EPFL

Thèse n° 10 384

Safe Deep Neural Networks

Présentée le 7 février 2024

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 2
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Kyle Michael MATOBA

Acceptée sur proposition du jury

Prof. A. M. Alahi, président du jury
Prof. P. Vandergheynst, Prof. F. Fleuret, directeurs de thèse
Dr T. Gehr, rapporteur
Prof. A. Kalousis, rapporteur
Prof. M. Jaggi, rapporteur

École
polytechnique
fédérale
de Lausanne

2024

Acknowledgements

No one obtains a PhD without help, and I had a lot of help!

First, I must acknowledge those intrepid researchers who stuck with neural network research during the latest AI winter. I am delighted that they are now receiving the praise and admiration that their courage and determination deserves.

Secondly, François Fleuret. François is a scholar to be admired – not only for his impressive knowledge and instincts, but also his earnest enthusiam and curiousity. In a world of online posturing and optics, I am honored to have been advised by someone of such genuine integrity and calm professionalism. I'll put it in a way that I know he will find complementary: François is the ultimate "nerd's nerd".

Third, Martin Jaggi: in addition to a scientist of admirable acumen, Martin is a compelling leader and just such a nice guy. From interviewing me for admission to EPFL, to my first course, to these closing months, Martin has always been there for me.

Deepest thanks as well to Timon Gehr, Pierre Vandergheynst, Alexandros Kalousis, and Alexandre Alahi. It is a pleasure and honor to have such fine scholars involved in my viva.

I am grateful to the Swiss National Science Foundation for funding the bulk of my PhD under grant number FNS-188758 "CORTI". And, more generally, I am grateful to Switzerland and its people, this is an amazing country of considerate, competent, and sensible people. I am proud and thankful to call Switzerland my home.

Some quick ones. Thank you very much

- Olivier Ledoit for showing me a wholly original approach to life.
- Paul Giannaros for being such a steadfast friend and interested observer of my doctoral studies.

- Ramana Kumar for some eye-opening early discussions about AI safety.
- Alexander Aue and John Fernald for your support of my PhD applications.

To those many unsung heros who contribute to open source software: a heartfelt thank you! A lot has changed for computer geeks since I was young, and it is heartening that powerful, reliable software tools can still be had simply and freely. An enumeration of the userland software that was indispensable during my PhD is in Appendix C.

Obviously, I am deeply thankful to all the wonderful friends I have made during my PhD: Lie H., Jessica P., Ognjen G., Michael L., Fabio F., Raphaëlle L., Florian M. Andrei C., Vedrana K., Michel S., Alex U., Teja S., Evann C., Sepehr J., Angelos K., Suraj S., Mattia R., Ketan K., Arnaud P., and surely many others.

To my colleagues on the EPFL LLM project ([22], that became Chen et al. [27]): what an interesting endeavor! Matteo P., Amirkeivan M., Alejandro H-C., Zeming C., and Axel M., notably, I benefitted tremendously from your expertise.

Many thanks to the team at Créche la Charade in Charrat (especially Laurène A.), Anne F., Fabiene C. and Felicia W. enseignantes d'1-2H à l'École de Charrat, and my mother-in-law Hoa Kim Nguyễn for their loving care of Madeleine and Edward.

My father Tim, mother Carolyn, and sister Karina: I was an awkward, weird kid that grew into a awkward, weird adult. I was not always (or even often!) easy to love. But with your unwavering support I have been able to build myself a happy life.

To my wife Trung-Hieu: many loving and supportive partners would stop short of the sacrifices you made for this PhD, and for my crazy schemes more generally. But you did not. You have been "down for me" for most of my adult life at this point :).

And lastly, my children Madeleine and Edward: you make my life so fun and satisfying. Thank you for putting up with my lame dad jokes and my terrible French, and especially my lame dad jokes in my terrible French. I am incredibly proud of the thoughtful and capable young people you are becoming and I cherish our time together. Love you little Hoa Hiéu and Phong Dũng!

Charrat, Switzerland / Cameron Park, USA
January 13, 2024 Kyle

Abstract

The capabilities of deep learning systems have advanced much faster than our ability to understand them. Whilst the gains from deep neural networks (DNNs) are significant, they are accompanied by a growing risk and gravity of a bad outcome. This is troubling because DNNs can perform well on a task *most of the time*, but can sometimes exhibit nonintuitive and nonsensical behavior for reasons that are not well understood.

I begin this thesis arguing that closer alignment between human intuition and the operation of DNNs is massively beneficial. Next, I identify a class of DNNs that are particularly tractable and which play an important role in science and technology. Then I posit three dimensions on which alignment can be achieved – (1) philosophy: thought exercises to understand the fundamental considerations, (2) pedagogy: to help fallible humans interact effectively with neural networks, and (3) practice: methods to impose desired properties upon neural network, without degrading their performance.

Then I present my work along these lines. Chapter 2 analyzes *philosophically* the issues of using penalty terms in criterion functions to avoid (negative) side effects via a three-way decomposition into the choice of (1) baseline, (2) deviation measure, and (3) scale of the penalty. Chapter 3 attempts to understand which inputs a DNN maps to an output class. I present two approaches to this problem, which can help users recognize unsafe behavior, even if they cannot formulate safety beforehand. Chapter 4 examines whether max pooling can be written as the composition of ReLU activations in order to investigate an open conjecture that max pooling is essentially redundant. These studies advance our *pedagogical* grasp of DNN modelling. Finally, Chapter 5 engages with *practice* by presenting a method for making DNNs more linear, and thereby more human-compatible.

Keywords: AI Safety, Deep Neural Network Interpretability, Max pooling, Adversarial Robustness, Verification, Polytopes.

Résumé

Les capacités des systèmes d'apprentissage profond ont progressé bien plus rapidement que notre capacité à les comprendre. Bien que les avantages des réseaux de neurones profonds (RNP) soient substantifs, ils s'accompagnent d'un risque et d'une gravité croissants d'un mauvais résultat. C'est inquiétant car les RNPs peuvent bien functionner sur une tâche *la plupart du temps*, mais peut parfois agir de manière non intuitive et non sensible pour des raisons qui ne sont pas bien comprises.

Je commence cette thèse en affirmant qu'alignement plus étroit entre l'intuition humaine et le fonctionnement des RNPs serait extrêmement bénéfique. Ensuite, j'identifie une classe de RNPs particulièrement solubles et qui jouent un rôle important en science et technologie. Ensuite, je pose trois dimensions sur lesquelles l'alignement peut être réalisé – (1) philosophie : exercices de réflexion pour comprendre les considérations fondamentales, (2) pédagogie : pour aider les humains faillibles à interagir efficacement avec les réseaux de neurones, et (3) pratique : méthodes pour imposer les propriétés souhaitées au réseaux de neurones, sans dégrader leurs performances.

Ensuite, je présente mon travail dans ce sens. Chapitre 2 analyse *philosophiquement* les problèmes liés à l'utilisation de termes de pénalité dans les fonctions de critères pour éviter les effets secondaires (négatifs) via une décomposition à trois voies dans le choix de (1) la ligne de base, (2) la mesure de l'écart et (3) l'ampleur de la pénalité. Chapitre 3 tente de comprendre quelles entrées un RNP mappe à une classe de sortie. Je présente deux approches à ce problème, qui peuvent aider les utilisateurs à reconnaître un comportement pas "safe", même s'ils ne peuvent pas formuler "safety" au préalable. Chapitre 4 examine si le max pooling peut être écrit comme la composition des activations ReLU afin d'étudier une conjecture ouverte selon laquelle le max pooling est essentiellement redondant. Ces études font progresser notre compréhension *pédagogique* de la modélisation RNP. Enfin, Chapitre 5 s'engage dans la *pratique* en présentant une méthode pour rendre les RNPs plus linéaires, et donc plus solubles par les humaines.

Mots-clés : Sûreté de l'IA, Interpretation des réseaux de neurones profonds, Max pooling, Robustesse adversariel, Vérification, Polytopes.

Contents

Chapter 1

Introduction

1.1 Introduction

> "Civilization advances by extending the number of important operations which we can perform without thinking about them."

This quote, from Whitehead [170, page 46] and relayed in Russell [133], is a prescient summary of our times. Whitehead is ruminating on how elegant notation hides complexity and thereby allows users of mathematics to achieve much more than they could with clumsier representations of the same underlying truth. Viewed more expansively, it means that humanity's wellbeing is advanced not by technology that is known to possible, but rather by what can be done in a simple, scalable fashion.

Synonyms for "without thinking" in this context might be "commoditized", "ordinary", or "routine". And every scientific endeavour has a flavour of this hierarchy. Producing ammonia from atmospheric nitrogen can now be done "without thinking", though this was not the case in Whitehead's time. Synthetic production of human insulin has passed from the possible to the pedestrian in our lifetimes. The complete sequencing of a human's genome is currently becoming routine, and the production of graphene looks to become commoditized within our lifetimes. It is easy to see how these four human-welfare-improving breakthroughs have negligible utilitarian effect until they could be done without thinking. Quantum computation, for example, is now theoretically possible, but lacks any definite path to becoming routine.

Deep neural networks trained on large datasets using hardware accelerators, rep-

resent a novel regime where models can be very useful without being routine. The competence of DNNs has run far ahead of our understanding and they cannot be used "without thinking": teams of humanity's brightest cannot prevent embarassing and dangerous errors. In Whitehead's sense, then, there is no guarantee that DNNs, however powerful they become, will be ultimately beneficial.

Interpretation and control is the bottleneck on making DNNs beneficial. The full promise of deep learning will be realized only once it can be treated like a self-contained tool that performs high-level tasks in a predictable and intuitive fashion. This thesis tackles one aspect of the interpretation and control ("safety" in the title of this thesis is shorthand for "interpretation and control") of DNNs: engineering DNNs.

1.2 Engineering DNNs

1.2.1 Characterization

Engineering DNNs are my own concept. They are architecturally simple DNNs used in narrow applications arising in engineering and science. They map from low dimensional domains, representing real-world quantities with definite intepretations, to low dimensional ranges, frequently representing some action. People tend to have some intuition for how they should operate, for example that certain regions of the input space should map to certain outputs. In many cases, the data used to train engineering DNNs is generated in an automated fashion, such as in-silico simulation or sensor readings. Engineering DNNs operate primarily on tangible objects that humans tend to understand in terms of simple physical quantities. Note that "engineering" here is used as a noun, the discipline practiced by engineers, not as a verb ("engineering DNNs" $\neq$ the process of designing DNNs).

Figure 1.1 summarizes these features.

Some examples and explanation for each point above follows. "Operates in natural quantities, not human constructions" means that the individual dimensions of the input space could be, say, angles, speeds, temperatures, or pressures, but not abstract and detached from nature like pixel values, stock price, language toxicity, or click through rates. "Mapping from low dimensional spaces" complements the previous, as very high dimensional spaces are almost always detached from the physical world, such as word embeddings. "Mapping to low dimensional spaces" restricts the DNN capabilities: a DNN could, say, maximize paperclip production by optimizing

- Operates in natural quantities, not human constructions
- Map low dimensional spaces to low dimensional spaces
- Humans can recognize properties that they should satisfy
- Replace earlier vintages of handcrafted logic
- Data often acquired automatically via experiment or simulation
- Simple architectures

Figure 1.1: Aspects of a model that characterize engineering DNNs

a parameterized form for its shape *only*, and not by indirectly pursuing instrumental goals. "Humans can recognize properties that they should satisfy" means that, for example, in a DNN that predicts risk of colorectal cancer, inputs such as body mass index, alcohol consumption, and age should exhibit a positive dependence (Dvijotham et al. [48] show how to model monotonicity constraints in a DNN). "Replace earlier vintages of handcrafted logic" reflects that if humans minimally understand the nature of a relationship in the wild, they will try to build a model of it. For example a linear-quadratic regulator. "Data often acquired automatically via experiment or simulation" is similar: if scientists understand a phenomenon well enought to generate data in an automated fashion then it largely overlaps with the above. It includes DNNs that act as the policy function for a reinforcement learning agent, partial differential equations solvers, symbolic equation solvers, and the like.

"Simple architectures" is distinct and is explained further in Section 1.2.2. It means that the composition of any pair of adjacent layers can be understood, and the complexity of the network arises from the iterated application of many layers.

The promise of engineering DNNs is greater efficiency. However, replacing classical decision logic (satisfying intuitive properties by construction) with complex, generically unsafe DNNs opens the door to dangerous failure modes.

The concept of engineering DNNs is inspired by now-ubiquitous tools like CNC (computer numerical control) machining or spreadsheets. These technologies are impactful and beneficial, with few safety concerns, because they solve well-scoped problems transparently. These technologies do not advance the possible in the narrow sense of greenfield research – anything that these tools do was possible before their creation – but because of their ergonomics and economics, they expand the operations that

humanity can perform without thinking.

Engineering DNNs tend to address the fundamental aspects of existence: hunger, disease, physical danger, communication, transportation, shelter or sanitation, say. They tend not to make possible wholly new technologies such as predicting stock returns, automated number plate recognition, or detecting hate speech. Obviously, what constitutes a novel technology and what solves an inalienable difficulty of existence is open for debate, but this is a useful framework for thinking about the differential risk profile of application areas.

Bostrom [16] makes a profound observation: humans have no idea how much "smarter" (intelligence is a multifaceted concept, but here the magnitude alone suffices) than humans a future artificial general intelligence (AGI) might become. AGI that is dramatically smarter than us would be a severe threat, and engineering DNNs (almost) cannot be massively generally capable by construction.

At present, there is no clear path to using general DNNs without thinking, but for engineering DNNs there is perhaps hope. By better understanding the micro-foundations of DNNs, and devising methods for training models more amenable to analysis, as well as more targetted analysis methods, practical solutions seem possible, even if the problem is NP-complete in theory (Katz et al. [87]). That is the goal of this thesis.

1.2.2 Mathematical Form

The mathematical constraints on engineering DNNs are that they be "interpretable". I phrase it in this indirect fashion because what is interpretable will evolve, and such a formulation is more future-proof. That said, I do take a definite stance on what this means currently.

One condition that implies "interpretable" is that the composition of any pair of adjacent layers within a network can be comprehensively intuited by a numerate person. Th is the framework adopted here, and it is true if a network is comprised of only two types of operations: (1) linear, and (2) ReLU. This is restrictive is in some ways, but it *does* encompass more than it might seem at a first glance.

Firstly, this condition imposes no direct limits on the width or depth of a network. While clear, this fact should be appreciated: deep neural networks are powerful because of *scale* more than because they implement varied and complex logic.

Many operations are linear. Notably, convolution is a linear operation, as is average

pooling, batch norm, instance norm, and the like. Any kind of reindexing, repetition, padding, Kronecker product, etc. are also linear. Even Fourier transforms, used in Li et al. [101], are linear. Residual blocks (He et al. [73]) are not linear, but can be handled in a similar fashion via doubling the layer width, then summing the two halves of a layer (see Section 3.B).

Some layers are conventions or can be absorbed into the loss function. For example, the softmax layer that conventionally forms the final layer of DNN classifier. This layer can usually be elided if it is not necessary to interpret outputs as a probability distribution; for example, if classification alone suffices then a softmax layer is redundant since $\operatorname{argmax}_j x_j = \operatorname{argmax}_j \operatorname{softmax}(x)_j$.

ReLU can be used to construct most piecewise linear activations, for example hard tanh $= x \mapsto \mathrm{ReLU}(x+1) - 1 - \mathrm{ReLU}(x-1)$ (Collobert [33]), leaky ReLU $= x \mapsto \mathrm{ReLU}(x) - .01 \times \mathrm{ReLU}(-x)$ (Maas, Hannun, and Ng [104]), ReLU6 $= x \mapsto \mathrm{ReLU}(x) - \mathrm{ReLU}(x-6)$ (Krizhevsky [96]), and hard sigmoid $= x \mapsto (\mathrm{ReLU}(x+3) - \mathrm{ReLU}(x-3))/6$ (Courbariaux, Bengio, and David [34]). Unless there is a non-performance reason, such as efficiency or comparison with an existing model (such as in Chapter 5), usually piecewise linear activations suffice (there are techniques for bounding the difference of a general activation function to a piecewise linear approximation, such as Zhang et al. [174]).

Lastly, interpretability is a feature of the architecture of a network, not of any aspect of training. It admits any sort of initialization, optimizer, regularization, data augmentation, label smoothing, learning rate schedule, or stopping criterion. Anything that does not affect the operation of a network as a function. This means that any operation that is piecewise linear at inference time, such as dropout (Srivastava et al. [150]), shake-shake (Gastaldi [58]), or spectral normalization (Miyato et al. [112]) can be included in an engineering DNN.

As an example of an interpretable network, consider the "all convolutional net" of Springenberg et al. [147, Table 5]. This network repeatedly applies convolution, leaky ReLU, and dropout, with a final softmax layer at the end. Reasoning that: (1) convolution is linear, (2) leaky ReLU can be written as the composition of linear and ReLU operations, (3) dropout is linear at inference time, and (4) the ultimate softmax layer is a convention that can be absorbed into the optimization criterion, we conclude that this network is an engineering DNN.

Engineering DNNs do not include some useful primitives – notably attention. That said, the linear-ReLU structure of engineering DNNs are a crucial building block of essentially *all* DNNs. Thus, even if incomplete, interpreting engineering DNNs is a necessary step to an understanding of all DNNs. Or: if we cannot learn to fully interpret

and trust engineering DNNs, then it seems unlikely that we will succeed on more complex models.

1.3 Abbreviations

In this thesis, I use the following abbreviations:

AGI	Artifical General Intelligence
AUP	Attainable Utility Preservation ([161])
CIFAR	Canadian Institute for Advanced Research
CURE	Curvature Regularization ([115])
DARPA	Defense Advanced Research Projects Agency
DNN	deep neural network
EPFL	École Polytechnique Fédérale de Lausanne
GPU	graphics processing unit
LCNN	low curvature neural network ([149])
LLM	large language model
LP	linear program
ML	machine learning
MSE	mean squared error
NTK	neural tangent kernel
PGD	projected gradient descent ([105])
PRNG	pseudrandom number generator
QMC	quasi monte carlo
RL	reinforcement learning
RR	Relative Reachability ([93])
SEC	Securities and Exchange Commission
SGD	stochastic gradient descent
SVHN	Street View House Numbers ([117])
UAT	universal approximation theorem ([37])
VD	Value Difference ([93])

1.4 Notation

This section introduces the mathematical notation used throughout the thesis.

There is no equivalence between notation (capital, Greek, bold, tildes, etc.) and mathematical objects (tensors of any order, random variables, sets, mappings, etc.). Nonetheless, common conventions in form generally hold, for example f, g, h are generally functions, i, j, k are tensor indices, m, n, p are tensor dimensions, s and t are times, x, y, z are throwaway variables such as function arguments, ϵ generally means a small positive number, and similar.

Common abbreviations like "std", and near-ubiquitous mathematical constants like $e = 2.71828.$ are used without further introduction. For an integer n, $n!$ is "n factorial" $= n \times (n-1) \times (n-2) \times \ldots 2 \times 1$. O is the "big-O" notation: "$f(n)$ is $O(g(n))$" means that there exists n', m such that $n \geq n' \implies |f(n)| \leq m \times g(n)$. E (for "expectation" – $E[X]$ is the expectation of a random variable X), and $\Pr[A]$ is the probability of the event A. $a \triangleq b$ means that "a is defined to equal b".

For a matrix A, rank A gives the matrix rank of a matrix A, $\det A$ gives the determinant, $A^\dagger$ is the pseudoninverse of A, and $A^\top$ denotes matrix transpose. For $A \in \mathbb{R}^{r_1 \times c_1}, B \in \mathbb{R}^{r_1 \times c_2}, C \in \mathbb{R}^{r_2 \times c_1}, D \in \mathbb{R}^{r_2 \times c_2}$,

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

is the $(r_1 + r_2) \times (c_1 + c_2)$ (block) matrix where the first r_1 rows are the horizontal concatenation of A and B, and the $(r_1 + 1)$st through $(r_1 + r_2)$th rows are the horizontal concatenation of C and D.

I_n is the identity matrix in n dimensions. "0_n" and "1_n" denote n-dimensional vectors of zeros and ones. Throughout, the dimension will be dropped if it is clear. ι_j is a vector of size that will be evident from the context of all zeros, except for the jth element, which is 1. diag acts like in numpy and many other matrix languages, with two distinct meanings that will be clear from the context (1) mapping a vector $x \in \mathbb{R}^n$ to a matrix $\in \mathbb{R}^{n \times n}$ with ijth element equal to zero, unless $i = j$, in which case it is x_i, and also (2) extracting the diagonal of an $n \times n$ matrix. In Section 5.A.2 some more high-powered notation for tensors is necessary and will be introduced only there because it is clumsy and not needed elsewhere.

For a set A, A^n is the n-fold Euclidean product, e.g. $\{0,1\}^n$ is the set of all length-n vectors with 0 and 1 valued elements or $[0,\infty)^n$ is the set of n-dimensional nonnegative numbers. For a finite set A, $|A| \in \mathbb{N}$ denotes the size of A. For a set A and a function f, $f(A) = \{f(a) : a \in A\}$, and for two sets A and B, the set $A+B = \{a+b : a \in A, b \in B\}$ is the Minkowsi sum. For sets A, B $A \subsetneq B$ means that A is a strict subset of B (there is some $b \in B, \notin A$), and $A \backslash B$ means the subset of A that is not in B.

In my terminology $x \mapsto f(x)$ is an anonymous argument synonym for f. The bit after the colon in $f : D \to R$ is a way to optionally specify the domain (D) and range (R) of a function f, and the inverse of a function $f : X \to Y$ is denoted $f^{-1} : Y \to X$. "$\circ$" represents function composition, e.g. $(g \circ f)(x) = g(f(x))$. So, for example, $(g^{-1} \circ h^{-1})(S) = g^{-1}(h^{-1}(S)) = \{z : g(h(z)) \in S\}$.

$\mathbb{1}_A$ is the function given by $\mathbb{1}_A(x)$ equals 1 if $x \in A$ and 0 otherwise. $\mathbb{1}_A$ is often called the indicator function of A. $\text{softmax} : \mathbb{R}^n \to [0,1]^d$ is the function $x \mapsto \exp(x)/\sum_j \exp(x)_j$, and $\text{logsoftmax} = \log \circ \text{softmax}$. $||f||_n$ means the L_n-norm of f, i.e. $\left(\int |f(x)|^n \mathrm{d}x\right)^{1/n}$, for some measure $\int$ – possibly discrete – that will be obvious from the context. For a matrix A, $||A||_n$ means the n-norm of a matrix, for example $||A||_2$ is the spectral norm. $\nabla_x f$ means the derivative of f with respect to x, and $\nabla_x^2 f$ means the Hessian. When it is clear I drop the subscript to ∇. "==" is an infix binary function that is true if its two arguments are equal. This notation is used to emphasize that the equality of two quantities itself is of interest (for example in an algorithm).

1.5 Organization

What does it mean for DNNs to be more understandable in simple terms? An analogy is helpful. Consider games of chance prior to the development of probability theory. The bible mentions Roman soldiers gambling in the time of Jesus, whilst Pascal, Fermat, and Huygens did their early work (e.g. Hugeni [81]) in the 1650s. Thus, for thousands of years, people avidly and earnestly played games of chance for high stakes with an incomplete understanding of the basic principles.

Given the strange fallacies and superstitions that exist even today around gambling and investing one can only imagine the strange and mistaken views that even the most scientifically-minded people of this era held (see Cardano [24] for some examples). I can recall playing board and card games as a child, and can critically reflect on my own thought process. In retrospect, the heuristics guiding my play were generally correct, but missed some important dynamics and lacked a cohesive underlying principle.

What did coming to have a mature understanding of probability theory look like for me? It was a mix of abstract arguments and thought experiments (e.g. Pascal's wager, St. Petersburg paradox, martingale betting strategies), developing and refining the primitive concepts and notation (probability space, Law of Large numbers, standard deviation), reading empirical studies and seeing the principle demonstrated (for example, in Thorp [157]), and working out simplified examples.

This process echoes progress in probability more broadly. To coin a phrase: "philosophy, pedagogy, and practice" – in order to be useful, a concept needs to be well understood through profound thought-experiments, it needs to be developed in a matter that facilitates subsequent analysis, and it needs to admit practically interesting and relevant application.

