approximate rank is $\mathcal{NP}$-complete, by reduction from uniform point partition.

This shows that, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial-time algorithm that computes the parametric approximate rank, underscoring the (computational) complexity of the neighbourhoods of degenerate regions of parameter space.

[1] I consider proximity in parameter space as the most relevant for learning algorithms that operate by local search. In Appendix A, I define a second measure of approximate degeneracy based on proximity in function space, and investigate the relationship between these measures. As for the use of uniform distance, this metric is computationally convenient (since it somewhat decouples dimensions).

[2] The "approximation" in this notion is of the parameter, as opposed to the rank (compare, "measure of approximate degeneracy," above, with "approximate measure of degeneracy"). There is little need to approximate the rank itself, which can be readily computed (Section 4.2).

6.1 Parametric approximate rank

In this section, I study the following measure of approximate degeneracy for simple neural network parameters based on (uniform) proximity to low-rank parameters.

Definition 6.1 (Parametric approximate rank)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a neural network parameter $w \in \mathcal{W}_h^{n,m}$ and a positive radius $\varepsilon \in \mathbb{R}^+$, define the *parametric approximate rank* of w, denoted $\operatorname{prank}_\varepsilon(w)$, as the rank of the lowest-rank parameter within a closed uniform neighbourhood of w with radius ε. That is,

$$\operatorname{prank}_\varepsilon(w) = \min \left\{ \operatorname{rank}(u) \in \mathbb{N} \,\middle|\, u \in \bar{B}_\infty(w;\varepsilon) \right\}.$$

Properties of parametric approximate rank

Proposition 6.2 provides some basic consequences of the definition, and Proposition 6.3 clarifies the role of the uniform radius.

Proposition 6.2. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $w \in \mathcal{W}_h^{n,m}$ and $\varepsilon \in \mathbb{R}^+$.*

(i) For all $u \in \bar{B}_\infty(w;\varepsilon)$, $\operatorname{prank}_\varepsilon(w) \le \operatorname{rank}(u)$.

(ii) If $\operatorname{prank}_\varepsilon(w) = r$, then $\exists u \in \bar{B}_\infty(w;\varepsilon)$ with $\operatorname{rank}(u) = r$.

(iii) $\operatorname{prank}_\varepsilon(w) \le \operatorname{rank}(w)$.

(iv) $\operatorname{prank}_\varepsilon(w) \le r$ if and only if $\exists u \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ with $\|u - w\|_\infty \le \varepsilon$.

Proof. (i) and (ii) by definition (as a minimum). (iii) by (i). (iv) by (i) and (ii). □

Proposition 6.3. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Fix $w \in \mathcal{W}_h^{n,m}$ and consider $\operatorname{prank}_\varepsilon(w)$ as a function of ε. Then,*

(i) $\operatorname{prank}_\varepsilon(w)$ is antitone in ε: if $\varepsilon \ge \varepsilon'$ then $\operatorname{prank}_\varepsilon(w) \le \operatorname{prank}_{\varepsilon'}(w)$; and

(ii) $\operatorname{prank}_\varepsilon(w)$ is right-continuous in ε: $\lim_{\delta\to0^+} \operatorname{prank}_{\varepsilon+\delta}(w) = \operatorname{prank}_\varepsilon(w)$.

Proof. For (i), put $u \in \bar{B}_\infty(w;\varepsilon')$ such that $\operatorname{rank}(u) = \operatorname{prank}_{\varepsilon'}(w)$. Then $u \in \bar{B}_\infty(w;\varepsilon)$, so $\operatorname{prank}_{\varepsilon'}(w) = \operatorname{rank}(u) \ge \operatorname{prank}_\varepsilon(w)$. Then for (ii), the limit exists by the monotone convergence theorem (and Proposition 6.2(iii)). Proceed to bound $\operatorname{prank}_\varepsilon(w)$ above and below by $r = \lim_{\delta\to0^+} \operatorname{prank}_{\varepsilon+\delta}(w)$. For the lower bound, for $\delta \in \mathbb{R}^+$, $\operatorname{prank}_{\varepsilon+\delta}(w) \le \operatorname{prank}_\varepsilon(w)$ by (i), so $r = \lim_{\delta\to0^+} \operatorname{prank}_{\varepsilon+\delta}(w) \le \operatorname{prank}_\varepsilon(w)$.

For the upper bound, since the parametric approximate rank is natural, the limit is achieved for some positive δ. That is, $\exists \Delta \in \mathbb{R}^+$ such that $\operatorname{prank}_{\varepsilon+\Delta}(w) = r$. Then for $k = 1, 2, \dots$ put $u_k \in \bar{B}_\infty(w; \varepsilon + \Delta/k)$ with $\operatorname{rank}(u_k) = r$. Since $\bar{B}_\infty(w; \varepsilon + \Delta)$ is compact the sequence $u_1, u_2, \dots$ has an accumulation point—call it u. Now, since $u_1, u_2, \dots \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$ (parameters of rank at most r, Definition 4.18), a closed set (Corollary 4.22), the accumulation point $u \in \mathfrak{B}_r[\mathcal{W}_h^{n,m}]$. Thus, $\operatorname{rank}(u) \leq r$. Finally, $u \in \bigcap_{k=1}^\infty \bar{B}_\infty(w; \varepsilon + \Delta/k) = \bar{B}_\infty(w; \varepsilon)$, so $r \geq \operatorname{rank}(u) \geq \operatorname{prank}_\varepsilon(w)$. □

Remark 6.4. A similar proof implies that $\operatorname{prank}_\varepsilon(w)$ achieves its upper bound $\operatorname{rank}(w)$ for small enough ε. The lower bound 0 is also achieved, for $\varepsilon \geq \|w\|_\infty = \|w - 0\|_\infty$.

Parametric instability of parametric approximate rank

Unlike for rank, two parameters with the same function may not share the same parametric approximate rank—consider Example 6.5.[3] However, a limited form of parameter independence holds as shown by Proposition 6.6 and Corollary 6.7.

Example 6.5. Let $\varepsilon \in \mathbb{R}^+$. Consider the neural network parameters $w, w' \in \mathcal{W}_2^{1,1}$ with $w = (2\varepsilon, 0, 2\varepsilon, 0, 0, 0, 0)$ and $w' = (2\varepsilon, 0, 2\varepsilon, 5\varepsilon, 0, 0, 0)$. Then

$$\operatorname{prank}_\varepsilon(w) = \operatorname{rank}(\varepsilon, -\varepsilon, \varepsilon, \varepsilon, 0, 0, 0) = 0 \neq 1 = \operatorname{prank}_\varepsilon(w'),$$

but, for $x \in \mathbb{R}$, $f_w(x) = 2\varepsilon \tanh(2\varepsilon x) + 0 \tanh(0x) = 2\varepsilon \tanh(2\varepsilon x) + 0 \tanh(5\varepsilon x) = f_{w'}(x)$.

Proposition 6.6. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$; a permutation $\pi \in S_h$; a sign vector $\sigma \in \{-1, +1\}^h$; and the corresponding transformations T_π and T_σ (cf. Definitions 3.32 and 3.35). For $w \in \mathcal{W}_h^{n,m}, \varepsilon \in \mathbb{R}^+$,*

$$\operatorname{prank}_\varepsilon(w) = \operatorname{prank}_\varepsilon(T_\sigma(T_\pi(w))).$$

Proof. Let $w' = T_\sigma(T_\pi(w))$. Let $u \in \bar{B}_\infty(w; \varepsilon)$ with $\operatorname{rank}(u) = \operatorname{prank}_\varepsilon(w)$, and let $u' = T_\sigma(T_\pi(u))$. Then $u' \in \bar{B}_\infty(w'; \varepsilon)$ since T_π and T_σ are isometries (Proposition 3.37):

$$\|u' - w'\|_\infty = \|T_\sigma(T_\pi(u)) - T_\sigma(T_\pi(w))\|_\infty = \|u - w\|_\infty \leq \varepsilon.$$

It follows that $\operatorname{prank}_\varepsilon(w) = \operatorname{rank}(u) = \operatorname{rank}(u') \geq \operatorname{prank}_\varepsilon(w')$. Moreover, by the same argument, $\operatorname{prank}_\varepsilon(w') \geq \operatorname{prank}_\varepsilon(T_{\pi^{-1}}(T_\sigma(w'))) = \operatorname{prank}_\varepsilon(w)$. □

Corollary 6.7. *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Consider two irreducible parameters $w, w' \in \mathfrak{I}[\mathcal{W}_h^{n,m}]$ and a radius $\varepsilon \in \mathbb{R}^+$. If $f_w = f_{w'}$ then* $\operatorname{prank}_\varepsilon(w) = \operatorname{prank}_\varepsilon(w')$.

Proof. Since w, w' are functionally equivalent and irreducible, they are related by some permutation and negation (Theorem 3.39). The result follows by Proposition 6.6. □

[3] Similar counterexamples hold for other natural choices of metric—parameter dependence is a consequence of using a "local" definition. A more stable notion of parametric approximate degeneracy must somehow aggregate neighbourhoods across equivalent parameters (see also Appendix A).

Approximating parametric approximate rank

Algorithm 6.10 is a polynomial-time algorithm computing an upper bound on the parametric approximate rank. The algorithm essentially functions by replacing each of the three main steps of Algorithm 4.15 (for computing the rank) with an *approximate* version, as follows.

1. Instead of eliminating units with zero incoming weight vector, eliminate units with *approximately* zero incoming weight vector (there is a nearby parameter where they are zero).
2. Instead of partitioning the remaining units by lexicographic absolute incoming weight and bias vector, cluster them into groups with *similar* lexicographic absolute incoming weight and bias vectors (there is a nearby parameter where they are equal).
3. Instead of eliminating unit groups with zero outgoing weight, eliminate unit groups with *approximately* zero incoming weight (there is a nearby parameter where the outgoing weights cancel).

Step (2) is non-trivial, I use a greedy approach, described separately as Algorithm 6.8. This algorithm builds an approximate partition of a list of vectors by iteratively inserting vectors into nearby groups, or, if there are no nearby groups, starting a new group.

Algorithm 6.8 (Greedy approximate partition)**.** Given a vector space $\mathbb{R}^p$, proceed:

```
1: procedure ApproxPartition($\varepsilon \in \mathbb{R}^+$, $u_1, \ldots, u_K \in \mathbb{R}^p$)
2:     $J \leftarrow 1$
3:     $v_1, \Pi_1 = u_1, \{1\}$                                  ▷ the first vector starts a group
4:     for $i \leftarrow 2, \ldots, K$ do                              ▷ for the remaining vectors
5:         for $j \leftarrow 1, \ldots, J$ do
6:             if $\|u_i - v_j\|_\infty \leq \varepsilon$ then                 ▷ if near a group-starter
7:                 $\Pi_j \leftarrow \Pi_j \cup \{i\}$                            ▷ join that group
8:                 skip to next iteration of outer loop ($i$)      ▷ (skip to next vector)
9:             end if
10:        end for                                                ▷ else, start a group
11:        $J \leftarrow J + 1$
12:        $v_J, \Pi_J \leftarrow u_i, \{i\}$
13:    end for
14:    return $\Pi_1, \ldots, \Pi_J$
15: end procedure
```

Remark 6.9. Under the usual assumptions (cf. Section 4.2), the worst-case runtime for Algorithm 6.8 is $\mathcal{O}(pK^2)$.

Algorithm 6.10 (Greedy bound for parametric approximate rank)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$, proceed:

```
1:  procedure BOUND($\varepsilon \in \mathbb{R}^+$, $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$)
2:      ▷ Identify units with non-near-zero incoming weight ◁
3:      $I \leftarrow \{\, i \in \{1, \ldots, h\} \mid \|b_i\|_\infty > \varepsilon \,\}$
4:      ▷ Compute outgoing weight for approximately-mergeable units ◁
5:      $\Pi_1, \ldots, \Pi_J \leftarrow$ APPROXPARTITION($\varepsilon, \mathrm{abs}_{\mathrm{lex}}(b_i, c_i)$ for $i \in I$)    ▷ Algorithm 6.8
6:      for $j \leftarrow 1, \ldots, J$ do
7:          $\alpha_j \leftarrow \sum_{i \in \Pi_j} \mathrm{sign}_{\mathrm{lex}}(b_i)\, a_i$
8:      end for
9:      ▷ Count approximately-mergeable units with non-near-zero outgoing weights ◁
10:     return $\left|\{\, j \in \{1, \ldots, J\} \mid \|\alpha_j\|_\infty \leq \varepsilon \cdot |\Pi_j| \,\}\right|$    ▷ $|S|$ denotes set cardinality
11: end procedure
```

Remark 6.11. By Remark 6.9, Algorithm 6.10 runs in worst-case time $\mathcal{O}(nh^2 + mh)$.

Correctness Theorem 6.12 (Algorithm 6.10)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. For $w \in \mathcal{W}_h^{n,m}$ and $\varepsilon \in \mathbb{R}^+$,*

$$\mathrm{prank}_\varepsilon(w) \leq \textsc{Bound}(\varepsilon, w).$$

Proof. Trace the algorithm to construct a parameter $u \in \bar{B}_\infty(w; \varepsilon)$ with $\mathrm{rank}(u) \leq \textsc{Bound}(\varepsilon, w)$. Construct $u = (a_1^{(u)}, \ldots, a_h^{(u)}, b_1^{(u)}, \ldots, b_h^{(u)}, c_1^{(u)}, \ldots, c_h^{(u)}, d^{(u)}) \in \mathcal{W}_h^{n,m}$ as follows. Set $d^{(u)} = d$. For $i \notin I$, $\|b_i\|_\infty \leq \varepsilon$, so set $b_i^{(u)} = 0$, leaving $a_i^{(u)} = a_i$ and $c_i^{(u)} = c_i$. For $i \in \Pi_j$, note that $\|\mathrm{abs}_{\mathrm{lex}}(b_i, c_i) - v_j\|_\infty \leq \varepsilon$, so set $(b_i^{(u)}, c_i^{(u)}) = \mathrm{sign}_{\mathrm{lex}}(b_i) v_j$. If $\|\alpha_j\|_\infty \leq \varepsilon \cdot |\Pi_j|$, then set $a_i^{(u)} = a_i - \mathrm{sign}_{\mathrm{lex}}(b_i)\alpha_j |\Pi_j|^{-1}$, else set $a_i^{(u)} = a_i$.

By construction, $u \in \bar{B}_\infty(w; \varepsilon)$. Moreover, $\mathrm{rank}(u) = \textsc{Bound}(\varepsilon, w)$: run Algorithm 4.15: (1) the first step finds the same I, since those $b_i^{(u)} = 0$; (2) the partitioning step finds the same $\Pi_1, \ldots, \Pi_J$, since $\mathrm{abs}_{\mathrm{lex}}(b_i^{(u)}, c_i^{(u)}) = v_j$ (note that $\mathrm{sign}_{\mathrm{lex}}(b_i^{(u)}) = \mathrm{sign}_{\mathrm{lex}}(b_i)$); (3) the counting step excludes the same α_j, since those outgoing weights sum to 0. □

Remark 6.13. Algorithm 6.10 does *not* compute the parametric approximate rank—merely an upper bound. For example, there may be a more effective approximate partition than that found by the greedy approximate partitioning algorithm (see also Remark 6.45).

Actually, this suboptimality is fundamental—computing a smallest approximate partition is $\mathcal{NP}$-hard, and can be reduced to computing the parametric approximate rank. The remainder of this chapter formally proves this observation.

6.2 Uniform point partition and related problems

The main aim of the rest of this chapter is to reduce `xSAT` (Problem 3.56) to the problem of computing parametric approximate rank. To conceptually simplify the reduction, I first reduce `xSAT` to a closer but simpler problem. Before giving *that* reduction (Section 6.3), I dedicate *this* section to exploring the novel problem in detail, from three perspectives:

1. *Uniform point partitioning:* given some points in the plane, can the points be partitioned into a small number of groups with small uniform diameter?
2. *Uniform point covering:* given some points in the plane, is there a small number of new "covering" points so that each (original) point is near a covering point?
3. *Clique partitioning on unit square graphs:* given a special kind of graph called a *unit square graph,* can the vertices be partitioned into a small number of cliques?

I show that these perspectives emphasise different aspects of the same computational problem and suggest distinct connections to related work.

Perspective 1: Uniform point partition

The first perspective is inspired by Algorithm 6.8 and Remark 6.13. This problem captures the task of partitioning points in the plane into a small number of groups such that each group fits within a small uniform neighbourhood. A decision variant is as follows.

Definition 6.14 ((k, ε)-partition)**.** Consider n points $x_1, \dots, x_n \in \mathbb{R}^2$. A (k, ε)*-partition* is a partition of $\{1, \dots, n\}$ into k subsets $\Pi_1, \dots, \Pi_k$ such that the uniform distance between points in any subset is at most ε:

$$\forall p \in \{1, \dots, k\}, \forall i, j \in \Pi_p, \|x_i - x_j\|_\infty \leq \varepsilon\,.$$

Problem 6.15 (Uniform point partition, `UPP`)**.** Given n points $x_1, \dots, x_n \in \mathbb{R}^2$; a diameter $\varepsilon \in \mathbb{R}^+$; and some $k \in \mathbb{N}$, determine if there is a (k, ε)-partition of the points.

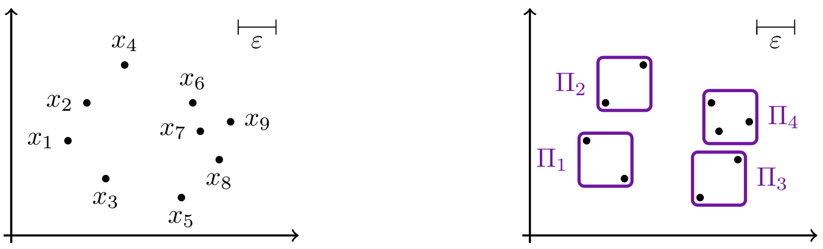


Figure 6.1: Example instance for uniform point partition (`UPP`, Problem 6.15). Left: nine points $x_1, \dots, x_9$. Right: a $(4, \varepsilon)$-partition $\Pi_1, \dots, \Pi_4$ $(= \{1, 3\}, \{2, 4\}, \{5, 8\}, \{6, 7, 9\})$.

Perspective 2: Uniform point cover

The second, closely related perspective is the task of finding central points within small uniform neighbourhoods (in simple neural networks, such points allow the approximate merging of hidden units; cf. Correctness Theorem 6.12).

Definition 6.16 ((k, ε)-cover)**.** Consider n *source points* in the plane, $x_1, \ldots, x_n \in \mathbb{R}^2$. A (k, ε)*-cover* is a list of k *covering points* $y_1, \ldots, y_k \in \mathbb{R}^2$ such that each source point is at most uniform distance ε from at least one covering point:

$$\forall i \in \{1, \ldots, n\}, \exists j \in \{1, \ldots, k\}, \|x_i - y_j\|_\infty \leq \varepsilon\,.$$

Problem 6.17 (Uniform point cover, `UPC`)**.** Given n source points $x_1, \ldots, x_n \in \mathbb{R}^2$; a radius $\varepsilon \in \mathbb{R}^+$; and some $k \in \mathbb{N}$, determine if there is a (k, ε)-cover of the source points.

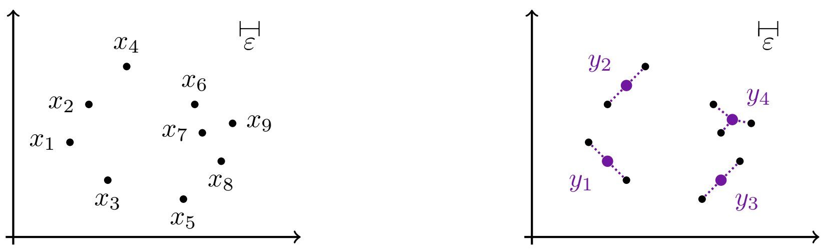


Figure 6.2: Example instance for uniform point cover (`UPC`, Problem 6.17). Left: nine source points $x_1, \ldots, x_9$. Right: a $(4, \varepsilon)$-cover $y_1, \ldots, y_4$.

Remark 6.18. The covering points are not necessarily source points—they are unconstrained within the plane. This can affect the decision. For example, with $x_1 = (0, 0)$, $x_2 = (2, 0)$, $\varepsilon = 1$, and $k = 1$, neither x_i is a (k, ε)-cover, but $y = (1, 0)$ is.