That is the approach taken in this thesis: Chapter 2 introduces some of the fundamental concepts (philosophy), a method for clarifying and simplifying the operation of DNNs in proposed in Chapter 3, Chapter 4 presents a fundamental analysis of one primitive operation in DNNs (pedagogy), and Chapter 5 formulates and tests a theory-driven hypothesis about the empirical behavior of DNNs (practice). Although the methodology for each chapter is different, this thesis is unified in its *goal* – which is understanding – rather than predictive accuracy, computational efficiency, or theoretical generality. Engineering DNNs are the motivating class of networks, though the analysis will in places stray – for example to basic test problems in image classification – in order to more effectively tie it to the literature.

Chapter 2

Challenges for Using Impact Regularizers to Avoid Negative Side Effects

This chapter is substantively [102] and I would like to thank Andreas Krause, François Fleuret and Benjamin Grewe for their valuable comments on the original paper. I would also like to thank Ramana Kumar for introducing me to AI Safety, and the 2019-vintage members of the EPFL AI Safety reading group for many interesting discussions.

The paper grew out of a desire to know whether in the literature is there a satisfactory approach to stopping unanticipated side effects, even if it is not implementable in practice. Sadly, once we learned that the answer was negative, we turned to understanding the problem as well as possible, and to determine where future efforts might be best allocated. David Lindner, Alexander Meulemans, and myself were all equally involved with all aspects of the conception, analysis, and writing.

2.1 Introduction

Specifying a reward function in reinforcement learning (RL) that completely aligns with the designer's intent is a difficult task. Besides specifying what is important to solve the task at hand, the designer also needs to specify how the AI system should behave in the environment in general, which is hard to fully cover. For example, RL agents playing video games can learn to achieve a high score without solving the desired task

by exploiting the game engine (Saunders et al. [138]). Side effects occur when the behavior of the AI system diverges from the designer's intent because of considerations that were not anticipated beforehand, such as the possibility to exploit a game. This work focuses on side effects that are tied to the reward function, defined as side effects that would occur even if we could find an optimal policy for any reward function. We explicitly do not consider side effects resulting from the RL algorithm, which are often discussed using the term *safe exploration* (Garcıa and Fernández [57]).

In practice, the designer would typically iterate the reward specification to optimize the agent's performance and minimize side effects. This can be tedious and there is no guarantee that the agent will not exhibit side effects when it encounters new situations. In fact, such problems with misspecified reward functions have been observed in practical applications of RL (Krakovna et al. [94]).

In most situations, it is useful to decompose the reward $R(s)$ into a task-related component $R^{\text{task}}(s)$ and an environment-related component $R^{\text{env}}(s)$, where the latter specifies how the agent should behave in the environment, regardless of the task. We write the reward function only as a function of states only for simplicity, as the state-space can be formally extended to include the last action. R^{env} is more prone to misspecification, because it needs to specify everything that can happen beyond a task, which can result in undesired outcomes. Because the designer builds an RL agent to solve a specific problem, it is relatively easy to anticipate considerations directly related to solving the task in R^{task}. Shah et al. [142] points out that environments are generally already optimized for humans, hence defining R^{env} primarily requires specifying which attributes of the environment the AI systems should not disturb. Therefore, penalizing large changes in the current state of the world can be thought of as a coarse approximation for R^{env}.

Impact regularization has emerged as a tractable and effective way to approximate R^{env} (Armstrong and Levinstein [7], Krakovna et al. [93], and Turner, Hadfield-Menell, and Tadepalli [161]). The main idea is to approximate R^{env} through a measure of "impact on the environment", which avoids negative side effects and reduces the burden on the reward designer.

This chapter discusses impact regularizers of the form

$$R(s_t) = R^{\text{spec}}(s_t) - \lambda \cdot d(s_t, b(s_0, s_{t-1}, t)) \tag{2.1}$$

where s_t denotes the state at time step t, R^{spec} denotes the reward function specified by the designer, containing the specified parts of both R^{task} and R^{env}, and:

- the *baseline* $b(s_0, s_{t-1}, t)$ provides a state obtained by following a "default" or "safe" policy at timestep t and uses either the initial state and the current time (s_0, t) to compute it, or else the current state s_{t-1},
- d measures the *deviation* of the realized state from the baseline state, and
- $\lambda \geq 0$ gives a global *scale* at which to trade off the specified reward and the regularization.

Composing these three terms gives a general formulation of regularization that encompasses most proposals found in the literature, but permits separate analysis.

We start by giving an overview of the related work on impact regularizers (Section 2.2), before discussing the three main design decisions. First, we discuss how to choose a *baseline* (Section 2.3), emphasizing considerations of environment dynamics and a tendency for agents to offset their actions. Second, we discuss how to quantify *deviations* from the baseline (Section 2.4), especially the distinction between negative, neutral, and positive side effects. Third, we discuss how to calibrate the scale λ (Section 2.5) of the penalty. Finally, we propose some directions to improve the effectiveness of impact regularization (Section 2.6) .

The main contribution of this work is to discuss in detail the current main challenges of impact regularization, building upon previous work, and to suggest possible ways to overcome these challenges.

2.2 Related Work

Amodei et al. [3] reviewed negative side effects as one of several problems in AI safety, and discussed using impact regularization to avoid negative side effects. Since then, several concrete approaches to impact regularization have been proposed. Equation (2.1) generalizes the majority of these and gives a common framework for comparing approaches. Armstrong and Levinstein [7] proposed measuring the impact of the agent compared to the *inaction baseline*, starting from the initial state s_0. The inaction baseline assumes the agent does nothing, formalized by assuming a non-action exists. Armstrong and Levinstein [7] emphasized the importance of a semantically meaningful state representation for the environment when measuring distances from the inaction baseline.

While Armstrong and Levinstein [7] discussed the problem of measuring the impact

of an agent abstractly, Krakovna et al. [93] proposed a concrete deviation measure called *Relative Reachability* (RR). RR measures the average reduction in the number of states reachable from the current state, compared to a baseline state. This captures the intuition that irreversible changes to the environment should be penalized more, and has advantages over using irreversibility as a measure of impact (as in Eysenbach et al. [50]), such as permitting the quantification of the magnitude of different irreversible changes.

Turner, Hadfield-Menell, and Tadepalli [161] and Krakovna et al. [93] generalized RR to *Attainable Utility Preservation* (AUP) and *Value Difference* (VD) respectively, which share the same form for the deviation measure:

$$d_{\text{VD}}(s_t, s'_t) = \sum_x w_x f\left(V_x(s'_t) - V_x(s_t)\right), \tag{2.2}$$

where x ranges over some sources of value, $V_x(s_t)$ is the value of state s_t according to x, w_x is its weight in the sum and f is a function characterizing the deviation between the values. AUP is a special case of VD with w_x constant and $f = x \mapsto |x|$. This formulation captures the same intuition as RR, but measures the impact of the agent in terms of different value functions, instead of counting states. AUP measures the agent's ability to achieve high utility on a range of different goals in the environment, and penalizes any change that reduces this ability. Turner, Hadfield-Menell, and Tadepalli [161] introduced the *stepwise inaction baseline* to mitigate offsetting behavior (c.f. Section 2.3.2). This baseline follows an inaction policy starting from the previous state s_{t-1} rather than the starting state s_0.

Krakovna et al. [92] built upon the VD measure by introducing an auxiliary loss representing how well the agent could solve future tasks in the same environment, given its current state. This is a deviation measure in the form of Equation (2.1) that rewards similarity with a baseline instead of penalizing deviation from it. Eysenbach et al. [50]'s irreversibility penalty is a special case of Krakovna et al. [92].

Rahaman et al. [127] proposed learning an *arrow of time*. This directed reachability measure observes that irreversible actions leave the environment in a more disorderly state, making it possible to define an arrow of time with methods from thermodynamics. As another alternative to impact regularization, Zhang, Durfee, and Singh [176] and Zhang, Durfee, and Singh [175] propose that an AI learn how the environment can be changed by asking a human overseer and derived an active querying approach for maximally informative queries. Shah et al. [142] also learns which parts of the environment a human cares about by assuming that the world is optimized to suit

humans. Saisubramanian, Kamar, and Zilberstein [135] use a multi-objective Markov Decision Process to learn a separate reward function penalizing negative side effects and optimize this secondary objective while staying close to the optimal policy of the task objective. Saisubramanian, Zilberstein, and Kamar [136] provide a broad overview of the various approaches for mitigating negative side effects, while we zoom in on one class of approaches, and discuss the corresponding challenges in detail.

2.3 Choosing a Baseline

Recent work mainly uses two types of baselines: (i) the inaction baseline $b(s_0, s_t, t) = T(s_t|s_0, \pi_{\text{inaction}})$ and (ii) the stepwise inaction baseline $b(s_0, s_t, t) = T(s_t|s_{t-1}, \pi_{\text{inaction}})$, where T is the distribution over states s_t when starting at state s_0 or s_{t-1} respectively and following the inaction policy π_{inaction} that always takes an action a_{nop} that does nothing.

Unfortunately, the inaction baseline can lead to undesirable offsetting behavior, where the agent tries to undo the outcomes of the task after collecting the reward, by moving back towards the initial baseline. The stepwise inaction baseline removes the offsetting incentive of the agent by branching off from the previous state instead of the starting state (Turner, Hadfield-Menell, and Tadepalli [161]). However, Krakovna et al. [92] argued that offsetting behavior is desirable in many cases. Section 2.3.2 contributes to this discussion by breaking down in detail when offsetting behavior is desirable, and Section 2.3.3 argues that the inaction baseline and step-wise inaction baseline can lead to *inaction incentives*. We start by observing that the inaction baseline and stepwise inaction baseline do not always represent safe policies in Section 2.3.1.

2.3.1 Inaction Baselines are not Always Safe

The baseline used in impact regularization should be a safe policy where the AI system does not harm its environment or itself. In many cases, taking no actions is a safe policy for the agent, e.g. for a cleaning robot. However, if the AI system performs a task requiring continuous control, inaction can be disastrous. For example, for an agent driving a car, doing nothing likely results in a crash. This is particularly problematic for the stepwise inaction baseline, which follows an inaction policy starting from the previous state. The inaction policy starting from the initial state can be unsafe, for example, if an agent takes over the control of the car from a human, and therefore the initial state s_0 already has the car driving.

For this reason, designing a safe baseline for a task or environment that requires continuous control is a hard problem. One possible approach is to design a policy that is known to be safe based on expert knowledge. However, this can be a time-consuming process, and is not always feasible. Designing safe baselines for tasks and environments that require continuous control is an open problem that has to be solved before impact regularization can be used in these applications.

2.3.2 Offsetting

An agent engages in offsetting behavior when it tries to undo the outcomes of previous actions. Offsetting behavior can be desirable or undesirable, depending on which outcomes the agent counteracts.

Undesirable offsetting

Impact regularizers with an inaction baseline can lead to undesirable offsetting behavior where the agent counteracts the outcomes of its task (Turner, Hadfield-Menell, and Tadepalli [161]). For example, Krakovna et al. [93] consider a vase on a conveyor belt. The agent is rewarded for taking the vase off the belt, preventing it from falling off the belt. The desired behavior is to take the vase and stay put. The offsetting behavior is to take the vase off the belt, collect the reward, and afterwards put the vase back on the conveyor belt to reduce deviation from the baseline. To understand this offsetting behavior recall the decomposition of the true reward into a task-related and an environment-related component from Section 2.1. A designer usually specifies a task reward $R^{\text{task}}_{\text{spec}}$ that rewards states signaling task completion (taking the vase off the belt). However, each task has consequences for the environment, which often are the reason why the task should be completed in the first place (the vase being not broken). In all but simple tasks, assigning a reward to every task consequence is impossible, and so by omission, they have a zero reward. When impact regularization penalizes consequences of completing the task, because they differ from the baseline, this results in undesirable offsetting behavior. The stepwise inaction baseline from Turner, Ratzlaff, and Tadepalli [162] successfully removes offsetting incentives. However, in other situations offsetting might be desired.

Desirable Offsetting

In many cases, to prevent unnecessary side effects, offsetting is desired. Krakovna et al. [92] give an example of an agent which is asked to go shopping, and needs to open the front door of the house to go to the shop. If the agent leaves the door open, wind can knock over a vase inside, which the agent can prevent by closing the door after leaving the house. When using the stepwise inaction baseline (with rollouts, c.f. Section 2.4.2), the agent is penalized once when opening the door for knocking over the vase in the future, independent of whether it closes the door afterwards or not. Hence, for this example, the offsetting behavior (closing the door) is desirable. The reasoning behind this example can be generalized to all cases where the offsetting behavior concerns states that are instrumental towards achieving the task (e.g. opening the door) and not a consequence of completing the task (e.g. the vase being not broken).

A Need for a New Baseline

Recently proposed baselines either remove offsetting incentives altogether or allow for both undesirable and desirable offsetting, which are both unsatisfactory solutions. Krakovna et al. [92] proposed allowing all offsetting (e.g. by using the inaction baseline) and rewarding all states where the task is completed in the specified reward function. However, there are three important downsides to this approach. First, states that occur after task completion can still have negative side effects; if the reward in these states is high enough to prevent offsetting, the agent may pursue them and ignore their negative side effects. Second, not all tasks have a distinct goal state that indicates the completion of a task, but rather accumulate task-related rewards over time steps during an episode. Third, since rewards continue after task completion, the agent will try to prevent being shut down, a concern raised by Hadfield-Menell et al. [70].

In conclusion: offsetting is still an unsolved problem, with no baseline able to prevent undesirable offsetting behavior but allow for desirable offsetting.

2.3.3 Environment Dynamics and Inaction Incentives

In dynamic environments that are sensitive to the agent's actions, there will be *inaction incentives*. Either the agent will be insufficiently regularized and possibly exhibit undesired side effects (for small λ) or will not act at all (otherwise).

Sensitivity to Typical Actions

One useful refinement for improving upon a simple action/inaction dichotomy is to distinguish *typical* actions. An action is *typical* if it is commonly used for solving a wide variety of tasks (e.g. moving). In an environment that is sensitive to typical actions, impact regularizers with the current baselines may excessively prevent the agent from engaging in normal operations. Thus, by penalizing typical actions less, it may be possible to discourage atypical actions (e.g. discharging weaponry), without impeding the normal operation of the agent.

Agent Capability and State Features

The inaction incentive is more apparent for agents highly capable of understanding the detailed consequences of their actions, for example with a powerful physics engine. For agents that can very accurately predict the implications of their actions, an accompanyingly intelligent impact regularizer is also necessary. Correctly discounting for uncertainty over factors outside of the agent's control is a clear refinement. Other possible steps could be to specifically distinguish and downweight higher-order consquences or even adapt methods from psychology to combat "analysis paralysis".

A related point is that one should not represent states with overly fine-grained features, as too much information risks an agent making decisions on irrelevancies (Armstrong and Levinstein [7]). For example, it would be counterproductive for an demand forecasting agent in an online sales environment to model each customer separately, when broader aggregates would suffice. However, there are two issues with this approach to mitigate the inaction incentive. First, the intrinsic dynamics of the environment remains sensitive to small perturbations, even under coarser features (e.g. the specific weather conditions). Second, for advanced AI systems, it might be beneficial to change their feature representation to become more capable of predicting the consequences of their actions. In this case, one would have no control over the granularity of the features.

2.4 Choosing a Deviation Measure

A baseline defines a "safe" counterfactual to the agent's actions. The deviation measure determines how much a deviation from this baseline should be penalized or rewarded. Currently, the main approaches to a deviation measure are the relative reachability

(RR) measure, the attainable utility preservation (AUP) measure and the future task reward (Krakovna et al. [92]). AUP and future task reward require a specification of which tasks the agent might want to achieve in future. This section argues that the current deviation measures still require specifying the *value* of the impact to avoid unsatisfactory performance of the agent and that new rollout policies should be designed that allow for a proper incorporation of delayed effects into the deviation measure.

2.4.1 Which Side Effects are Negative?

An impact regularizer is meant to approximate R^{env} for all states in a tractable manner. It does this by penalizing impact on the environment, built upon the assumption that the environment is already optimized for human preferences (Shah et al. [142]). The impact regularizer aims to penalize impact proportionally to the magnitude of this impact assumed to be the magnitude of the side effect. However, not all impact is negative – impact can be neutral or even positive. R^{env} does not only consider the magnitude the impact on the environment, but also to which degree this impact is negative, neutral or positive. Neglecting the associated value of impact can lead to suboptimal agent behavior, as highlighted in the example below.

Example: The Chemical Production Plant

Consider an AI system controlling a plant producing a chemical product for which various reactions exist, each producing a different combination of waste products. The task of the AI system is to optimize the production rate of the plant. To minimize the impact of the plant on the environment, the reward function of the agent is augmented with an impact regularizer, which penalizes the mass of waste byproducts, compared to an inaction baseline (where the plant is not operational). Some waste products are harmless (e.g. O_2), whereas others can be toxic. When the deviation measure of the impact regularizer does not differentiate between negative, neutral or positive impact, at any λ the AI system would choose a toxic byproduct as long as it is emitted at a lower rate per unit of output. And moreover, if the AI were to discover a technique to costlessly sequester carbon dioxide alongside its other tasks, a naïve impact regularizer would not tend to adopt it.

Value Differences

To distinguish between positive, neutral and negative side effects, an approximation of R^{env} that goes beyond measuring impact as a sole source of information is necessary. Attainable utility preservation allows for differentiating between positive and negative impact by defining the deviation measure as a sum of differences in value between a baseline and the agent's state-action pair for various value functions. Hence, it is possible to reflect how much the designer values different kinds of side effects in these value functions. However, the challenge remains to design value functions that approximate R^{env} sufficiently on the complete state space, which is again prone to reward misspecification. So although the value difference framework permits specifying values for side effects, *how* to specify this notion of value is still an open problem.

2.4.2 Rollout Policies

The actions of an agent are not always immediately apparent after taking the action. The stepwise inaction baseline (Turner, Hadfield-Menell, and Tadepalli [161]) ignores all actions that took place before $t-1$, hence, to correctly penalize delayed effects, the deviation measure needs to incorporate future effects. This can be done by collecting rollouts of future trajectories using a model of the environment. These rollouts depend on which *rollout policy* is followed by the agent in the simulation. For the baseline states, the inaction policy is logical. For the future effects of the agent's action, it is less clear which rollout policy should be used. Turner, Hadfield-Menell, and Tadepalli [161] use the inaction policy. considering a rollout where the agent takes its current action, after which it cannot take any further actions. This approach has significant downsides, because it may not allow the agent to consider future actions when determining the impact penalty (e.g. the agent can take an action to jump, but cannot plan for its landing accordingly in the rollout). Therefore, it seems that future work should develop rollout policies different from the inaction policy, such as the current policy of the agent.

2.5 Choosing the Magnitude of the Regularizer

To combine an impact regularizer with a specified reward function, the designer must choose the magnitude of the regularizer λ. Turner, Hadfield-Menell, and Tadepalli [161] say that "loosely speaking, λ can be interpreted as expressing the designer's beliefs about the extent to which R [the specified reward] might be misspecified".

If λ is too small, the regularizer may not reduce the risk of undesirable side effects. If λ is too big, the regularizer will overly restrict necessary effects of the agent on the environment, and the agent will be less effective at achieving its goal. While the regularizers proposed by Krakovna et al. [93] and Turner, Hadfield-Menell, and Tadepalli [161] already measure utility, in general λ must also handle a unit conversion of the regularizer to make it comparable with the reward function.

Some intuition for choosing λ comes from a Bayesian perspective, where the regularizer encodes prior knowledge and λ controls how far from the prior the posterior should have moved. Another distinct view on setting λ comes from the dual optimization problem, where it represents the Lagrange multiplier on an implied set of constraints: λ is the magnitude of the regularizer for which the solution to the penalized optimization problem coincides with a constrained optimization problem. Hence, the designer can use λ to communicate constraints to the AI system, which is a natural way to pose some common safety problems (Ray, Achiam, and Amodei [129]).

Armstrong and Levinstein [7] discuss tuning λ and note that unintuitively, the region of useful λs can be small and hard to find safely, for example, by starting with a high λ and reducing it until the agent achieves the desired behavior. For a fixed step-size in decreasing λ, tuning might always jump from a λ implying inaction, to a λ that yields unsafe behavior. The same holds for other common procedures to tune hyperparameters.

2.6 Ways Forward

This section puts forward promising future research directions to overcome the challenges discussed in the previous sections.

2.6.1 A Causal Framing of Offsetting

Section 2.3.2 highlighted that some offsetting behavior is desired and some undesired. To design an impact regularizer that allows for desired offsetting but prevents undesired offsetting, one first needs a mechanism that can predict and differentiate between these two types of offsetting. Undesired offsetting concerns the environment states that are a consequence of the task. The difficulty lies in determining which states are a causal consequence of the task being completed and differentiate them from states that could have occurred regardless of the task.

When the goal is to reach a certain state, the consequences of performing a task can be formalized in a causal framework (Pearl [123]). Given a causal graph of the environment-agent-interaction, the states that are a consequence of the task can be obtained as the causal children nodes of the goal state. Hence, a baseline that allows for desired offsetting but prevents undesired offsetting prevents the agent from interfering with the children nodes of the goal states, while allowing for offsetting on other states.

Not all tasks have a distinct goal state indicating the completion of a task, but accumulate instead task-related rewards during an episode. Extending this argument to general tasks remains an open issue, for which causal influence diagrams (Everitt et al. [49]) can provide a mathematical framework.

2.6.2 Probabilities as Baseline

Armstrong and Levinstein [7] argue that probabilities are better suited than counterfactuals for measuring the impact of actions. Because the baseline is one specific trajectory, it can differ considerably from the actual trajectory of the agent in environments that exhibit random dynamics. One approach to a more robust measure of the agent's impact on the environment is to compare probabilities that marginalize over all external perturbations instead of comparing specific trajectories. Specifically, the difference between the probability of reaching a state given that the actions that the agent took, and that under the baseline. All influences of perturbations that did not arise from the agent are marginalized out (averaged over) in these probabilities. Hence, a divergence measure between these two probabilities can give a more robust measure of potential impact of the agent, without being susceptible to non-necessary inaction incentives. To the best of our knowledge, this idea has not yet been implemented as a concrete impact regularization method.

2.6.3 Improved Human-Computer interaction

Side effects occur if there is a difference between the outcome an AI system achieves and the intent of its (human) designer. Thus improving how well the designer can communicate their intent to the AI system is an important aspect of eliminating side effects (Leike et al. [100]). This emphasis on the human component of learning to avoid negative side effects connects it closely to the problem of *scalable oversight* highlighted by Amodei et al. [3].

Improved Tools for Reward Designers

Commonly, when choosing an impact regularizer, a designer will iteratively test the choice of baseline, deviation measure, and regularization strength in a sequence of environments that increasingly resemble the production environment. At each iteration, the designer identifies and corrects weaknesses, such that the criterion being optimized becomes increasingly true to the designer's intent. For example, an AI with the goal to trade financial assets may be run against historical data ("backtested") in order to understand how it might have reacted in the past, and presented with deliberately extreme inputs ("stress-tested") in order to understand likely behavior in out of sample situations. To design a reward function and a regularizer, it is crucial for the designer to be able to understand how the system would react in novel situations and how to fix it in case it exhibits undesired behavior. Further research to increase the designer's ability to understand how a system will react, will substantially help the designer to communicate their intent more effectively. Work in the direction of *interpretability* such as Gilpin et al. [63], and *verification* for example Huang et al. [80] of machine learning models appear promising.

Actively Learning from Humans

Considering the problem from the perspective of the AI system, the goal is to better understand the designer's intent, especially in unanticipated scenarios. Instead of the designer *telling* the system their intent, the system can also *ask* the designer about their intent. To decide what to ask the designer, the system may be able to determine which states it is most uncertain about, even if it is not able to accurately ascribe values to some of them. Recent work, such as Christiano et al. [29], shows that such an approach can be effectively used to learn from the human about a task, but it may also be used to learn about the constraints of the environment and which side effects are desired or undesired (Zhang, Durfee, and Singh [176]). Active learning could also provide a different perspective on impact regularizers: instead of only penalizing impact on the environment, a high value of the regularization term could be understood as indicating feedback is needed. In particular, this approach could help to resolve situations in which a positive task reward conflicts with the regularization term.

2.7 Conclusion

Avoiding negative side effects in systems that have the capacity to cause harm is necessary to fully realize the promise of AI. This chapter discussed a popular approach to reduce negative side effects: impact regularization. We discussed the practical difficulty of choosing each of the three components: a baseline, a deviation measure and a regularization strength. Furthermore, we pointed to fundamental problems that are currently not addressed by state-of-the-art methods, and presented several new future research directions to address them. While our discussion showed that current approaches still leave significant opportunities for future work, impact regularizers are a promising idea for building the next generation of safe AI systems, and we hope that our discussion is valuable for researchers trying to build new penalization functions.

Chapter 3

Inversion and Safety

This chapter is mostly an expanded version of Matoba and Fleuret [109]. Section 3.4 is Matoba [107]. I wish to thank Arnaud Pannatier, Suraj Srinivas, Martin Jaggi, and Pascal Frossard for their valuable feedback on the original works.

The guiding principle of this chapter is that although it may be difficult or impossible to characterize *ex nihilo* failure modes, it is relatively easy for a domain expert to recognize when shown examples. This meta-principle holds across fields. The phrase "I know it when I see it" (cf. Gewirtz [60]) is used to describe situations where a comprehensive definition is difficult, but judging individual instances is trivial. A proactive approach to DNN safety is to build tools to enable neural network designers to "know it when they see it" by inspecting their networks directly via the input-output mapping at senstive inputs.

3.1 Introduction

Although deep neural networks (DNNs) can achieve excellent predictive accuracy, reasoning about their performance is difficult, even for experts. Our goal is to enable non-expert stakeholders, such as clinical health workers, investors, or military commanders to trust a statistical model in high-stakes environments. To do this, we posit that decisionmakers want to understand a model in both directions, both from inputs to outputs, but also being able to start with outputs, and understand the inputs that lead to them.

This chapter develops an equivalent representation of a certain class of DNN classifiers.

This representation requires only a basic numeracy to productively interact with and can be used by domain experts to build intuition and trust. We apply this method to a reinforcement learning agent trained to solve the cart-pole problem, and find that a DNN implementing a successful policy makes a particular type of mistake on 24% of the mass of the 1/8th of the state space for which the optimal action is known (Section 3.3.2). Section 3.3.3 shows how using the preimage in place of verification can yield an efficient and interpretable end-to-end system for analyzing aircraft collision avoidance systems.

3.1.1 Previous Work

DNNs have the property that knowing the output tells us very little about the input it corresponds to. This is most apparent in image classifiers, where totally different outputs can arise from inputs that are visually indistinguishable, see Szegedy et al. [152]. This chapter builds upon the mathematical framework developed for verification of DNNs, for example Tjeng, Xiao, and Tedrake [158] and Wong and Kolter [171]. Table 3.1 orients the present work to the literature.

Commonly called	What is computed	Examples
Verification	$(f, X, Y) \mapsto f^{-1}(Y) \cap X == \emptyset (= f(X) \cap Y == \emptyset)$	[171]
Reachability	$(f, X) \mapsto f(X)$	[173]
Inversion	$(f, y) \mapsto f^{-1}(\{y\})$	[25]
Preimage	$(f, Y) \mapsto f^{-1}(Y)$	Present work

Table 3.1: A taxonomy of previous work on inversion and verification. Here $f : \mathbb{R}^{n_1} \to \mathbb{R}^{n_L}$ is a DNN, $X \subseteq \mathbb{R}^{n_1}$, $x \in \mathbb{R}^{n_1}$, $Y \subseteq \mathbb{R}^{n_L}$, and $y \in \mathbb{R}^{n_L}$.

Phrasing verification in this fashion facilitates comparison with the other points. In image classification, X could be an epsilon ball around an input, with Y the halfspace where one coordinate is higher than all others.

High level analyses typically entails many verifications: for example, studies often verify the absence of adversarial examples around the entire training set, or give a complete tradeoff between adversarial robustness and accuracy. Reachability is an interesting extension to verification in that it computes the entire image of convex input set. For example, the set of all logits attained within an epsilon ball around a data point. Given this image, a verification can be performed by seeing if the image intersects with a set Y, and any number of useful properties besides, with minimal additional computation. Thus, reachability can be significantly more useful than

verification for answering higher-level modelling questions.

Computing about the outputs that arise from a set of inputs is only half of the picture, however. Ascertaining the inputs corresponding to a set of outputs gives a workable approach to anticipate unforseen problems. And, since $f^{-1}(Y) \cap X = \emptyset \iff f(X) \cap Y = \emptyset$ gives a clear connection to verification. Carlsson, Azizpour, and Razavian [25] and Behrmann et al. [12] are oriented backwards: they reconstruct the inputs that result in an output. These papers study the statistical invariances that nonlinear layers encode. Behrmann et al. [12] examines the preimage of a single point through a single ReLU layer, analyzing stability via an approximation-based experiment. Carlsson, Azizpour, and Razavian [25] analyzes theoretically the preimage of a single point through the repeated application of a nonlinearity. This chapter looks at the preimage of non-singleton subsets of the codomain, and requires considerable extension to their approaches.

3.2 Method

The method is easily stated: it builds up the preimage of a DNN from the preimage of its layers, using analytical formulae. We start by developing some properties of the preimage operator, then we describe the class of sets that we compute the preimage of, and finally we discuss the class of DNNs that our algorithm addresses.

3.2.1 Preimage Properties

Abstractly, a "layer" in a feedforward neural network is a function, and the entire neural network is the composition of the action of each layer. Let $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ denote the network in this form, where the function f_i characterizes the ith layer. Layers operate on flattened tensors, for instance that a 2d convolutions on $3 \times 32 \times 32$ tensor takes as input 3072×1 vectors. This convention enforces compatibility with the polytope formulation (Section 3.2.2), and amounts to a simple reindexing at worse.

Lemma 1 shows how to build up the preimage of a DNN from the preimages of its constitutent layers. The proof of all statements is in Section 3.A.

Lemma 1 (Preimage of composition is reversed composition of preimages)**.** *For functions $f_j : \mathbb{R}^{n_j} \to \mathbb{R}^{n_{j+1}}$,*

$$(f_{\ell+k} \circ f_{\ell+k-1} \circ \ldots \circ f_\ell)^{-1} = f_\ell^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_{\ell+k}^{-1}. \tag{3.1}$$

Secondly, is an intuitive property of f^{-1} that is handy for building up the preimage of any set from the preimages of any partition of that set.

Lemma 2 (Preimage of union is union of preimages)**.**

$$f^{-1}\left(\cup_{i=1}^{N} S_i\right) = \cup_{i=1}^{N} f^{-1}(S_i).$$

3.2.2 Polytopes

The method is not applicable to arbitrary sets Y, but rather to sets that are defined by piecewise linear inequalities. The basic building block of these sets are polytopes.

Definition 1 (Polytope)**.** *A* polytope *in* $\mathbb{R}^n$ *is a set that can be written as* $\{x \in \mathbb{R}^n : b - Ax \geq 0\}$ *for some* $m \in \mathbb{N}$*,* $b \in \mathbb{R}^m$*, and* $A \in \mathbb{R}^{m \times n}$*.*

Definition 1 does not require that polytopes be bounded, but polytopes are convex. Sets with linear boundaries, though generally non-convex, can decomposed into the union of polytopes. We term such sets *region-unions*, and the set of polytopes which comprise them, *regions*.

Definition 2 (Region and region-union)**.** *For* $N \in \mathbb{N}, b_i \in \mathbb{R}^{m_i}$*,* $A_i \in \mathbb{R}^{m_i \times n}$*, with* $m_i \in \mathbb{N}$*, a region is a set of polytopes,* $\{\{x : b_i - A_i x \geq 0\}; i = 1, \ldots, N\}$*. A region-union is a set* $\cup_{r \in R} r$ *for some region* R*.*

Region-unions are interesting because the preimage of polytopes under piecewise linear functions are regions-unions. Thus, the preimages of region-unions through both ReLU and linear layers are region-unions. It is necessary to also keep information on how to form a region-union, hence the notion of a region. Note that if R_1 and R_2 are regions, then $R_1 \cup R_2$ is a region, and correspondingly for region-unions.

3.2.3 Linear and ReLU Polytope Preimages

This section introduces formulae for the preimage of linear and ReLU functions, giving content to Lemma 1. The preimage of a polytope under linear mappings is a polytope:

Lemma 3 (Preimage of Linear Layer)**.**

$$(x \mapsto Wx + a)^{-1}(\{x : b - Ax \geq 0\}) = \{x : (b - Aa) - AWx \geq 0\}. \tag{3.2}$$

ReLU is a piecewise linear function, so carefully treating the subsets of the domain on which it exhibits different behavior, gives a similar formulation for each:

Lemma 4 (Preimage of ReLU Layer)**.**

$$\begin{aligned}&\mathrm{ReLU}^{-1}(\{x : b - Ax \geq 0\}) \\ &= \bigcup_{v \in \{0,1\}^n} \left\{x : b - A\mathrm{diag}(v)x \geq 0, -\mathrm{diag}(1-v)x \geq 0, \mathrm{diag}(v)x \geq 0\right\}.\end{aligned} \tag{3.3}$$

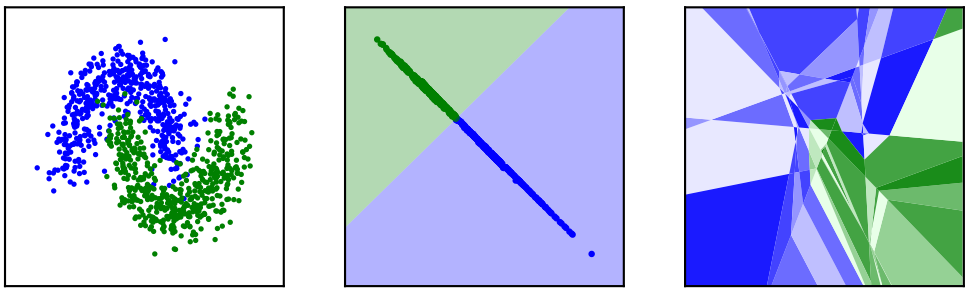
To understand Lemma 4 let $s(x)$ be the vector given by $s(x)_i = 1$ if $x_i \geq 0$ and zero otherwise. Then $\mathrm{diag}(s(x))x = \mathrm{ReLU}(x)$. This expression separates $x \mapsto \mathrm{ReLU}(x)$ into a pattern of signs over its coordinates and x itself. Thus, restricting attention to a set on which the sign does not change, familiar linear algebra routines can be used to compute the preimage set, akin to Lemma 3. The nonnegative values are denoted by $v \in \{0,1\}^n$ in the above, and the set of x such that $x_i \geq 0 \iff v_i = 1$ is given by $\mathrm{diag}(v)x \geq 0$. Similarly, $x_i \leq 0 \iff v_i = 0$ for $i = 1, 2, \ldots, n$ if and only if $-\mathrm{diag}(1-v)x \geq 0$. Equation (3.3) follows by partitioning $\mathbb{R}^n$ into the 2^n sets where each coordinate is nonnegative or not.

Computing the preimage of a ReLU layer is intractable at scale, though the problem exhibits considerable structure. The fundamental problem of computing a preimage is the same as that of verification – searching all possible sign patterns – thus it appears likely that it is possible to compute the preimage of networks of a similar scale to those that can be verified. Preimages are most insightful and useful when the inputs and outputs have definite interpretation. Thus, it is possible and useful to compute the preimage of engineering DNNs.

3.3 Experiments

3.3.1 Two Moons Classification

The first experiment fits a DNN $f : [-3, +3]^2 \to \mathbb{R}^2$ consisting of two nonlinear layers with eight neurons each, on an instance of the "two moons" dataset. This data is shown in Figure 3.1a (further details of details of f and the data are in Section 3.D.1). Figure 3.1b plots the corresponding logits, along with the sets to be inverted $\{x : x_1 \lesseqgtr x_2\} \subseteq \mathbb{R}^2$. Figure 3.1c shows the corresponding preimages, with different hues of the same color corresponding to different sign patterns v in Equation (3.3).



(a) (b) (c)

Figure 3.1: Inversion of a simple DNN $\mathbb{R}^2 \mapsto \mathbb{R}^2$ fit to the "two moons" data shown in Figure 3.1a. Figure 3.1b is the logits from a simple DNN corresponding to each data point, along with the decision boundary. Figure 3.1c shows the preimages comprised of polytopes that form the region-union.

3.3.2 Cart-pole Reinforcement Learning Agent

In the "cart pole" control problem a pole is balanced atop a cart which moves along a one dimensional track (Figure 3.2). Gravity pulls the pole downward, the falling of the pole pushes the cart, and external movement of the cart pushes the pole in turn. The goal is to keep the pole upright by accelerating the cart.

In the formulation of Brockman et al. [19] controller inputs are: the position of the cart, x, velocity of the cart $\dot{x}$, the angle of the pole from upright θ, and the angular velocity of the pole $\dot{\theta}$. Possible actions are to accelerate the cart in the positive or negative x direction. The environment encourages balancing by a unit reward per period before failure, where failure means that the pole is not sufficiently upright ($\theta \notin [-\pi/15, +\pi/15]$), or the cart is not near enough the origin ($x \notin [-2.4, +2.4]$). There are no prescribed limits for $\dot{x}$ and $\dot{\theta}$, but via a methodology described in Section 3.D.2, these states take values in $[-3.0, +3.0] \times [-3.5, +3.5]$.

Consider a still cart and pole ($\dot{x} = \dot{\theta} = 0$), with the cart left of zero ($x \leq 0$) and the pole left of vertical ($\theta \leq 0$). Keeping x and θ near zero is preferable, since these are further from failure, so moving left will steady θ but worsen x. Nonzero velocities are more complicated, but one configuration is unambiguous: if $x \leq 0, \dot{x} \leq 0, \theta \geq 0, \dot{\theta} \geq 0$, then pushing right is clearly the correct action. Figure 3.2 depicts a value in this orthant. Let $D_{+1} = (-\infty, 0]^2 \times [0, \infty)^2$, and correspondingly, let $D_{-1} = [0, +\infty)^2 \times (-\infty, 0]^2$.

We fit a one hidden layer neural network control function $f : \mathbb{R}^4 \to \mathbb{R}^2$ using policy

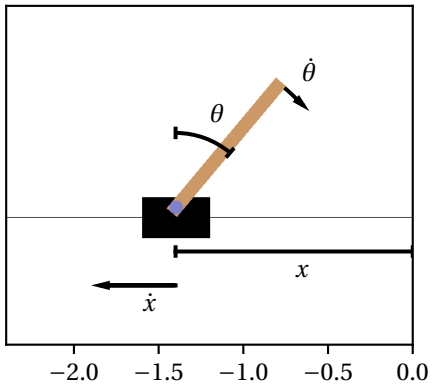


Figure 3.2: The state space of the cart pole problem, schematically. Here $x \le 0$ (the cart is left of the origin), $\dot{x} \le 0$ (the cart is moving leftward), $\theta \ge 0$ (the pole is right of vertical), and $\dot{\theta} \ge 0$ (the pole is moving rightward).

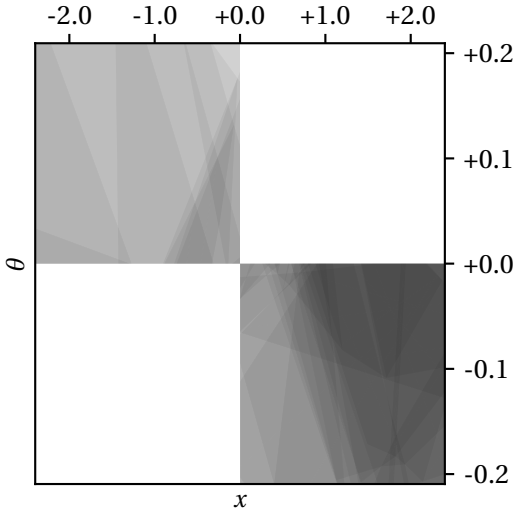


Figure 3.3: Projection of subsets of the domain where the wrong action is taken, with the hue of the area being proportional to the volume of the wrong sets, divided by the volume of the projection.

gradient reinforcement learning. Details of this calculation are in Section 3.D.2. This agent succeeds in balancing the pole: in 1000 trials of 200 periods, (x,θ) remains uniformly in $[-.75,+.75]\times[-.05,+.05]$ with very low velocities. Nonetheless there are many states for which pushing right is clearly the correct action, but for which the DNN controller predicts -1: in the same simulation of 1000 trials of 200 steps, roughly 7% of actions performed by the agent fail this sanity check. Such behavior is not a numerical fluke – it holds if states only nonnegligibly interior to D_{+1} and D_{-1} are considered, and also if only predictions that are made with probability greater than .51 are counted. One such pocket of clearly wrong behavior is

$$[-2.399,-1.462]\times[-2.922,-2.262]\times[+1.798\times10^{-8},+0.1067]\times[+1.399,+1.728]\subseteq$$
$$D_{+1}\cap f^{-1}(\{x\in\mathbb{R}^2 : x_1 > x_2\}).$$

This box is large – for example the first coordinate comprises almost 20% of that dimension of the state space. And moreover, this box is inscribed within a larger polytope (using the algorithm of Bemporad, Filippi, and Torrisi [13]) that has a volume about 40 times larger. The total volume in $\mathbb{R}^4$ of these sets is 3% of the state space volume, and thus 24% of the volume of $D_{-1}\cup D_{+1}$. Figure 3.3 parses this surprising fact a bit further by plotting the projection of the four-dimensional domain onto the

(x,θ) plane. The hue of the gray is proportional to the volume of the four-dimensional polytope divided by the volume of the two-dimensional projection, so darker areas mean more $(\dot{x},\dot{\theta})$ mass that is wrong. Since the entirety of the second and fourth quadrants are grey at *every* $(x,\theta) \in [-2.4,+2.4] \times [-\pi/15,+\pi/15]$ there are some $(\dot{x},\dot{\theta})$ where the cart will be pushed left, when right is clearly the correct action, or vice versa.

3.3.3 Collision Avoidance Systems

The final application shows how to use domain knowledge to anticipate dangerous behavior of a DNN in a complex modelling domain.

Background

Aircraft automated collision avoidance systems (ACAS) are navigational aids that use data on aircraft positions and velocities to issue guidance on evasive actions to prevent collisions with an intruding aircraft. The ACAS developed in Kochenderfer and Chryssanthacopoulos [90] uses dynamic programming to formulate the optimal control of a partially observed Markov process, and issues advisories to optimize a criterion that penalizes near collisions and raising false or inconsistent warnings. Evaluating the policy function is too resource-intensive to run on certified avionics hardware. DNNs have been been found to be adequate approximators that require little storage and can perform inference quickly. Unfortunately, even accurate DNNs can give very wrong predictions on some inputs – Katz et al. [87], for example show that when another aircraft is nearby and approaching from the left, a DNN-based approximation need not advise the correct action of turning right aggressively.

Verification can check that one-step behavior in a DNN-based ACAS behaves as intended. However, it cannot answer higher level questions like "will a near-collision occur if this policy is followed?" The idea of Julian and Kochenderfer [84] is to verify dynamic properties of such systems by combining single-step verification with worst-case assumptions about randomness in state transitions and (constrained) behavior of other aircraft.

Discretize and Verify

In Julian and Kochenderfer [84], the state consists of distances (x,y) between two aircraft, and an angle of approach angle between them, ψ. The actions are five turn-

g	Volume fraction
40	0.05128
80	0.02532
120	0.01681
160	0.01267
200	0.01005

Table 3.2: Quantifying the inefficiency of discretization: Each of the three dimensions, (x, y, ψ) is discretized into a grid of size g, so that the domain is partitioned into g^3 cubes. "Volume fraction" gives the fraction of cubes for which all eight corners of this cube do not evaluate to the same prediction, a sufficient condition for the cell to intersect with a decision boundary.

ing advisories: (1) "clear of conflict" (COC), (2) weak left [turn] (WL), (3) strong left (SL), (4) weak right (WR), and (5) strong right (SR). The initial condition is given by the boundary of the domain where the distance of the intruding aircraft are at their maxima. Transition dynamics are denoted by $\Psi(a, S)$, a set-valued function giving the set of states that are reachable from states S under action a. Ψ encompasses both randomness in transition, and behavior of the other aircraft. The change in (x, y) is controlled by the angle between the crafts, and the update to the angle is the difference between the turning of the two crafts, with some randomness. To compute the states that can arise under a policy, the idea is to begin from an initial set of states known to be reachable, and to iteratively append states that are reachable from any of those states, until a fixed point is reached. U are the states to preclude.

This idea is formalized by Julian and Kochenderfer [84] as Algorithm 1. This algorithm loops through starting cells, advisories, and all cells accessible from the starting cell under the advsories until either a fixed point is reached, an unsafe state is visited, or the encounter ends. Because multiple advisories are issued whenever a cell straddles the decision boundary, the discretized algorithm will wrongly include some states as reachable since a worst-case analysis needs to take account of *all* reachable states. Table 3.2 indicates the magnitude of overestimation, presenting how much of the state space will lead to multiple advisories under a simple discretization scheme.

Julian and Kochenderfer [84] use a dynamic grid with finer resolution where the function changes more;an improvement on an equispaced grid that not fundamentally address discretization error. And any false positives in a single-step decision function will be amplified in the dynamic analysis, as more reachable states at one step lead to even more reachable points subsequently, so overestimation at one step will compound through the dynamics. Not coincidentally, Julian and Kochenderfer [84] reach a

usable solution, but are unable to guarantee the absence of near collisions under some realistic parameter configuations.

Data: Initial state set $\mathscr{R}_0$, policy f, unsafe set U, transition dynamics Ψ, encounter length T.
Result: Guaranteed to not reach an unsafe state from $\mathscr{R}_0$ under policy f?
initialization: $t = 0$, done = False;
Partition the state space into cells $c \in \mathscr{C}$;
while *not done* **do**
  $t = t + 1$;
  $\mathscr{R}_t = \emptyset$;
  for $c \in \mathscr{C}$ *such that* $c \cap \mathscr{R}_{t-1} \neq \emptyset$ **do**
    for i *such that* $f(c) \cap \{x : x_i \geq x_j \text{ for } j \neq i\} \neq \emptyset$ **do**
      for $c' \in \mathscr{C}$ *such that* $c' \cap \Psi(i, c) \neq \emptyset$ **do**
        $\mathscr{R}_t \leftarrow \mathscr{R}_t \cup c'$
      end
    end
  end
  done = $\mathscr{R}_t == \mathscr{R}_{t-1}$ or $U \cap \mathscr{R}_t \neq \emptyset$ or $t > T$.
end
Return $\mathscr{R}_t \cap U == \emptyset$

Algorithm 1: Algorithm from Julian and Kochenderfer [84] for computing whether an unsafe set U can be reached under a policy f beginning from $\mathscr{R}_0$ under transition dynamics Ψ.

Note how the cells can be traversed in any order, indicating that this algorithm is not fully using the spatial structure of the problem. Algorithm 2 incorporates the knowledge that behavior should be spatially autocorrelated.

A Preimage-Based Alternative

Rather than looping first the domain, then over actions at those points, using the preimage Algorithm 2 loops over actions i, computing all reachable points under a.

Algorithm 2 is exact – it will never wrongly say that a state can be reached – while the accuracy of Algorithm 1 is controlled by the number of cells, $|\mathscr{C}|$. This is because it is necessary to perform n_L verifications for each reachable cell, and the number of reachable cells is proportional to $|\mathscr{C}|$. Let V denote the cost of a verification. Verification is known to be NP-complete (see Katz et al. [87]), so V dominates all other calculations such as computing intersections or evaluating $\Psi(i, c)$. Thus, the computational cost of Algorithm 1 is $O(|\mathscr{C}|Vn_L)$. Algorithm 2 must initially compute n_L preimages which

Data: $\mathscr{R}_0$, f, U, Ψ, T.
Result: Guaranteed to not reach an unsafe state from $\mathscr{R}_0$ under policy f?
initialization: $t = 0$, done = False;
for $i = 1, 2, \ldots, n_L$ **do**
  $\Xi_i = f^{-1}(\{x : x_i \geq x_j \text{ for } j \neq i\})$
end
while *not done* **do**
  $t = t + 1$;
  $\mathscr{R}_t = \emptyset$;
  for $i = 1, 2, \ldots, n_L$ **do**
    $\mathscr{R}_t \leftarrow \mathscr{R}_t \cup \Psi(i, \Xi_i \cap \mathscr{R}_{t-1})$;
  end
  done = $\mathscr{R}_t == \mathscr{R}_{t-1}$ or $U \cap \mathscr{R}_t \neq \emptyset$ or $t > T$.
end
Return $U \cap \mathscr{R}_t == \emptyset$.