Remark 6.19. Uniform point cover is reminiscent of well-known hard clustering problems, with some subtle differences. *Planar k-means* (see, e.g., Mahajan et al., 2012), concerns finding centroids minimising a *weighted sum* of *squared Euclidean* distances between source points and nearby centroids. Uniform point cover concerns minimising a *maximum* of *uniform* distances instead. *Vertex k-center* (see, e.g., Garcia-Diaz et al., 2017) concerns maximum (Euclidean) distances, but the covering points are constrained to be source points. Remark 6.18 clarifies this crucial difference.

Remark 6.20. The closest related problem is perhaps a point covering problem studied by Supowit (1981, §4.3.2). Supowit's problem matches uniform point cover, except uniform distance is replaced by Euclidean distance. Supowit showed that this variant is $\mathcal{NP}$-complete by direct reduction from 3-`SAT`. My reduction (Section 6.3) is somewhat similar, but with some simplifications afforded by the additional restrictions of `xSAT`.

Perspective 3: Clique partition for unit square graphs

The third perspective strays from the simple neural network context, but reveals further related work. This perspective is based on the problem of *clique partition* restricted to a family of graphs I call *unit square graphs.* Unit square graphs are a uniform-distance variant of *unit disk graphs* (based on Euclidean distance; cf. Clark et al., 1990).

Definition 6.21 (Unit square graph)**.** Consider n points in the plane, $x_1, \dots, x_n \in \mathbb{R}^2$, and a diameter $\varepsilon \in \mathbb{R}^+$. Thus define an undirected graph (V, E) with vertices $V = \{1, \dots, n\}$ and edges $E = \left\{ \{i, j\} \,\middle|\, i \neq j, \|x_i - x_j\|_\infty \leq \varepsilon \right\}$. A *unit square graph* is any graph that can be constructed in this way.

Remark 6.22. "Unit square graph" comes from an equivalent definition of these graphs as intersection graphs of unit squares. To see the equivalence, scale the collection of squares by ε and then consider their centres. The same idea relates the *proximity* and *intersection* models for unit disk graphs (Clark et al., 1990).

Definition 6.23 (Clique partition)**.** Consider an undirected graph (V, E). A *clique partition* of size k is a partition of the vertices V into k subsets $\Pi_1, \dots, \Pi_k$ such that each subset is a clique:

$$\forall p \in \{1, \dots, k\}, \forall v_i \neq v_j \in \Pi_p, \{v_i, v_j\} \in E\,.$$

Problem 6.24 (Clique partition for unit square graphs, `usgCP`)**.** Given a unit square graph (V, E) and some $k \in \mathbb{N}$, determine if there is a clique partition of size k.

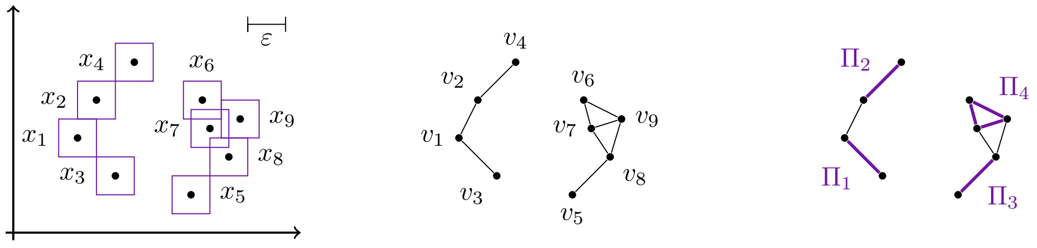


Figure 6.3: Example instance of clique partition for unit square graphs (`usgCP`, Problem 6.24). Left: nine points $x_1, \dots, x_9$, along with their ε-width squares. Middle: the corresponding unit square graph with vertices $v_1, \dots, v_9$. Right: a partition of the unit square graph into four cliques $\Pi_1, \dots, \Pi_4$.

Remark 6.25. The clique partition problem, long known to be $\mathcal{NP}$-complete in general graphs (Karp, 1972), remains $\mathcal{NP}$-complete when restricted to unit disk graphs (Cerioli et al., 2004; 2011).[4] Cerioli et al. (2004; 2011) gave a reduction from `planar 3-SAT`$_3$ that is somewhat similar to my reduction for unit square graphs (Section 6.3).

[4]Actually, Cerioli et al. (2004; 2011) studied *penny graphs,* which are special unit disk graphs in which the Euclidean distance between points is at least ε (evoking the contact relationships among non-overlapping coins). My reduction (Section 6.3) happens to produce a graph satisfying a uniform distance version of this condition, thus showing that clique partition remains $\mathcal{NP}$-complete in this case as well.

Remark 6.26. I haven't ruled out the case that unit disk graphs and unit square graphs coincide, or that every unit disk graph is also a unit square graph.[5] In either case, the hardness of clique partition for unit square graphs would follow from the result of Cerioli et al. (2004; 2011). Remembering the goal of studying parametric approximate rank, I pursue a direct reduction for unit square graphs in lieu of clarifying this relationship.

Equivalence of the three perspectives

The above three problems—`UPP`, `UPC`, and `usgCP`—are equivalent, in the sense that there is an immediate reduction between any pair of them, as follows.

Theorem 6.27 (Equivalence of `UPP`, `UPC`, and `usgCP`)**.** *Let $x_1, \dots, x_n \in \mathbb{R}^2$, $\varepsilon \in \mathbb{R}^+$, and $k \in \mathbb{N}$. The following conditions are equivalent:*

(i) there exists a (k, ε)-partition of $x_1, \dots, x_n$;

(ii) there exists a (k, ε)-cover of $2x_1, \dots, 2x_n$; and

(iii) the unit square graph on $x_1, \dots, x_n$ (diameter ε) has a clique partition of size k.

Proof. (i ⇒ ii): Let Π be a (k, ε)-partition of the points $x_1, \dots, x_n$. Construct a (k, ε)-cover of $2x_1, \dots, 2x_n$ as follows. For $p = 1, \dots, k$, define $x_{\min}, x_{\max}, y_p \in \mathbb{R}^2$ with components $x_{\max,j} = \max_{i \in \Pi_p} x_{i,j}$, $x_{\min,j} = \min_{i \in \Pi_p} x_{i,j}$, and $y_{p,j} = (x_{\max,j} + x_{\min,j})/2$ $(j = 1, 2)$. Then for $i \in \Pi_p$ and $j \in \{1, 2\}$,

$$\begin{aligned}
|2x_{i,j} - 2y_{p,j}| &= |2x_{i,j} - (x_{\max,j} + x_{\min,j})| \\
&\leq |x_{i,j} - x_{\min,j}| + |x_{i,j} - x_{\max,j}| && \text{(triangle inequality)} \\
&= x_{\max,j} - x_{\min,j} && (x_{\min,j} \leq x_{i,j} \leq x_{\max,j}) \\
&= x_{\alpha,j} - x_{\beta,j} && (\alpha = \operatorname{argmax}_{a \in \Pi_p} x_{a,j}, \beta = \operatorname{argmin}_{b \in \Pi_p} x_{b,j}) \\
&\leq \|x_\alpha - x_\beta\|_\infty \leq \varepsilon. && (\alpha, \beta \in \Pi_p)
\end{aligned}$$

It follows that $\|2x_i - 2y_p\|_\infty \leq \varepsilon$, thus $2y_1, \dots, 2y_k$ is a (k, ε)-cover of $2x_1, \dots, 2x_n$.

(ii ⇒ iii): Let $y_1, \dots, y_k \in \mathbb{R}^2$ be a (k, ε)-cover of $2x_1, \dots, 2x_n$. Partition $\{1, \dots, n\}$ into $\Pi_1, \dots, \Pi_k$ by grouping points according to the nearest covering point (break ties arbitrarily). Then for $i, j \in \Pi_p$, $\{i, j\} \in E$ of the unit square graph, since

$$\|x_i - x_j\|_\infty = \frac{1}{2}\|2x_i - 2x_j\|_\infty \leq \frac{1}{2}\left(\|2x_i - y_p\|_\infty + \|y_p - 2x_j\|_\infty\right) \leq \frac{1}{2}(\varepsilon + \varepsilon) = \varepsilon.$$

Thus $\Pi_1, \dots, \Pi_k$ is a clique partition of size k.

(iii ⇒ i): Let Π be a clique partition of size k. Then for $i, j \in \Pi_p$, $\{i, j\} \in E$, and so $\|x_i - x_j\|_\infty \leq \varepsilon$. Thus Π is a (k, ε)-partition. □

[5]Switching to uniform distance will often change the graph, but there may be *other* points defining the same graph. Consider "unit diamond graphs" (L_1 distance), which are "rotated" unit square graphs.

6.3 Complexity of uniform point partition

In this section, I show that `UPP` is $\mathcal{NP}$-complete.[6] The main part of the proof is a reduction from the restricted Boolean satisfiability problem (`xSAT`, Problem 3.56) to `UPP`. I conclude this section by commenting on various generalisations.

Reduction overview

The idea of the `UPP` instance is to build a collection of points mirroring the structure of the bipartite variable–clause incidence graph (Definition 3.54) of the `xSAT` instance.

To each variable vertex, clause vertex, and edge corresponds a collection of points. The points for each variable vertex can be partitioned in one of two configurations, based on the value of the variable in a truth assignment. Each determines the partitioning of the edge points so as to propagate these values to the clauses. The allowed number of groups is carefully controlled so that there are enough to include the points of each clause vertex if and only if some variable satisfies that clause in the assignment.

I detail the construction sketched above in several steps, below. I prove equivalence to the original `xSAT` instance in Theorem 6.34.

Reduction step 1: Lay out the graph on a grid

Due to the restrictions on the `xSAT` instance, the bipartite variable–clause incidence graph is planar with maximum degree three. Therefore there exists a graph layout where (1) the vertices are positioned at integer coordinates, and (2) the edges comprise horizontal and vertical segments between adjacent pairs of integer coordinates (Valiant, 1981, §IV).

Moreover, such a *(planar, rectilinear, integer) grid layout* can be constructed in polynomial time (see, e.g., Valiant, 1981; Liu et al., 1998).[7] Figure 6.4 shows three examples.

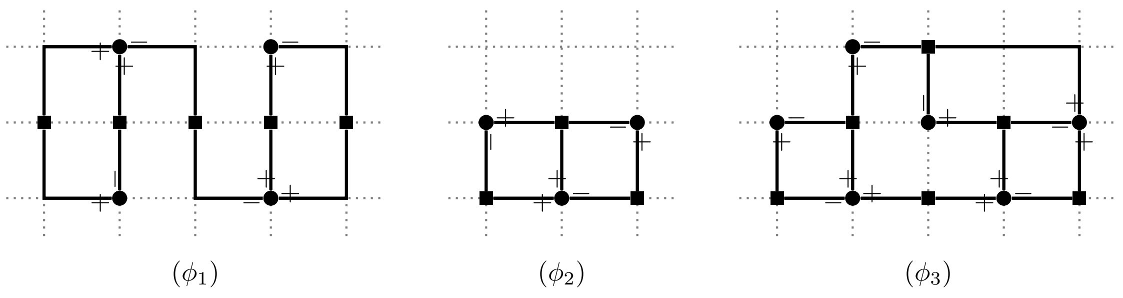


Figure 6.4: Example planar, rectilinear, integer grid layouts of the bipartite variable–clause incidence graphs from Figure 3.8. These layouts are computed by hand—those produced by standard algorithms (e.g., Valiant, 1981; Liu et al., 1998) may be larger.

[6] By Theorem 6.27, it follows that `UPC` and `usgCP` are also $\mathcal{NP}$-complete. I use `UPP` to streamline the presentation by de-emphasising the graph structure and the choice of covering points.

[7] There is no requirement to produce an "optimal" layout—just a polynomial-time computable layout.

Reduction step 2: Divide the layout into tiles

The grid layout serves as a blueprint for a `UPP` instance: it governs how the points corresponding to each variable vertex, clause vertex, and edge are arranged in the plane. The idea is to conceptually divide the plane into unit square *tiles,* with one tile for each coordinate of the integer grid occupied by a vertex or edge in the grid layout. The tile divisions for the running examples are shown in Figure 6.5.

Due to the restrictions on the `xSAT` instance, any tile division uses just forty distinct *tile types* (just nine up to rotation and reflection). In particular, there are straight edge segments and corner edge segments, plus clause and variable vertices with two or three edges in any direction. Moreover, for variable vertices, exactly one direction corresponds to a negative occurrence. Figure 6.6 enumerates these types.

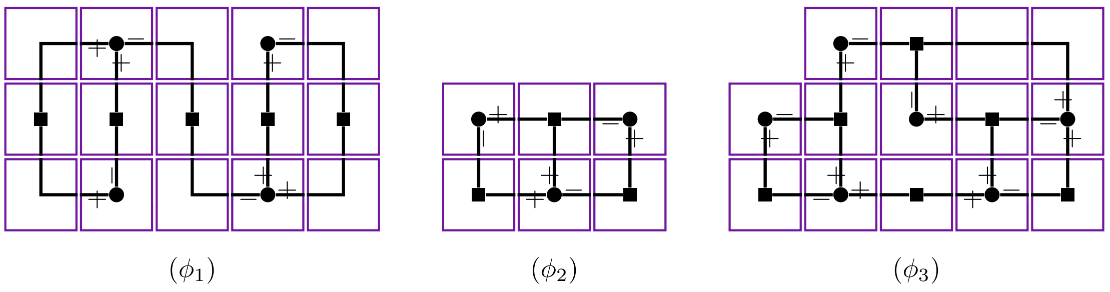


Figure 6.5: Example division of the grid layouts from Figure 6.4 into tiles.

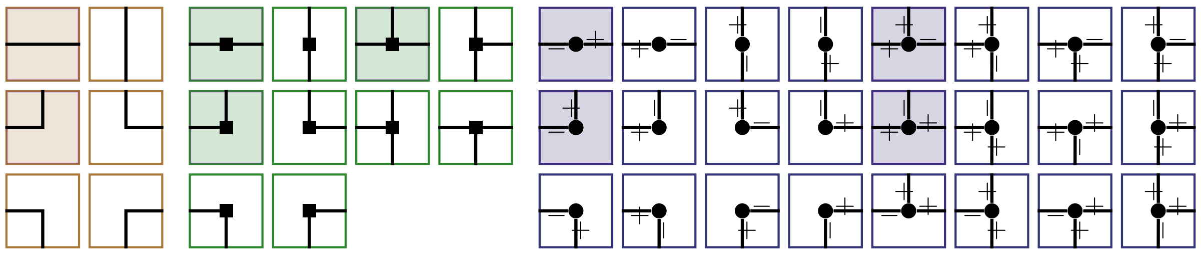


Figure 6.6: Just forty tile types suffice to construct any tile division of a grid layout. Up to rotation and reflection, just nine distinct types (for example, those highlighted) suffice.

Reduction step 3: Populate the instance with points

The points of the `UPP` instance are of two kinds (described in more detail below):

1. *boundary points* between neighbouring pairs of tiles; and
2. *interior points* within each tile in a specific arrangement depending on the tile type.

The boundary point can be grouped with interior points of one or the other neighbouring tile. In this way, boundary points couple the choice of how to partition the interior points of neighbouring tiles, creating the global constraint that corresponds to satisfiability.

Reduction step 3a: Boundary points between neighbouring tiles

There is one boundary point at the midpoint of the boundary between each pair of neighbouring tiles. A pair of *neighbouring tiles* is one for which there is an edge crossing the boundary. It is not sufficient for the tiles to be adjacent. Figure 6.7 clarifies this distinction using the running examples.

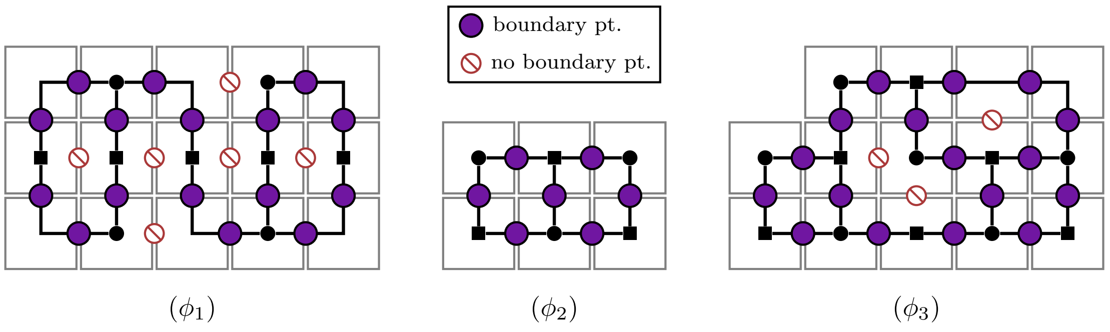


Figure 6.7: Example of the placement of boundary points between neighbouring tiles. Boundary points are not placed between adjacent tiles if no edge crosses this tile boundary.

Reduction step 3b: Interior points for variable tiles

Due to the restrictions on the `xSAT` instance, each variable tile has one *negative boundary point* and one or two *positive boundary point(s)* (corresponding to the occurrences of the variable). The interior point arrangement for variable tiles ensures that, with a small number of groups, it is possible to partition the interior points such that *either* (1) the positive boundary point(s) are included; *or* (2) the negative boundary point is included; but *not both.* Then, the choice of which boundary point(s) to include corresponds to the value of the variable in a truth assignment.

Table 6.8 shows arrangements of interior points for each type of variable tile (up to rotation and reflection). The exclusivity condition is formalised in Lemma 6.28.

Lemma 6.28. *Consider an interior and boundary point arrangement from Table 6.8, or a rectilinear rotation or reflection of such an arrangement. Let $k \in \{2, 3\}$ be the allocated number of groups, and let $\varepsilon \in \mathbb{R}^+$ be the scale.*

(i) There is no (r, ε)-partition of the interior points if $r < k$.

(ii) For any (k, ε)-partition of the interior points, the negative boundary point is within uniform distance ε of all points in some group, if and only if (neither of) the positive boundary point(s) are within uniform distance ε of all points in any group.

Proof. It suffices to consider the arrangements in Table 6.8 because the uniform distance is invariant to rectilinear rotation and reflection. The claims are then verified by exhaustive consideration of all possible partitions of the interior points into at most k groups. □

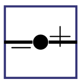
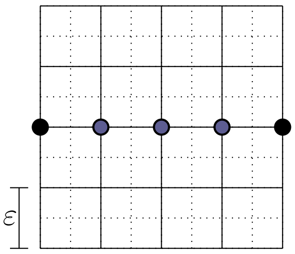
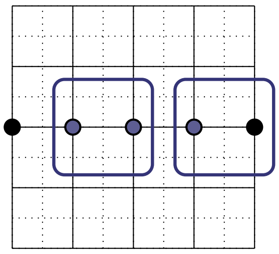
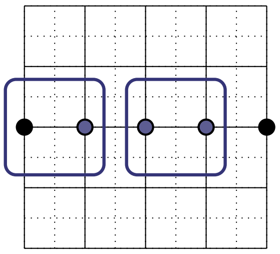
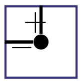
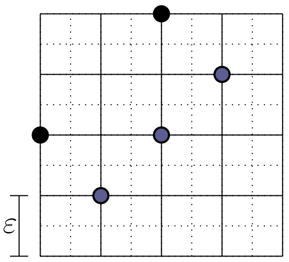
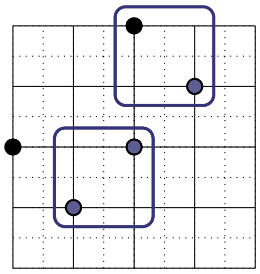
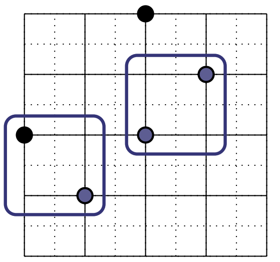
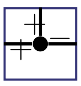
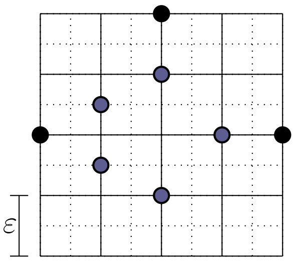
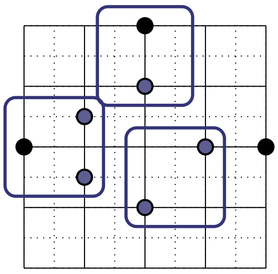
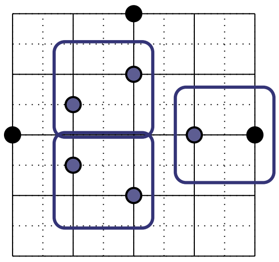
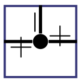
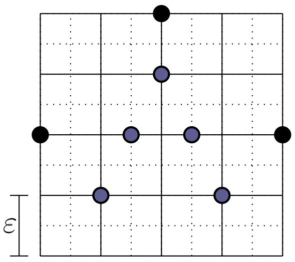
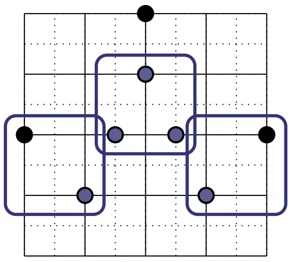
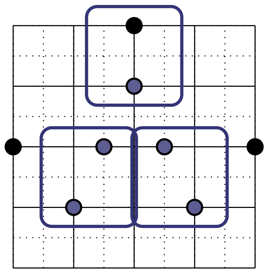
Type	n	k	Arrangement	True partition	False partition
	3	2			
	3	2			
	5	3			
	5	3			

Table 6.8: Arrangement of interior points for variable tiles. The number n represents the number of interior points (coloured). Nearby boundary points are also shown (black). The number k represents the number of groups allocated to the tile during the reduction.