Algorithm 2: Our preimage-based, exact algorithm for computing the dynamically reachable states in an ACAS.

dominates the entire calculation, consisting of fast operations – applying the dynamics and computing intersections up to T times, for T a number around 40.

Let P denote the cost of computing a preimage, then the cost of Algorithm 2 is $O(Pn_L)$. So whilst it dispenses with the need to solve $O(|\mathscr{C}|)$ verifications, it may be more intractable if P is significantly greater than V. However, *exact verification for even a single cell is impossible at present for large networks* and computing preimages is computationally hard for the exact same reason that verification is complex – the need to check most of the sign patterns. Thus, it seems very plausible that the computational cost of computing a preimage is proportional to that of performing a verification on the same network: $P = O(V)$. The practical tractability of both P and V hinges importantly upon theoretical arguments showing that not all 2^n configurations of the nonlinearities of an n-dimensional layer can be achieved (Serra, Tjandraatmadja, and Ramalingam [141] and Hanin and Rolnick [72]), and clever implementations that take account of the structure of the problems (e.g. Tjeng, Xiao, and Tedrake [158] and Katz et al. [87]).

An *encounter plot* such as Figure 3.4 makes clear the distinction between the two algorithms. Encounter plots depict the advisories, for a fixed angle of approach (conveyed by the angle between the red and black aircrafts), for each relative position. This figure replicates Julian and Kochenderfer [84, Figure 4] except in one crucial respect: the preimage of the five sets where each of the advisories are issued are *analytically-computed* (details of the experiment are in Section 3.D.3). The shaded areas arise from plotting polytopes, as in Algorithm 2. Julian and Kochenderfer [84], on the other

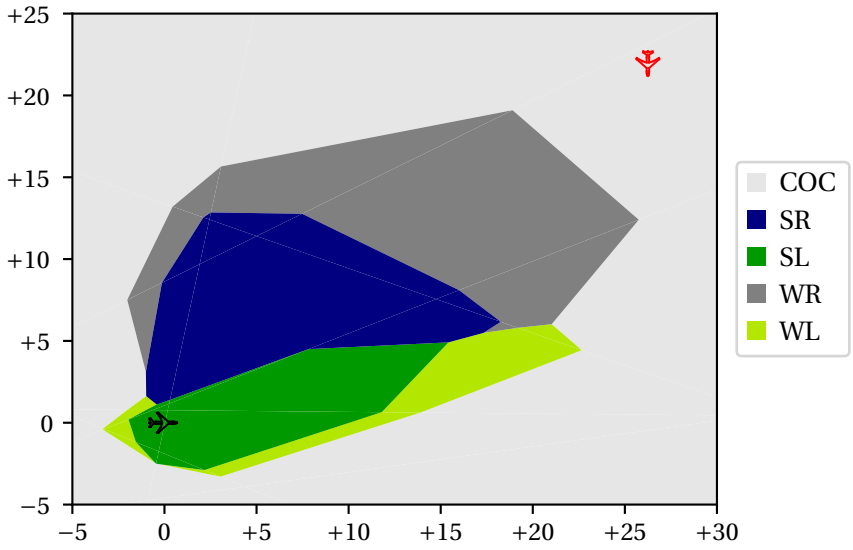


Figure 3.4: An encounter plot showing the optimal action at each (x, y) for a fixed angle of approach, indicated by the perpendicular orientation of the red (intruder) aircraft (here $\pi/2$). Distances are measured in kilofeet.

hand, produce such plots by evaluating the predictions of the network on a fine grid. The different manner in which the plots are produced is an exact analogue of the different way that the networks are summarized and analyzed through time. In particular, discretization can never preclude small (but practically significant) pockets of unexpected behavior, but preimages can. The next section gives some preliminary analysis that permits tackling this problem with a clear tradeoff between completeness and computation.

3.4 Locating Unsafe Regions

Tha analysis above showed that deep neural networks can demonstrate excellent performance on a task *most of the time*, but exhibit nonsensical behavior at isolated points in the domain for reasons that are not well understood. Preimages offer a way to observe this phenomenon if present, but is intractable at scale. This section presents a novel approach that gives a clear tradeoff between computation and the possible volume of a a connected set of adversarial inputs. Specifically, it shows that for a set of inputs with sufficiently low-discrepancy (a measure of uniformity), it is possible to guarantee that adversarial inputs will be observed if they exist. Or, that for inputs

that are of sufficiently low discrepancy, the set of adversarial inputs cannot be too large. Like existing methods (Wong and Kolter [171], Zhang et al. [174], and Virmaux and Scaman [166]), the effectiveness of this approach hinges on the smoothness of the input-output mapping, but instead of optimization, abstract interpretation (Gehr et al. [59]) or some other way to summarize the behavior of the network over all inputs, it uses only function evaluations and relies on the coverage of the input space to furnish the needed structure via a powerful result from the theory of quasi monte carlo integration.

3.4.1 A Simple Model of Robustness

As a simple, extensible model of uncovering unanticipated behaviour, akin to Section 3.3.2, consider the following scenario. Domain knowledge (such as the argument that $\forall x \in D_{+1}, f(x) = 1$) specifies that a neural network f is expected to take the value one[1] uniformly over values on the unit cube, $[0,1]^d$. The verification problem is to determine whether there is a connected region of volume ϵ on which the output is zero. For brevity, call this subset the "unsafe region". The goal is thus to determine if a network has an unsafe region. Although unsafe regions of measure zero are conceptually interesting and practically relevant, they cannot be addressed by this method.

3.4.2 Verification by Sampling

The proposed approach is to "brute force" the model, by evaluating it at many points, designed so that the fewest number of points are needed to discover an unsafe region of a given volume, ϵ. Let $\mathscr{P} = \{x_1, \ldots, x_n\} \subsetneq [0,1)^d$ denote an arbitrary point set of size n in dimension d to search for unsafe points. One nondeterministic baseline would be to choose identically, independently distributed inputs uniformly, so that the probability of a single sample falling in the unsafe region would be ϵ, and the probability of having at least one of n samples in the unsafe region would be $1-(1-\epsilon)^n$. A deterministic baseline is an equispaced regular grid of $p \triangleq \log_d(n)$ points along each of d dimensions: $\{(i_1/(p+1), i_2/(p+1), \ldots, i_d/(p+1)) : i_k = 1, \ldots, p, k = 1, \ldots, d\}$. However, the function

[1] This argument considers the network as including a thresholding to $\{0,1\}$, representing a binary classifier. This fits seamlessly with the smoothness measure to be used subsequently. The argument for real-valued networks is similar if the condition of equality with one is replaced by being above some threshold.

$$z \mapsto \begin{cases} 1 & \text{if } |z_i - k/(p+1)| \le \left(\frac{1-\epsilon}{n}\right)^{1/d} \text{ for some } i,k \\ 0 & \text{otherwise} \end{cases}$$

satisfies $f(x_j) = 1$ for all $x_j \in \mathscr{P}$, and yet also characterizes a connected set of mass ϵ on which $f(x) = 0$. The remainder of this section shows how to reduce the search for an unsafe region to bounding the integration error. Then next section introduces a decomposition of this error, concluding with a lower bound on the size of a data set necessary to discover an unsafe region of volume ϵ, if it exists.

3.4.3 Discrepancy

Quasi monte carlo (QMC) pointsets exhibit low *discrepancy*, a notion of maximal deviation from uniformity. Let $a = (a_1, a_2, \ldots, a_n) \in [0,1)^d$, and let

$$\delta(a; \mathscr{P}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{[0,a_i)}(x_i) - \prod_{j=1}^{d} a_j.$$

$\delta(a; \mathscr{P})$ measures the difference between the empirical fraction of points in $[0,a)$ and the volume of $[0,a)$: a natural measure of the deviation from uniformity of $\mathscr{P}$ on $[0,a)$. The (star) disrepancy of $\mathscr{P}$ is the maximum over all intervals starting at the origin:

$$D^\star(\mathscr{P}) = \sup_{a \in [0,1)^d} |\delta(a; \mathscr{P})|. \tag{3.4}$$

In one dimension, the star discrepancy is the Kolmogorov-Smirnoff statistic for testing whether $\mathscr{P}$ is a realization from a uniform distribution. Figure 3.5 presents a QMC pointset visually in agreement with the definition above: the number of points in each subset is roughly proportional to the volume, for every subset. This is not true of pseudorandomly generated random numbers, which exhibit "clumps". QMC pointsets are especially important in high dimensions, where clumps of points necessarily lead to voids of higher volume.

3.4.4 Hardy-Krause Variation

Controlling the Lipschitz constant of a DNN *is* a reliable method to improve the adversarial robustness of a deep neural network (Zhang et al. [174] and Virmaux and Scaman

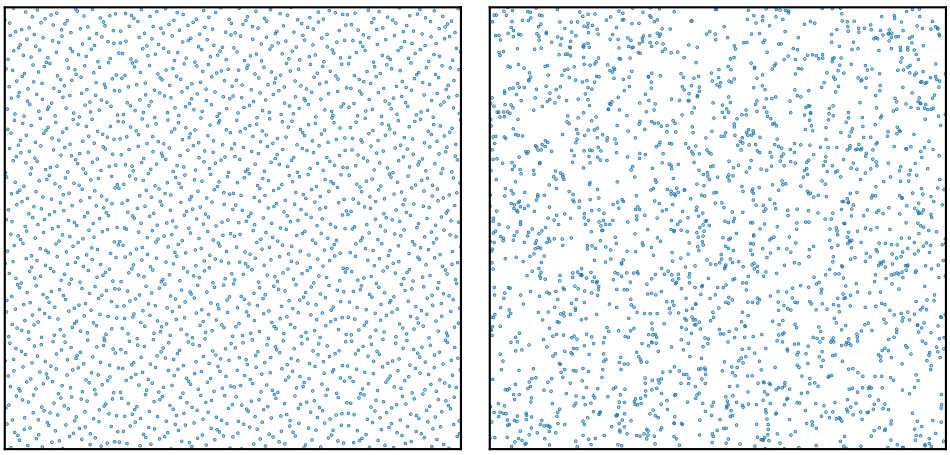


Figure 3.5: Left: Scrambled Sobol sequence of 2048 points on the 2-d unit cube. Right: Equal number of uniformly distributed points. It is visually apparent that the Sobol sequence has fewer gaps.

[166]). The *Hardy-Krause variation* is a similar measure of regularity. The simplest way to understand the Hardy-Krause variation is as a multidimensional analogue of the total variation, to which it reduces in one dimension. Let

$$B_n = \{b : [b_0, b_1] \cup [b_1, b_2] \cup \ldots \cup [b_n, b_{n+1}] = [0,1], 0 = b_0 < b_1 < b_2 < \ldots < b_n < b_{n+1} = 1\},$$

then the total variation of a function f is $\max_n \sup_{b \in B_n} \sum_{j=0}^{n} |f(b_{j+1}) - f(b_j)|$. Hardy-Krause variation gives quantitatively meaningful bounds even for discontinuous functions, e.g. step functions have bounded Hardy-Krause variation, but an unbounded Lipschitz constant. The definition for general dimension is complex, and so omitted here – a concise definition is given in Owen [119, Definition 2]. Basu and Owen [11] give more detail on the Hardy-Krause variation, including an upper bound in terms of average mixed derivatives. Recent work, such as Pausinger and Svane [122], appears to be moving towards alternative measures of variation that are more tractable. Overall, it appears that the best method for bounding the (Hardy-Krause, or otherwise) variation of a function is unresolved, and is a topic of ongoing research.

3.4.5 A Dataset Size Bound

If an unsafe region exists, then $\int_{[0,1]^d} f(x)\mathrm{d}x = 1-\epsilon$. Given $\mathscr{P}$, an unsafe input will be found if $\frac{1}{n}\sum_{i=1}^{n} f(x_i) < 1$, if

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) - 1 + \epsilon \leq \left| \frac{1}{n}\sum_{i=1}^{n} f(x_i) - (1-\epsilon) \right| < \epsilon.$$

The Koksma-Hlawka inequality (Hlawka [77]) states that

$$\left| \frac{1}{n}\sum_{i=1}^{n} f(x_i) - \int_{[0,1]^d} f(x)\mathrm{d}x \right| \leq D^\star(\mathscr{P}) \times V_{\mathrm{HK}}(f), \tag{3.5}$$

where $D^\star(\mathscr{P})$ is the *star discrepancy* of $\mathscr{P}$ (Equation (3.4)) and $V_{\mathrm{HK}}(f)$ is the *Hardy-Krause variation* of f. This elegant result bounds the error of an approximate integration as the product of a component that depends only on how uniform is the pointset, and another that depends on how regular is the integrand. And more importantly, for any f with $V_{\mathrm{HK}}(f) < \infty$, it characterizes how large a point set need be to achieve a particular level of error. If $\mathscr{P}$ is a Halton sequence – a QMC pointset construction contained in scikit-learn – then $D^\star(\mathscr{P}) \leq c(d)(\log n)^d / n$ for a constant $c(d)$ that can be derived from the expression in Atanassov [8, Theorem 2.1]. Let i_d denote the inverse of $n \mapsto (\log n)^d / n$,[2] then for $\mathscr{P}$ constructed via a Halton sequence, if

$$V_{\mathrm{HK}}(f)c(d)(\log n)^d / n \leq \epsilon \iff n \geq i\left(\frac{\epsilon}{V_{\mathrm{HK}}(f)c(d)}\right),$$

then $D^\star(\mathscr{P}) < \epsilon / V_{\mathrm{HK}}(f)$, meaning that if an unsafe region of volume ϵ exists, at least one element of the pointset will fall within it. More sophisticated point set contructions can improve the error rate, and this is an active area of QMC research. For instance, Dick [41] improves the effective $D^\star(\mathscr{P})$ to $O((\log n)^d / n^{3/2})$ in some situations.

This result still faces impediments to its practical application. Software to work with low discrepancy point sets is immature; for example, the Sobol pointsets built into PyTorch hardcodes certain parameters from Joe and Kuo [83] that affect discrepancy. V_{HK} is complicated to bound, and the number of required points may be large. Nonetheless, this analysis shows that with a principled construction and an adequate number of

[2] Technically, this function is monotonic only for $n \geq \exp(d)$ – this is a very weak condition.

points, it is possible to guarantee robustness, and using only function evaluations.

3.5 Conclusion

The first part of this chapter proposed computing the preimage of the decisions of a DNN as an intuitive diagnostic that can anticipate problems and help domain experts gain trust in a DNN, even if they cannot formally articulate what makes a DNN trustworthy. In order to do this, it developed the preimage of a DNN and presented an algorithm to compute it. Section 3.3 demonstrated the utility of the preimage to understand counterintuitive behavior from a cart pole agent, and to more precisely characterize the set of states that would be reachable in an existing application of DNNs to aircraft automated collision avoidance systems.

Section 3.4 took a different approach, transposing a foundational result from quasi monte carlo integration theory to analyze a simple model of DNN safety. It identified the Hardy-Krause variation of a DNN as an interesting quantity (alongside its Lipschitz constant and other measures of smoothness), and furthermore showed that low discrepancy point sets are a good approach to summarizing the behavior of a deep neural network, especially in high dimensions.

3.A Proofs

3.A.1 Proof of Lemma 1

Proof. Unroll Equation (3.1). Let $S \subseteq \mathbb{R}^{n_{\ell+k}}$ be arbitrary.

$$\begin{aligned}
&(f_{\ell+k} \circ f_{\ell+k-1} \circ \ldots \circ f_\ell)^{-1}(S) \\
=&\{x : (f_{\ell+k} \circ f_{\ell+k-1} \circ \ldots \circ f_\ell)(x) \in S\} \\
=&\{x : f_{\ell+k}((f_{\ell+k-1} \circ f_{\ell+k-2} \circ \ldots \circ f_\ell)(x)) \in S\} \\
=&\{x : (f_{\ell+k-1} \circ f_{\ell+k-2} \circ \ldots \circ f_\ell)(x) \in f_{\ell+k}^{-1}(S)\} \\
=&\vdots \\
=&\{x : (f_{\ell+1} \circ f_\ell)(x) \in (f_{\ell+2}^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_{\ell+k}^{-1})(S)\} \\
=&\{x : f_\ell(x) \in (f_{\ell+1}^{-1} \circ f_{\ell+2}^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_{\ell+k}^{-1})(S)\} \\
=&(f_\ell^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_\ell + k)^{-1}(S).
\end{aligned} \tag{3.6}$$

□

3.A.2 Proof of Lemma 2

Proof.

$$
\begin{aligned}
&x \in f^{-1}(\cup_{i=1}^{N} S_i) \iff \\
&f(x) \in \cup_{i=1}^{N} S_i \iff \\
&f(x) \in S_1 \text{ or } f(x) \in S_2 \text{ or } \ldots \text{ or } f(x) \in S_N \iff \\
&x \in f^{-1}(S_1) \text{ or } x \in f^{-1}(S_2) \in S_2 \text{ or } \ldots \text{ or } x \in f^{-1}(S_N) \iff \\
&x \in \cup_{i=1}^{N} f^{-1}(S_i).
\end{aligned}
$$

□

An identical argument shows that $f^{-1}\left(\cap_{i=1}^{N} S_i\right) = \cap_{i=1}^{N} f^{-1}(S_i)$, which can be useful in an application where, for $S_i = \Psi \cap \Xi_i$, writing $\cup_i S_i$ as $\Psi \cap \cup_i \Xi_i$ may be more efficient.

3.B The Inverse of a Residual Block

The key function in a residual block is $x \mapsto W_2\text{ReLU}(W_1 x) + x$. Combining arguments similar to Lemma 3 and Lemma 4, gives Lemma 5.

Lemma 5 (Preimage of residual block)**.**

$$
\begin{aligned}
&(z \mapsto W_2\text{ReLU}(W_1 z) + z)^{-1}(\{x : b - Ax \geq 0\}) \\
=&\{x : b - A(W_2\text{ReLU}(W_1 x) + x) \geq 0\} \\
=&\bigcup_{v \in \{0,1\}^n} \left\{x : b - A(W_2\text{diag}(v)W_1 + I)x \geq 0, -\text{diag}(1-v)W_1 x \geq 0, \text{diag}(v)W_1 x \geq 0\right\}.
\end{aligned}
\tag{3.7}
$$

3.C Collecting This All Up

Combining Section 3.2.1, Section 3.2.2, and Section 3.2.3 give a recipe for computing the preimage of region-unions for DNNs which can be written as the composition of linear and ReLU functions. To summarize the steps are:

1. Put the network into "standard form":
   (a) Embed any transformations that are "off" at inference time, such as dropout or batch normalization into the weights.
   (b) Rewrite the network in flattened form.
   (c) Rewrite all transformations as compositions of linear and ReLU functions (cf. Section 1.2.2).
2. Let $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ denote the network in this form.
3. Let $R_L = \cup_i \Delta_i$ be the image set to be inverted, for example $R_L = \Delta_1 = \{x : x_1 \geq x_2\} \subseteq \mathbb{R}^2$ in a binary classifier.
4. Compute $f_L^{-1}(\Delta_i)$ for all i, using Lemma 3 or Lemma 4.
5. Each term above is a region-union, thus $\cup_i f_L^{-1}(\Delta_i)$ is a region-union.
6. By Lemma 2, $R_{L-1} \triangleq f_L^{-1}(R_L) = \cup_i f_L^{-1}(\Delta_i)$.
7. R_{L-1} is a region-union $\implies R_{L-2} \triangleq f_{L-1}^{-1}(R_{L-1}) = f_{L-1}^{-1}(f_L^{-1}(R_L))$.
8. Repeat for $\ell = L-2, \ldots, 1$ to compute $R_0 = f_1^{-1}(R_1) = \ldots = (f_1^{-1} \circ f_2^{-1} \circ \ldots \circ f_L^{-1})(R_L)$.
9. Appeal to Equation (3.1) to conclude that $R_0 = f^{-1}(R_L)$.

3.D Details of Experiments

3.D.1 Experiment in Section 3.3.1

The dataset is constructed using the function `sklearn.datasets.make_moons` from scikit-learn (Buitinck et al. [21]) with `noise = .2` to generate 500 samples.

Weights are initialized according to a uniform$\left(-\sqrt{\text{in features}}, +\sqrt{\text{in features}}\right)$ distribution (the PyTorch default), and were run for 1000 epochs of with a batch size of 128. Gradient steps were chosen by the Adam optimizer (Kingma and Ba [89]) with learning rate of 0.005 and default $(\beta_1, \beta_2) = (0.9, 0.999)$.

3.D.2 Experiment in Section 3.3.2

This experiment is based upon the "CartPole-v1" environment from Brockman et al. [19]. The fitting procedure is based upon the Monte Carlo policy gradient vignette

from the PyTorch project, with a considerably simplified control policy, consisting of DNN wih only five hidden units.

Velocity magnitude

There seems to be no single methodology for computing limits on the velocities $(\dot{x}, \dot{\theta})$ in the cart pole problem. In premise, very high velocities could be supported by the discretization scheme, but these are unlikely to be achieved by any feasible sequence of actions.

An interpretable baseline that gives quantitative bounds robust to details of parameterizations would is best. For this, we chose limits on $\dot{x}, \dot{\theta}$ as the values that answer the question "how fast can the cart and pole be moving if we start from rest with the cart all the way to the right, and the pole all the way to the left, and continually push left until failure?". This experiment is plotted in Figure 3.6, where we see that the implied limits are ± 3.0 and ± 3.5 for $\dot{x}$ and $\dot{\theta}$, respectively. These limits largely agree with three the other candidates:

- The same experiment, although constrained to obey the same initialization as in Brockman et al. [19].
- A simple agent which seeks out high velocities by pushing first in a uniform direction, then switching to the opposite direction.
- The most extreme values emitted from a small DNN that is able to eventually achieve good performance.

3.D.3 Experiment in Section 3.3.3

The analysis presented in Section 3.3.3 uses data generated by Julian and Kochenderfer [84]'s software that formulates and solves dynamic programs to deliver lookup tables of optimal collision avoidance behavior in the same manner as the FAA's proprietary software (cf. Kochenderfer and Chryssanthacopoulos [90]). Our DNN modelling makes some methodological improvements in order to achieve better results at smaller network sizes – necessary for the feasibility of inversion. This section details the aspects of the analysis that differ from Julian and Kochenderfer [84], with the bottom line being that our results *can* be directly compared to theirs.

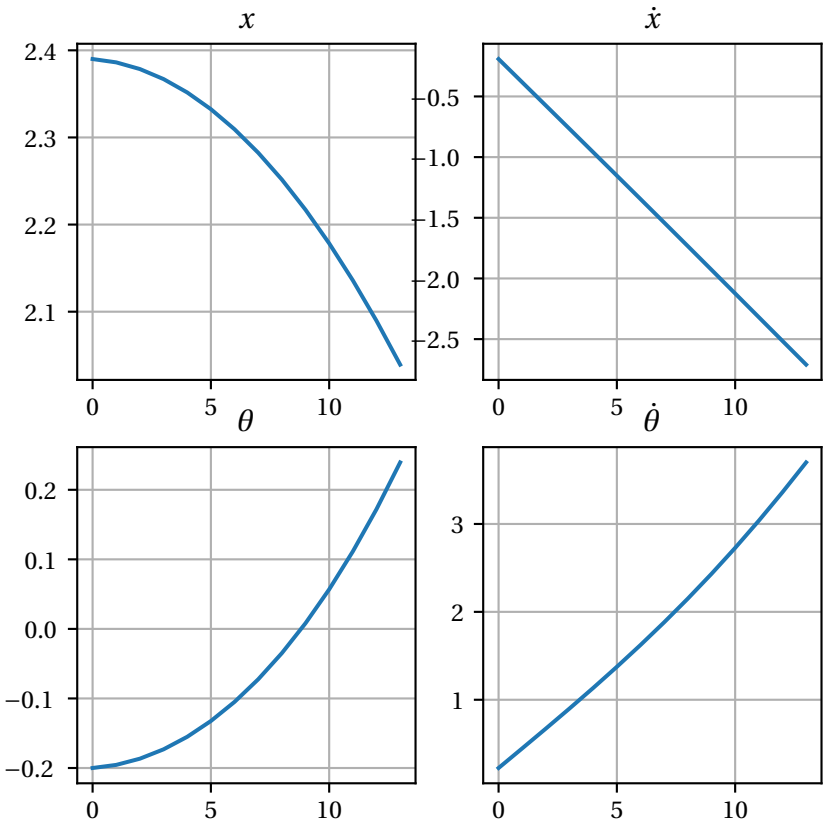


Figure 3.6: The evolution of the cart pole state under a policy which continually accelerates left until failure (when $\theta \geq \pi/15$), starting from rest, with the cart near the right boundary and the pole near its right boundary ($(x, \dot{x}, \theta, \dot{\theta}) = (2.39, 0, -.20, 0)$).