Remark 6.29. The true and false partitions indicated in the table are used in the forward direction of the reduction proof to construct a partition given a satisfying assignment. In the converse direction, one must consider other possible partitions. With the exception of the arrangements in the fourth row, if the positive (negative) boundary point(s) are to be grouped with the interior points of the tile, then the true (false) partition is the only possible partition of the interior points using k groups. For the final row, there are some other partitions where only one true boundary point is included, but, as suffices for the reduction, there are no partitions including both positive and negative boundary points.

Reduction step 3c: Interior points for edge tiles

Once the partition of a variable tile includes either the positive boundary point(s) or negative boundary point, the role of the edge tiles is to propagate this choice to the clause in which the variable occurs. This information can be communicated with simple arrangements of points as summarised in Table 6.9. The arrangements are identical to those for variables with two occurrences. As such the exclusivity condition in Lemma 6.30 is a special case of Lemma 6.28.

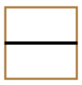
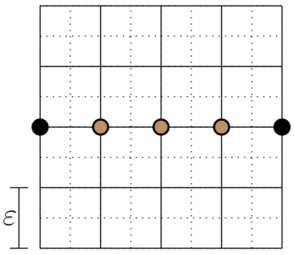
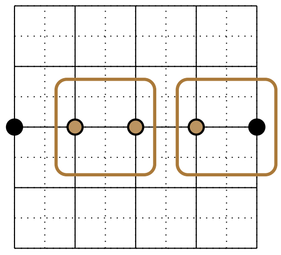
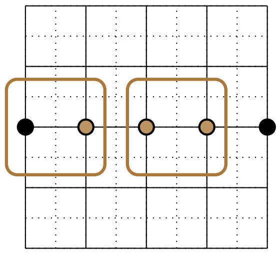
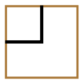
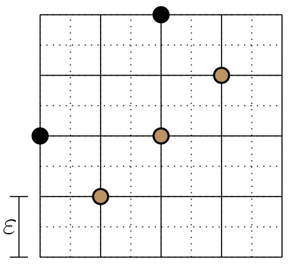
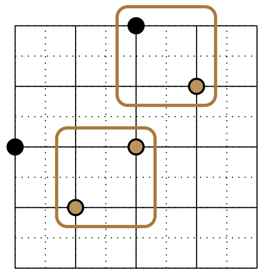
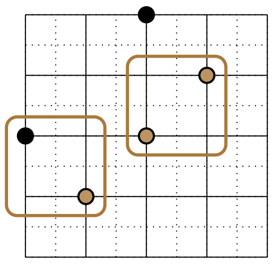
Type	n	k	Arrangement	Partition 1	Partition 2
	3	2			
	3	2			

Table 6.9: Arrangement of interior points for edge tiles. The number n represents the number of interior points (coloured). The boundary points are also shown (black). The number k represents the number of groups allocated to the tile during the reduction.

Lemma 6.30. *Consider an interior and boundary point arrangement from Table 6.9, or a rectilinear rotation or reflection of such an arrangement. Let $\varepsilon \in \mathbb{R}^+$ be the scale.*

(i) There is no (r, ε)-partition of the interior points if $r < 2$.

(ii) For any $(2, \varepsilon)$-partition of the interior points, either boundary point is within uniform distance ε of all points in some group, if and only if the other boundary point is not within uniform distance ε of all points in any group.

Proof. Special case of Lemma 6.28. □

Remark 6.31. The partitions indicated in Table 6.9 are the only possible $(2, \varepsilon)$-partitions of the interior points.

Reduction step 3d: Interior points for clause tiles

The interior points of the clause tile can be partitioned if and only if one of the boundary points is included by a neighbouring tile. Table 6.10 outlines such arrangements, and Lemma 6.32 formalises the desired property.

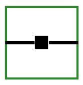
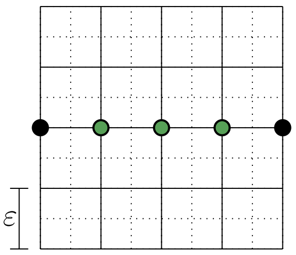
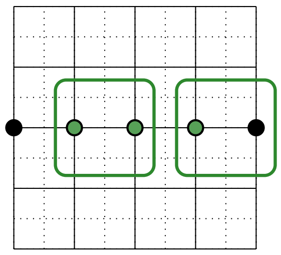
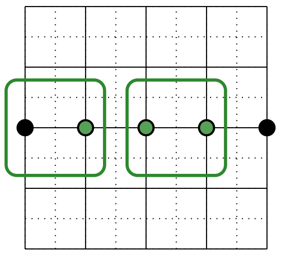
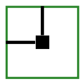
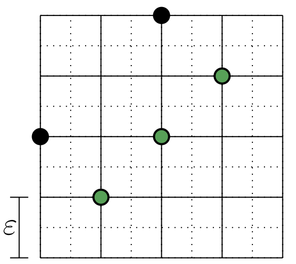
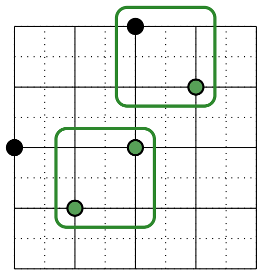
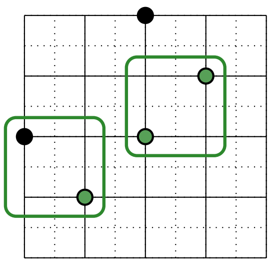
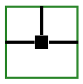
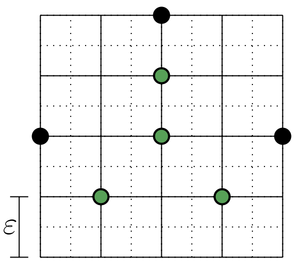
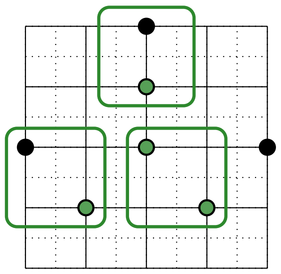
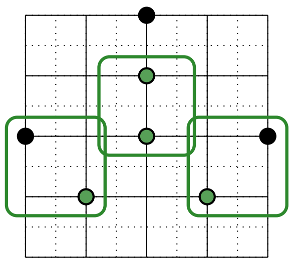
Type	n	k	Arrangement	Possible partitions
	3	2		 
	3	2		 
	4	3		 

Table 6.10: Arrangement of interior points for clause tiles. The number n represents the number of interior points (coloured). The boundary points are also shown (black). The number k represents the number of groups allocated to the tile during the reduction.

Lemma 6.32. *Consider an interior and boundary point arrangement from Table 6.10, or a rectilinear rotation or reflection of such an arrangement. Let $k \in \{2, 3\}$ be the allocated number of groups, and let $\varepsilon \in \mathbb{R}^+$ be the scale.*

(i) There is no (r, ε)-partition of the interior points if $r < k$.

(ii) For any (k, ε)-partition of the interior points, there is at least one boundary point that is not within uniform distance ε of all points in any group.

Proof. Following Lemma 6.28, the conditions can be checked exhaustively. □

Remark 6.33. Table 6.10 shows the only possible (k, ε)-partitions of the interior points, except in the third row, where a reflected version of the left partition is also possible.

Reduction step 4: Set the scale and number of groups

The reduction works at any (polynomial-time computable) scale—simply rescale the points. For concreteness, set the diameter to 1/4, giving each tile unit width. For the number of groups, total the allocations for the interior points of each tile (k in Tables 6.8, 6.9, and 6.10). This completes the construction. Figure 6.11 shows the full UPP instances for the running examples (cf. Examples 3.48 and 3.57 and Figures 3.8, 6.4, 6.5 and 6.7).

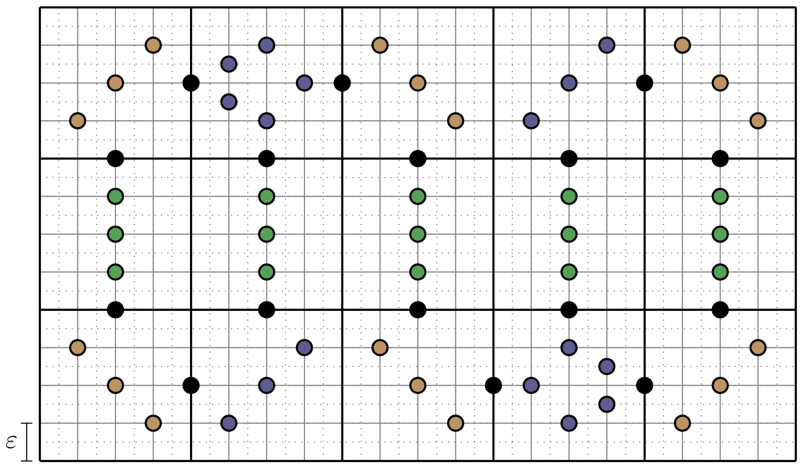
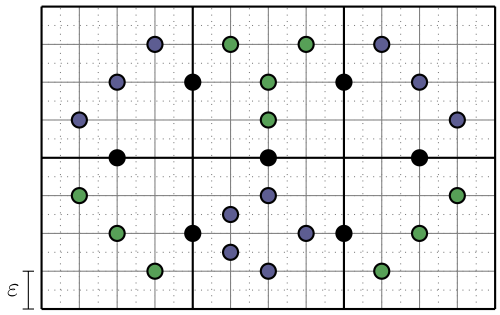
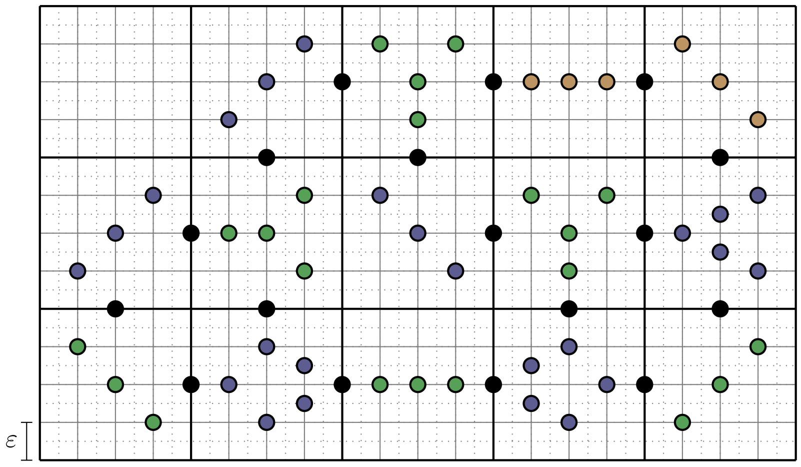
Formula	n	k	Source points
ϕ_1	65	32	
ϕ_2	28	14	
ϕ_3	68	34	

Table 6.11: Full examples of the reduction from `xSAT` to `UPP`, based on `xSAT` instances described in Examples 3.48 and 3.57. Exercise: is there a (k, ε)-partition in each case?

Proof of the reduction

I formally prove that the above construction indeed reduces `xSAT` to `UPP`.

Theorem 6.34 (`xSAT` $\xrightarrow{\mathcal{P}}$ `UPP`)**.**

Proof. Given an instance of `xSAT`, that is, a Boolean formula in conjunctive normal form ϕ with variables $v_1, \ldots, v_n$ and clauses $c_1 \wedge \cdots \wedge c_m$ satisfying the additional restrictions outlined in Definition 3.55, construct an instance of `UPP` as described above, namely, the points $x_1, \ldots, x_N$ (as described in step 3, the interior points from all tiles and the boundary points between them), a number of groups k (as described in step 4, the total allocated groups from all of the tiles), and an arbitrary diameter ε (such as 1/4, as described in step 4). Step 1 (grid layout) runs in polynomial time (Liu et al., 1998), and the remaining steps run in linear or constant time.

It remains to show that the constructed instance of `UPP` is equivalent to the original `xSAT` instance. That is, ϕ is satisfiable if and only if there exists a (k, ε)-partition of $x_1, \ldots, x_N$.

($\Rightarrow$): Suppose ϕ is satisfiable. Let θ be a satisfying truth assignment. Produce a (k, ε)-partition as follows.

1. Partition the interior points of each variable tile using the true partition or false partition (Table 6.8), based on the value of this variable under θ. Include the boundary points as indicated.

2. Following the included boundary points through zero or more edge tiles to the clause tile, partition the interior points of each edge tile according to Table 6.9 such that the boundary point in the direction of the clause tile is included.

3. Since θ is a satisfying assignment, every clause tile is reached in this way at least once, and thus has at least one of its boundary points included in the groups described in steps (1) and (2). For each clause tile, partition the interior points according to Table 6.10, including the remaining boundary points, if any.

4. Following the included boundary points through zero or more edges back to a variable tile, partition the interior points of each edge tile according to Table 6.9 such that the boundary point in the direction of the variable tile is included.

The final step includes exactly the boundary points that were left out in step (1), since these are the edges for which the clause tiles were left to include the boundary point on that side in step (3). Thus, all tiles have their interior and boundary points included. The number of groups is exactly in accordance with the allocated number of groups per tile. Therefore, the constructed partition is a (k, ε)-partition, as required.

($\Leftarrow$): Suppose there is a (k, ε)-partition of the points. Since the interior points of each tile are separated from the tile boundaries by at least ε, no group can include interior points from two separate tiles. It follows that the interior points of each tile must be partitioned into their allocated number of groups: if the interior points of one tile were to use more than the allocated number of groups, then there would not be enough groups left to include all of the interior points of some other tile, by Lemmas 6.28(i), 6.30(i), and 6.32(i). Moreover, since the allocated number of groups leaves no space for the boundary points to have their own groups, each boundary point must be in a group with interior points from one of its neighbouring tiles.

Consider each clause tile. By Lemma 6.32(ii), there must be at least one boundary point that is included with the interior points of the *neighbouring* tile. Pick one such direction for each clause and use this to construct a satisfying assignment for ϕ as follows.

By Lemma 6.30(ii), each boundary point must be included with the interior points of the *next* edge tile in the sequence of zero or more edge tiles on the way to the variable. In turn, the boundary point at the variable tile must be included with the interior points of the variable tile. If this is a positive boundary point, set this variable "true" in a truth assignment θ, and if it is a negative boundary point, set the variable "false".

This uniquely defines θ for all variables reached in this way at least once—if a variable is reached twice, it must be to two positive boundary points, since by Lemma 6.28(ii), it is impossible for the partition to have both a negative and a positive boundary point included with the interior points of a variable tile.

Since some variables may not be reached by following the chosen edges, complete the definition of θ by assigning arbitrary truth values to such variables. These variables are not necessary to see that θ is a satisfying assignment for ϕ. □

$\mathcal{NP}$-completeness proof

Theorem 6.35 (`UPP` is $\mathcal{NP}$-complete)**.**

Proof. `UPP` $\in \mathcal{NP}$: a (k, ε)-partition of the points acts as a certificate. Such a partition can be verified in polynomial time by computing the pairwise uniform distances within each group. Since `xSAT` is $\mathcal{NP}$-complete by Theorem 3.58 and `xSAT` $\xrightarrow{\mathcal{P}}$ `UPP` by Theorem 6.34, `UPP` is $\mathcal{NP}$-complete. □

Corollary 6.36 (`UPC` and `usgCP` are both $\mathcal{NP}$-complete)**.**

Proof. By Theorem 6.27, `UPP` $\xrightarrow{\mathcal{P}}$ `UPC` $\xrightarrow{\mathcal{P}}$ `usgCP` $\xrightarrow{\mathcal{P}}$ `UPP`. It then follows by Theorem 6.35 that all three problems are in $\mathcal{NP}$ and, moreover, are $\mathcal{NP}$-complete. □

Complexity of generalisations beyond the plane

All three problems permit an obvious generalisation beyond $\mathbb{R}^2$ to $\mathbb{R}^p$ for any $p \geq 1$. For example, adopting the partitioning perspective, and with a suitable generalisation of the notion of (k, ε)-partition, consider the following problem:

Problem 6.37 (Uniform point partition in $\mathbb{R}^p$, $\texttt{UPP}^p$)**.** Let $p \in \mathbb{N}$. Given n points $x_1, \ldots, x_n \in \mathbb{R}^p$; a diameter $\varepsilon \in \mathbb{R}^+$; and some $k \in \mathbb{N}$, determine if there is a (k, ε)-partition of the points.

Corollary 6.38 (If $p \geq 2$, then $\texttt{UPP}^p$ is $\mathcal{NP}$-complete)**.**

Proof. $\texttt{UPP}^p \in \mathcal{NP}$ with a (k, ε)-partition as a polynomial-time verifiable certificate. Then for $p \geq 2$, $\texttt{UPP} \xrightarrow{\mathcal{P}} \texttt{UPP}^p$ by embedding the points in the plane into, for example, the first two dimensions of $\mathbb{R}^p$ (leaving the remaining components zero)—any (k, ε)-partition of the higher-dimensional points is also a (k, ε)-partition of the 2-dimensional points, and vice versa. □

Remark 6.39. By a p-dimensional generalisation of Theorem 6.27, p-dimensional generalisations of `usgCP` and `UPC` are also $\mathcal{NP}$-complete for $p \geq 2$.

Remark 6.40. In contrast, $\texttt{UPP}^1$ is easy. A minimal partition can be constructed using a greedy algorithm with runtime $\mathcal{O}(n \log n)$ (where n is the number of points): first, sort the n points by their value. Then, proceeding through the points in increasing order, if a point, x, is not yet grouped, group all points in the neighbourhood $[x, x + \varepsilon]$. Once all points have been processed in this manner, the resulting partition is minimal, and so a (k, ε)-partition exists if and only if there are at most k groups in the current partition.

6.4 Complexity of parametric approximate rank

In this section, I show that the following decision problem is $\mathcal{NP}$-complete.

Problem 6.41 (Parametric approximate rank, `PAR`)**.** Given a simple neural network architecture $\mathcal{A}_h^{n,m}$, a neural network parameter $w \in \mathcal{W}_h^{n,m}$, a uniform radius $\varepsilon \in \mathbb{R}^+$, and a maximum rank $r \in \mathbb{N}$, determine whether $\operatorname{prank}_\varepsilon(w) \leq r$.

Equivalently, deciding if a parameter is within some uniform distance of a bounded rank region is $\mathcal{NP}$-complete (Proposition 6.2(iv)). Moreover, it follows that computing the parametric approximate rank itself is $\mathcal{NP}$-hard (with the parametric approximate rank, one can immediately answer Problem 6.41).

The foundation for this result has been established in Sections 3.4 and 6.3. It remains only to reduce `UPC` (Problem 6.17) to `PAR` and to show that `PAR` $\in \mathcal{NP}$.

Reduction from uniform point covering

Theorem 6.42 shows a reduction based on the following approach. Given an instance of `UPC`, consider an architecture with one hidden unit per source point in the instance, one input unit, and one output unit. Construct a parameter using the coordinates of the source points as the incoming weights and biases for the hidden units. Actually, to avoid issues with zeros and lexicographic sign, first translate all of the source points well into the positive quadrant. Likewise, set the outgoing weights to some high positive value.

If there is a small cover of the source points, then the hidden units can be perturbed by a small amount so that they match up with the (translated) covering points. Since there are few covering points, many units can be merged in the perturbed parameter, so the original parameter has low parametric approximate rank.