Fitting – Optimization Criterion

The first manner in which our approach is different is the fitting criterion: Julian and Kochenderfer [84] classify indirectly: by first fitting the continuation value to taking each action, and then choosing the action with the highest predicted continuation value. This oblique approach is an understandable continuation of their extended project to build (Kochenderfer and Chryssanthacopoulos [90]) and compress the Q-table (Julian, Kochenderfer, and Owen [85], Katz et al. [87]).

However, issuing advisories requires only the argmax. In order to recognize the invari-

ance of the prediction to any increasing transformation we replace Julian, Kochenderfer, and Owen [85]'s mean squared error (MSE)-based criterion with cross entropy loss to directly model the optimal decision. This achieves better performance with smaller networks. For example, Julian and Kochenderfer [84] use a five layer fully connected layers with 25 neurons each to achieve an accuracy of 97% to 98% while we achieve the same accuracy with *two* layers of 25 neuron (a network of the same size using Julian, Kochenderfer, and Owen [85]'s method attains an accuracy around 93%).

Why is anything less than complete fidelity to the Q-table acceptable in an approximation? The answer is twofold. Firstly the Q-table is itself not perfect, because of discretization artifacts in the original data. One can observe physically implausible sawtooth-like decision boundaries that arise from a coarse grid in the top plot of Julian and Kochenderfer [84, Figure 4]. The second is that accuracy alone does not capture the genuine metric of goodness, for example in the bottom plot of Julian and Kochenderfer [84, Figure 4] a highly accurate network exhibits unusual "islands" of SR completely encompassed by a region of WR that are both not present in the ground truth, and also prescribe a conceptually wrong relationship.[3]

Fitting – Symmetry

The second manner in which our approach differs from Julian and Kochenderfer [84] is the domain. Julian and Kochenderfer [84] fixed a lookup table over $(x, y, \psi) \in [-56000, +56000]^2 \times [-\pi, +\pi)$. However, if $Q : \mathbb{R}^3 \to \mathbb{R}^5$ denotes the Q-function as a function of the state $s = (x, y, \psi)$, then the physics of the problem ensure that

$$Q(T_i s) = T_o Q(s) \text{ where } T_i = \begin{pmatrix} +1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \text{ and } T_o = \begin{pmatrix} +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & +1 & 0 \end{pmatrix}.$$

This relationship only holds for $a_{\text{prev}} = \text{COC}$, but similar symmetries exist more generally. Thus, only half of the lookup table is needed, and moreover it is wasteful to ask a network to learn the same thing twice. Our method is to only fit f over $(x, y, \psi) \in$

[3] A pilot could be initially advised a strong right turn, then after some period of lessened danger have it downgraded to a weak right, only to have it re-upgraded to a strong right, although the danger continues to lessen.

$[-56000, +56000]^2 \times [0, +\pi)$, and when needed to infer $f(s) = T_o f(T_i s)$ for $s = (x, y, \psi)$ with $\psi < 0$. To continue the analysis above: exploiting symmetry achieves accuracy above 97% from a one layer, 24 neuron network. Figure 3.4 is computed on a 16 neuron network that achieves about 96% accuracy.

Inversion – Projection

Figure 3.4 was formed by taking the fitted $n_0 = 3$ DNN, fixing ψ to a given value, and working with the resultant $n_0 = 2$ DNN. If W_1, b_1 denote the weights and bias of the first linear layer of the original DNN, then this amounts to replacing these coefficients with $W_1' = W_1 \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, b_1' = b_1 + \psi W_1 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.

3.D.4 Polytope Dimension

The terms in Equation (3.3) overlap at the boundaries, for example if $b \geq 0$ then the origin is contained in every term in the union. This work only forms *full dimensional* polytopes. Lower-dimensional sets do not change the union, so the preimage is not affected by this choice.

Empirically, *most* sets in the region-union comprising Equation (3.3) are not full dimensional. And if an element of a preimage region is known to not be full dimensional, it need not be further considered. Thus, most computational effort is spent in calculations of the form of Equation (B.4). We tested both Mosek ([6]) and Gurobi ([69]) as software for solving linear programs such as Equation (B.4), and found Gurobi to be faster.

3.D.5 Clock Time to Solve Problems of Varying Sizes

Table 3.3 indicates the computational difficulty of inverting small DNN. The problem is as described in Section 3.3.1, with further detail given in Section 3.D.1. Though the essential computation, checking polytope emptiness, is embarassingly parallel the computation does not use any multiprocessing. All timings were performed on a 1.6 GHz Dual-Core Intel Core i5 CPU.

Hidden layer widths	Accuracy	# parameters	Storage (MB)	Time (s)
8	0.973	42	1.105	0.205
12	0.975	62	35.256	2.907
16	0.974	82	985.767	52.912
$4 \to 4$	0.972	42	0.260	0.269
$6 \to 6$	0.967	74	6.764	3.205
$8 \to 8$	0.971	114	42.089	26.256
$4 \to 4 \to 4$	0.969	62	3.125	1.802
$6 \to 6 \to 6$	0.968	116	279.138	117.248
$4 \to 4 \to 4 \to 4$	0.973	82	43.695	27.253

Table 3.3: Complexity of inverting DNNs of various size. Model accuracy and time to compute the preimage averaged over ten fittings. Storage (MB) = disk storage necessary to hold both the H and V forms of a complete preimage partition (pickled dense numpy arrays of `float64`s).

Chapter 4

The Expressiveness of Max Pooling

Max pooling is notably omitted from the function class discussed in Section 1.2.2. This is not accidental: max pooling is unlike other elementwise nonlinearities. The precise manner in which this is true is the content of this section, which is based on Matoba, Dimitriadis, and Fleuret [108]. This work is overwhelmingly my own, with Nikoloas Dimitriadis offering some feedback and assistance with experiments and proof details.

I would like to thank Guillermo Ortiz-Jiménez, Apostolos Modas, Arnaud Pannatier, and Suraj Srinivas, for their valuable comments on the original paper. Nikos was supported by Swisscom (Switzerland) AG.

4.1 Introduction

When convolutional neural networks first became state of the art image classifiers in the early 2010s, max pooling featured prominently. For example, in VGG (Simonyan and Zisserman [143]) and AlexNet (Krizhevsky, Sutskever, and Hinton [98]). Largely coincident with this period, however, Springenberg et al. [147] argued that max pooling operations were not necessary because strided convolutions composed with nonlinearity is simpler and more flexible. And subsequent architectures have continued to move in such a direction – ResNets (He et al. [73]) have a single max pooling layer, and some – such as InceptionV3 (Szegedy et al. [153]) and mobilenetV3 (Howard et al. [79]) – have none at all.[1] Max pooling is mostly omitted in the attention-based image

[1] All statements about historical models refer to their implementation in Torchvision (Marcel and Rodriguez [106]), described at `https://pytorch.org/vision/stable/models.html`.

classifiers that began to see use in the early 2020s, such as the Vision Transformer (Dosovitskiy et al. [46]). Examining whether max pooling can truly be dropped from the toolbox of neural network image classifiers is worthwhile because it would ease architecture design and reduce development burden.[2]

This chapter examines whether max pooling is a remnant of an earlier era when image classifiers were motivated by the visual cortex. It finds that for some inputs, max pooling constructs very different features than an optimal approximation by ReLUs, thus it expresses a different inductive bias and can sometimes be appropriate. It derives comprehensive bounds on the error realized by approximating max functions with the composition of ReLU and linear operations, finds that a simple divide and conquer algorithm cannot be improved upon. Hence, accurate approximations must be computationally complex.

The main result of this work does not imply that omitting max pooling from image classifiers is wrong. Rather, it says when it *could be wrong*. Max pooling will be strictly more expressive than a ReLU-based approximation on inputs with a large difference between the maximum and other values within pools.

The contributions are (1) introducing a novel generalization of max pooling (Section 4.4), (2) proving that the max function cannot be represented by simpler elements of this function class (Theorem 4), (3) analyzing experimentally the size and quality of approximations (Section 4.5), (4) formulating a notion of separation for max pooling (Theorem 1, Theorem 2), and (5) connecting mathematically the average of subpool maximums with order statistics (Theorem 3).

4.2 Background and Related work

Early work examined the neurological antecedents of artificial neural networks, and experiments with mammal's eyes and brains showed max pooling to be a biologically plausible operation (Fukushima [56] and Riesenhuber and Poggio [130]). Practice followed this observation, with max pooling being important when deep neural networks first began achieving competitive performance. For example, AlexNet (Krizhevsky, Sutskever, and Hinton [98]) dominated the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 (Russakovsky et al. [132]), and had max pooling in more

[2] For example, in the modern era of extremely specialized hardware, it is common for primitive operations to be reimplemented several times for different computational "backends", such as CUDA device capabilities.

than half of its "blocks".[3] However, the ILSVRC 2017 leaderboard was dominated by ResNets, for which max pooling is not central. Similarly with DawnBench (Coleman et al. [32]). This move away from max pooling was consistent with an influential study, Springenberg et al. [147], arguing that strided convolution composed with nonlinearity is preferable to max pooling because it can fit a max pooling mapping if appropriate, and can seamlessly fit another functional form if not.

We examine this assertion by questioning whether (1) max pooling can be simplified to ReLU nonlinearity, and if not, (2) what pattern in the data would make max pooling and ReLU comparably expressive *empirically*.

The present inquiry is preceded by a literature examining the expressiveness of piecewise linear networks via bounds on the number of linear regions they express, for example Montúfar et al. [114], Serra, Tjandraatmadja, and Ramalingam [141], Hanin and Rolnick [71], and Montúfar, Ren, and Zhang [113]. Several of these studies examine the maxout activation function (Goodfellow et al. [64]), which is equivalent to our mathematical model of max pooling. Maxout networks generally express many more linear regions than equally-sized networks using ReLU activations (Montúfar et al. [114] and Tseran and Montúfar [160]).

Compared to these studies, we assess the increased expressivity of max pooling networks as the error attained by a single max pooling operation with ReLUs; this it more direct and practically interpretable. However, the conclusions also need more thought when transposed to architecture design since it may be that the outputs of a network using max pooling is efficiently approximated, even though an individual neuron is not.

These narrow assumptions in turn suggest a potentially fruitful direction of inquiry into maxout activations: eschew general maxout assumptions for architectures consistent with max-pooling (e.g. output stricly smaller than input, inputs are grouped into disjont "pools", etc.). These stronger assumptions may be the missing link since maxout, although more general, never gained traction (it is not implemented natively in PyTorch, for example, and practically never seen in the wild).

Like most related studies, this work easily extends to many other (elementwise) piecewise linear activations and does not apply to many other sources of nonlinearity, such as tanh. However, there are standard techniques for extending work on piecewise linear activations to general activations based on bounding the difference to a piecewise linear approximation, such as Zhang et al. [174]. Attention mappings (Vaswani et al.

[3] The leaderboard is [99], AlexNet is a product of the "SuperVision" team.

[165]), notably, do not fit within the scope of the analysis. Nonetheless, the recent move towards of fewer max pooling layers has been continued in the emergence of transformer-based image classifiers. The Vision Transformer (Dosovitskiy et al. [46]), for example, uses no max pooling layers.

Hertrich et al. [76] is closely related to the theoretical component of our work. This study addresses aspects of ReLU-based approximations to the max function, and proves a technical sense in which max pooling is more expressive than ReLUs. However, their proof technique relies crucially upon the kink at zero, a dynamic not shared by our analysis, which is restricted to nonnegative inputs. We give precise error bounds, and are able to compare the errors of a large parameterized family of approximations with considerable precision to offer a more precise grasp of the practical tradeoff that applied approximations entail. Table 4.1 summarizes several results on the approximation of the maximum function by linear-ReLUs blocks.

Depth	Function Class	Result
$\lceil \log_2 d \rceil$	width $\leq d/2$	exact trivial, Theorem 2
1	width $\uparrow \infty$	approximation possible, Hornik [78] and Cybenko [37]
$\lceil \log_2 d \rceil - 1$	width $\uparrow \infty$	exact impossible on $\mathbb{R}^d$, Hertrich et al. [76]
$\lceil \log_2 d \rceil - 1$	$\mathscr{M}_d(R)$	exact impossible on $[0,1]^d$, Theorem 4
$\lceil \log_2 d \rceil - 1$	width $\uparrow \infty$	approximation difficult on $[0,1]^d$, Section 4.5

Table 4.1: A taxonomy of approximations to the max function by linear-ReLUs blocks in d dimensions. $\mathscr{M}_d(R)$ is the function class introduced in Section 4.4. The quality of the approximation depends crucially on depth, and as far as we are aware, this is the first work to examine the finite width case.

Boureau, Ponce, and LeCun [17] is an early work comparing average and max pooling from an average-case statistical perspective. They also identify the input dimensionality as a key factor in complexity. In their framework, sparse or low-probability features correspond to the corners of the input domain in our analysis.

Grüning and Barth [68] find that *min* pooling can also be a useful pooling method, a finding that supports and is rationalized by the finding here that the quantiles of the input are the basis for linear combinations of maxes. If the true data is strongly determined by the nonlinear behavior of quantiles, then ReLU-based approaches are relatively disadvantaged.

4.3 The Complexity of Max Pooling Operations

This section proves that in a simplified model, max pooling *requires* depth – multiple layers of ReLU nonlinearity are necessary to effect the same computation, and more layers are needed for larger kernel sizes.

4.3.1 Simplifications

In the design of deep learning architectures, max pooling reduces dimensionality by summarizing the values of spatially nearby inputs. To simplify, this chapter examines the approximation of a max *function*, putting aside "pooling"-specific considerations like stride, padding, and dilation that are ultimately linear pre-and post-processing. We summarize the size of the input using the term "order" so that, for example, the maximum over a 3×3 input is an order 9 max function. In this chapter, we call engineering DNNs comprised of (see Section 1.2.2) alternating ReLU and linear layers "(feedforward) ReLU networks" to emphasize that they use only ReLU nonlinearity to approximate max pooling.

4.3.2 Max pooling as a feature-builder

Telgarsky [156] showed that deep neural networks cannot be concisely simulated by shallow networks. Their approach is to demonstrate a classification problem that is easy for deep networks, but is provably difficult for shallow networks. We do similarly by building a test problem on which max pooling succeeds and ReLU fails. First, we investigate the appropriate metric for comparison. Theorem 1 shows that for any dimensionality, a narrow, shallow engineering network with a single source of nonlinearity can emit the same output as max pooling. Thus, prediction accuracy is not the correct metric by which to compare nonlinearities.

Theorem 1. *There exists a feedforward ReLU network $f : \mathbb{R}^d \to \mathbb{R}$ with d hidden neurons such that for all $\xi \in \mathbb{R}$, $f(x - \xi) \leq 0 \iff \max\{x_1, \ldots, x_d\} \leq \xi$.*

Proof.

$$\max\{x_1, \ldots, x_d\} \leq \xi \iff x_1 \leq \xi \text{ and } \ldots \text{ and } x_d \leq \xi \iff \sum_{k=1}^{d} \text{ReLU}(x_k - \xi) \leq 0.$$

□

Max pooling is used in the construction of intermediate layer features, and not directly in the computation of final logits. Thus, the ability of a feedforward ReLU network to achieve the same real-valued output as a max pooling operation with L_∞ error is primary. And since diminishing the representation capacity of a single neuron necessarily reduces the expressivity of the whole network, the remainder of this chapter concerns the expressivity of a single neuron.

4.3.3 Computing max using ReLU

Theorem 1 is a positive result on the complexity of functions that the composition of linear and ReLU functions can represent. This section begins developing a negative result.

The maximum of two values can be computed using the relationship

$$\max(a,b) = (\text{ReLU}(a-b) + \text{ReLU}(b-a) + a + b)/2. \tag{4.1}$$

Mukherjee and Basu [116] and Hertrich et al. [76] prove that the max of 2^k variables cannot be written as a deep neural network with a k ReLU layer engineering DNN, in particular that the maximum of three variables cannot be written as a linear combination of ReLU features. Thus, there is no generalization of this formula to three or higher dimensions – indicating the fundamental inadequacy of using linear combination and ReLU nonlinearity as a drop-in replacement for max pooling. Nevertheless, Theorem 2 shows that with additional depth a feedfoward network can compute the maximum of many variables by recursively forming pairwise maxes.

Theorem 2. $\max: \mathbb{R}^d \to \mathbb{R}$ *can be written as a* $\lceil \log_2(d) \rceil$*-hidden layer feedforward ReLU network, where the* k*th hidden layer has width* $2^{2+\lceil \log_2(d) \rceil - k}$.

Sketch of Proof. A variant of Equation (4.1) applicable to feedforward ReLU networks is

$$\begin{aligned} \max(x,y) &= (g \circ \text{ReLU} \circ f)(x,y) \text{ where} \\ f(x,y) &= \begin{pmatrix} +x-y \\ -x+y \\ +x+y \\ -x-y \end{pmatrix} \\ g(x) &= (x_{0:n/4} + x_{n/4:n/2} + x_{n/2:3n/4} - x_{3n/4:n})/2. \end{aligned} \tag{4.2}$$

Here $x_{n_1:n_2}$ means the n_1th through the (n_2-1)th elements of x (inclusive), and n is the dimension of the input. f and g are linear. At the cost of quadrupling every layer width, ReLU can evaluate pairwise maxes and $\lceil \log_2(d) \rceil$ iterations of pairwise maxima can compute the maximum of d variables.

□

As an example, the max of five variables is three iterations of pairwise max:

$$\begin{aligned} &\max(x_1,x_2,x_3,x_4,x_5) = \max(z_1,z_2) \\ \text{where } &z_1 = \max(z_3,z_4), z_2 = \max(z_5,z_6) \\ \text{where } &z_3 = \max(x_1,x_2), z_4 = \max(x_2,x_3), z_5 = \max(x_3,x_4), z_6 = \max(x_4,x_5). \end{aligned}$$

Theorem 2 is an upper bound on the width and depth necessary to evaluate a max function. Corresponding lower bounds are more intricate. Table 4.1 outlines several, and the next section discusses the function class $\mathscr{M}_d(R)$ that approximating max in the main result.

4.4 The Class of Subpool Max Averages, $\mathscr{M}_d(R)$

This work pertains to a particular function class, $\mathscr{M}_d(R)$. Section 4.4.1 describes $\mathscr{M}_d(R)$, and Section 4.5 justifies the relevance of this function class to deep learning.

4.4.1 Subpool Maxes

For a vector $x \in \mathbb{R}^d$ and index set $J \subseteq \{1,\dots,d\}$ let $\max\{x_j : j \in J\}$ be called the J-subpool max of x. For example, if $x = (3,2,10,5)$, then the $\{1,2,4\}$-subpool max of x is $\max(3,2,5) = 5$. The J-subpool max is a generalization of the max that trades off complexity and accuracy in the sense that for $J_1 \subseteq J_2$, the J_1-subset max is simpler to compute than the J_2-subset max, but the error of a J_1-subpool max will always be at least that of a J_2-subpool max.

Let $C(j,r,d)$ denote the jth (out of $\binom{d}{r}$) subset of $\{1,\dots,d\}$ of size r in the lexicographic ordering. For example, $C(1,2,3) = \{1,2\}, C(2,2,3) = \{1,3\}$ and $C(3,2,3) = \{2,3\}$. Let $\omega_{jr}(x)$

denote the $C(j,r,d)$-subpool max of x (the dimension d can be inferred from the size of x). In order to model a constant intercept, let the $C(1,0,d)$-subpool max, $\omega_{10}(x)$, be = 1.

Linear combinations of J-subpool maxes are a natural class of estimators to the max function. We organize subsets J by (1) the size of the subset, r, and (2) the subset index of that size, j. For $R \subseteq \{0,1,\ldots,d-1,d\}$ let

$$\mathscr{M}_d(R) = \left\{ x \mapsto \sum_{r \in R} \sum_{j=1}^{\binom{d}{r}} \beta_r^j \omega_{jr}(x) : \beta_r^j \in \mathbb{R} \right\} \tag{4.3}$$

be the set of all linear combinations of r-subpool maxes, $r \in R$. Let a general element of $\mathscr{M}_d(R)$ be called an R-estimator, and because any estimator in $\mathscr{M}_d(R)$ can be written as a feedforward ReLU network with $\lceil \log_2(\max R) \rceil$ layers, call this the *depth* of $\mathscr{M}_d(R)$. Theorem 4 shows that $\max \notin \mathscr{M}_d(\{0,1,\ldots,d-1\})$, with a bound on the L_∞ error from this function class. For this, Theorem 3 is instrumental.

Theorem 3. *Let* $S(x;r,d) \triangleq \frac{1}{\binom{d}{r}} \sum_{j=1}^{\binom{d}{r}} \omega_{jr}(x)$ *be the average of all subpool maxes of* $x \in \mathbb{R}^d$ *of order* $r \geq 1$. *Let* $x_{(j)}$ *(the subscripts being enclosed in parentheses) denote the* j*th largest element of a vector* x *(*order statistics *notation).*

$$S(x;r,d) = \frac{1}{\binom{d}{r}} \sum_{j=1}^{d-r+1} \binom{d-j}{r-1} x_{(j)}. \tag{4.4}$$

Sketch of Proof. $x_{(j)}$ is the largest value within a subpool if and only if all indices less than j are excluded from that subpool and j is not excluded. For a subpool of size r, the $r-1$ remaining values must be among the $d-j$ values $x_{(j+1)},\ldots,x_{(d)}$. Thus, there will be $\binom{d-j}{r-1}$ subpools of size r in which the largest value is $x_{(j)}$. □

The average of subpool maxes of an order $r \in R$ is a summary of the quantiles of the distribution via a particular weighted average, with better fidelity to the max for larger r. For example, $S(x;1,d) = (x_{(1)} + \ldots + x_{(d)})/d = (x_1 + \ldots + x_d)/d$ is a simple average, but $S(x;d-1,d) = ((d-1)x_{(1)} + x_{(2)})/d$, is mostly the largest value, with only the second-largest value contributing. At $x = \begin{pmatrix} 0 & 0 & \ldots & 1 \end{pmatrix}$, the error of an order-1 approximation is $x_{(1)} - S(x;1,d) = 1 - 1/d$. While the error of an order $d-1$ approximation is $x_{(1)} - S(x;d-1,d) = 1/d$. The idea is demonstrated further in Figure 4.1, with the individual subpool maxes easily interpreted via shading, and the higher-order subpool max averages increasingly resembling the actual max.

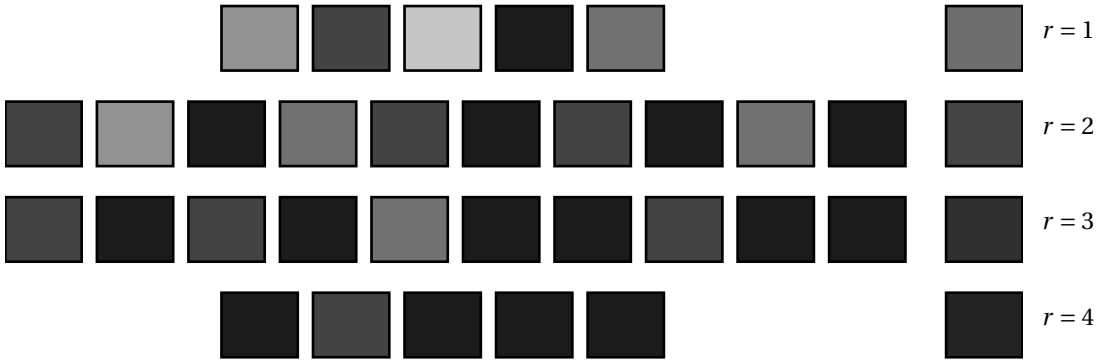


Figure 4.1: Subpool maxes (left) and their average (right) for $d = 5$. The value of each node is presented as inverted grayscale. Each row $r = 1,2,3,4$ gives the pool order r. As r grows, the average value grows darker towards the actual max.

4.4.2 Optimal Approximation Over $\mathscr{M}_d(R)$

Theorem 4 is the main result of this chapter. It gives the minimal errors achievable with approximations in $\mathscr{M}_d(R)$, for various R, over the d-dimensional unit cube. For example, Equation (4.8) states that the lowest L_∞ error achievable using elements of $R = \{0,1,d-1\}$ is $1/(2(d+1))$, with $\begin{pmatrix}\beta_0 & \beta_1 & \beta_{d-1}\end{pmatrix} = \begin{pmatrix}1/2 & -d/(d-2) & d(d-1)/(d-2)\end{pmatrix}/(d+1)$.

Theorem 4. *Let* $||f||_\infty$ *denote the the* L_∞ *norm over the unit cube:* $||f||_\infty = \sup_{x\in[0,1]^d} |f(x)|$. *Let* $\text{dist}(R) = \min_{m\in\mathscr{M}_d(R)} ||m - \max||_\infty$, *where* $\mathscr{M}_d(R)$ *is as defined in Equation* (4.3).

$$\text{dist}(\{0,1\}) = \frac{1}{2}\frac{d-1}{d} \tag{4.5}$$

$$\text{dist}(\{d-1\}) = 1/(2d-1) \tag{4.6}$$

$$\text{dist}(\{0,d-1\}) = 1/(2d) \tag{4.7}$$

$$\text{dist}(\{0,1,d-1\}) = 1/(2(d+1)) \tag{4.8}$$

$$\text{dist}(\{0,1,d-2,d-1\}) = 1/d^2 \tag{4.9}$$

$$\text{dist}(\{0,1,2,\ldots,d-1\}) = 1/2^d. \tag{4.10}$$

Sketch of Proof. We first establish symmetry as a property of any optimal estimator. Then we assume that the L_∞ norm of the error is characterized by $|R|$ nonzero corners of the unit cube and zero. Under this conjecture, the norm is optimized by evaluating the error as a function of the coefficients, and choosing coefficients which equate

them. Generally, changing coefficients will increase the error at one point and lower it at others, and balancing these effects characterizes optimality. For example with $R = \{0, d-1\}$, the optimal coefficients are $(\beta_0, \beta_{d-1}) = (1/(2d), 1)$. At $x = (0,\dots,0), (1,0,\dots,0)$ and $(1,1,\dots,1)$ the error of this estimator is $1/(2d)$. Given the conjectured error and coefficients, it is proven optimal by contradiction: by showing that any other coefficients attain a higher criterion at some x.

To continue the example from the proof of Equation (4.7), the only way that both $-1/(2d) < 1 - \beta_0 - \beta_{d-1}$ (the condition at $x = (1,\dots,1)$) and $1 - \beta_0 - \beta_{d-1}(d-1)/d < 1/(2d)$ (the condition at $x = (1,0,\dots,0)$) is for $\beta_0 > 1/(2d)$. But this clearly precludes the error from being $< 1/(2d)$ at $x = (0,0,\dots,0)$.

□

Even with the largest R, $\{0,1,2,\dots,d-1\}$, the error is > 0, meaning that insufficiently deep networks in $\mathcal{M}_d(R)$ cannot model max pooling, and giving a partial converse to Theorem 2.

Equation (4.5) gives a linear model as a baseline for the max. The error is high, and not much different to a constant model in high dimensions (an intercept-only model can obtain .5 = dist($\{0\}$)). Equation (4.6) shows that error declining with dimensionality is achievable with a higher order term. Contrasting Equation (4.7) to Equation (4.6) quantifies the additivity of the intercept. Including an intercept helps, but does not scale with dimension. Similarly for the grand mean: it reduces the error from $1/(2d)$ to $1/(2(d+1))$ (Equation (4.8)). Neither the intercept nor the mean as a feature entails no meaningful further nonlinearity, thus it is assumed that $\{0,1\} \subseteq R$ subsequently.

Equation (4.9) is important for understanding how the error falls with the addition of further strongly dimension-sensitive subpool orders: appending $d-2$ to $\{0,1,d-1\}$ improves the rate of convergence from $O(1/d)$ to $O(1/d^2)$. Equation (4.10) gives the best-case rate of convergence: if *all* lower order $r < d$ are included, then the error is $O(1/2^d)$. Contrasting this with Equation (4.9), shows that the inclusion of many r is necessary for low error. Overally, Equation (4.10) implies that $\max \notin \mathcal{M}_d(\{0,1,\dots,d-1\})$, though it can be approximated well within the function class.

Let $f_R^\star : \mathbb{R}^d \to \mathbb{R}$ denote the optimal estimate based on terms in R. L_∞ error can be high, even if $f_R^\star \approx \max$ on most of the domain. If the the high error could not be realized in practice, because the measure of the domain on which it arises is miniscule, then Theorem 4 would be only a technicality with little practical relevance. In the literature studying the number of linear regions of a piecewise linear network, this distinction

is recognized as the difference between the maximum and average number of linear regions (Tseran and Montúfar [160]). Theorem 5 shows that this is not the case, by showing that the error is almost surely nonzero. The proof is in Section 4.B.

Theorem 5. *If x is uniformly distributed over the unit cube, then* $\text{vol}(\{x \in [0,1]^d : |x_{(1)} - f_R^\star(x)| = 0\}) = 0$.

4.5 Experimental Evidence on the Relevance of $\mathscr{M}_d(R)$

For $\mathscr{M}_d(R)$ to be relevant, the results must not be dramatically different for a class of feedforward ReLU networks of the same depth, but somewhat more general than it. This section presents experimental evidence that $\mathscr{M}_d(R)$ is an adequate proxy for all networks of the same depth in empirically approximating the max function. It does this by showing that additional capacity does not appear to improve the quality of the approximation.

Section 4.D presents an algorithm for writing $f_R^\star$ as a feedforward ReLU network of depth $\lceil \log_2(\max R^\star) \rceil$ based on reusing lower order subpool maxes to evaluate higher order subpool maxes in a way that skips unneeded orders. For a f given in this form, let $w(f)$ denote the layer widths. For example, via this procedure with $d = 9$ and $R = \{0,1,7,8\}$, $f_R^\star$ can be computed by a feedforward ReLU network with hidden layer widths $w(f_R^\star) = (78, 122, 182)$. Finally, let $\mathscr{G}_{d,k}$ denote the set of all depth k feedforward ReLU networks $\mathbb{R}^d \to \mathbb{R}$, and for $\mu > 0$ let

$$\mathscr{G}_d(R,\mu) = \{g \in \mathscr{G}_{d,\lceil \log_2(\max R) \rceil} : w(g) \leq \mu \times w(f_R^\star)\} \tag{4.11}$$

be the set of all neural networks that are at most μ times as wide as $f_R^\star$. $\mathscr{G}_d(R,\mu)$ represents a parameterized interpolation between $\mathscr{M}_d(R)$ and all networks of a given depth in the sense that $\mathscr{M}_d(R) \subseteq \mathscr{G}_d(R,1), \mu_1 < \mu_2 \implies \mathscr{G}_d(R,\mu_1) \subseteq \mathscr{G}_d(R,\mu_2)$ and $\lim_{\mu \uparrow \infty} \mathscr{G}_d(R,\mu) = \mathscr{G}_{d,\lceil \log_2(\max R) \rceil}$, and width, via μ, is the notion of "capacity" when discussing the addition of capacity to $\mathscr{M}_d(R)$.

Rhetorically, $\mathscr{M}_d(R)$ will be more relevant if elements of $\mathscr{G}_d(R,\mu)$ do not achieve low error, even for μ high. This is an awkward result because failing to achieve low error on a deep learning task is not difficult, for example it could result from a coding mistake or a bad hyperparameterization. That said, given the simplicity of the modelling problem, excluding bugs is plausible – we code it in an idiomatic fashion, check that the code can solve problems that should be solvable (for example, replacing the max with the mean), and transparently release the source code. We endeavor to show that our

results are not caused by poor modelling choices with extensive ablation studies in the appendix. Finally, there is an important precedent for this type of result in machine learning experiments: interpreting any result showing that one method is better than another relies upon the inferior method implemented correctly and reasonably. Thus, although demonstrating experimentally the correctness of a failing method requires a high standard of evidence, the nature of the argument is not inherently problematic.

4.5.1 Experimental setup

Independent test and train datasets are generated uniformly on the unit cube with 10,000 rows. L_∞ loss is directly optimized by an Adam optimizer (with PyTorch default parameters). Results are similar with MSE loss. Parameters are initialized according to the PyTorch default. A batch size of 512 is used throughout, and the data set is shuffled over each of 300 epochs. The experiments are all repeated over ten pseudorandom number generator seeds, with both the data being generated differently, and different randomness in the fitting (e.g. the shuffling over minibatches).

4.5.2 Results

The experimental evidence in this section consists of assessing how the expressiveness of trained models $\in \mathcal{G}_d(R,\mu)$ depends on μ. A DNN $\in \mathcal{G}_d(R,\mu)$ optimized to model $x \mapsto x_{(1)}$ is more expressive if it achieves a lower empirical test L_∞ error. We term this quantity $\text{ERR}(R,\mu)$. If expressiveness is not substantially increased ($\iff \text{ERR}(R,\mu)$ is not substantially decreased) for progressively greater μ, then a lower bound on the error of approximating the max function over $\mathcal{M}_d(R)$ empirically also holds for $\mathcal{G}_d$. And this is observed.

The results focus on μ around 1, because it corresponds to the width of $f_R^\star$.[4] To prove the desired point – that adding capacity does not significantly increase expressiveness – requires only $\mu \geq 1$, however the results for $\mu < 1$ are included to help foster intuition. $\mu < 1$ also demonstrates an expected result from the literature on the *neural tangent kernel* (NTK), for instance Allen-Zhu, Li, and Liang [2], showing that for a fixed dataset size, error falls from high levels for small models, but rapidly levels off. Note, however,

[4] There are non-width reasons that, even for $\mu = 1$, $\mathcal{M}_d(R) \subsetneq \mathcal{G}_d(R,1)$. One is that $\mathcal{G}_d(R,\mu)$ allows intercepts in the linear layers, despite not being present in $f_R^\star$. $\mathcal{G}_d(R,\mu)$ also imposes no low-rank structure on the weight matrices, despite a straightforward representation of the theoretical weights as the outer product of matrices roughly one quarter as large (via the quadrupling of layer sizes implied by Equation (4.2)). The larger is the class of models that does not substantially reduce error, the more conservative are the experimental results and the stronger is the conclusion.

that not all the assumptions of the NTK, such as a very small learning rate, hold. Thus, prescriptions of this model are indicative. Neither the NTK nor universal approximation theorems (UATs) ensure that there is a network achieving arbitrarily low error as $\mu \uparrow \infty$.

We examine $R = \{0, 1, d-1\}, \{0, 1, d-2, d-1\}$, and $\{0, 1, 2, \ldots, d-2, d-1\}$ as prototypical "small", "medium", and "large" approximations, respectively. Although larger models do empirically achieve lower errors, the overall flattening trajectory of their dependence on increased capacity is uniform.

For a single order d max-pool layer the maximum expected number of regions is, from Tseran and Montúfar [160], Theorem 8: $1 + (d-1)d/2$. Contrast this with Montúfar et al. [114]'s bound for a single-output, d-input fully connected network, of $(1 + w(f)_1) \times \ldots \times (1 + w(f)_k)$. Thus, by this metric, $\mu = 4$ more than suffices to model any reasonable d.

Figure 4.2 is the main experimental result: fitting error does not reliably fall as μ rises. Even quadrupling all layer widths does not noticeably increase expressiveness. Clearly, the amount of data available to train models of differing capacities is an important determinant of performance. Figure 4.3 examines the effect of dataset size on training error for the $R = \{0, 1, 2, \ldots, d-2, d-1\}$ model. Here the x-axis is on a logarithmic scale in μ space. Observe that although test error reliably falls with the addition of more data, there is a marked flattening for all dataset sizes – beyond a certain point, simply adding capacity does not make a more accurate model. That the results are uniform across dataset sizes indicates that the results are not driven by overfitting.

Figure 4.4 analyzes each model separately in greater detail. Shaded around the sample mean is the region encompassed by ± 1.96 standard deviations. The max and min values over the ten seeds are plotted in dotted lines. The min and max are roughly coincident with ± 1.96 standard deviations above and below the sample mean, so the distribution of results over seeds is even more thin-tailed than a Gaussian. The train error is plotted as a dashed line: slightly lower than test, but following the same trajectory.

This analysis shows that greater width does not seem to decrease approximation error. From this, we conclude that although the main results are proven only for $\mathcal{M}_d(R)$, empirically they translate well to more general and powerful function classes within the space of all feedforward ReLU networks. Section 5.D contains additional experimental results in order to further establish the robustness of this conclusion.

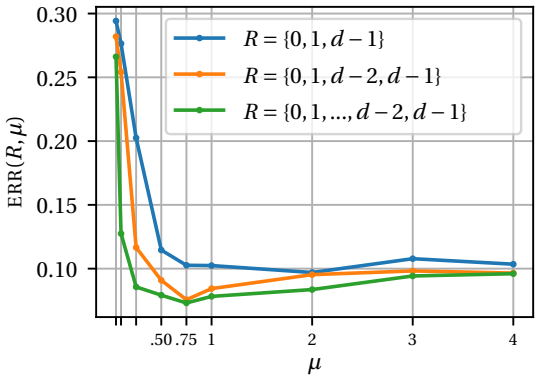


Figure 4.2: ERR(R,μ) (y-axis) does not continue to fall with greater μ (x-axis), meaning that expressivity is not increased with more capacity. Here $d = 8$.

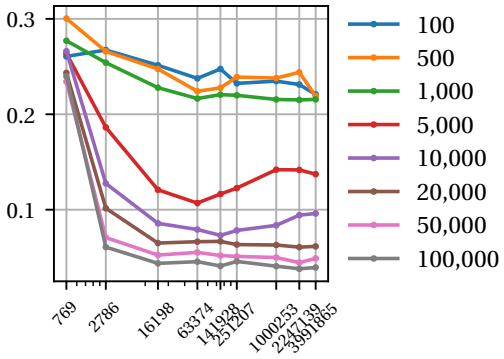


Figure 4.3: x-axis: number of parameters in a $R = \{0, 1, \ldots, d-2, d-1\}$ model, y-axis: error. Each line is a dataset size. Error declines with more data, but the pattern of Figure 4.2 is unchanged.

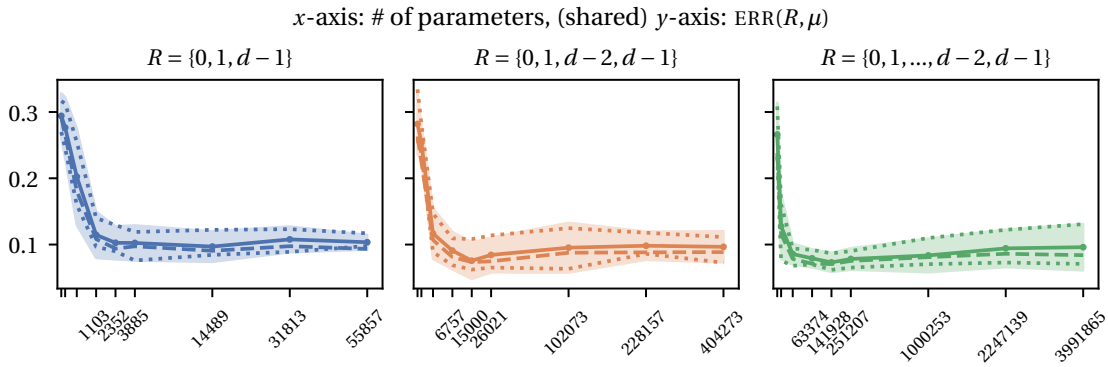
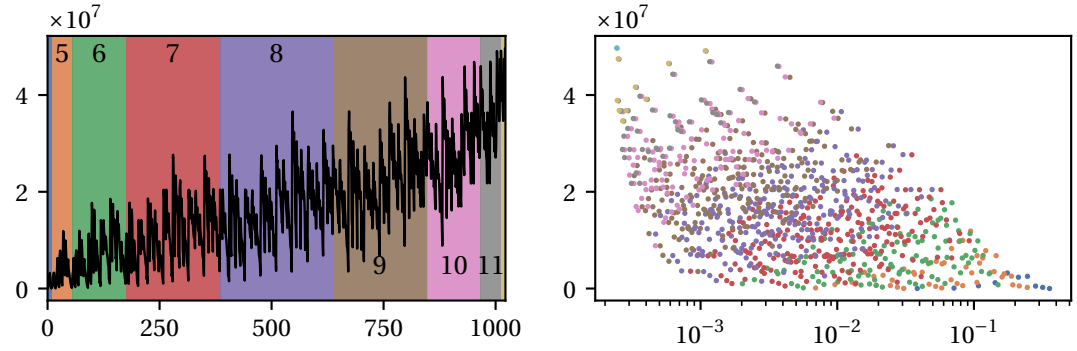


Figure 4.4: x-axes: number of parameters in R-estmators, y-axis: solid line: average test error, dashed line: average training error, dotted-lines: min / max over trials, shaded area: averaged test error ± 1.96 standard deviations. Here $d = 8$ and the parameter counts are dictated by varying μ from near zero up to 4.



(a) Parameter count for R ordered by size, then lexicographically within sizes. Each $|R|$ is distinguished by background color.

(b) (Logarithmic) x-axis: L_∞ error, y-axis: parameter count. Marker color is as in Figure 4.5a.

Figure 4.5: Estimator size and error in $d = 12$.

4.5.3 The Complexity of $\mathscr{M}_d(R)$

Equation (4.6) and Equation (4.10) represent quite disparate orders of error. This is because $\mathscr{M}_d(\{0,1,2,\ldots,d-1\})$ is much larger than $\mathscr{M}_d(\{0,1,d-1\})$, and there are many models in between. Figure 4.5 plots the number of parameters in a deep neural network representation of an R-estimator. Figure 4.5a relates the parameter counts to R. For $d = 12$, this ranges from less than 3500 parameters for $R = \{0,1,2\}$, to nearly 50 million for $R = \{0,1,\ldots,d-1\}$. Figure 4.5b further relates the model size to the L_∞ error bound computed in Section 4.C to convey a sense of how many parameters are needed to achieve a given error. In both plots, we group models by $|R|$, indicated by color.

For another view on model size that offers more insight on how network architecture is affected by dimension, Figure 4.6 plots the model sizes for networks implementing two $f_R^\star$ for two R, as a function of the model dimension, d. The depth of these models is always $\lfloor \log_2(\max(R)-1) \rfloor + 1$.

4.6 Conclusion

Motivated by a clear trend in computer vision architectures, we have posed and answered the question: can max pooling be replaced by linear mappings composed with ReLU activations? And when would it give a model that is considerably different?

To do this, we first established distance in intermediate feature space as the notion of comparison. Next, we established a simple baseline: max pooling with kernel

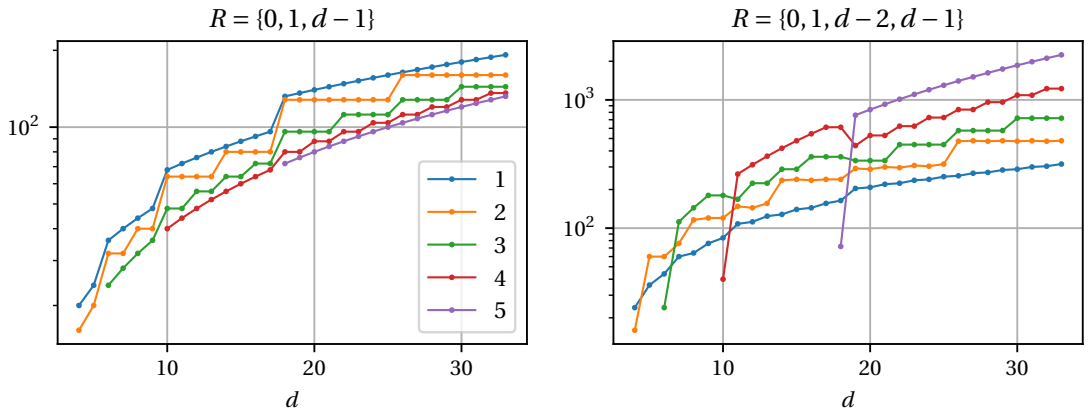


Figure 4.6: Model sizes for neural networks representing $f_R^\star$ for two different R. y-axis (logarithmic): hidden layer width, x-axis: d. Layer numbers by color are shown in the legend on the left plot. Depth of the requisite network at each dimension d is implied by the number of lines present, for example a $d = 17, R = \{0, 1, d-1\}$-estimator has $w(f) = (96, 80, 72, 68)$.

size of d can be computed by a block of $\log_2(d)$ narrow layers. We next introduced subpool max averages as a tractable class of approximators, and proved that the max function in d dimensions cannot be written as the linear combination of subpool max averages of order $< d$, though the error can be made as low as $1/2^d$. This novel approach complements existing work on the number of linear regions in piecewise linear networks. We experimentally extended our analysis to networks not constrained to have a fixed weight pattern by establishing that the class of subpool max averages was not less expressive than more general function classes.

4.A Proofs

4.A.1 Proof of Theorem 4

Proof. Let perm(d) denote the set of all permutations of $\{1, 2, \ldots, d\}$, and let

$$\mathcal{M}_d^s(R) = \{m \in \mathcal{M}_d(R) : m(x_{\sigma_1}, x_{\sigma_2}, \ldots, x_{\sigma_d}) = m(x_1, x_2, \ldots, x_d) \text{ for all } \sigma \in \text{perm}(d)\}$$

denote the restriction of $\mathscr{M}_d(R)$ to those elements that are invariant to a reordering of its arguments. Because max is symmetric, any optimal approximation to it should be in $\mathscr{M}_d^s(R)$ – that is the essence of Theorem 6.

Theorem 6. *For all R,*

$$\min_{m\in\mathscr{M}_d(R)} ||m-\max||_\infty = \min_{m\in\mathscr{M}_d^s(R)} ||m-\max||_\infty.$$

Proof. Assume otherwise, $\iff$ there is an m that is not symmetric and an x such that $m(x') < m^s(x)$ for all symmetric m^s and all x'. In particular, $m(x_\sigma) < m^s(x)$ for all permutations x_σ of x. Thus, since $m(x_\sigma) < m^s(x_\sigma) = m^s(x)$:

$$\frac{1}{d!}\sum_{\sigma\in\text{perm}(x)} m(x'_\sigma) < m^s(x).$$

However, $x \mapsto \frac{1}{d!}\sum_{\sigma\in\text{perm}(d)} m(x_\sigma)$ is evidently symmetric, a contradiction.

□

Thus, it is without loss of generality to optimize over $\mathscr{M}^s(R)$ rather than $\mathscr{M}(R)$. For $r>1$, $\omega_{jr}(1_d-\iota_k) = 1$[5] for all $k = 1,2,\ldots,d$ and all $j = 1,\ldots,\binom{d}{r}$ since there is only a single non-one value. This implies that all subpool maxes of order greater than 1 are equal, and for all k_1, k_2:

$$\sum_{r\in R\setminus\{0,1\}}\sum_{j=1}^{\binom{d}{r}} \beta_r^j\omega_{jr}(1_d-\iota_{k_1}) = \sum_{r\in R\setminus\{0,1\}}\sum_{j=1}^{\binom{d}{r}} \beta_r^j\omega_{jr}(1_d-\iota_{k_2}).$$

The restriction to symmetric functions moreover requires that

$$\sum_{j\neq k_1}\beta_1^j = \sum_{j=1}^{d}\beta_1^j\omega_{j1}(1_d-\iota_{k_1}) = \sum_{j=1}^{d}\beta_1^j\omega_{j1}(1_d-\iota_{k_2}) = \sum_{j\neq k_2}\beta_1^j.$$

[5] $\omega_{jr}(x)$ is defined in Section 4.4.1, the $C(j,r,d)$-subpool max of x.