Conversely, since all of the weights and biases are highly positive, then any nearby low-rank parameter must be due to the approximate mergeability of the units. Therefore if the parameter has low parametric approximate rank, there is a small cover of the translated points (and, in turn, the original points).

Theorem 6.42 (`UPC` $\xrightarrow{\mathcal{P}}$ `PAR`)**.**

Proof. Given an instance of `UPC`—that is, n source points in the plane, $x_1, \ldots, x_n \in \mathbb{R}^2$; a uniform radius $\varepsilon \in \mathbb{R}^+$; and some $k \in \mathbb{N}$, construct an instance of `PAR` with radius ε, maximum rank k, and parameter for architecture $\mathcal{A}_n^{1,1}$ constructed as follows.

1. Find the minimum coordinates amongst the source points. Define $x_{\min} \in \mathbb{R}^2$ as

$$x_{\min} = \left(\min_{i=1,\ldots,n} x_{i,1}, \min_{i=1,\ldots,n} x_{i,2} \right).$$

   Note that the minimisation runs over each dimension independently.

2. Define a translation $T : \mathbb{R}^2 \to \mathbb{R}^2$ such that

$$T(x) = x - x_{\min} + (2\varepsilon, 2\varepsilon).$$

Translate the source points $x_1, \ldots, x_n$ to $x'_1, \ldots, x'_n$ where $x'_i = T(x_i)$. Note (for later) that all components of the translated points are at least 2ε by step (1).

3. Construct the neural network parameter:

$$w = (a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n, 0) \in \mathcal{W}_n^{1,1} = \mathbb{R}^{3n+1}$$

where for $i = 1, \ldots, n$, $a_i = 2\varepsilon$, $b_i = x'_{i,1}$, and $c_i = x'_{i,2}$.

This construction requires only linear time. It remains to show that the constructed instance of PAR is equivalent to the given instance of UPC, that is, there exists a (k, ε)-cover of the source points if and only if the constructed parameter has $\operatorname{prank}_\varepsilon(w) \le k$.

($\Rightarrow$): Suppose there exists a (k, ε)-cover $y_1, \ldots, y_k$. Define $\kappa : \{1, \ldots, n\} \to \{1, \ldots, k\}$ such that x_i is nearest to $y_{\kappa(i)}$ (breaking ties arbitrarily). Then for $j = 1, \ldots, k$, define $y'_j = T(y_j)$, where T is the translation defined in step (2) above. Define the parameter

$$w^\star = (a^\star_1, \ldots, a^\star_n, b^\star_1, \ldots, b^\star_n, c^\star_1, \ldots, c^\star_n, 0) \in \mathcal{W}_n^{1,1}$$

where, for $i = 1, \ldots, n$, $a^\star_i = 2\varepsilon$, $b^\star_i = y'_{\kappa(i),1}$, and $c^\star_i = y'_{\kappa(i),2}$. Then $\operatorname{rank}(w^\star) \le k$, since there are at most k distinct incoming weight and bias vectors (namely $y'_1, \ldots, y'_k$). Moreover, $\|w - w^\star\|_\infty \le \varepsilon$: both have no output bias, and, for $i = 1, \ldots, n$, $a^\star_i = a_i$ and

$$\|(b_i, c_i) - (b^\star_i, c^\star_i)\|_\infty = \left\|x'_i - y'_{\kappa(i)}\right\|_\infty = \left\|T(x_i) - T(y_{\kappa(i)})\right\|_\infty = \left\|x_i - y_{\kappa(i)}\right\|_\infty \le \varepsilon$$

by the defining property of the cover. Therefore $\operatorname{prank}_\varepsilon(w) \le \operatorname{rank}(w^\star) \le k$.

($\Leftarrow$): Suppose $\operatorname{prank}_\varepsilon(w) \le k$, with $w^\star \in \bar{B}_\infty(w; \varepsilon)$ such that $\operatorname{rank}(w^\star) = r \le k$. It suffices to produce an (r, ε)-cover, which can always be extended to a (k, ε)-cover by adding $k - r$ arbitrary covering points. Now, in general, the only ways that $w^\star$ could have reduced rank compared to w are the following (cf. Algorithm 4.12).

1. Some b_i could be perturbed to zero to allow the corresponding unit to be removed.

2. Two units with (b_i, c_i) and (b_j, c_j) within 2ε could be perturbed to have identical weight and bias vectors, allowing them to be merged.

3. Two units with (b_i, c_i) and $-(b_j, c_j)$ within 2ε could be perturbed to have identically negative weight and bias vectors, again allowing them to be merged.

4. Some m units, merged through the above options (or $m = 1$) with total outgoing weight within $m\varepsilon$ of zero, could be perturbed to make the total zero.

But, by construction, all $a_i, b_i, c_i \geq 2\varepsilon > 0$, ruling out (1) and (3), and also (4) since any such total outgoing weight is $2m\varepsilon > m\varepsilon$. This leaves option (2) alone responsible.

Accordingly, there must be exactly r distinct vectors among the incoming weight and bias vectors of $w^\star$. Denote these vectors $y'_1, \ldots, y'_r$—they constitute an (r, ε)-cover of the incoming weight and bias vectors of w, $x'_1, \ldots, x'_n$ (as $w^\star \in \bar{B}_\infty(w; \varepsilon)$). To complete the proof invert T to produce an (r, ε)-cover of $x_1, \ldots, x_n$. □

$\mathcal{NP}$-completeness proof

Based on the above reduction, I prove that `PAR` is $\mathcal{NP}$-complete.

Theorem 6.43 (`PAR` is $\mathcal{NP}$-complete)**.**

Proof. First, `UPC` is $\mathcal{NP}$-complete (Corollary 6.36) and `UPC` $\xrightarrow{\mathcal{P}}$ `PAR` (Theorem 6.42). To show `PAR` $\in \mathcal{NP}$: given a parameter $w \in \mathcal{W}_h^{n,m}$, radius $\varepsilon \in \mathbb{R}^+$, and maximum rank $r \in \mathbb{N}$, use as a certificate an approximate partition (radius ε) of the vectors $\text{abs}_\text{lex}(b_i, c_i)$ for i with $\|b_i\|_\infty > \varepsilon$, such that when this partition stands in for the result of Algorithm 6.8 in Algorithm 6.10, the result is at most r. Such a certificate can be verified in polynomial time by checking the diameter of each group and then following Algorithm 6.10.

As to the validity of the certificate: if the instance has parametric approximate rank at most r then there is a low-rank parameter within its neighbourhood from which the certificate can be constructed by identifying the units that would be merged if the parameter were reduced. Conversely, if such a certificate exists, then, *à la* Correctness Theorem 6.12, a nearby parameter with rank at most r can be constructed (to produce the new incoming weight and bias vectors, apply the construction in Theorem 6.27).[8] □

Remark 6.44. Theorem 6.42 only uses architectures with a single input unit and a single output unit. `PAR` remains hard for architectures with *more* input or output units: extend the reduction to set additional outgoing and incoming weights to 0. Given p incoming weights, compare with `UPP`$^{p+1}$ (Problem 6.37, the $+1$ comes from counting biases).

Remark 6.45. While `UPP`^{1} is in $\mathcal{P}$ (Remark 6.40), it does *not* follow that a variant of `PAR` for an *unbiased* single-input simple neural networks is easy. In fact, this variant remains $\mathcal{NP}$-complete—a reduction from the known $\mathcal{NP}$-complete *subset sum* problem (Karp, 1972)[9] is (still) possible. Describing this reduction is beyond the scope of this chapter, but this underscores a "second layer" of complexity in `PAR`—one must seek an approximate partition that *jointly* maximises the numbers of units eliminated (1) by approximate merging *and* (2) with total outgoing weight near zero.

[8] Why not just use a nearby low-rank parameter itself as the certificate? An arbitrary nearby low-rank parameter is unsuitable because the components of the parameter could have unbounded description lengths (or even be uncomputable). The proof essentially establishes that there is always a nearby equally-low-rank parameter that can be (indirectly) described and verified in polynomial time.

[9] Karp (1972) studied the subset sum problem under the name "knapsack." It is a special case of the knapsack problem as described in, for example, Garey and Johnson (1979).

Chapter 7

Discussion

In this chapter, I briefly discuss the implications of my results for understanding deep learning, and promising directions for future work opened up by my investigation.

7.1 Degeneracy beyond simple neural networks

The basis for the algorithmic framework for analysing degeneracy in simple neural networks is the correspondence between *reducibility,* or, local redundancy, and *non-minimality,* or, global redundancy (cf. Section 2.4). Due to this equivalence, the rank can be efficiently computed as if by the repeated application of operations that eliminate local redundancy. All results in this thesis essentially stem from this framework.

The first and clearest presentation of the equivalence between local and global redundancy is due to (Sussmann, 1992; cf. Section 3.3 in this thesis), motivating the choice of the simple neural network setting in this thesis. However, having taken this first step, similar analyses can be conducted for any neural network architecture in which a similar equivalence holds. This should include more practical neural network architectures, including with multiple layers[1] and the rectified linear unit activation function.

Certain modifications will be necessary to account for the different conditions of reducibility. For example, in the case of the rectified linear unit activation function, rather than computing signs and absolute values, one must instead perform positive scale normalisation operations. I note that the operations of merging units with the same incoming weights and biases and eliminating units with zero outgoing weights are universal.

A sensible starting point for such future work would be the very general analysis of Vlačić and Bölcskei (2021), in terms of arbitrary connection graphs and based on the affine symmetries of various activation functions.

[1]For multi-layered architectures, the rank can be defined on a per-layer basis, summarised in a tuple. Likewise, it should be possible to decompose the minimisation problem layer-wise.

7.2 Degenerate geometry and deep learning

Perhaps due to the *a priori* atypicality of degenerate parameters, the field has largely worked under an implicit assumption that non-degenerate neural network geometry describes neural networks learned in practice. Degenerate neural network geometry has the potential to clarify several topics of current empirical and theoretical[2] investigation in the field. In particular, consider the following topics.

1. *Overparameterisation and memorisation.* It is routine practice to use neural network architectures with more than enough units to precisely memorise the examples in the data set. The generalisation question becomes, which of the available memorising parameters will be found by a learning algorithm? (cf. Zhang et al., 2017; 2020; 2021; Belkin et al., 2019; Belkin, 2021; Bartlett et al., 2021; Dar et al., 2021).

2. *Curvature of the loss landscape.* Empirical studies have recently investigated the curvature of the loss landscape, especially at the solutions found by standard deep learning algorithms, consistently finding many approximately flat directions[3] (as indicated by the Hessian eigenspectrum being largely concentrated near zero; Sagun et al., 2017; 2018; Chaudhari et al., 2017; 2019; Papyan, 2018; Ghorbani et al., 2019).

3. *Connectivity of the loss landscape.* Recent empirical studies have observed that low-lying regions of the loss landscape are often connected by (sometimes indirect) paths of low-loss parameters (Freeman and Bruna, 2017; Sagun et al., 2018; Draxler et al., 2018; Garipov et al., 2018; Entezari et al., 2021).

Observe that if a parameter implements a function that achieves low loss or memorises a data set, then any functionally equivalent parameter does so as well. Therefore, the so-called *critical locus* of memorising parameters, and the low-lying regions of the parameter space more broadly, can be characterised as a union of functional equivalence classes.

In the case of memorisation, once one passes the minimum number of units required to perfectly encode the data set, then the critical locus necessarily contains degenerate parameters. This means that the richer geometry of degenerate functional equivalence classes (cf. Section 5.2), is part of the structure of the critical locus (and degenerate parameters themselves are potential candidates for selection).

[2]Existing theory addresses these topics (see, e.g., Freeman and Bruna, 2017; Cooper, 2018; 2020; Nguyen et al., 2019; Nguyen, 2019; 2021; Brea et al., 2019; Kuditipudi et al., 2019; Venturi et al., 2020; Şimşek et al., 2021). Applying the perspective of degenerate neural network geometry to interpret this work—especially that based on unit pruning (Kuditipudi et al., 2019), permutation symmetries (Brea et al., 2019), and embedding (Şimşek et al., 2021)—is important future work.

[3]"Flat minima" have long been associated with reliable generalisation, based on the minimum description length principle (Hinton and van Camp, 1993; Hochreiter and Schmidhuber, 1994; 1997a) and Bayesian statistical arguments (Hochreiter and Schmidhuber, 1997a; Langford and Caruana, 2001).

Moreover, parameters with low parametric approximate rank would make appealing memorising solutions. Such parameters implement similar functions to those of their nearby low-rank parameters, which may have desirable generalisation properties due to parsimony. In particular, the additional complexity in the solution could serve to drive the loss to zero on the observed examples without largely changing the function (cf. the "simple-plus-spiky" decomposition posited by Bartlett et al., 2021, §7).

Conversely, suppose that degenerate parameters lie (approximately) within the low-lying regions. Then their richer functional equivalence classes (as a union of manifolds, Theorem 5.13) would explain (approximately) flat directions, and Theorem 5.25 would similarly imply (approximate) connectivity of certain low-lying regions.

Consider the following concrete example in the simple setting. A non-degenerate parameter with low parametric approximate rank and low loss has a bounded-loss path to a nearby low-rank parameter. This low-rank parameter has a piecewise linear path connected and fixed-loss functional equivalence class innervating the parameter space, including approaching all transformed versions of the original parameter (cf. Corollary 6.7).

No individual functional equivalence class describes the low-lying region. In practice, observed low-loss paths appear to cut across multiple functional equivalence classes (Garipov et al., 2018). This suggests a need for future work on *partial neural network geometry,* studying classes of parameters with equal outputs on a (finite) subset of inputs.

Parametric instability

A lesson to be taken from degenerate neural network geometry is that functional equivalence does *not* imply that parameters are similar in other respects. This lesson arises in prior work studying the classification of degenerate critical points (Fukumizu and Amari, 2000; Fukumizu et al., 2019; cf. Section 2.5, above), and recurs in this thesis. Theorem 5.13 reveals that the degenerate functional equivalence class is a non-manifold whose dimension varies at different points. Example 6.5 demonstrates equivalent degenerate parameters with varying degrees of approximate degeneracy (proximity to even *more* degenerate parameters). Simple examples (cf. Table 2.7) show that parametric norms vary for equivalent degenerate parameters. These differences are consequential for learning (which operates by local search) or in any attempt to judge neural networks by the size, flatness, or compressibility of their parameters.

Yet this lesson is not conveyed by non-degenerate parameters, for which there is no local variation,[4] and for which the only symmetries are linear isometries (cf. Corollary 6.7). This suggests that, to account for possible degeneracy, one should base metrics on the whole functional equivalence class, a canonical representative, or the function itself.

[4]The non-degenerate functional equivalence class is discrete for simple neural networks (Theorem 3.39). With the rectified linear unit activation function, positive scaling allows continuous variation. It has been noted that various notions of flatness, for example, are sensitive to scaling (Dinh et al., 2017).

7.3 Measuring degeneracy

A key motivation for studying degeneracy in neural networks is that it provides a natural measure of the true complexity of a neural network, as distinct from problematic measures such as the mere number of hidden units available.

The rank is primarily a theoretical measure of degeneracy. It serves as an ideal basis on which to develop the theoretical results in Chapters 5 and 6, however, it is of limited suitability as a practical measure of neural network complexity (if for no other reason than limitations in numerical precision preventing the observation of identically equal, negative, or zero weights and biases in a practical setting).

The parametric approximate rank is a natural attempt to relax the precise constraints of the rank and could serve as a meaningful practical measure of neural network complexity. However, Chapter 6 shows that while it is easy to verify degeneracy given a nearby low-rank parameter, computing the parametric approximate rank itself is intractable.

There is still hope for developing an effective (parametric approximate) rank-based measure of neural network complexity. First, $\mathcal{NP}$-hardness does not preclude the development of fast approximation algorithms that can produce effective bounds, and effective bounds are still useful as a one-sided measure of complexity. Algorithm 6.10 provides one such algorithm (albeit with superlinear runtime). Chapter 6, especially the discussion of related problems, connects this problem to the substantial literature on approximation algorithms for hard problems (see, e.g., Garey and Johnson, 1979, §6, as a starting point). Future work can develop more efficient and effective bounding algorithms.

Second, the difficulty of computing the parametric approximate rank for *all* instances does not necessarily imply the difficulty of computing the parametric approximate rank in *typical* instances. The reductions in Chapter 6 essentially construct pathological parameters, poised between nearby regions of low-rank parameters just such that choosing the optimal direction in which to perturb the parameter involves solving a highly complex combinatorial optimisation problem equivalent to (a hard instance of) Boolean satisfiability. It is far from clear that typical learned parameters have this character.[5] Consider an analogy to clustering problems such as k-means (cf. Remark 6.19), well known to be $\mathcal{NP}$-complete in general, but for which near-optimal solutions can often be found efficiently for typical instances (Daniely et al., 2012; though see also Ben-David, 2015).

Having proved that computing the parametric approximate rank precisely in the most difficult instances is as hard as Boolean satisfiability, this thesis leaves as an open problem the difficulty of effectively approximating the parametric approximate rank of typical neural network parameters encountered in practice.

[5]Degenerate parameters are themselves, *a priori,* atypical (cf. Section 2.5 and Remark 3.45), and approximately degenerate parameters may or may not arise typically in practice. Here I claim that even among approximately degenerate parameters, *really* hard instances of `PAR` may be atypical.

Chapter 8

Conclusion

It is well known that neural network parameters do not uniquely implement functions. By exchanging unit weights, or through a small number of additional operations depending on the activation function, one can generate many distinct but functionally equivalent neural network parameters. For almost all neural networks, such operations exhaustively describe the *functional equivalence class.* For certain *degenerate* neural networks, there are additional parameters implementing the same function. Since some of these implementations correspond to smaller networks, studying such structural degeneracy has the potential to clarify the nature of neural network complexity.

In this thesis, I comprehensively investigate structural degeneracy in single-hidden-layer biased hyperbolic tangent networks. While the degenerate case is far more complex than the non-degenerate case (because there are additional possibilities to consider), it is still possible to make progress with an algorithmic approach. Based on Sussmann's characterisation of degeneracy in terms of conditions of *reducibility* (Sussmann, 1992), I develop an algorithmic framework for analysing and measuring degeneracy, including the notion of neural network *rank* (the minimum number of hidden units required to implement a given function). This algorithmic framework leads to a characterisation of the complete functional equivalence class in the degenerate case. Moreover, I develop a measure of *approximate* degeneracy based on proximity to low-rank parameters.

Despite having measure zero, the set of degenerate parameters has a rich internal structure. I study the basic properties of sets of bounded-rank parameters (which are closed and algebraic), bounded-rank functions (which are highly non-convex), the degenerate functional equivalence classes themselves (which are piecewise linear path connected), and the neighbourhoods of low-rank regions of the parameter space (which have a highly complex structure, in that determining membership therein is $\mathcal{NP}$-complete).

This thesis lays a foundation for future work developing efficient measures of approximate degeneracy, for empirically investigating the prevalence of approximate degeneracy in learned neural networks, and for clarifying the role of degeneracy in neural network learning more broadly.

References

Nur Ahmed and Muntasir Wahed. The de-democratization of AI: Deep learning and the compute divide in artificial intelligence research. Working paper, **2020**. Preprint arXiv:2010.15581 [cs.CY]. Cited on pages 1 and 10.

Hirotugu Akaike. Information theory and an extension of the maximum likelihood principle. In *2nd International Symposium on Information Theory*, pages 267–281. Akadémiai Kiadó, **1973**. Reprinted by Springer in 1998. Access via Crossref. Cited on page 11.

Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, **1974**. Access via Crossref. Cited on page 11.

Francesca Albertini and Eduardo D. Sontag. For neural networks, function determines form. Technical Report SYCON-92-03, Rutgers Center for Systems and Control, **1992**. Expanded version of Albertini and Sontag (1993a). Cited on page 18.

Francesca Albertini and Eduardo D. Sontag. For neural networks, function determines form. *Neural Networks*, 6(7):975–990, **1993**a. Access via Crossref. Cited on pages 18 and 114.

Francesca Albertini and Eduardo D. Sontag. Identifiability of discrete-time neural networks. In *Proceedings of the European Control Conference 1993*, volume 2, pages 460–465. European Control Association, **1993**b. Access via Francesca Albertini. Cited on page 18.