Thus: $\beta_1^1 = \beta_1^2 = \ldots = \beta_1^d$. Call this single value β_1, and (for $\{0,1\} \subseteq R$)

$$\mathscr{M}_d^s(R) = \left\{ x \mapsto \beta_0 + \beta_1 S(x;1,d) + \sum_{r \in R \setminus \{0,1\}} \sum_{j=1}^{\binom{d}{r}} \beta_r^j \omega_{jr}(x) : \beta_r^j, \beta_0, \beta_1 \in \mathbb{R} \right\}.$$

Repeating this process for pools consisting of entirely of 1, except for $2,3,4,\ldots,d-1$ zeros in turn implies that the estimator must be a function of $S(x;r,d)$ alone, and not the individual terms of the sum separately for $r = 2,3,\ldots,d-1$:

$$\mathscr{M}_d^s(R) = \left\{ x \mapsto \sum_{r \in R} \beta_r S(x;r,d) : \beta_r \in \mathbb{R} \right\}. \tag{4.12}$$

So, the maximization over all $m \in \mathscr{M}_d(R)$ can be reduced to the maximization of $|R|$ scalar coefficients.

Put simply, to prove Theorem 4, we want to show that

$$\min_{\beta_0,\beta_1} ||x_{(1)} - \beta_0 - \beta_1 S(x;1,d)||_\infty = \frac{1}{2}\frac{d-1}{d} \tag{4.13}$$

$$\min_{\beta_{d-1}} ||x_{(1)} - \beta_{d-1} S(x;d-1,d)||_\infty = \frac{1}{2d-1} \tag{4.14}$$

$$\min_{\beta_0,\beta_{d-1}} ||x_{(1)} - \beta_0 - \beta_{d-1} S(x;d-1,d)||_\infty = \frac{1}{2d} \tag{4.15}$$

$$\min_{\beta_0,\beta_1,\beta_{d-1}} ||x_{(1)} - \beta_0 - \beta_1 S(x;1,d) - \beta_{d-1} S(x;d-1,d)||_\infty = \frac{1}{2(d+1)} \tag{4.16}$$

$$\min_{\beta_0,\beta_1,\beta_{d-2},\beta_{d-1}} ||x_{(1)} - \beta_0 - \beta_1 S(x;1,d) - \beta_{d-2} S(x;d-2,d) - \beta_{d-1} S(x;d-1,d)||_\infty = \frac{1}{d^2} \tag{4.17}$$

$$\min_{\beta_0,\beta_1,\ldots,\beta_{d-1}} ||x_{(1)} - \beta_0 - \beta_1 S(x;1,d) - \ldots - \beta_{d-1} S(x;d-1,d)||_\infty = \frac{1}{2^d} \tag{4.18}$$

where the L_∞ norm is taken over values of $x \in [0,1]^d$. The proof of all the above equations are similar, and share this prototype:

1. Conjecture coefficients, $\beta^\star$ with the help of Section 4.C.

2. Lower bound: for any f and for any set of points $P \subseteq [0,1]^d$,

$$\min_{\beta} \max_{x} |f(\beta, x)| \geq \min_{\beta} \max_{x \in P} |f(\beta, x)| = \max_{x \in P} |f(\beta^\star, x)|$$

   and prove the equality by contradiction: suppose there is a better β', and derive a contradiction.

3. Upper bound: for any f,

$$\min_{\beta} \max_{x} |f(\beta, x)| \leq \max_{x} |f(\beta^\star, x)|.$$

   Then show that $\max_x |f(\beta^\star, x)| \leq \beta_0^\star$ by writing it as a linear combination of order statistics.

For Equation 4.13, let $\begin{pmatrix}\beta_0^\star & \beta_1^\star\end{pmatrix} = \begin{pmatrix}(d-1)/(2d) & 1\end{pmatrix}$. Suppose that there were some $\begin{pmatrix}\beta_0' & \beta_1'\end{pmatrix}$ achieving a criterion $< \beta_0^\star$. Evaluating the error at $x = \begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix}$ and $x = \begin{pmatrix}1 & 0 & \dots & 0\end{pmatrix}$, this implies

$$\begin{aligned} 1 - \beta_0' - \beta_1'/d < +\beta_0^\star \text{ and } 1 - \beta_0' - \beta_1' > -\beta_0^\star &\implies (d-1)(1-\beta_0') < (d+1)\beta_0^\star \\ &\iff \beta_0' > 1 - \frac{d+1}{d-1}\beta_0^\star = \beta_0^\star. \end{aligned}$$

A contradiction to the condition at $x = 0$ that $|\beta_0'| \leq (d-1)/(2d)$. Thus, $\min_{\beta_0,\beta_1} ||x_{(1)} - \beta_0 - \beta_1 S(x;1,d)||_\infty \geq \beta_0^\star$. For the upper bound, from Theorem 3

$$x_{(1)} - \beta_0^\star - S(x;1,d) = \left(1 - \frac{1}{d}\right) x_{(1)} - \frac{1}{d}\left(x_{(2)} + \ldots + x_{(d)}\right) - \beta_0^\star \in \left[-\beta_0^\star, +\beta_0^\star\right].$$

□

For Equation 4.14, let $\beta_{d-1}^\star = \frac{2d}{2d-1}$. Suppose that there were some β_{d-1}' achieving a criterion strictly less than $1/(2d-1)$. Evaluating the criterion at $x = \begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix}$ and $x = \begin{pmatrix}1 & 1 & \dots & 1 & 0\end{pmatrix}$, this is only possible if

$$1 - \beta_{d-1}' > -\frac{1}{2d-1} \text{ and } 1 - \beta_{d-1}'\frac{d-1}{d} < \frac{1}{2d-1} \iff \frac{2d}{2d-1} > \beta_{d-1}' \text{ and } \beta_{d-1}' > \frac{2d}{2d-1},$$

which is to say it is impossible. For the upper bound, from Theorem 3

$$\begin{aligned} x_{(1)} - \frac{2d}{2d-1} S(x; d-1, d) &= x_{(1)} - \frac{2d}{2d-1}\left(\frac{d-1}{d} x_{(1)} + \frac{1}{d} x_{(2)}\right) \\ &= \frac{x_{(1)} - 2x_{(2)}}{2d-1} \in \left[-\frac{1}{2d-1}, +\frac{1}{2d-1}\right]. \end{aligned}$$

□

For Equation (4.15), let $\begin{pmatrix}\beta_0^\star & \beta_{d-1}^\star\end{pmatrix} = \begin{pmatrix}1/(2d) & 1\end{pmatrix}$. Suppose that there are some $\begin{pmatrix}\beta_0' & \beta_{d-1}'\end{pmatrix}$ achieving a criterion $< \beta_0^\star$. Evaluating the error at $x = \begin{pmatrix}1 & 1 & 1 & \dots & 1\end{pmatrix}$ implies

$$-1/(2d) < 1 - \beta_0' - \beta_{d-1}' \iff \beta_{d-1}' < 1 - \beta_0' + 1/(2d). \tag{4.19}$$

Evaluating the error at $x = \begin{pmatrix}1 & 0 & 0 & \dots & 0\end{pmatrix}$ implies

$$\begin{aligned} 1 - \beta_0' - \beta_{d-1}'(d-1)/d < +1/(2d) &\iff 1 - \beta_0' - 1/(2d) < \beta_{d-1}'(d-1)/d \\ &\iff \frac{d}{d-1}\left(1 - \beta_0' - 1/(2d)\right) < \beta_{d-1}'. \end{aligned} \tag{4.20}$$

Combining Equation 4.19 and Equation 4.20

$$\begin{aligned} \frac{d}{d-1}\left(1 - \beta_0' - 1/(2d)\right) < 1 - \beta_0' + 1/(2d) &\iff \left(\frac{d}{d-1} - 1\right)(1 - \beta_0') < \frac{1}{2d}\left(1 + \frac{d}{d-1}\right) \\ &\iff (1 - \beta_0') < \frac{2d-1}{2d} \\ &\iff \beta_0' > \frac{1}{2d}. \end{aligned} \tag{4.21}$$

A contradiction to the condition at $x = 0$ that $|\beta_0'| \leq 1/(2d)$. For the upper bound, from Theorem 3

$$x_{(1)} - \frac{1}{2d} - S(x; d-1, d) = \frac{1}{d}\left(x_{(1)} - x_{(2)}\right) - \frac{1}{2d} \in \left[-\frac{1}{2d}, +\frac{1}{2d}\right]. \tag{4.22}$$

□

For Equation (4.16), let $\begin{pmatrix}\beta_0^\star & \beta_1^\star & \beta_{d-1}^\star\end{pmatrix} = \begin{pmatrix}\frac{1}{2} & \frac{-d}{d-2} & \frac{d(d-1)}{d-2}\end{pmatrix}\frac{1}{d+1}$. Suppose that there were a $\begin{pmatrix}\beta_0' & \beta_1' & \beta_{d-1}'\end{pmatrix}$ achieving a criterion less than $\beta_0^\star$, then the condition at $x = \begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix}, \begin{pmatrix}1 & 1 & 0 & \dots & 0\end{pmatrix}$, and $\begin{pmatrix}1 & 0 & 0 & \dots & 0\end{pmatrix}$ imply:

$$\begin{aligned}
1-\beta_0'-\beta_1'-\beta_{d-1}' < +\beta_0' &\iff 1-\beta_1'-\beta_{d-1}' &< +2\beta_0' \\
1-\beta_0'-\beta_1'\frac{2}{d}-\beta_{d-1}' > -\beta_0' &\iff 1-\beta_1'\frac{2}{d}-\beta_{d-1}' &> 0 \\
1-\beta_0'-\beta_1'\frac{1}{d}-\beta_{d-1}'\frac{d-1}{d} < +\beta_0' &\iff 1-\beta_1'\frac{1}{d}-\beta_{d-1}'\frac{d-1}{d} &< +2\beta_0'
\end{aligned}$$

combining the first and the second implies $-\frac{2d}{d-2}\beta_0' \le \beta_1'$, while combining the second and third implies $\beta_1' \le \frac{2d^2}{d-2}\beta_0' - \frac{d}{d-2}$. Combining these

$$-\frac{2d}{d-2}\beta_0' < \frac{2d^2}{d-2}\beta_0' - \frac{d}{d-2} \iff \beta_0' > \frac{1}{2(d+1)},$$

a contradiction to the condition at $x = 0$ that $|\beta_0'| \le 1/(2(d+1))$. For the upper bound, from Theorem 3

$$\begin{aligned}
&x_{(1)} - \frac{1}{2(d+1)} + \frac{1}{(d+1)(d-2)}(x_{(1)} + \ldots + x_{(d)}) - \frac{(d-1)}{(d+1)(d-2)}\left((d-1)x_{(1)} + x_{(2)}\right) \\
=&\frac{1}{d+1}(x_{(1)} - x_{(2)}) + \frac{1}{(d+1)(d-2)}(x_{(3)} + \ldots + x_{(d)}) - \frac{1}{2(d+1)} \in \left[-\frac{1}{2(d+1)}, +\frac{1}{2(d+1)}\right].
\end{aligned}$$

□

For Equation (4.17): $\begin{pmatrix}\beta_0^\star, \beta_1^\star, \beta_{d-2}^\star, \beta_{d-1}^\star\end{pmatrix} = \begin{pmatrix}\frac{1}{d^2} & \frac{2}{d(d-3)} & -1-\frac{2}{d(d-3)} & 2\end{pmatrix}$. Suppose that there were a $\begin{pmatrix}\beta_0' & \beta_1' & \beta_{d-2}' & \beta_{d-1}'\end{pmatrix}$ achieving a criterion less than $\beta_0^\star$. In order to scale up the proof by contradiction, we use Theorem 7.

Theorem 7 (Carver [26], Theorem 3)**.** *$Ax < b$ is consistent $\iff$ $y = 0$ is the only solution for $y \ge 0$, $y^\top A = 0$, $y^\top b \le 0$.*

Applying the assumed condition at $x = (0,0,\ldots,0)$, $(1,1,\ldots,1)$, $(1,1,1,0,\ldots,0)$, $(1,1,0,\ldots,0)$, and $(1,0,\ldots 0)$ imply that:

$$\begin{aligned}
\beta'_0 &< +\beta^\star_0 \\
1-\beta'_0-\beta'_1-\beta'_{d-2}-\beta'_{d-1} &> -\beta^\star_0 \\
1-\beta'_0-\beta'_1\frac{3}{d}-\beta'_{d-2}-\beta'_{d-1} &< +\beta^\star_0 \\
1-\beta'_0-\beta'_1\frac{2}{d}-\beta'_{d-2}\frac{(d+1)(d-2)}{d(d-1)}-\beta'_{d-1} &> -\beta^\star_0 \\
1-\beta'_0-\beta'_1\frac{1}{d}-\beta'_{d-2}\frac{d-2}{d}-\beta'_{d-1}\frac{d-1}{d} &< +\beta^\star_0
\end{aligned}$$

or $Ax < b$ for

$$A=\begin{pmatrix} +1 & 0 & 0 & 0 \\ +1 & +1 & +1 & +1 \\ -1 & -\frac{3}{d} & -1 & -1 \\ +1 & +\frac{2}{d} & \frac{(d+1)(d+2)}{d(d-1)} & +1 \\ -1 & -\frac{1}{d} & \frac{d-2}{d} & -\frac{d-1}{d} \end{pmatrix}, x=\begin{pmatrix} \beta'_0 \\ \beta'_1 \\ \beta'_{d-2} \\ \beta'_{d-1} \end{pmatrix}, \text{ and } b=\begin{pmatrix} \beta^\star_0 \\ \beta^\star_0+1 \\ \beta^\star_0-1 \\ \beta^\star_0+1 \\ \beta^\star_0-1 \end{pmatrix}.$$

It is straightforward to verify that

$$y=\begin{pmatrix} 1/d \\ (d-2)/(2d) \\ (d/2-1) \\ (d-1)/2 \\ 1 \end{pmatrix}$$

satisfes $y^\top A = 0$ and $y^\top b = (\beta^\star_0 d - 1/d) = 0$. The last equality is because $\beta^\star_0 = 1/d^2$. This demonstrates a $y \geq 0, y \neq 0$ with $y^\top b \leq 0$, thus the system is not consistent, which forms a contradiction to the supposition that there exists a $\begin{pmatrix}\beta'_0 & \beta'_1 & \beta'_{d-2} & \beta'_{d-1}\end{pmatrix}$ achieving a criterion less than $\beta^\star_0$. For the upper bound, from Theorem 3

$$\begin{aligned}
&x_{(1)}-\frac{1}{d^2}-\frac{2}{d(d-3)}\frac{1}{d}(x_{(1)}+\ldots+x_{(d)})+\frac{(d-2)(d-1)}{d(d-3)}2S(x;d-2,d)-\frac{2}{d}\left((d-1)x_{(1)}+x_{(2)}\right) \\
=&\frac{2}{d^2}x_{(1)}-\frac{2}{d^2}x_{(2)}+\frac{2}{d^2}x_{(3)}-\frac{2}{d^2}\frac{1}{d-3}(x_{(4)}+\ldots+x_{(d)})-\frac{1}{d^2} \in \left[-\frac{1}{d^2},+\frac{1}{d^2}\right].
\end{aligned}$$

□

The idea of Equation (4.18) is the same, but handling $d+1$ equations simultaneously requires more powerful notation. Let $B(d)$ be the $d-1 \times d$ matrix with (r,c)th element $\binom{d-c}{r-1}/\binom{d}{r}$ if $r+c \leq d+1$, and zero otherwise. Write the condition Equation (4.4) simultaneously for all r as

$$S(x;d)^\top \triangleq \begin{pmatrix} S(x;0,d) \\ S(x;1,d) \\ S(x;2,d) \\ S(x;3,d) \\ \vdots \\ S(x;d-1,d) \end{pmatrix}^\top = \begin{pmatrix} 1 \\ x_{(1)} \\ x_{(2)} \\ \vdots \\ x_{(d)} \end{pmatrix}^\top \begin{pmatrix} 1 & 0_{d-1}^\top \\ 0_d & B(d)^\top \end{pmatrix} \in \mathbb{R}^{1\times d}. \tag{4.23}$$

Let $V(d)$ be the $d \times d+1$ matrix of points at which the estimator is evaluated.

$$V(d) \triangleq \begin{pmatrix} 0 & 1 & 1 & \dots & 1 & 1 \\ 0 & 0 & 1 & \dots & 1 & 1 \\ \vdots & \dots & & \vdots & & \\ 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \tag{4.24}$$

Let $s(d)$ be the $(d+1)$-dimensional vector starting and ending with jth element $(-1)^{j-1}$; $\mathrm{diag}(s(d))$ encodes the signs of the binding inequalities. Let

$$A = \mathrm{diag}(s(d)) \begin{pmatrix} 1_{1\times d+1} \\ B(d)V(d) \end{pmatrix}^\top, b = 1/2^d + \mathrm{diag}(s(d)) \begin{pmatrix} 0 \\ 1 \\ 1 \\ \dots \\ 1 \end{pmatrix}.$$

where we indicate the dimensionality of the $d+1$-dimensional vector of ones. Here

$$y = \begin{pmatrix} \binom{d}{0} \\ \binom{d}{1} \\ \binom{d}{2} \\ \vdots \\ \binom{d}{d-1} \\ \binom{d}{d} \end{pmatrix}$$

satisfies $y^\top A = 0$ and $y^\top b = 0$. Thus, there are no parameters achieving a criterion $< 1/2^d$, by Theorem 7. For the other direction, let

$$\beta^\star = \begin{pmatrix} \beta_0^\star \\ \beta_1^\star \\ \vdots \\ \beta_{d-1}^\star \end{pmatrix} = -1 \times \begin{pmatrix} -1/2^d \\ (-1/2)^{d-1}\binom{d}{1} \\ (-1/2)^{d-2}\binom{d}{2} \\ \vdots \\ (-1/2)^1\binom{d}{d} \end{pmatrix}.$$

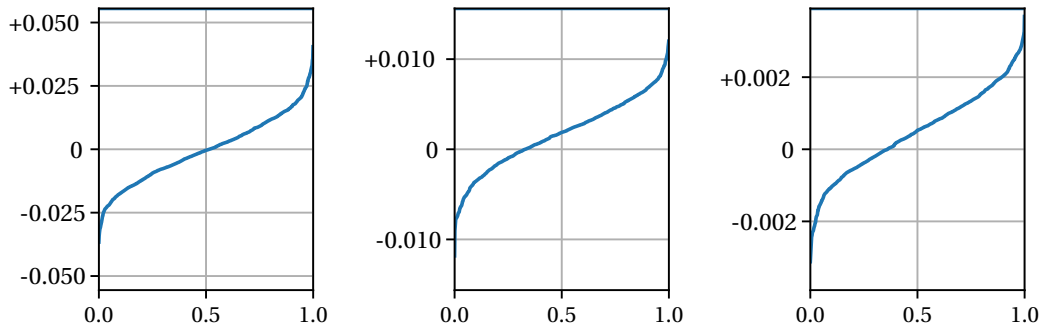
Then, by the binomial theorem, $\sum_{k=0}^n \binom{n}{k} r^k = (1+r)^n$:

$$-\begin{pmatrix} 1 & 0_{1\times d-1} \\ 0_{d\times 1} & B(d)^\top \end{pmatrix} \beta^\star = \begin{pmatrix} 1/2^d \\ 1 - 2/2^d \\ +2/2^d \\ -2/2^d \\ \vdots \end{pmatrix}.$$

Thus

$$x_{(1)} - S(x;d)^\top \beta^\star = 1/2^d + \sum_{j=1}^{d} \frac{2}{2^d}(-1)^{j+1} x_{(j)} \in \left[-\frac{1}{2^d}, +\frac{1}{2^d} \right]. \tag{4.25}$$

□



(a) $R = \{0, 1, d-1\}$ (b) $R = \{0, 1, d-2, d-1\}$ (c) $R = \{0, 1, d-2, d-1\}$

Figure 4.7: Simulated empirical cumulative distribution functions for $x_{(1)} - f_R^\star(x)$ over 1500 uniform random points in the unit cube, for three R. The y axis is scaled between $\pm$ the maximum L_∞ error.

4.B Distribution of Optimal Error

This section derives the distribution of the optimal residual.

4.B.1 A Foundational Result

Weisberg [169] gives this result:

> Let $x_{(0)}, x_{(1)}, x_{(2)}, \ldots, x_{(d)}, x_{(d+1)}$ be the order statistics from x sampled uniformly on the d-dimensional unit cube with the convention that $x_{(0)} = 0, x_{(d+1)} = 1$.
>
> Let $\alpha_j \geq 0, j = 1, \ldots, d$ be positive weights, and let $a_j = \sum_{k=j}^{d} \alpha_k$, so $a_1 \geq a_2 \geq \ldots \geq a_d$. Let $c_1 > c_2 > \ldots > c_s$ be the unique (strictly ordered) values of a_k, and k_ℓ denotes the number of a_j that c_ℓ covers.
>
> Then

$$\Pr\left[\sum_{j=1}^{d} x_{(j)}\alpha_j \leq z\right] = 1 - \sum_{i=1}^{r} \frac{\mathrm{d}^{k_i-1}}{\mathrm{d}x^{k_i-1}} \frac{(x-z)^d}{\prod_{\ell \neq i}(x-c_\ell)}\bigg|_{x=c_i}, \tag{4.26}$$

> where r is the largest i where $z \leq c_i$.

Matsunawa [110] also gives a characteristic function-based analysis.

In our application, the weights in the linear combination are not nonnegative. Happily, there is a reduction to this case presented by Diniz, Silva, and Gail [43] (modernized from Dempster and Kleyle [40]):

> Let α_j be as before, though not necessarily nonnegative this time, and as before let $a_j = \sum_{k=j}^{d} \alpha_k$. Let $a^0 = (0, a_1, \ldots, a_d)$ be a prepended with a zero and let σ be the permutation of the $\{1, \ldots, d+1\}$ so that $a^0_{\sigma(1)} \geq \ldots \geq a^0_{\sigma(d+1)}$. Then

$$\Pr\left[\sum_{j=1}^{d} x_{(j)}\alpha_j \leq z\right] = \Pr\left[\sum_{j=1}^{d} x_{(j)}(a^0_{\sigma(j)} - a^0_{\sigma(j+1)}) \leq z - a^0_{\sigma(d+1)}\right]. \tag{4.27}$$

Basically: including zero with the a as calculated previously, sort them, then using the previous logic with the differences (guaranteed to be nonnegative), and a modified argument, gives the same probability. In this equation, some differences may be zero, and that is covered in Equation (4.26) with zero α_js.

Actually evaluating the derivatives in Equation (4.26) is not difficult – there is an efficent recursive formulation – but is also not totally straightforward. Thus, for brevity we do not go into too more detail and instead simply observe that the cumulative distribution function is a continuously differentiable function of its input. To give an idea of the distribution, Figure 4.7 plots the empirical distribution for 1500 points.

4.B.2 The Residual as a Linear Combination of Order Statistics

The optimal residual is a linear combination of order statistics, from Equation (4.12) and Equation (4.4):

$$\begin{aligned} f^\star_R(x) &= \sum_{r\in R} \beta^\star_r S(x; r, d) \\ &= \sum_{r\in R} \beta^\star_r \frac{1}{\binom{d}{r}} \sum_{j=1}^{d-r+1} \binom{d-j}{r-1} x_{(j)} \\ &= \sum_{k=1}^{d} a^\star_k x_{(k)} \text{ where } a^\star_k = \sum_{r\in R\cap\{0,\ldots,d-k+1\}} \frac{\beta^\star_r}{\binom{d}{r}} \binom{d-k}{r-1}. \end{aligned}$$

Thus, the optimal residual is a linear function of the order statistics:

$$x_{(1)} - f_R^\star(x) = (1 - a_1^\star)x_{(1)} - \beta_0^\star - \sum_{r \in R \setminus \{0,1\}} a_r^\star x_{(r)}.$$

The cumulative distribution of this quantity then follows from plugging these coefficients into Equation (4.27). This distribution is a a polynomial, so continuous, thus the probability that it takes any discrete value is zero.