Francesca Albertini and Eduardo D. Sontag. Uniqueness of weights for recurrent nets. In *Systems and Networks: Mathematical Theory and Applications: Proceedings of the International Symposium MTNS 1993*, volume II, pages 599–602. Akademie Verlag, **1993**c. Access via Francesca Albertini or via Eduardo D. Sontag. See also extended version, access via Eduardo D. Sontag. Cited on page 18.

Francesca Albertini, Eduardo D. Sontag, and Vincent Maillot. Uniqueness of weights for neural networks. In *Artificial Neural Networks for Speech and Vision*, pages 113–125. Chapman & Hall, London, **1993**. Proceedings of a workshop held at Rutgers University in 1992. Access via Eduardo D. Sontag. Cited on pages 17, 18, and 23.

Shun-ichi Amari, Hyeyoung Park, and Tomoko Ozeki. Singularities affect dynamics of learning in neuromanifolds. *Neural Computation*, 18(5):1007–1065, **2006**. Access via Crossref. Cited on page 25.

Shun-ichi Amari, Tomoko Ozeki, Ryo Karakida, Yuki Yoshida, and Masato Okada. Dynamics of learning in MLP: Natural gradient and singularity revisited. *Neural Computation*, 30(1):1–33, **2017**. Access via Crossref. Cited on page 25.

Dario Amodei and Danny Hernandez. AI and compute. Blog post, **2018**. Access via OpenAI. Cited on page 9.

Sheldon Axler. *Linear Algebra Done Right*. Springer, third edition, **2015**. Access via Crossref. Cited on pages 28 and 55.

Caglar Aytekin, Francesco Cricri, and Emre Aksu. Compressibility loss for neural network weights. **2019**. Preprint arXiv:1905.01044 [cs.LG]. Cited on page 55.

Pierre F. Baldi and Kurt Hornik. Learning in linear neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(4):837–858, **1995**. Access via Crossref. Cited on page 13.

Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, **1993**. Access via Crossref. Cited on pages 2, 17, 141, 152, and 155.

Peter L. Bartlett, Andrea Montanari, and Alexander Rakhlin. Deep learning: A statistical viewpoint. *Acta Numerica*, 30:87–201, **2021**. Access via Crossref. Cited on pages 110 and 111.

Mikhail Belkin. Fit without fear: Remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248, **2021**. Access via Crossref. Cited on page 110.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, **2019**. Access via Crossref. Cited on pages 11 and 110.

Shai Ben-David. Computational feasibility of clustering under clusterability assumptions. **2015**. Preprint arXiv:1501.00437 [cs.CC]. Cited on page 112.

Piotr Berman, Alex D. Scott, and Marek Karpinski. Approximation hardness and satisfiability of bounded occurrence instances of SAT. Technical Report IHES/M/03/25, Institut des Hautes Études Scientifiques [Institute of Advanced Scientific Studies], **2003**. Access via CERN. Cited on page 50.

Elettra Bietti and Roxana Vatanparast. Data waste. *Harvard International Law Journal*, 61, **2020**. Access via Harvard ILJ. Online issue. Cited on page 10.

Or Biran and Courtenay Cotton. Explanation and justification in machine learning: A survey. Presented at the Twenty-sixth International Joint Conference on Artificial Intelligence, Explainable AI workshop, **2017**. Access via Or Biran. Cited on page 10.

Cristopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, **2006**. Cited on page 6.

Joachim Bona-Pellissier, François Bachoc, and François Malgouyres. Parameter identifiability of a deep feedforward ReLU neural network. **2021**. Preprint arXiv:2112.12982 [math.ST]. Cited on pages 17 and 18.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, **2018**. Access via Crossref. Cited on pages 7 and 16.

Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9:16, **2018**. Access via Crossref. Cited on page 8.

Johanni Brea, Berfin Şimşek, Bernd Illing, and Wulfram Gerstner. Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape. **2019**. Preprint arXiv:1907.02911 [cs.LG]. Cited on page 110.

Vinicius Andrade Brei. Machine learning in marketing: Overview, learning strategies, applications, and future developments. *Foundations and Trends in Marketing*, 14(3):173–236, **2020**. Access via Crossref. Cited on page 8.

Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, **2001**. Access via Crossref. Cited on page 6.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33*, pages

1877–1901. Curran Associates, **2020**. Access via NeurIPS. Cited on pages 1, 2, 8, and 9.

Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541. ACM, **2006**. Access via Crossref. Cited on pages 25, 141, 152, and 154.

Liam Carroll. *Phase Transitions in Neural Networks.* Master's thesis, School of Mathematics and Statistics, The University of Melbourne, **2021**. Access via Daniel Murfet. Cited on page 18.

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1721–1730. ACM, **2015**. Access via Crossref. Cited on page 10.

Stephen Casper, Xavier Boix, Vanessa D'Amario, Ling Guo, Martin Schrimpf, Kasper Vinken, and Gabriel Kreiman. Frivolous units: Wider networks are not really that wide. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 8, pages 6921–6929. AAAI Press, **2021**. Access via Crossref. Cited on page 25.

Márcia R. Cerioli, Luerbio Faria, Talita O. Ferreira, and Fábio Protti. On minimum clique partition and maximum independent set on unit disk graphs and penny graphs: Complexity and approximation. *Electronic Notes in Discrete Mathematics*, 18:73–79, **2004**. Access via Crossref. Cited on pages 50, 94, and 95.

Márcia R. Cerioli, Luerbio Faria, Talita O. Ferreira, and Fábio Protti. A note on maximum independent sets and minimum clique partitions in unit disk graphs and penny graphs: Complexity and approximation. *RAIRO: Theoretical Informatics and Applications*, 45(3):331–346, **2011**. Access via Crossref. Cited on pages 50, 94, and 95.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. In *5th International Conference on Learning Representations*. OpenReview, **2017**. Access via OpenReview. Cited on pages 11, 110, and 118.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics:*

Theory and Experiment, 2019(12):124018, **2019**. Access via Crossref. Updated version of Chaudhari et al. (2017). Cited on pages 11 and 110.

An Mei Chen and Robert Hecht-Nielsen. On the geometry of feedforward neural network weight spaces. In *Second International Conference on Artificial Neural Networks*, pages 1–4. IET, **1991**. Access via IEEE Xplore. Cited on pages 19 and 20.

An Mei Chen, Haw-minn Lu, and Robert Hecht-Nielsen. On the geometry of feedforward neural network error surfaces. *Neural Computation*, 5(6):910–927, **1993**. Access via Crossref. Cited on pages 2, 17, 19, and 20.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, **2018**. Access via Crossref. Cited on pages 55 and 118.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. **2020**. Preprint arXiv:1710.09282v9 [cs.LG]. Updated version of Cheng et al. (2018). Cited on page 55.

Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53(7):5113–5155, **2020**. Access via Crossref. Cited on page 55.

Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 2, pages 1237–1242. IJCAI, **2011**. Access via Crossref. Cited on pages 2 and 8.

Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, **1990**. Access via Crossref. Cited on page 94.

Donald L. Cohn. *Measure Theory*. Birkhäuser, second edition, **2013**. Access via Crossref. Cited on pages 45, 142, and 143.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, **1971**. Access via Crossref. Cited on pages 48 and 50.

Yaim Cooper. The loss landscape of overparameterized neural networks. **2018**. Preprint arXiv:1804.10200 [cs.LG]. Cited on page 110.

Yaim Cooper. The critical locus of overparameterized neural networks. **2020**. Preprint arXiv:2005.04210 [cs.LG]. Cited on page 110.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, third edition, **2009**. Cited on page 30.

Florent Cousseau, Tomoko Ozeki, and Shun-ichi Amari. Dynamics of learning in multilayer perceptrons near singularities. *IEEE Transactions on Neural Networks*, 19(8):1313–1328, **2008**. Access via Crossref. Cited on page 25.

David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, fourth edition, **2015**. Access via Crossref. Cited on page 45.

Harald Cramér. *Mathematical Methods of Statistics*. Princeton University Press, **1946**. Cited on page 11.

Kate Crawford. *The Atlas of AI: Power, Politics, and the Planetary Costs of Artificial Intelligence*. Yale University Press, **2021**. Cited on pages 1 and 10.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, **1989**. Access via Crossref. Cited on pages 2 and 17.

Amit Daniely, Nati Linial, and Michael Saks. Clustering is difficult only when it does not matter. **2012**. Preprint arXiv:1205.4891 [cs.LG]. Cited on page 112.

Yehuda Dar, Vidya Muthukumar, and Richard G. Baraniuk. A farewell to the bias-variance tradeoff? An overview of the theory of overparameterized machine learning. **2021**. Preprint arXiv:2109.02355 [stat.ML]. Cited on page 110.

Wim De Mulder, Steven Bethard, and Marie-Francine Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, **2015**. Access via Crossref. Cited on page 14.

Li Deng, Geoffrey E. Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE, **2013**. Access via Crossref. Cited on page 8.

Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, **1996**. Access via Crossref. Cited on page 6.

Christopher DiMattina and Kechen Zhang. How to modify a neural network gradually without changing its input-output functionality. *Neural Computation*, 22(1):1–47, **2010**. Access via Crossref. Cited on pages 17, 18, and 19.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1019–1028. PMLR, **2017**. Access via PMLR. Cited on pages 19 and 111.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1309–1318. PMLR, **2018**. Access via PMLR. Cited on pages 11 and 110.

Dominic B. Dwyer, Peter Falkai, and Nikolaos Koutsouleris. Machine learning approaches for clinical psychology and psychiatry. *Annual Review of Clinical Psychology*, 14(1):91–118, **2018**. Access via Crossref. Cited on page 8.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. **2021**. Preprint arXiv:2110.06296 [cs.LG]. Cited on page 110.

Charles Fefferman. Reconstructing a neural net from its output. *Revista Matemática Iberoamericana*, 10(3):507–555, **1994**. Access via Crossref. Cited on pages 2, 17, and 18.

Charles Fefferman and Scott Markel. Recovering a feed-forward net from its output. In *Advances in Neural Information Processing Systems 6*, pages 335–342. Morgan Kaufmann, **1993**. Access via NeurIPS. Cited on page 18.

Joseph Fourier. *Théorie Analytique de la Chaleur [Analytical Theory of Heat]*. F. Didot, Paris, **1822**. In French. Translated into English by Freeman (1878). Reprinted by Cambridge University Press in 2009. Access via Crossref. Cited on pages 17 and 120.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations*. OpenReview, **2019**. Access via OpenReview. Cited on page 11.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3259–3269. PMLR, **2020**. Access via PMLR. Cited on page 11.

Alexander Freeman. *The Analytical Theory of Heat*. Cambridge University Press, **1878**. Access via Crossref. English translation of Fourier (1822). Reprinted 2009. Cited on page 120.

C. Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization. In *5th International Conference on Learning Representations*. OpenReview, **2017**. Access via OpenReview. Cited on pages 11 and 110.

Kenji Fukumizu. A regularity condition of the information matrix of a multilayer perceptron network. *Neural Networks*, 9(5):871–879, **1996**. Access via Crossref. Cited on pages 11, 18, and 37.

Kenji Fukumizu and Shun-ichi Amari. Local minima and plateaus in hierarchical structures of multilayer perceptrons. *Neural Networks*, 13(3):317–327, **2000**. Access via Crossref. Cited on pages 25, 79, and 111.

Kenji Fukumizu, Shoichiro Yamaguchi, Yoh-ichi Mototake, and Mirai Tanaka. Semi-flat minima and saddle points by embedding neural networks to overparameterization. In *Advances in Neural Information Processing Systems 32*, pages 13868–13876. Curran Associates, **2019**. Access via NeurIPS. Cited on pages 25, 79, and 111.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, **1980**. Access via Crossref. Cited on page 7.

Jesus Garcia-Diaz, Jairo Sanchez-Hernandez, Ricardo Menchaca-Mendez, and Rolando Menchaca-Mendez. When a worse approximation factor gives better performance: A 3-approximation algorithm for the vertex k-center problem. *Journal of Heuristics*, 23(5):349–366, **2017**. Access via Crossref. Cited on page 93.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, **1979**. Cited on pages 46, 48, 108, and 112.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew G. Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *Advances in Neural Information Processing Systems 31*, pages 8789–8798. Curran Associates, **2018**. Access via NeurIPS. Cited on pages 11, 110, and 111.

Carl Friedrich Gauss. *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium [Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections]*. Perthes and Besser, Hamburg, **1809**. In Latin. As cited in Stewart (1995). Cited on page 16.

Carl Friedrich Gauss. *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae [Theory of the Combination of Observations Least Subject to Error]*. H. Dieterich, Göttingen, **1821**. First published in Latin in multiple parts starting in 1821. Reproduced in Latin and translated into English by Stewart (1995). Cited on pages 16 and 133.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via Hessian eigenvalue density. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2232–2241. PMLR, **2019**. Access via PMLR. Cited on pages 11 and 110.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. PMLR, **2011**. Access via PMLR. Cited on page 13.

Herman H. Goldstine and John von Neumann. Planning and coding of problems for an electronic computing instrument, part II, volume II. Technical report, The Institute for Advanced Study, Princeton, New Jersey, **1948**. Cited on page 30.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, **2016**. Cited on pages 1, 7, 9, 13, and 16.

Mary L. Gray and Siddharth Suri. *Ghost Work: How to Stop Silicon Valley from Building a New Global Underclass*. Eamon Dolan Books, **2019**. Cited on page 10.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1737–1746. PMLR, **2015**. Access via PMLR. Cited on page 33.

Katsuyuki Hagiwara, Naohiro Toda, and Shiro Usui. On the problem of applying AIC to determine the structure of a layered feedforward neural network. In *1993 International Joint Conference on Neural Networks*, volume 3, pages 2263–2266. IEEE, **1993**. Access via Crossref. Cited on page 11.

Paul R. Halmos. *Finite-Dimensional Vector Spaces*. Van Nostrand, Princeton, New Jersey, second edition, **1958**. Reprinted by Springer. Access via Crossref. Cited on pages 28 and 55.

Paul R. Halmos. *Naive Set Theory*. Van Nostrand, Princeton, New Jersey, **1960**. Reprinted by Springer in 1974. Access via Crossref. Cited on page 28.

Egbert Harzheim. *Ordered Sets*. Springer, **2005**. Access via Crossref. Cited on pages 28 and 30.

Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, **2017**. Access via Crossref. Cited on page 7.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second edition, **2009**. Access via Crossref. Cited on page 6.

Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, **1949**. Reprinted by Psychology Press in 2002. Access via Crossref. Cited on page 7.

Brent Hecht, Lauren Wilcox, Jeffrey P. Bigham, Johannes Schöning, Ehsan Hoque, Jason Ernst, Yonatan Bisk, Luigi De Russis, Lana Yarosh, Bushra Anjum, Danish Contractor, and Cathy Wu. It's time to do something: Mitigating the negative impacts of computing through a change to the peer review process. ACM Future of Computing Academy blog post, **2018**. Access via Crossref. Cited on page 9.

Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, volume 1, pages 593–605. IEEE, **1989**. Access via Crossref. Cited on page 17.

Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. North-Holland, Amsterdam, **1990**. Access via Crossref. Cited on pages 17, 19, and 20.

Danny Hernandez and Tom B. Brown. Measuring the algorithmic efficiency of neural networks. **2020**. Preprint arXiv:2005.04305 [cs.LG]. Cited on pages 9 and 10.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. **2017**. Preprint arXiv:1712.00409 [cs.LG]. Cited on page 11.

Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13. ACM, **1993**. Access via Crossref. Cited on pages 55 and 110.

Geoffrey E. Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. Presented at Twenty-eighth Conference on Neural Information Processing Systems, Deep Learning workshop, **2014**. Preprint arXiv:1503.02531 [stat.ML]. Cited on pages 25, 141, 152, and 154.

C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, **1962**. Access via Crossref. Cited on page 30.

Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in Neural Information Processing Systems 7*, pages 529–536. MIT Press, **1994**. Access via NeurIPS. Cited on page 110.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, **1997**a. Access via Crossref. Cited on page 110.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, **1997**b. Access via Crossref. Cited on page 14.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, **1989**. Access via Crossref. Cited on pages 2 and 17.

David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, **1959**. Access via Crossref. Cited on page 7.

David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, **1962**. Access via Crossref. Cited on page 7.

David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of Neurophysiology*, 28(2):229–289, **1965**. Access via Crossref. Cited on page 7.

Klaus Jansen and Haiko Müller. The minimum broadcast time problem for several processor networks. *Theoretical Computer Science*, 147(1-2):69–85, **1995**. Access via Crossref. Cited on page 50.

Michael I. Jordan and Tom M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, **2015**. Access via Crossref. Cited on pages 1, 6, and 9.

Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar. *Adversarial Machine Learning*. Cambridge University Press, **2019**. Access via Crossref. Cited on pages 8 and 9.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, **2021**. Access via Crossref. Cited on pages 1 and 8.

Paul C. Kainen, Věra Kůrková, Vladik Kreinovich, and Ongard Sirisaengtaksin. Uniqueness of network parametrization and faster learning. *Neural, Parallel & Scientific Computations*, 2(4):459–466, **1994**. Cited on page 18.

Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, **1972**. Access via Crossref. Cited on pages 48, 94, and 108.

Andrej Karpathy. Software 2.0. Blog post, **2017**. Access via Medium. Cited on pages 6 and 8.

Andrei N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung [Foundations of the Theory of Probability]*. Springer, **1933**. In German. Translated into English by Morrison (1950). Cited on pages 128 and 142.

Tjalling C. Koopmans. Identification problems in economic model construction. *Econometrica*, 17(2):125–144, **1949**. Access via Crossref. Cited on page 18.

Tjalling C. Koopmans and Olav Reiersol. The identification of structural characteristics. *The Annals of Mathematical Statistics*, 21(2):165–181, **1950**. Access via Crossref. Cited on page 18.

Erwin Kreyszig. *Introductory Functional Analysis with Applications*. Wiley, **1978**. Cited on page 28.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, **2012**. Access via NeurIPS. Cited on pages 2 and 8.

Věra Kůrková and Paul C. Kainen. Functionally equivalent feedforward neural networks. *Neural Computation*, 6(3):543–558, **1994**. Access via Crossref. Cited on pages 17, 18, 20, and 23.

Rohith Kuditipudi, Xiang Wang, Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Rong Ge, and Sanjeev Arora. Explaining landscape connectivity of low-cost solutions for multilayer nets. In *Advances in Neural Information Processing Systems 32*, pages 14601–14610. Curran Associates, **2019**. Access via NeurIPS. Cited on page 110.

Samuel Lalmuanawma, Jamal Hussain, and Lalrinfela Chhakchhuak. Applications of machine learning and artificial intelligence for Covid-19 (SARS-CoV-2) pandemic: A review. *Chaos, Solitons & Fractals*, 139:110059, **2020**. Access via Crossref. Cited on page 8.

John Langford and Rich Caruana. (Not) bounding the true error. In *Advances in Neural Information Processing Systems 14*, pages 809–816. MIT Press, **2001**. Access via NeurIPS. Cited on page 110.

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten ZIP code recognition. *Neural Computation*, 1(4):541–551, **1989**a. Access via Crossref. Cited on pages 7 and 14.

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan Kaufmann, **1989**b. Access via NeurIPS. Cited on pages 7 and 14.

Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, **2015**. Access via Crossref. Cited on pages 1 and 7.

Adrien-Marie Legendre. *Nouvelles Méthodes pour la Détermination des Orbites des Comètes [New Methods for the Determination of the Orbits of Comets]*. F. Didot, Paris, **1805**. In French. As cited in Stewart (1995). Cited on page 16.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, **1993**. Access via Crossref. Cited on pages 2 and 17.

Esther Levin, Naftali Tishby, and Sara A. Solla. A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574, **1990**. Access via Crossref. Cited on page 18.

Leonid A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii [Problems of Information Transmission]*, 9(3):115–116, **1973**. In Russian. Translated into English in Trakhtenbrot (1984). Cited on page 48.

Arthur Lewbel. The identification zoo: Meanings of identification in econometrics. *Journal of Economic Literature*, 57(4):835–903, **2019**. Access via Crossref. Cited on page 18.

Konstantinos G. Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. Machine learning in agriculture: A review. *Sensors*, 18(8):2674, **2018**. Access via Crossref. Cited on page 8.

David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, **1982**. Access via Crossref. Cited on page 50.

Timothy P. Lillicrap and Konrad P. Körding. What does it mean to understand a neural network? **2019**. Preprint arXiv:1907.06374 [cs.LG]. Cited on pages 7 and 11.

Tom L. Lindstrøm. *Spaces: An Introduction to Real Analysis*. American Mathematical Society, **2017**. Cited on page 28.

Zachary C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *ACM Queue*, 16(3):31–57, **2018**. Access via Crossref. Cited on pages 1 and 10.

Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. **2015**. Preprint arXiv:1506.00019 [cs.LG]. Cited on page 14.

Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, **2011**. Access via Crossref. Cited on page 8.

Yanpei Liu, Aurora Morgana, and Bruno Simeone. A linear algorithm for 2-bend embeddings of planar graphs in the two-dimensional grid. *Discrete Applied Mathematics*, 81(1-3):69–91, **1998**. Access via Crossref. Cited on pages 96 and 103.

David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, **2003**. Cited on page 6.

Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, **2012**. Access via Crossref. Cited on page 93.

Charles F. Manski. *Partial Identification of Probability Distributions*. Springer, **2003**. Access via Crossref. Cited on page 18.

Adam H. Marblestone, Greg Wayne, and Konrad P. Körding. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10:94, **2016**. Access via Crossref. Cited on page 7.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, **1943**. Access via Crossref. Cited on page 7.

Hrushikesh N. Mhaskar. Neural networks for optimal approximation of smooth and analytic functions. *Neural Computation*, 8(1):164–177, **1996**. Access via Crossref. Cited on pages 2, 17, 141, 152, and 155.

Tim Miller. "But why?" Understanding explainable artificial intelligence. *XRDS*, 25(3):20–25, **2019**a. Access via Crossref. Cited on page 10.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, **2019**b. Access via Crossref. Cited on page 10.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, expanded edition, **1988**. Cited on page 17.

Shira Mitchell, Eric Potash, Solon Barocas, Alexander D'Amour, and Kristian Lum. Algorithmic fairness: Choices, assumptions, and definitions. *Annual Review of Statistics and Its Application*, 8(1):141–163, **2021**. Access via Crossref. Cited on page 9.

Tom M. Mitchell. The discipline of machine learning. Technical Report CMU-ML-06-108, Carnegie Mellon University, Machine Learning Department, **2006**. Access via Tom M. Mitchell. Cited on pages 6 and 8.

Tom M. Mitchell. *Machine Learning*, chapter draft: Key Ideas in Machine Learning. McGraw-Hill, draft of forthcoming second edition, **2017**. Access via Tom M. Mitchell. Cited on page 6.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, **2015**. Access via Crossref. Cited on pages 1, 2, and 8.

Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Self-published, second edition, **2022**. Access via Christoph Molnar. Cited on pages 1 and 10.

Guido F. Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems 27*, pages 2924–2932. Curran Associates, **2014**. Access via NeurIPS. Cited on page 2.

Nathan Morrison. *Foundations of the Theory of Probability*. Chelsea Publishing Company, **1950**. English translation of Kolmogorov (1933). Cited on page 125.

Ghulam Mujtaba, Liyana Shuib, Ram Gopal Raj, Nahdia Majeed, and Mohammed Ali Al-Garadi. Email classification research trends: Review and open issues. *IEEE Access*, 5:9044–9064, **2017**. Access via Crossref. Cited on page 8.

James R. Munkres. *Topology: A First Course.* Prentice Hall, **1974**. Cited on page 45.

Noboru Murata. An integral representation of functions using three-layered networks and their approximation bounds. *Neural Networks*, 9(6):947–956, **1996**. Access via Crossref. Cited on pages 2, 17, and 155.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, **2012**. Cited on pages 6 and 24.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814. Omnipress, **2010**. Access via ICML. Cited on page 13.

Whitney K. Newey and Daniel McFadden. Large sample estimation and hypothesis testing. *Handbook of Econometrics*, 4:2111–2245, **1994**. Access via Crossref. Cited on page 11.

Quynh Nguyen. On connected sublevel sets in deep learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4790–4799. PMLR, **2019**. Access via PMLR. Cited on page 110.

Quynh Nguyen. A note on connectivity of sublevel sets in deep learning. **2021**. Preprint arXiv:2101.08576 [cs.LG]. Cited on page 110.

Quynh Nguyen, Mahesh Chandra Mukkamala, and Matthias Hein. On the loss landscape of a class of deep neural networks with no bad local valleys. In *7th International Conference on Learning Representations*. OpenReview, **2019**. Access via OpenReview. Cited on page 110.

Peter Norvig. Deep learning and understandability versus software engineering and verification. Talk presented at Silicon Valley Deep Learning Group, **2016**. Access via YouTube. Cited on page 6.

Mícheál Ó Searcóid. *Metric Spaces.* Springer, **2006**. Access via Crossref. Cited on page 28.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. **2019**. Preprint arXiv:1912.06680 [cs.LG]. Cited on page 8.

Vardan Papyan. The full spectrum of deepnet Hessians at scale: Dynamics with SGD training and sample size. **2018**. Preprint arXiv:1811.07062 [cs.LG]. Cited on pages 11 and 110.

Raja Parasuraman and Victor Riley. Humans and automation: Use, misuse, disuse, abuse. *Human Factors*, 39(2):230–253, **1997**. Access via Crossref. Cited on page 8.

David Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. **2021**. Preprint arXiv:2104.10350 [cs.LG]. Cited on page 10.

Amandalynne Paullada, Inioluwa Deborah Raji, Emily M. Bender, Emily Denton, and Alex Hanna. Data and its (dis)contents: A survey of dataset development and use in machine learning research. *Patterns*, 2(11):100336, **2021**. Access via Crossref. Cited on page 10.

Philipp Petersen, Mones Raslan, and Felix Voigtlaender. Topological properties of the set of functions generated by neural networks of fixed size. *Foundations of Computational Mathematics*, 21(2):375–444, **2021**. Access via Crossref. Cited on pages 66, 70, and 153.

Mary Phuong and Marcus Hutter. Formal algorithms for transformers. **2022**. Preprint arXiv:2207.09238 [cs.LG]. Cited on page 14.

Mary Phuong and Christoph H. Lampert. Functional vs. parametric equivalence of ReLU networks. In *8th International Conference on Learning Representations*. OpenReview, **2020**. Access via OpenReview. Cited on pages 2, 3, 17, 18, and 19.

Allan Pinkus. Weierstrass and approximation theory. *Journal of Approximation Theory*, 107(1):1–66, **2000**. Access via Crossref. Cited on page 17.

R. Piziak and P. L. Odell. Full rank factorization of matrices. *Mathematics Magazine*, 72(3):193–201, **1999**. Access via Crossref. Cited on page 55.

Adnan Qayyum, Junaid Qadir, Muhammad Bilal, and Ala Al-Fuqaha. Secure and robust machine learning for healthcare: A survey. *IEEE Reviews in Biomedical Engineering*, 14:156–180, **2020**. Access via Crossref. Cited on page 8.

Zhi-Yong Ran and Bao-Gang Hu. Parameter identifiability in statistical machine learning: A review. *Neural Computation*, 29(5):1151–1203, **2017**. Access via Crossref. Cited on page 18.

Amal Rannen Triki. *Function Norms for Neural Networks: Theory and Applications*. Ph.D. thesis, Faculty of Engineering Science, KU Leuven, **2020**. Access via KU Leuven Lirias. Cited on page 66.

Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, **2011**. Access via Crossref. Cited on page 6.

Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, **2017**. Access via Crossref. Cited on page 14.

Blake A. Richards, Timothy P. Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J. Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W. Lindsay, Ken Miller, Richard Naud, Christopher C. Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P. Körding. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, **2019**. Access via Crossref. Cited on pages 7 and 11.

Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, **1996**. Cited on page 6.

David Rolnick and Konrad P. Körding. Reverse-engineering deep ReLU networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8178–8187. PMLR, **2020**. Access via PMLR. Cited on page 18.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, **2019**. Access via Crossref. Cited on pages 1 and 10.

Cynthia Rudin and Kiri L. Wagstaff. Machine learning for science and society. *Machine Learning*, 95(1):1–9, **2014**. Access via Crossref. Cited on page 8.

Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, third edition, **1976**. Cited on page 17.

Stefan M. Rüger and Arnfried Ossen. The metric structure of weight space. *Neural Processing Letters*, 5(2):1–9, **1997**. Access via Crossref. Cited on pages 17, 19, 20, 72, and 73.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, fourth edition, **2021**. Cited on pages 6, 8, and 9.

Stuart J. Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, **2015**. Access via Crossref. Cited on page 9.

Levent Sagun, Léon Bottou, and Yann LeCun. Eigenvalues of the Hessian in deep learning: Singularity and beyond. **2017**. Preprint arXiv:1611.07476 [cs.LG]. Cited on pages 11 and 110.

Levent Sagun, Utku Evci, V. Ugur Guney, Yann Dauphin, and Léon Bottou. Empirical analysis of the Hessian of over-parametrized neural networks. In *6th International Conference on Learning Representations: Workshop Track*. OpenReview, **2018**. Access via OpenReview. Cited on pages 11 and 110.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. Presented at the Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing, **2019**. Preprint arXiv:1910.01108 [cs.CL]. Cited on page 25.

Andrew Saxe, Stephanie Nelli, and Christopher Summerfield. If deep learning is the answer, what is the question? *Nature Reviews Neuroscience*, 22(1):55–67, **2021**. Access via Crossref. Cited on page 7.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, **2015**. Access via Crossref. Cited on pages 1 and 7.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, **2020**. Access via Crossref. Cited on page 10.

Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, **2020**. Access via Crossref. Cited on page 8.

Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks*. IEEE, **2022**. Access via Crossref. Cited on page 9.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, **2016**. Access via Crossref. Cited on page 8.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, **2017**. Access via Crossref. Cited on page 8.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, **2018**. Access via Crossref. Cited on pages 1, 2, and 8.

Berfin Şimşek, François Ged, Arthur Jacot, Francesco Spadaro, Clément Hongler, Wulfram Gerstner, and Johanni Brea. Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. In *Proceedings of the 38th International Conference on Machine Learning*, pages 9722–9732. PMLR, **2021**. Access via PMLR. Cited on pages 25, 79, 82, and 110.

Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, third edition, **2013**. Cited on page 48.

G. W. Stewart. *Theory of the Combination of Observations Least Subject to Error*. SIAM, **1995**. Reproduction and English translation of Gauss (1821). Cited on pages 121, 122, and 126.

Pierre Stock and Rémi Gribonval. An embedding of ReLU networks and an analysis of their identifiability. *Constructive Approximation*, **2022**. Access via Crossref. Cited on pages 17, 18, and 19.

Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books, Reading, Massachusetts, **1994**. Cited on page 24.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650. Association for Computational Linguistics, **2019**. Access via Crossref. Cited on pages 1, 10, and 11.

Kenneth J. Supowit. *Topics in Computational Geometry*. Ph.D. thesis, University of Illinois at Urbana-Champaign, **1981**. Access via ProQuest. Cited on page 93.

Harry Surden. Machine learning and law: An overview. In *Research Handbook on Big Data Law*, pages 171–184. Edward Elgar Publishing, **2021**. Access via Crossref. Cited on page 8.

Héctor J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5(4):589–593, **1992**. Access via Crossref. Cited on pages 2, 3, 4, 13, 17, 18, 21, 22, 23, 27, 31, 36, 37, 38, 39, 42, 51, 109, and 113.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, **2018**. Cited on page 6.

Taiji Suzuki, Hiroshi Abe, Tomoya Murata, Shingo Horiuchi, Kotaro Ito, Tokuma Wachi, So Hirai, Masatoshi Yukishima, and Tomoaki Nishimura. Spectral pruning: Compressing deep neural networks via spectral analysis and its generalization error. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2839–2846. IJCAI, **2020**a. Access via Crossref. Cited on page 55.

Taiji Suzuki, Hiroshi Abe, and Tomoaki Nishimura. Compression based bound for non-compressed network: Unified generalization error analysis of large compressible deep neural network. In *8th International Conference on Learning Representations*. OpenReview, **2020**b. Access via OpenReview. Cited on page 55.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B (Methodological)*, 58(1):267–288, **1996**. Access via Crossref. Cited on page 24.

Naftali Tishby, Esther Levin, and Sara A. Solla. Consistent inference of probabilities in layered networks: Predictions and generalization. In *International 1989 Joint Conference on Neural Networks*, volume 2, pages 403–409. IEEE, **1989**. Access via Crossref. Cited on page 18.

Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, **1984**. Access via Crossref. Cited on page 50.

Boris A. Trakhtenbrot. A survey of Russian approaches to *perebor* (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384–400, **1984**. Access via Crossref. Cited on page 126.

Lloyd N. Trefethen and David Bau, III. *Numerical Linear Algebra*. SIAM, **1997**. Cited on page 24.

Alan M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, **1950**. Access via Crossref. Cited on page 6.

Leslie G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, 100(2):135–140, **1981**. Access via Crossref. Cited on page 96.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, **2000**. Access via Crossref. Cited on page 11.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, **2017**. Access via NeurIPS. Cited on pages 8 and 14.

Luca Venturi, Afonso S. Bandeira, and Joan Bruna. Spurious valleys in two-layer neural network optimization landscapes. **2020**. Preprint arXiv:1802.06384 [math.OC]. Cited on page 110.

Tom Viering and Marco Loog. The shape of learning curves: A review. **2021**. Preprint arXiv:2103.10948 [cs.LG]. Cited on page 11.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the real-time strategy game StarCraft II. Blog post, **2019**. Access via DeepMind. Cited on page 8.

Verner Vlačić and Helmut Bölcskei. Affine symmetries and neural network identifiability. *Advances in Mathematics*, 376:107485, **2021**. Access via Crossref. Cited on pages 17, 18, 23, and 109.

Verner Vlačić and Helmut Bölcskei. Neural network identifiability for a family of sigmoidal nonlinearities. *Constructive Approximation*, 55(1):173–224, **2022**. Access via Crossref. Cited on pages 18 and 23.

Sumio Watanabe. Almost all learning machines are singular. In *IEEE Symposium on Foundations of Computational Intelligence*, pages 383–388. IEEE, **2007**. Access via Crossref. Cited on page 11.

Sumio Watanabe. *Algebraic Geometry and Statistical Learning Theory*. Cambridge University Press, **2009**. Cited on page 11.

Sumio Watanabe. *Mathematical Theory of Bayesian Statistics*. CRC Press, **2018**. Access via Crossref. Cited on page 11.

Haikun Wei, Jun Zhang, Florent Cousseau, Tomoko Ozeki, and Shun-ichi Amari. Dynamics of learning near singularities in layered networks. *Neural Computation*, 20(3):813–843, **2008**. Access via Crossref. Cited on page 25.

Susan Wei, Daniel Murfet, Mingming Gong, Hui Li, Jesse Gell-Redman, and Thomas Quella. Deep learning is singular, and that's good. *IEEE Transactions on Neural Networks and Learning Systems*, **2022**. Access via Crossref. To appear in an upcoming volume. Cited on page 11.

Karl Weierstrass. Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen [On the analytic representability of an arbitrary function of a real variable]. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639,789–805, **1885**. In two parts in the same volume. In German. Cited on page 17.

Halbert White. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464, **1989**. Access via Crossref. Cited on page 18.

Jess Whittlestone, Rune Nyrup, Anna Alexandrova, Kanta Dihal, and Stephen Cave. *Ethical and Societal Implications of Algorithms, Data, and Artificial Intelligence: A Roadmap for Research*. Nuffield Foundation, **2019**. Access via LCFI. Cited on page 9.

Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*. MIT Press, second edition, **1961**. Cited on page 6.

Norbert Wiener. *God and Golem, Inc.: A Comment on Certain Points where Cybernetics Impinges on Religion*. MIT Press, **1964**. Cited on page 6.

J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, **1964**. In Algorithms, edited by G. E. Forsythe. Access via Crossref. Cited on page 30.

Robert S. Wolf. *A Tour Through Mathematical Logic*. The Mathematical Association of America, **2005**. Cited on page 47.

Spencer Wong. *From Analytic to Algebraic: The Algebraic Geometry of Two Layer Neural Networks*. Master's thesis, School of Mathematics and Statistics, The University of Melbourne, **2022**. Access via Daniel Murfet. Cited on page 80.

Dong Yu and Li Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer, **2015**. Access via Crossref. Cited on page 8.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. MetaFormer is actually what you need for vision. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10819–10829. IEEE, **2022**. Access via Crossref. Cited on page 14.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations*. OpenReview, **2017**. Access via OpenReview. Cited on pages 1, 9, 11, 110, and 137.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Michael C. Mozer, and Yoram Singer. Identity crisis: Memorization and generalization under extreme overparameterization. In *8th International Conference on Learning Representations*. OpenReview, **2020**. Access via OpenReview. Cited on page 110.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, **2021**. Access via Crossref. Republication of Zhang et al. (2017). Cited on pages 1, 9, 11, and 110.

Lintao Zhang and Sharad Malik. The quest for efficient Boolean satisfiability solvers. In *Computer Aided Verification*, pages 17–36. Springer, **2002**. Access via Crossref. Cited on page 47.

Appendices

Appendix A

Degenerate Neighbourhoods in Function Space

In this appendix, I introduce a second approximate measure of neural network degeneracy: the *functional approximate rank*. Like the parametric approximate rank (Chapter 6), the functional approximate rank of a neural network is defined based on the rank of nearby neural networks. The key difference is that for the functional approximate rank, I use neighbourhoods in function space rather than parameter space. The contents of this appendix are as follows.

1. In Appendix A.1, I review the necessary preliminary mathematical definitions and notation for defining the $\mathscr{L}_2$ semimetric between functions.

2. In Appendix A.2, I give a lemma relating uniform neighbourhoods in parameter space with $\mathscr{L}_2$ neighbourhoods in function space.

3. In Appendix A.3, I introduce the functional approximate rank for simple neural networks. I discuss the definition from several perspectives, including the relationships to the parametric approximate rank (using the result of Appendix A.2), neural network compressibility (Buciluă et al., 2006; Hinton et al., 2014), and approximation complexity theory (Barron, 1993; Mhaskar, 1996). Each perspective leads to a different method of bounding the functional approximate rank.

A.1 Function spaces

Neural networks implement functions mapping between an input vector space and an output vector space. The set of such mappings itself has the structure of an infinite-dimensional abstract vector space. In this section, I recall the measure theory required to imbue this space with a semimetric structure (as an $\mathscr{L}_2$ space).

For readers not already familiar with elementary functional analysis, I note that a somewhat informal familiarity with the contents of this section is sufficient to understand this appendix. For a comprehensive introduction, consult Cohn (2013, §§1–3,10).

The Euclidean metric

The uniform norm reigns in the chapters of this thesis, governing the measurement of parameter vectors (Section 3.1). In this appendix, Euclid sets the standard by which input and output vectors are measured. I recall the following elementary definitions.

Definition A.1 (Euclidean norm)**.** Consider a vector $x \in \mathbb{R}^n$. The *Euclidean norm* of x, denoted $\|x\|_2$, is defined as $\|x\|_2 = \sqrt{\sum_{i=1}^n {x_i}^2}$.

Definition A.2 (Euclidean distance)**.** Consider a pair of vectors $x, y \in \mathbb{R}^n$. The *Euclidean distance* between x and y, denoted $\|x - y\|_2$, is defined as the Euclidean norm of the difference between x and y: $\|x - y\|_2 = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$.

Input measures

To measure functions (in the $\mathscr{L}_2$ sense), it is first necessary to weigh their inputs.

Definition A.3 (Input measure)**.** Fix an input dimension $n \in \mathbb{N}^+$. Let $\mathscr{B}(\mathbb{R}^n)$ denote the Borel σ-algebra on $\mathbb{R}^n$ (for current purposes, a set of subsets of $\mathbb{R}^n$). An n-dimensional *input measure* is a probability measure q on $\mathbb{R}^n$: a function $q : \mathscr{B}(\mathbb{R}^n) \to [0, 1]$ satisfying

(i) $q(\emptyset) = 0$;

(ii) $q(\mathbb{R}^n) = 1$; and

(iii) if $A_1, A_2, \ldots \in \mathscr{B}(\mathbb{R}^n)$ is a sequence of disjoint sets, then $q\left(\bigcup_{i=1}^\infty A_i\right) = \sum_{i=1}^\infty q(A_i)$.

Remark A.4. For current purposes, it suffices to view q as a kind of probability distribution over input vectors in $\mathbb{R}^n$: a function that takes a subset $A \subset \mathbb{R}^n$ and returns the probability of the event that a randomly chosen input vector is in A. Then, the co-domain and the conditions (i)–(iii) ensure that these probabilities satisfy Kolmogorov's axioms (Kolmogorov, 1933).

Integration

Having weighed inputs, it is then possible to compute a weighted average output of a (sufficiently well-behaved) function.

Definition A.5 (Integral with respect to an input measure)**.** Consider an n-dimensional input measure q and a *q-integrable* function $f : \mathbb{R}^n \to \mathbb{R}$ (see Remark A.7). Let the expression

$$\int_{\mathbb{R}^n} f(x)\ dq(x)$$

denote the *integral of the function f with respect to the input measure q.*

Remark A.6. A full definition of the value of the integral is omitted for brevity but can be found in Cohn (2013, §2). For current purposes, it suffices to understand the integral as a weighted average of the function output, with inputs weighted according to the probability ascribed by the input measure q.

Remark A.7. Definition A.5 defines the integral for a q-integrable function f. A sufficient condition (and all that is necessary for this appendix) is that f is continuous and bounded, since q is a probability measure (Cohn, 2013, Examples 2.1.2(a) and 2.3.7(a)).

$\mathscr{L}_2$ space of square integrable functions

With input measures and integration, one can define the *size* of a function as the (square root of the) weighted average squared Euclidean size of the function's output. This requires the squared size of the output to be q-integrable (Definition A.5) leading to the following space of class of functions that can be measured in this way.

Definition A.8 (Square integrable functions)**.** Consider $n, m \in \mathbb{N}^+$, and an n-dimensional input measure q. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is *square integrable* with respect to the measure q if the function mapping $x \mapsto \|f(x)\|_2^2$ is q-integrable. Denote the set of such square integrable functions $\mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$.

Remark A.9. $\mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$ is closed under pointwise scalar multiplication and pointwise addition of functions, so forms an abstract vector space of functions.

Remark A.10. Following Remark A.7, for f to be square integrable with respect to any input measure, it suffices for f to be continuous and bounded.

The space of simple neural network functions (Section 3.2) is a subspace of the space of square integrable functions, as simple neural network functions are square integrable:

Proposition A.11. *Consider an n-dimensional input measure q. $\mathcal{F}_\infty^{n,m} \subset \mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$.*

Proof. The hyperbolic tangent is continuous and bounded, so any simple neural network function $f \in \mathcal{F}_h^{n,m}$ (as a componentwise hyperbolic tangent up to affine linear transformations) is as well. Therefore f is square integrable with respect to q (Remark A.10). $\square$

The $\mathscr{L}_2$ semimetric

It is time to define the $\mathscr{L}_2(q)$ seminorm and $\mathscr{L}_2(q)$ distance semimetric—measures of the size of and distance between square integrable functions.

Definition A.12 ($\mathscr{L}_2$ seminorm of a function)**.** Consider an n-dimensional input measure q. Let $f \in \mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$ be a square integrable function. Define the *$\mathscr{L}_2(q)$-seminorm of f,* denoted $\|f\|_{\mathscr{L}_2(q)}$, as

$$\|f\|_{\mathscr{L}_2(q)} = \left(\int_{\mathbb{R}^n} \|f(x)\|_2^2 \, dq(x) \right)^{\frac{1}{2}}.$$

Definition A.13 ($\mathscr{L}_2$ distance semimetric between two functions)**.** Consider an n-dimensional input measure q. Let $f, g \in \mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$ be a pair of square integrable functions. Define the *$\mathscr{L}_2(q)$-distance between f and g* as

$$\|f - g\|_{\mathscr{L}_2(q)} = \left(\int_{\mathbb{R}^n} \|f(x) - g(x)\|_2^2 \, dq(x) \right)^{\frac{1}{2}}.$$

Remark A.14. Semimetrics (seminorms) possess nearly all of the properties of metrics (norms), except they are not necessarily positive for distinct (non-zero) functions. Indeed if $f \neq g$ differ only for inputs that are not assigned positive probability by q, then $\|f - g\|_{\mathscr{L}_2(q)} = 0$. The remaining properties are sufficient for my analysis.

Finally, the $\mathscr{L}_2$ semimetric allows the following definition of an $\mathscr{L}_2$ neighbourhood (also $\mathscr{L}_2$ ball)—a set of functions with similar outputs to a given function.

Definition A.15 (Closed $\mathscr{L}_2$ neighbourhood)**.** Consider $n, m \in \mathbb{N}^+$ and an n-dimensional input measure q. Given a square integrable function $f \in \mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$ and a positive scalar $\varepsilon \in \mathbb{R}^+$, the *closed $\mathscr{L}_2(q)$ neighbourhood of f with radius ε,* denoted $\bar{B}_{\mathscr{L}_2(q)}(f; \varepsilon)$, is the set of square integrable functions with $\mathscr{L}_2(q)$ distance *at most* ε from f:

$$\bar{B}_{\mathscr{L}_2(q)}(f; \varepsilon) = \left\{ g \in \mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m) \,\middle|\, \|f - g\|_{\mathscr{L}_2(q)} \leq \varepsilon \right\}.$$

A.2 Parametric and functional neighbourhoods

This section proves the following lemma, which shows a link between uniform neighbourhoods in parameter space and $\mathscr{L}_2$ neighbourhoods in the space of simple neural networks.

Definition A.16 (Raw second moment of an input measure)**.** Given an n-dimensional input measure q, define the *raw second moment of* q, denoted $\mathbb{M}_q^2$, as

$$\mathbb{M}_q^2 = \int_{\mathbb{R}^n} \|x\|_2^2 \, dq(x).$$

Moreover, let $\mathbb{M}_q$ denote its square root $\mathbb{M}_q = \sqrt{\mathbb{M}_q^2}$.

Lemma A.17 (Uniform distance and $\mathscr{L}_2$ distance)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $w, w' \in \mathcal{W}_h^{n,m}$, and let q be an n-dimensional input measure. Define constants $M, W \in \mathbb{R}^+$ as follows:*

$$M = (h+2)\sqrt{(\mathbb{M}_q^2 + 1)(n+1)m}, \qquad W = \max\left\{\|w\|_\infty, 1\right\}.$$

Then

$$\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} \leq 3MW \|w - w'\|_\infty + M \|w - w'\|_\infty^2 .$$

The remainder of this section builds up to the proof of Lemma A.17:

1. I establish a bound on the $\mathscr{L}_2$ seminorm of a neural network function with a single unbiased hidden unit.
2. I establish a bound on the $\mathscr{L}_2$ distance between pairs of individual unbiased units that share the same incoming or outgoing weight vector.
3. I establish a bound on the $\mathscr{L}_2$ distance between arbitrary pairs of individual unbiased units.
4. I establish a bound on the $\mathscr{L}_2$ distance between arbitrary simple unbiased neural network functions.
5. I establish a bound on the $\mathscr{L}_2$ distance between simple unbiased neural network functions in terms of their parametric difference.
6. By realising simple neural networks with biases as unbiased simple neural networks with additional units, I use the latter bound to prove Lemma A.17.

Step 1: Seminorm bound for an individual unit

The following bound on the seminorm of a single unbiased hidden unit's function is used in several of the following steps.

Lemma A.18. *Let q be an n-dimensional input measure. For $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$,*

$$\|a \tanh(bx)\|_{\mathscr{L}_2(q)} \leq \|a\|_2 \, \|b\|_2 \, \mathbb{M}_q.$$

Proof. It suffices to show that $\|a \tanh(bx)\|^2_{\mathscr{L}_2(q)} \leq \|a\|_2^2 \, \|b\|_2^2 \, \mathbb{M}_q^2$:

$$\begin{aligned}
\|a \tanh(bx)\|^2_{\mathscr{L}_2(q)} &= \int_{\mathbb{R}^n} \|a \tanh(b \cdot x)\|_2^2 \; dq(x). \\
&= \|a\|_2^2 \int_{\mathbb{R}^n} (\tanh(b \cdot x))^2 \; dq(x) \\
&\leq \|a\|_2^2 \int_{\mathbb{R}^n} (b \cdot x)^2 \; dq(x) && (|\tanh(z)| \leq |z|) \\
&\leq \|a\|_2^2 \int_{\mathbb{R}^n} \|b\|_2^2 \, \|x\|_2^2 \; dq(x) && \text{(Cauchy–Schwarz)} \\
&= \|a\|_2^2 \, \|b\|_2^2 \, \mathbb{M}_q^2.
\end{aligned}$$

□

Step 2: Distance bounds for units with shared weights

Consider two units with the same incoming weight, or two units with the same outgoing weight, and no biases. The following results bound their $\mathscr{L}_2$ distance in function space.

Lemma A.19. *Let q be an n-dimensional input measure. For $a, a' \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$,*

$$\|a \tanh(bx) - a' \tanh(bx)\|_{\mathscr{L}_2(q)} < \|a - a'\|_2 \, \|b\|_2 \, \mathbb{M}_q.$$

Proof. By Lemma A.18,

$$\begin{aligned}
\|a \tanh(bx) - a' \tanh(bx)\|_{\mathscr{L}_2(q)} &= \|(a - a') \tanh(bx)\|_{\mathscr{L}_2(q)} \\
&\leq \|a - a'\|_2 \, \|b\|_2 \, \mathbb{M}_q.
\end{aligned}$$

□

Lemma A.20. *Let q be an n-dimensional input measure. For $a \in \mathbb{R}^m$ and $b, b' \in \mathbb{R}^n$,*

$$\|a \tanh(bx) - a \tanh(b'x)\|_{\mathscr{L}_2(q)} \leq 2 \, \|a\|_2 \, \|b - b'\|_2 \, \mathbb{M}_q.$$

Proof. The subtraction formula for hyperbolic tangent says

$$\tanh(\phi - \psi) = \frac{\tanh \phi - \tanh \psi}{1 - \tanh \phi \tanh \psi}.$$

Square both sides and re-arrange to find

$$\begin{aligned}(\tanh\phi - \tanh\psi)^2 &= \tanh^2(\phi-\psi)\cdot(1-\tanh\phi\tanh\psi)^2\\ &\le 4\tanh^2(\phi-\psi)\,. && (|\tanh\phi\tanh\psi|<1)\end{aligned}$$

It follows that

$$\begin{aligned}\|a\tanh(bx) - a\tanh(b'x)\|_{\mathscr{L}_2(q)}^2 &= \int_{\mathbb{R}^n} \|a\tanh(bx) - a\tanh(b'x)\|_2^2 \; dq(x)\\ &= \|a\|_2^2 \int_{\mathbb{R}^n} (\tanh(bx) - \tanh(b'x))^2 \; dq(x)\\ &\le 4\,\|a\|_2^2 \int_{\mathbb{R}^n} \tanh^2(bx - b'x)\; dq(x)\\ &= 4\,\|a\|_2^2\,\|\tanh((b-b')x)\|_{\mathscr{L}_2(q)}^2\\ &= 4\,\|a\|_2^2\,\|b-b'\|_2^2\,\mathbb{M}_q^2\,. && \text{(Lemma A.18)}\end{aligned}$$

The desired bound is recovered by taking the square root of both sides. □

Step 3: Distance bound for arbitrary individual units

From the above bounds on distances between units with shared weights, I derive the following bounds on distances between arbitrary unbiased units.

Lemma A.21. *Let q be an n-dimensional input measure. For $a, a' \in \mathbb{R}^m$ and $b, b' \in \mathbb{R}^n$,*

$$\|a\tanh(bx) - a'\tanh(b'x)\|_{\mathscr{L}_2(q)} \le 2\,\|a\|_2\,\|b-b'\|_2\,\mathbb{M}_q + \|b'\|_2\,\|a-a'\|_2\,\mathbb{M}_q.$$

Proof. The key is to decompose the difference into a difference between units with shared outgoing weight and a difference between units with shared incoming weight. Then, the triangle inequality and the above bounds (Lemmas A.19 and A.20) can be applied to each component. In detail,

$$\begin{aligned}&\|a\tanh(bx) - a'\tanh(b'x)\|_{\mathscr{L}_2(q)}\\ &\quad= \|a\tanh(bx) - a\tanh(b'x) + a\tanh(b'x) - a'\tanh(b'x)\|_{\mathscr{L}_2(q)}\\ &\quad\le \|a\tanh(bx) - a\tanh(b'x)\|_{\mathscr{L}_2(q)} + \|a\tanh(b'x) - a'\tanh(b'x)\|_{\mathscr{L}_2(q)} && \text{(tri. ineq.)}\\ &\quad\le 2\,\|a\|_2\,\|b-b'\|_2\,\mathbb{M}_q + \|b'\|_2\,\|a-a'\|_2\,\mathbb{M}_q && \text{(Lemmas A.19 and A.20)}\end{aligned}$$

□

Step 4: Distance bound for arbitrary network functions

By pairing the units in two simple neural networks and applying the triangle equality, I derive the following bounds on distances between arbitrary unbiased neural network functions.

Lemma A.22. *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $w, w' \in \mathcal{W}_h^{n,m}$ be unbiased parameters $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, 0, 0)$ and $w' = (a'_1, \ldots, a'_h, b'_1, \ldots, b'_h, 0, 0)$. Then*

$$\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} \leq \sum_{i=1}^{h} \left(2\|a_i\|_2 \|b_i - b'_i\|_2 + \|b'_i\|_2 \|a_i - a'_i\|_2\right) \mathbb{M}_q.$$

Proof. Pair off the units, use the triangle inequality, and then use the bound on the distance between arbitrary pairs of units (Lemma A.21).

$$\begin{aligned}
\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} &= \left\| \sum_{i=1}^{h} a_i \tanh(b_i x) - \sum_{i=1}^{h} a'_i \tanh(b'_i x) \right\|_{\mathscr{L}_2(q)} \\
&= \left\| \sum_{i=1}^{h} \big(a_i \tanh(b_i x) - a'_i \tanh(b'_i x)\big) \right\|_{\mathscr{L}_2(q)} \\
&\leq \sum_{i=1}^{h} \|a_i \tanh(b_i x) - a'_i \tanh(b'_i x)\|_{\mathscr{L}_2(q)} && \text{(triangle inequality)} \\
&\leq \sum_{i=1}^{h} \left(2\|a_i\|_2 \|b_i - b'_i\|_2 + \|b'_i\|_2 \|a_i - a'_i\|_2\right) \mathbb{M}_q. && \text{(Lemma A.21)}
\end{aligned}$$

□

Remark A.23. The above proof works with any pairing of units in the two networks. The strongest bound therefore comes from the minimum over all pairings (equivalently, over all unit permutations $\pi \in S_h$, where S_h is the symmetric group on h elements):

$$\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} \leq \min_{\pi \in S_h} \sum_{i=1}^{h} \left(2\|a_i\|_2 \left\|b_i - b'_{\pi(i)}\right\|_2 + \left\|b'_{\pi(i)}\right\|_2 \left\|a_i - a'_{\pi(i)}\right\|_2\right) \mathbb{M}_q.$$

Step 5: Distance bound for similar parameters

Lemma A.24. *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $w, w' \in \mathcal{W}_h^{n,m}$ be unbiased parameters, and let q be an n-dimensional input measure. Then*

$$\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} \leq \mathbb{M}_q h \sqrt{nm} \left(3\|w\|_\infty + \|w - w'\|_\infty\right) \|w - w'\|_\infty.$$

Proof.

$$\begin{aligned}
\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} &\leq \mathbb{M}_q \sum_{i=1}^{h} \left(2\|a_i\|_2 \|b_i - b_i'\|_2 + \|b_i'\|_2 \|a_i - a_i'\|_2\right) && \text{(Lemma A.22)}\\
&\leq \mathbb{M}_q \sqrt{nm} \sum_{i=1}^{h} \left(2\|a_i\|_\infty \|b_i - b_i'\|_\infty + \|b_i'\|_\infty \|a_i - a_i'\|_\infty\right) && (\dagger)\\
&\leq \mathbb{M}_q \sqrt{nm} \sum_{i=1}^{h} \left(2\|w\|_\infty \|w - w'\|_\infty + \|w'\|_\infty \|w - w'\|_\infty\right)\\
&\leq \mathbb{M}_q h\sqrt{nm} \left(2\|w\|_\infty + \|w'\|_\infty\right) \|w - w'\|_\infty\\
&\leq \mathbb{M}_q h\sqrt{nm} \left(3\|w\|_\infty + \|w - w'\|_\infty\right) \|w - w'\|_\infty . && (\ddagger)
\end{aligned}$$

Step ($\dagger$) uses the observation that for $u \in \mathbb{R}^p$, $\|u\|_2 \leq \sqrt{p}\,\|u\|_\infty$ and step ($\ddagger$) uses the observation that $\|w'\|_\infty \leq \|w\|_\infty + \|w - w'\|_\infty$ by the triangle inequality. □

Step 6: The main lemma

Lemma A.24 establishes a connection between variations in parameter space and variations in function space for simple unbiased neural networks. The main lemma aims to establish a similar connection for arbitrary networks. The result for unbiased networks is useful because a simple biased neural network can be implemented as an unbiased neural network with a few extra units and weights, if q is augmented to enforce a restriction on the inputs to an additional input unit:

Proposition A.25. *Consider two simple neural network architectures, $\mathcal{A}_h^{n,m}$, and $\mathcal{A}_{h+2}^{n+1,m}$. Then for each neural network parameter $w \in \mathcal{W}_h^{n,m}$, there exists an unbiased neural network parameter $u \in \mathcal{W}_{h+2}^{n+1,m}$ such that for $x \in \mathbb{R}^n$,*

$$f_u(x, 1) = f_w(x).$$

Proof. Let $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d) \in \mathcal{W}_h^{n,m}$. Define $b_0' = (0, \ldots, 0, 1) \in \mathbb{R}^{n+1}$. Construct

$$u = \Bigg(\underbrace{a_1, \ldots, a_h, \frac{d}{2\tanh(1)}, \frac{d}{2\tanh(1)}}_{h+2 \text{ outgoing weight vectors in } \mathbb{R}^m},\ \underbrace{\overbrace{b_1, c_1}^{(b_i,c_i)\in\mathbb{R}^{n+1}}, \ldots, b_h, c_h,\ \overbrace{0, 1}^{0\in\mathbb{R}^n, 1\in\mathbb{R}}, 0, 1,}_{h+2 \text{ incoming weight vectors in } \mathbb{R}^{n+1}}\ \underbrace{0, \ldots, 0}_{\text{biases}}\Bigg) \in \mathcal{W}_{h+2}^{n+1,m}.$$

Then for $x \in \mathbb{R}^n$,

$$\begin{aligned}
f_u(x,1) &= \sum_{i=1}^{h} a_i \tanh(b_i \cdot x + c_i \cdot 1) + \sum_{i=1}^{2} \frac{d}{2\tanh(1)} \tanh(0 \cdot x + 1 \cdot 1) \\
&= d + \sum_{i=1}^{h} a_i \tanh(b_i \cdot x + c_i) \\
&= f_w(x). \qquad \square
\end{aligned}$$

With this construction in mind, I restate and prove the main lemma.

Lemma A.17 (Uniform distance and $\mathscr{L}_2$ distance)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let $w, w' \in \mathcal{W}_h^{n,m}$, and let q be an n-dimensional input measure. Define constants $M, W \in \mathbb{R}^+$ as follows:*

$$M = (h+2)\sqrt{(\mathbb{M}_q^2 + 1)(n+1)m}, \qquad W = \max\{\|w\|_\infty, 1\}.$$

Then

$$\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} \le 3MW\|w - w'\|_\infty + M\|w - w'\|_\infty^2.$$

Proof. Let $w, w' \in \mathcal{W}_h^{n,m}$ and q an n-dimensional input measure, as above. Embed these parameters as unbiased parameters in a larger parameter space as per Proposition A.25, producing $u, u' \in \mathcal{W}_{h+2}^{n+1,m}$ respectively. Define $q' = q \times \delta_1$, where δ_1 is the Dirac measure (point mass) concentrated on 1. Then $\|f_u - f_{u'}\|_{\mathscr{L}_2(q')} = \|f_w - f_{w'}\|_{\mathscr{L}_2(q)}$, since

$$\begin{aligned}
\|f_u - f_{u'}\|_{\mathscr{L}_2(q')}^2 &= \int_{\mathbb{R}^{n+1}} \|f_u(x) - f_{u'}(x)\|_2^2 \, dq'(x) \\
&= \int_{\mathbb{R}^n} \int_{\mathbb{R}} \|f_u(x,y) - f_{u'}(x,y)\|_2^2 \, d\delta_1(y) dq(x) \\
&= \int_{\mathbb{R}^n} \|f_u(x,1) - f_{u'}(x,1)\|_2^2 \, dq(x) \\
&= \int_{\mathbb{R}^n} \|f_w(x) - f_{w'}(x)\|_2^2 \, dq(x) && \text{(Proposition A.25)} \\
&\|f_w - f_{w'}\|_{\mathscr{L}_2(q)}.
\end{aligned}$$

Therefore

$$\begin{aligned}
\|f_w - f_{w'}\|_{\mathscr{L}_2(q)} &= \|f_u - f_{u'}\|_{\mathscr{L}_2(q')}^2 \\
&\le \mathbb{M}_{q'}(h+2)\sqrt{(n+1)m}\,(3\|u\|_\infty + \|u - u'\|_\infty)\|u - u'\|_\infty && \text{(Lemma A.24)} \\
&\le M(3W + \|w - w'\|_\infty)\|w - w'\|_\infty && (\dagger) \\
&= 3MW\|w - w'\|_\infty + M\|w - w'\|_\infty^2.
\end{aligned}$$

Step (†) is based on the following three observations. First,

$$\begin{aligned}\mathbb{M}_{q'}^2 &= \int_{\mathbb{R}^n}\int_{\mathbb{R}} (\|x\|_2^2 + y^2) d\delta_1(y)\ dq(x) \\ &= \int_{\mathbb{R}^n} \|x\|_2^2\ dq(x) + \int_{\mathbb{R}} y^2 d\delta_1(y) \\ &= \mathbb{M}_q^2 + 1.\end{aligned}$$

Second, note that based on how u is constructed from w (by the proof of Proposition A.25), writing $w = (a_1, \ldots, a_h, b_1, \ldots, b_h, c_1, \ldots, c_h, d)$,

$$\begin{aligned}\|u\|_\infty &= \max\left\{\|a_1\|_\infty, \ldots, \|a_h\|_\infty, \|b_1\|_\infty, \ldots, \|b_h\|_\infty, |c_1|, \ldots, |c_h|, \frac{\|d\|_\infty}{2\tanh(1)}, 1\right\} \\ &\leq \max\{\|a_1\|_\infty, \ldots, \|a_h\|_\infty, \|b_1\|_\infty, \ldots, \|b_h\|_\infty, |c_1|, \ldots, |c_h|, \|d\|_\infty, 1\} \\ &= \max\{\|w\|_\infty, 1\}.\end{aligned}$$

Third, by similar reasoning, $\|u - u'\|_\infty \leq \|w - w'\|_\infty$. □

A.3 Functional approximate rank

Chapter 6 studies a measure of approximate degeneracy of a simple neural network parameter based on the rank of the most degenerate parameter in a small uniform neighbourhood in parameter space. In this section, I investigate the following alternative measure of approximate degeneracy for simple neural networks, based on the rank of the most degenerate neural network *function* within a small $\mathscr{L}_2$ neighbourhood in *function space*.

Definition A.26 (Functional approximate rank)**.** Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Given a parameter $w \in \mathcal{W}_h^{n,m}$, an n-dimensional input measure q, and a positive radius $\varepsilon \in \mathbb{R}^+$, define the *functional approximate rank* of w, denoted $\operatorname{frank}_{q,\varepsilon}(w)$, as the rank of the lowest-rank simple neural network function within a closed $\mathscr{L}_2(q)$ neighbourhood with radius ε. That is,

$$\operatorname{frank}_{q,\varepsilon}(w) = \min \left\{ \operatorname{rank}(g) \,\middle|\, g \in \bar{B}_{\mathscr{L}_2(q)}(f_w; \varepsilon) \cap \mathcal{F}_\infty^{n,m} \right\}.$$

(Recall that Definition 4.5 defines the rank of a neural network function.)

Remark A.27. Unlike the parametric approximate rank (but like the rank itself), the functional approximate rank is a well-defined property of the neural network function implemented by a parameter.

Below, I investigate several perspectives on the functional approximate rank, leading to various bounds for the quantity.

1. Based on the result of Appendix A.2, nearby degenerate parameters correspond to nearby degenerate functions. Therefore, one can bound the functional approximate rank using a related parametric approximate rank.

2. The functional approximate rank is a kind of lossy compressibility of a neural network parameter, specifying a search for a low-rank parameter that implements a similar function. Accordingly, computing the functional approximate rank can be viewed as a teacher–student learning problem, as in *model compression* (Buciluă et al., 2006) or *distillation* (Hinton et al., 2014), and standard learning algorithms can be used to compute bounds.

3. Neural network approximation complexity theory (e.g., Barron, 1993; Mhaskar, 1996) studies the number of units required to implement functions of a given degree of smoothness. By applying such results to neural network functions themselves one arrives at a bound for the functional approximate rank of a neural network function in terms of its degree of smoothness.

Bound by parametric approximate rank

Appendix A.2 shows that uniform neighbourhoods in parameter space can be found implementing functions within any neighbourhood in function space. Therefore, the parametric approximate rank provides a way to bound the functional approximate rank.

Theorem A.28 (Parametric approximate rank and functional approximate rank)**.** *Consider a simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Let q be an n-dimensional input measure. Given $w \in \mathcal{W}_h^{n,m}$ and $\varepsilon \in \mathbb{R}^+$,*

$$\operatorname{frank}_{q,\varepsilon}(w) \le \operatorname{prank}_\delta(w),$$

where $\delta = \sqrt{\frac{9}{4}W^2 + \frac{\varepsilon}{M}} - \frac{3}{2}W$, $M = (h+2)\sqrt{(\mathbb{M}_q^2 + 1)(n+1)m}$, *and* $W = \max\{\|w\|_\infty, 1\}$.

Proof. Let $w \in \mathcal{W}_h^{n,m}$ with $\operatorname{prank}_\delta(w) = r$. Let $u \in \bar{B}_\infty(w; \delta)$ with $\operatorname{rank}(u) = r$. Then

$$\begin{aligned}
\|f_w - f_u\|_{\mathscr{L}_2(q)} &\le 3MW\|w-u\|_\infty + M\|w-u\|_\infty^2 && \text{(Lemma A.17)}\\
&\le 3MW\delta + M\delta^2 && (\|w-u\|_\infty \le \delta)\\
&= \varepsilon.
\end{aligned}$$

The final step follows because the chosen δ is exactly the positive solution to the quadratic equation $\varepsilon = 3MW\delta + M\delta^2$. Since f_u is within ε of f_w, $\operatorname{frank}_{q,\varepsilon}(w) \le \operatorname{rank}(u) = r$. □

Remark A.29. The functions implemented by the uniform neighbourhood may not exhaust the $\mathscr{L}_2$ neighbourhood (cf. inverse stability results from, e.g., Petersen et al., 2021). Moreover, if some or all of $\|w\|_\infty$, $\mathbb{M}_q$, and h are large relative to ε (with large h in particular being reasonable in practice), this decreases the uniform radius, inflating the parametric approximate rank (Proposition 6.3) and therefore loosening the bound.

Remark A.30. Theorem A.28 holds for any parameter in any architecture implementing a given function. However, the parametric approximate rank may vary from one parameter to another (cf. Example 6.5). By finding an implementation with the lowest possible parametric approximate rank (given the associated radius δ, which also depends on the parameter) one would obtain the tightest bound available with this method.

Remark A.31. Theorem A.28 and Remark A.30 suggest a general method for computing an upper bound on the functional approximate rank of a parameter, by (1) choosing some functionally equivalent parameter w', (2) computing the appropriate parametric radius δ based on the parameter's norm, and (3) computing a bound for $\operatorname{prank}_\delta(w')$ (such as by Algorithm 6.10).

Bound by learning or compression

The above approach to bounding the functional approximate rank can be seen as a search for one *low-rank* function within a set of *nearby* functions. A dual approach is therefore to search for one *nearby* function within a set of *low-rank* functions.

If the set of low-rank functions is taken as the parameter space of a simple neural network architecture with a reduced number of hidden units, then the latter search can be viewed as a teacher–student learning problem. Given a neural network parameter $w \in \mathcal{W}_h^{n,m}$, and some prospective functional approximate rank bound r, proceed as follows.

1. Create a data set by sampling a large number of inputs from the input measure, and computing the corresponding outputs according to f_w.

2. Conduct a learning process to search $\mathcal{W}_r^{n,m}$ for a low-loss parameter $w_\star \in \mathcal{W}_r^{n,m}$ using the mean squared error loss function and the data set from step (1). Any standard learning algorithm can be used.

3. The mean squared error is an estimate of the squared $\mathscr{L}_2$ distance between f_w and $f_{w_\star}$. Therefore, if the learning process found $w_\star$ with loss less than ε^2, then $f_{w_\star}$ is (according to the estimate) within the $\mathscr{L}_2$ neighbourhood of f_w with radius ε. It follows that r—indeed, $\operatorname{rank}(w_\star)$—is an upper bound on the functional approximate rank $\operatorname{frank}_{q,\varepsilon}(w)$ (according to the estimate).

4. If instead the learning process found $w_\star$ with loss greater than ε^2, one cannot necessarily conclude that r is *not* an upper bound on the functional approximate rank $\operatorname{frank}_{q,\varepsilon}(w)$. There may be low-loss parameters that the learning algorithm failed to find. Therefore, in this case, do not report a bound.

Remark A.32. The above procedure for bounding the functional approximate rank is similar to the techniques of *model compression* (Buciluă et al., 2006) or *distillation* (Hinton et al., 2014), by which one attempts to learn a small neural network that mimics the function of an existing large neural network. Accordingly, these existing techniques can be interpreted as attempts to compute the functional approximate rank of the large neural network.

Bound by approximation complexity

Related to functional approximate rank is the theory of neural network approximation complexity. This subfield of neural network geometry studies the relationship between the number of hidden units and the achievable degree of approximation to an arbitrary function, in terms of properties of the function (see, e.g., Barron, 1993; Murata, 1996; Mhaskar, 1996). Results from this field often conform to the pattern "given a function f, for $h \in \mathbb{N}^+$, there exists a parameter $w_h \in \mathcal{W}_h^{n,m}$ such that $\|f - f_{w_h}\|_{\mathscr{L}_2(q)} \leq Q/h^p$, where $Q > 0$ is some measure of the smoothness of f and $p > 0$ is some order of approximation." Thus these results quantify the rate at which the ability to approximate functions of a given smoothness class improves with an increase in the number of hidden units available.

If neural network functions themselves satisfy the assumptions imposed by such results on the approximated functions, then such a rate of approximation allows one to calculate a number of hidden units above which there must be a neural network function within a given $\mathscr{L}_2$ radius. This can be used to derive a bound on the functional approximate rank, as follows.

Lemma A.33 (Approximation complexity and functional approximate rank)**.** *Consider the simple neural network architecture $\mathcal{A}_h^{n,m}$ with parameter space $\mathcal{W}_h^{n,m}$. Consider a parameter $w \in \mathcal{W}_h^{n,m}$, an n-dimensional input measure q, and a positive radius $\varepsilon \in \mathbb{R}^+$. Suppose there exist some constants $Q, p \in \mathbb{R}^+$ such that for all $r \in \mathbb{N}^+$ there exists $w_r \in \mathcal{W}_r^{n,m}$ such that*

$$\|f_w - f_{w_r}\|_{\mathscr{L}_2(q)} \leq \frac{Q}{r^p}.$$

Then

$$\operatorname{frank}_{q,\varepsilon}(w) \leq \left\lceil \left(\frac{Q}{\varepsilon}\right)^{\frac{1}{p}} \right\rceil.$$

Proof. Let $r = \left\lceil \left(\frac{Q}{\varepsilon}\right)^{\frac{1}{p}} \right\rceil$. Then $r \in \mathbb{N}^+$ since $Q, p, \varepsilon \in \mathbb{R}^+$. It follows by the assumption that there exists $w_r \in \mathcal{W}_r^{n,m}$ for which

$$\|f_w - f_{w_r}\|_{\mathscr{L}_2(q)} \leq \frac{Q}{r^p} = \frac{Q}{\left(\left\lceil \left(\frac{Q}{\varepsilon}\right)^{\frac{1}{p}} \right\rceil\right)^p} \leq \frac{Q}{\left(\left(\frac{Q}{\varepsilon}\right)^{\frac{1}{p}}\right)^p} = \varepsilon.$$

Thus $f_{w_r} \in \bar{B}_{\mathscr{L}_2(q)}(f_w; \varepsilon)$. Then, noting $\operatorname{rank}(w_r) \leq r$ (Proposition 4.6),

$$\operatorname{frank}_{q,\varepsilon}(w) \leq \operatorname{rank}(f_{w_r}) = \operatorname{rank}(w_r) \leq r = \left\lceil \left(\frac{Q}{\varepsilon}\right)^{\frac{1}{p}} \right\rceil. \qquad \square$$

Remark A.34. One always has $\operatorname{frank}_{q,\varepsilon}(w) \leq h$ for $w \in \mathcal{W}_h^{n,m}$ (cf. Proposition 6.2(iii)). Whether Lemma A.33 leads to a non-trivial bound depends on the smoothness of f_w and the order of approximation.

Appendix B

List of Lists

B.1 List of figures and tables

Tables and figures are numbered and listed together.

B.2 List of definitions and computational problems

B.3 List of algorithms and named theorems

Each algorithm is accompanied by a correctness theorem (not listed). First listing for Lemma A.17 is for statement, second for restatement and proof. Most propositions, lemmas, corollaries, and remarks are not listed.

B.4 List of symbols

Grouped by topic: (1) some basic mathematical objects; (2) simple neural networks and neural network geometry; (3) computational complexity theory; (4) measure theory, function space, and functional approximate rank (from Appendix A).

Symbol	Definition	Description
$\emptyset$	–	Empty set
$\mathbb{N}$	–	(Non-negative) natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$
$\mathbb{N}^+$	–	Positive natural numbers $\mathbb{N}^+ = \{1, 2, \ldots\}$
$\mathbb{R}$	–	Real scalars
$\mathbb{R}^+$	–	Positive real scalars $\mathbb{R}^+ = \{\, x \in \mathbb{R} \mid x > 0 \,\}$
$\mathbb{R}^p$	–	p-dimensional real vectors
$\mathbb{V}(\cdot)$	–	Zero locus of a given collection of polynomials
S_h	3.31	Symmetric group on $\{1, \ldots, h\}$ (set of permutations)
id	–	Identity permutation (also identity activation function)
$\{-1, +1\}^h$	3.34	Set of sign vectors of dimension h
$\|S\|$	–	Cardinality (number of elements) of set S (see also $\|s\|$)
$R \subset S$	–	R is a subset of S (including the case $R = S$)
$R \subsetneq S$	–	R is a proper/strict subset of S ($R \subset S$ and $R \neq S$)
$\|s\|$	–	Absolute value of scalar s (see also $\|S\|$ and $\mathrm{abs}_{\mathrm{lex}}(v)$)
$\\|v\\|_\infty$	3.1	Uniform norm of vector v
$\\|u - v\\|_\infty$	3.2	Uniform distance between vectors u and v
$\bar{B}_\infty(v; \varepsilon)$	3.3	Closed uniform neighbourhood of v with radius $\varepsilon \in \mathbb{R}^+$
$\preceq$ $(\prec, \succeq, \succ)$	3.4	Lexicographic order on $\mathbb{R}^p$
$\mathrm{sign}_{\mathrm{lex}}(v)$	3.7	Lexicographic sign of vector v
$\mathrm{abs}_{\mathrm{lex}}(v)$	3.9	Lexicographic absolute value of vector v
id	§2.2:1	Identity (activation) function (also identity permutation)
tanh	§2.2:2	Hyperbolic tangent (activation) function
relu	§2.2:3	Rectified linear unit (activation) function
$\mathcal{A}_h^{n,m}$	3.11	Simple neural network architecture with n input units, m output units, and h hidden units
$\mathcal{W}_h^{n,m}$	3.12	Space of simple neural network parameters for $\mathcal{A}_h^{n,m}$
$w \in \mathcal{W}_h^{n,m}$	3.13	Simple neural network parameter
f_w	3.18	Simple neural network function implemented by w

Symbol	Definition	Description
$\mathcal{F}_h^{n,m}$	3.18	Family of simple neural network functions for $\mathcal{A}_h^{n,m}$
$\mathcal{F}_\infty^{n,m}$	3.20	Extended family of simple neural network functions
T_π	3.32	Simple neural network permutation based on $\pi \in S_h$
T_σ	3.35	Simple neural network negation based on $\sigma \in \{-1,+1\}^h$
$\mathfrak{F}[w]$	3.38	Functional equivalence class of parameter w
$\mathfrak{R}[\mathcal{W}_h^{n,m}]$	3.40	Reducible region of parameter space $\mathcal{W}_h^{n,m}$
$\mathfrak{I}[\mathcal{W}_h^{n,m}]$	3.41	Irreducible region of parameter space $\mathcal{W}_h^{n,m}$
$\mathrm{rank}(w)$	4.1	Rank of simple neural network parameter w
$\mathrm{rank}(f)$	4.5	Rank of simple neural network function f
$\mathcal{R}_h^{n,m}$	4.7	Family of fixed-rank simple neural network functions
$\mathfrak{B}_r[\mathcal{W}_h^{n,m}]$	4.18	Bounded rank region of rank r of parameter space $\mathcal{W}_h^{n,m}$
$\Xi_{\mathrm{R}}(h,k)$	4.20	Set of reduction traces of length k on h units
$\Xi_{\mathrm{C}}(h,r)$	5.12	Set of canonicalisation traces of order r on h units
ρ	5.19	Piecewise linear path in a subset of the parameter space (the subset is implicit)
$\leftrightsquigarrow$	5.20	Piecewise linear path connectivity relation on a subset of the parameter space (the subset is implicit)
$\mathrm{prank}_\varepsilon(w)$	6.1	Parametric approximate rank of w given radius $\varepsilon \in \mathbb{R}^+$
$\mathcal{O}(f(n))$	–	Class of problems solvable within time asymptotically equivalent (or less) than $f(n)$ given instances of size n
$\mathcal{P}$	§3.4	Class of polynomial-time solvable decision problems
$\mathcal{NP}$	§3.4	Class of decision problems for which a "yes" output (accompanied by a certificate) is verifiable in polynomial time
$X \xrightarrow{\mathcal{P}} Y$	3.51	Decision problem X is (polynomial-time) reducible to decision problem Y
`SAT`	3.49	Boolean satisfiability
`xSAT`	3.56	Restricted Boolean satisfiability (see also Theorem 3.58 defining 3-`SAT`, `planar` 3-`SAT`, and `planar` 3-$\texttt{SAT}_3$)
`UPP`	6.15	Uniform point partition
`UPC`	6.17	Uniform point cover
`usgCP`	6.24	Clique partition for unit square graphs
$\texttt{UPP}^p$	6.37	Uniform point partition in $\mathbb{R}^p$ (as opposed to $\mathbb{R}^2$)
`PAR`	6.41	Parametric approximate rank (decision variant)

Symbol	Definition	Description
$\lVert x \rVert_2$	A.1	Euclidean norm of vector x
$\lVert x - y \rVert_2$	A.2	Euclidean distance between vectors x and y
$\mathscr{B}(\mathbb{R}^n)$	A.3	Borel σ-algebra on $\mathbb{R}^n$
q	A.3	Input (probability) measure on $\mathbb{R}^n$
$\int_{\mathbb{R}^n} f(x)\ dq(x)$	A.5	Integral of (q-integrable) function f with respect to q
$\mathscr{L}_2(q, \mathbb{R}^n, \mathbb{R}^m)$	A.8	Space of functions from $\mathbb{R}^n$ to $\mathbb{R}^m$ that are square integrable with respect to input measure q
$\lVert f \rVert_{\mathscr{L}_2(q)}$	A.12	$\mathscr{L}_2(q)$ seminorm of a (square-integrable) function f
$\lVert f - g \rVert_{\mathscr{L}_2(q)}$	A.13	$\mathscr{L}_2(q)$ distance (semimetric) between f and g
$\bar{B}_{\mathscr{L}_2(q)}(f; \varepsilon)$	A.15	Closed $\mathscr{L}_2(q)$ neighbourhood of f with radius $\varepsilon \in \mathbb{R}^+$
$\mathbb{M}_q^2$	A.16	Raw second moment of input measure q
$\mathbb{M}_q$	A.16	Square root of raw second moment $\mathbb{M}_q^2$
$\operatorname{frank}_{q,\varepsilon}(w)$	A.26	Function approximate rank of simple neural network parameter w given input measure q and radius $\varepsilon \in \mathbb{R}^+$
$\lceil a \rceil$	–	Ceiling bracket: least integer larger or equal to $a \in \mathbb{R}$