4.C Bounding the Error and Complexity of a General R-Estimator

This section presents a method for computing nontrivially tight lower bounds on the error from a general $R \subseteq \{0, 1, 2, \ldots, d-1\}$-estimator. For any set of points $P \subseteq [0,1]^d$,

$$\begin{aligned} ||m - \max||_\infty \geq \max_{x \in P} |m(x) - \max(x)| \implies \\ \min_{m \in \mathcal{M}_d^s(R)} ||m - \max||_\infty \geq \min_{m \in \mathcal{M}_d^s(R)} \max_{x \in P} |m(x) - \max(x)|. \end{aligned}$$

Apply this with P equal to $d+1$ corners of the unit cube containing $0, 1, \ldots$ ones:

$$\begin{aligned} ||m - \max||_\infty \geq \min_{m \in \mathcal{M}_d^s(R)} \max \{ & |m(\begin{pmatrix} 0 & 0 & \ldots & 0 & 0 \end{pmatrix})|, \\ & |m(\begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \end{pmatrix}) - 1|, \\ & |m(\begin{pmatrix} 1 & 1 & \ldots & 0 & 0 \end{pmatrix}) - 1|, \\ & \ldots \\ & |m(\begin{pmatrix} 1 & 1 & \ldots & 1 & 0 \end{pmatrix}) - 1|, \\ & |m(\begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \end{pmatrix}) - 1| \}. \end{aligned}$$

Finally, write this as a convex optimization problem in $2(d+1)$ constraints and $1 + |R|$ variables, using a a standard trick for rewriting L_∞ optimization (see Boyd and Vandenberghe [18]).

$$\min_{g,\beta_0,(\beta_r,r\in R\setminus\{0\})} g \text{ subject to } \begin{aligned}
&\left|\beta_0 + \sum_{r\in R\setminus\{0\}} \beta_r S(\begin{pmatrix}0 & 0 & \dots & 0 & 0\end{pmatrix}; r, d)\right| \le g,\\
&\left|\beta_0 - \sum_{r\in R\setminus\{0\}} \beta_r S(\begin{pmatrix}1 & 0 & \dots & 0 & 0\end{pmatrix}; r, d) - 1\right| \le g,\\
&\left|\beta_0 - \sum_{r\in R\setminus\{0\}} \beta_r S(\begin{pmatrix}1 & 1 & \dots & 0 & 0\end{pmatrix}; r, d) - 1\right| \le g,\\
&\dots\\
&\left|\beta_0 - \sum_{r\in R\setminus\{0\}} \beta_r S(\begin{pmatrix}1 & 1 & \dots & 1 & 0\end{pmatrix}; r, d) - 1\right| \le g,\\
&\left|\beta_0 - \sum_{r\in R\setminus\{0\}} \beta_r S(\begin{pmatrix}1 & 1 & \dots & 1 & 1\end{pmatrix}; r, d) - 1\right| \le g.
\end{aligned}$$

This computation scales well: linear programs such as this can be simply solved on a desktop computer using standard software for thousands of variables and constraints.

4.D Implementing an R Estimator as a Feedforward Network

This section demonstrates how to cast a general R-estimator as the forward pass of a feedfoward network. This analysis is necessary to give a benchmark against which to compare stochastic gradient descent fitting.

First, we describe a concept called the R-mapping and an algorithm for computing R-mappings, and then we show how to use an R-mapping to construct a feedforward ReLU network that is an R-estimator.

4.D.1 R-Mapping Definition and Motivation

An *R-mapping* describes an R-estimator as a sequence of pairwise maxes. For $d \in \mathbb{N}$ and $r \in \{1,2,\dots,d\}$ let $C(r,d) = \{\{1,2,\dots,r\},\{1,3,\dots,r+1\},\dots,\{d-r,\dots,d\}\}$ denote the set of size $\binom{d}{r}$ of all subsets of $\{1,2,\dots,d\}$ of size r.

Definition 3. *An R-mapping for $R \subseteq \{0,1,2,\dots,d\}$ is a sequence of sets $t_1, t_2, \dots, t_s$ with $s \le \lceil \log_2 d \rceil$ satisfying:*

- *for all* $r \in R, r > 0$ *there is some* j *such that* $C(r,d) \subseteq t_j$, *and*
- $t_{j+1} \subseteq \{\tau_1 \cup \tau_2 : \tau_1, \tau_2 \in t_j\}$.

Each element of an R-mapping corresponds to a set of indices into the input, and at the jth layer computes the max of the input over all indices in t_j. The first defining characteristic of an R-mapping ensures that the indices permit the evaluation of all required subpool max averages. And the second condition ensures that a feedforward ReLU network can compute the maxes over the implied indices.

4.D.2 Computing R-Mappings

This section shows how to compute an R-estimator. Say that R is *adequate* if $\max(R) \leq 2 \times \max\left(R \cap [0, 2^{\lfloor \log_2(\max(R)-1) \rfloor}]\right)$. If R is adequate then it possible to form the greatest remaining term from pairwise maxes of terms that are a lower power of two – a condition necessary to enforce the second condition in Definition 3. If R is not adequate, then it can be brought closer to adequacy by appending an additional term. Let $\widetilde{R} = a(R)$, where $a : \{0,\ldots,d\} \mapsto \{0,\ldots,d\}$ is defined recursively as:

$$a(R) = \begin{cases} R & \text{if } R = \{0,1\} \\ \{\max(R)\} \cup a(R \backslash \{\max(R)\}) & \text{if } R \text{ is adequate} \\ \{\max(R)\} \cup a(R \backslash \{\max(R)\}) \cup \{\lceil \max(R)/2 \rceil\} & \text{otherwise.} \end{cases} \tag{4.28}$$

The third case covers the situation where it would not be possible to compute an R-mapping out of terms in R, and so an additional term is appended. For example, $a(\{0,1,2,5,6\}) = \{0,1,2,3,5,6\}$: 3 has been appended since it is impossible to compute the maxes of five and six terms using only pairwise maxes ($r = 2$) of pairs of variables. $a(R)$ reduces its argument by one term with each recursive call, and thus it is fast and straightforward to evaluate.

By construction, every truncation of $\widetilde{R}$ is adequate, thus for every $\tilde{r} \in \widetilde{R}$ with $\tilde{r} > 1$ there exists a $\tilde{r}' \in \widetilde{R}$ with $\tilde{r}' \geq \tilde{r}/2$. This means that

$$C(1,d), C(2,d), \cup_{r \in R \cap \{3,4\}} C(r,d), \ldots, \cup_{r \in R \cap \{d/2,\ldots,d\}} C(r,d)$$

is a R-mapping, however if $R \subsetneq \widetilde{R}$, then there will be smaller R-mappings since it is possible to skip the computation of some terms in $C(r,d)$. For example, there are 56

subsets of size 3 of $d = 8$ values, but 36 terms of length 3 can be combined to form all subsets of size 5 and 6.

For a vector $A \in \mathbb{R}^\ell$ with $\ell > k$, let $\text{SPLIT}_k : \mathbb{R}^\ell \to \mathbb{R}^k \times \mathbb{R}^k$ be the function that splits its argument into the first and last k elements: $\text{SPLIT}_k(A) = (A_{(1)}, \ldots, A_{(k)}), (A_{(\ell-k)}, \ldots, A_{(\ell)})$. And let FLATSPLIT_k be the set-valued function that applies SPLIT to each of its inputs and collects the outputs into a single set

$$\begin{aligned}&\text{FLATSPLIT}_k(\{A^1, A^2, \ldots, A^n\}) \\ &= \{(A^1_{(1)}, \ldots, A^1_{(k)}), (A^2_{(\ell-k)}, \ldots, A^2_{(\ell)}), \ldots, (A^n_{(1)}, \ldots, A^n_{(k)}), (A^n_{(\ell-k)}, \ldots, A^n_{(\ell)})\}.\end{aligned}$$

Note that computing all subpool maxes entails some redundancy: for example

$$\begin{array}{l} \max(x_1, x_2, x_3) = \max(z_1, z_2) \\ \max(x_1, x_2, x_4) = \max(z_1, z_3) \\ \max(x_1, x_3, x_4) = \max(z_2, z_3) \\ \max(x_2, x_3, x_4) = \max(z_4, z_5) \end{array} \quad \text{where} \quad \begin{array}{l} z_1 = \max(x_1, x_2) \\ z_2 = \max(x_1, x_3) \\ z_3 = \max(x_1, x_4). \\ z_4 = \max(x_2, x_3) \\ z_5 = \max(x_2, x_4) \end{array}$$

In particular, just 5 out of the $\binom{4}{2} = 6$ terms suffice. This is the idea of Algorithm 3, which gives one approach to compute an R-mapping that computes only a minimal subset of terms necessary to support the computation of subsequent terms.

4.D.3 R-Mappings to Neural Network R-Estimators

An R-mapping (Definition 3), is a sequence of sets where each element of a set is associated with a pair of elements in the previous set. Thus, it is well-suited to compute pairwise maxes via a simple linear-ReLU-linear block as shown in Equation (4.2). Computing the average of all subpooled values is a linear operation.

An important book-keeping challenge with this approach is to enable the network to convey the average of low-order subpool maxes through the network. To do this, we append to the network a "memory" – additional neurons which carry forward values computed earlier in the network via identity mappings (propagated through ReLUs via $x = \text{ReLU}(+x) - \text{ReLU}(-x)$) through until the end – a layer of width $|R|$, containing all needed subpool max averages. Finally, these values are aggregated according to β.

Data: $R \subseteq \{0, 1, \ldots, d\}$
Result: R-mapping suitable for evaluating $f_R^\star$
$\widetilde{R} \leftarrow a(R)$ // Augment R with needed terms
$s = \lceil \log_2(\max R) \rceil$
$R_j \leftarrow R \cap \{i : i \in \mathbb{N}, \lceil \log_2 i \rceil = j\}$ for $j = 1, 2, \ldots, s$ // group R by power of 2
$\widetilde{R}_j \leftarrow \widetilde{R} \cap \{i : i \in \mathbb{N}, \lceil \log_2 i \rceil = j\}$ for $j = 1, 2, \ldots, s$ // group $\widetilde{R}$ by power of 2
for $i = 0, 1, \ldots, s-1$ **do**
  $j \leftarrow s - i - 1$ // backward index
  if $i = 0$ **then**
    $N_j \leftarrow \emptyset$
  else
    $N_j \leftarrow t_{j+1}$ // terms needed for subsequent layer
  end
  $X_j \leftarrow \{C(k, r, d) : k = 1, \ldots, \binom{d}{r}, r \in R_j\}$ terms needed for this layer
  $Y_j \leftarrow X_j \cup N_j$
  $k_j \leftarrow \max(\widetilde{R}_j)$ // tuple width in this mapping term
  $t_j \leftarrow \text{FLATSPLIT}_{k_j}(Y_j)$
end
Return $t_1, t_2, \ldots, t_s$

Algorithm 3: Computation of an R-mapping

The code is written in three stages: (1) compute a base network consisting of the linear layers implied by Equation (4.2), then (2) append the subpool averages and their attendant memory neurons, and finally (3) aggregate the penultimate layer value with the coefficients. One helpful trick to developing this logic is to leave each step above as consecutive linear layers, then once everything is complete, to fuse them all together.

This approach perhaps does not result in the smallest possible network, but it is simple, fast, and the architecture is very descriptive of the logic the network implements. It seems unlikely that a large improvement on this general scheme is possible, though we do not attempt to prove this speculation.

4.E Additional Experiments

This section repeats Figure 4.2 for configurations different than the one described in Section 4.5.1 in exactly one way, described in the caption. Across Figure 4.8, Figure 4.9, Figure 4.10, Figure 4.11, Figure 4.12, and Figure 4.13 the basic pattern of quickly falling error that levels out around $\mu = 1$ for all three models is repeated.

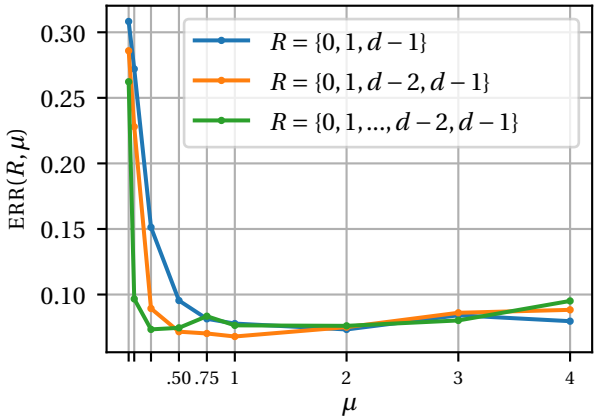


Figure 4.8: Figure 4.2 with Kaiming initialization for weights and small positive constant for bias.

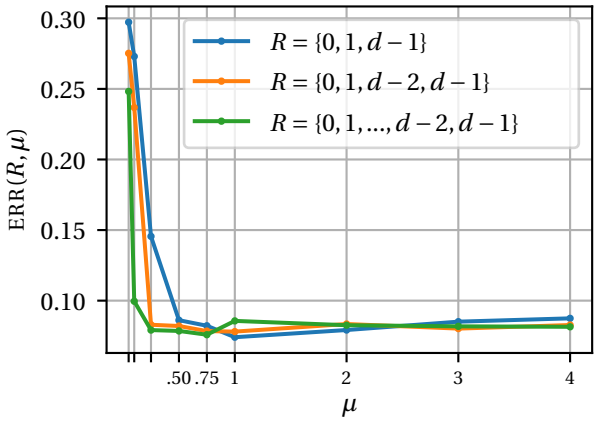


Figure 4.9: Figure 4.2 with Xavier initialization for weights and small positive constant for bias.

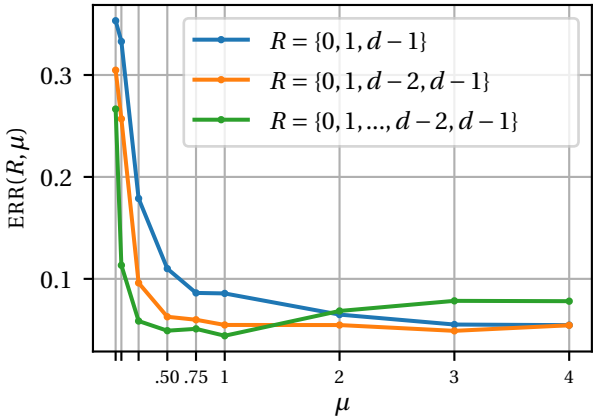


Figure 4.10: Figure 4.2 with L_∞ criterion replaced with L_2 criterion.

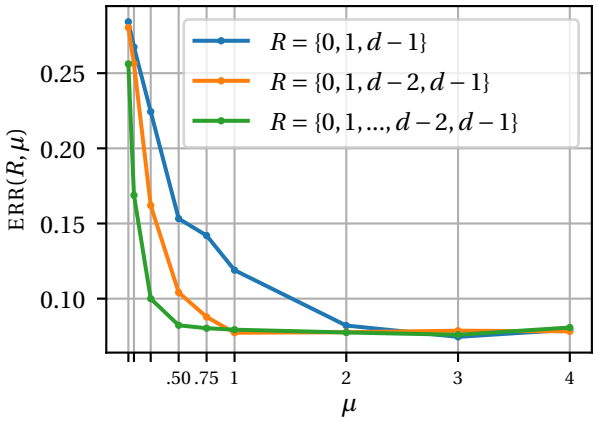


Figure 4.11: Figure 4.2 with Adam optimizer replaced with AdamW optimizer (PyTorch default parameters).

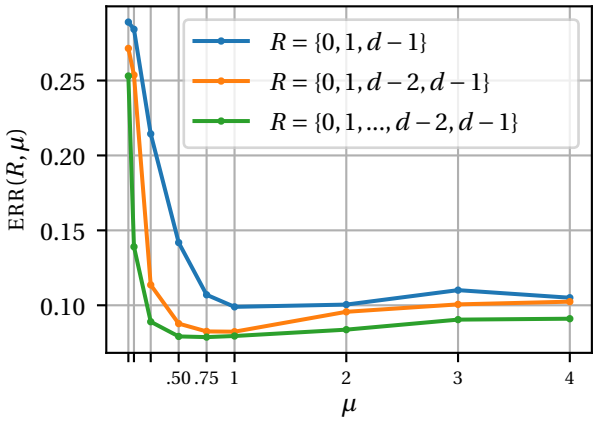


Figure 4.12: Figure 4.2 with pseudorandom uniform data replaced with Sobol sequence.

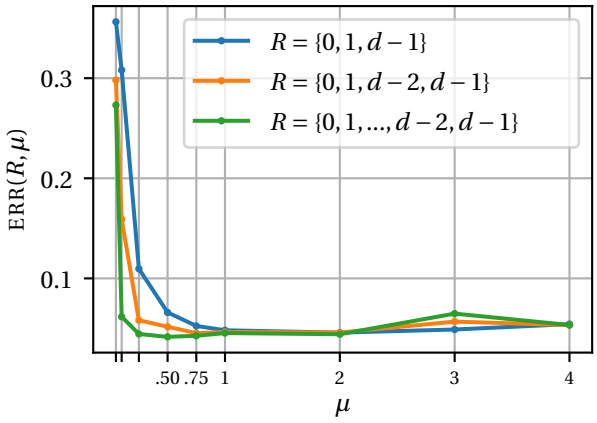


Figure 4.13: Figure 4.2 with pseudorandom uniform data replaced with Dirichlet$(1, \ldots, 1)$ data.

Chapter 5

Curvature

An important takeaway from Chapter 3 is that linearity is essentially "free" as it pertains to model complexity. Moreover, small reductions in nonlinearity can imply large reductions in complexity. Thus, a principled way to make a network "more linear" is called for. That is the idea of this chapter, which is based on Srinivas et al. [149].

Suraj and I jointly and equally developed the idea, the main theorem, the novel activation function, and the other layer modifications and criterion that finally characterized a LCNN. In the latter stages, I validated the Lipschitz one-ness conditions, ran more of the experiments and developed the tensor-based proof of the curvature bound, whilst Suraj did more of the writing up of the experiments, and proof of the gradient stability and adversarial robustness.

Suraj was supported in part by NSF awards #IIS-2008461 and #IIS-2040989, and research awards from Google, JP Morgan, Amazon, Harvard Data Science Initiative, and the D^3 Institute at Harvard.

Note that differently to the other work in this thesis, we here do not use piecewise linear activations (an important contribution is to posit a new activation). This is only to enable simple calculus only; by standard arguments for approximating nonsmooth functions as the limit of smooth functions, the intuition holds for piecewise linear activations. A formal argument to this effect is in Dombrowski et al. [45].

To motivate more and less linear activations, consider the "leaky relu" activation function of Maas, Hannun, and Ng [104], $\text{LeakyReLU}_\alpha = x \mapsto \text{ReLU}(x) + \alpha \times \min(0, x)$ with $\alpha \in [0, 1]$. Clearly, $\text{LeakyReLU}_\alpha(x) = (1-\alpha)\text{ReLU}(x) + \alpha x$, thus the leaky relu activation interpolates between the highly nonlinear relu activation (for $\alpha = 0$), and the identity

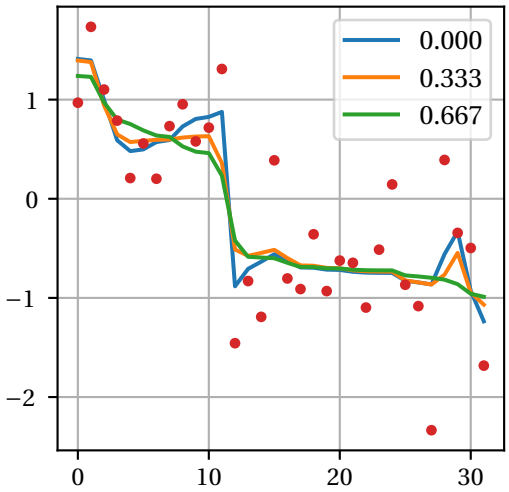


Figure 5.1: More "leakiness" α in a LeakyReLU$_\alpha$ activation makes for a more linear and less expressive model.

mapping (for $\alpha = 1$). In this sense, $1 - \alpha$ is in an increasing correspondence with curvature, and its consequences are demonstrated in Figure 5.1, where we train three deep neural networks $\mathbb{R} \to \mathbb{R}$, identical except for α in the leaky relu activation function, are trained on a noisy dataset. Less curved (higher α) models overfit less.

A basic premise of this is analysis is that linearity itself is valuable. The chapter captures this goal circumspectly, via adversarial robustness, gradient stability, etc. But the motivation is deeper in that linearity implies untold attractive properties. For example, *reproducbility* (similar results for similar trainings), robustness to label noise, composability with other optimizations, or compressability are a few of the many attributes that are implied by linearity.

5.1 Introduction

The nonlinearity of deep neural networks is critical to achieving good performance in complex tasks such as image classification, language modelling and generative modelling of images (He et al. [74], Chowdhery et al. [28], and Ramesh et al. [128]). However, excessive flexibility (nonlinearity) is undesirable as this can lead to model under-specification (see D'Amour et al. [38]) which results in unpredictable behaviour on out-of-domain inputs, such as vulnerability to adversarial examples. This work

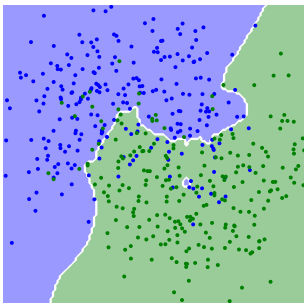
develops a method for training neural network models without excess non-linearity (see Figure 5.2b for example), such that predictive performance remains unaffected.

To do this, a precise notion of curvature – a mathematical quantity that encodes the flexibility or the degree of non-linearity of a function – is required. In deep learning, the curvature of a function at a point is often quantified as the norm of the Hessian (matrix of mixed second derivatives) at that point, for example Moosavi-Dezfooli et al. [115] and Dombrowski et al. [45]. The Hessian is zero everywhere if and only if the function is linear, making Hessian norm a suitable measure of non-linearity. However, the Hessian depends on the scaling of model gradients, which makes it unsuitable to study its interplay with model robustness. In particular, Hein and Andriushchenko [75] shows that robust models have small gradient norms, which naturally imply smaller Hessian norms. But are they truly more linear as a result? To be able to study robustness independent of non-linearity, we propose *normalized curvature*, which normalizes the Hessian norm by its corresponding gradient norm, thereby disentangling the two measures. Experimentally, it seems that normalized curvature is a stable measure across train and test samples (see Table 5.3), whereas existing curvature measures are not.

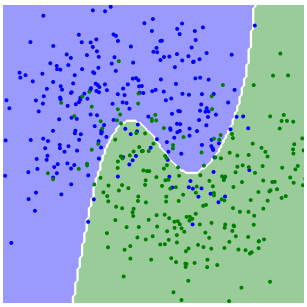
One approach to train low-curvature models is to directly penalize curvature at training samples, see Kanbak, Moosavi-Dezfooli, and Frossard [86] and Qin et al. [126]. However, these methods require expensive Hessian computations, and only minimize local point-wise curvature and not curvature everywhere. A complementary approach is presented in Dombrowski et al. [45], who propose architectures that have small global curvature, but do not penalize curvature during training. In contrast, this chapter proposes efficient mechanisms to directly penalize the *normalized curvature* globally. In addition, while Moosavi-Dezfooli et al. [115] and Dombrowski et al. [45] penalize the Hessian Frobenius norm, here the spectral norm is penalized, providing tighter and more interpretable robustness bounds.

The overall contributions of this chapter are:

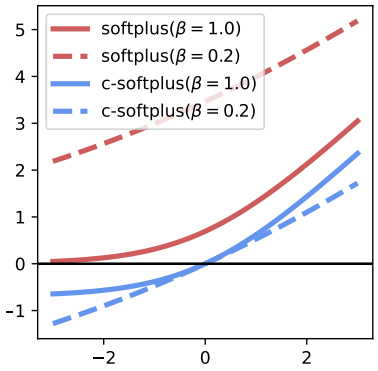
1. Section 5.3.1 proposes measuring the curvature of deep models via *normalized curvature*, which is invariant to scaling of the model gradients, in particular, the magnitude of linear layers.
2. Section 5.3.2 shows that the normalized curvature of DNNs can be upper bounded via a decomposition into normalized curvatures and slopes of individual layers.
3. Developing an architecture for training low curvature neural networks combining a novel activation function (Section 5.3.3) with recent innovations to constrain



(a) Decision boundary with standard NN (β = 1000)



(b) Decision boundary with LCNN ($\beta = 1$)



(c) Softplus vs Centered-Softplus

Figure 5.2: Decision boundaries of (a) standard NN and (b) Low Curvature NN trained on the two moons dataset. The low curvature NN recovers a highly regular decision boundary. (c) The softplus and centered-softplus non-linearities (defined in Section 5.3.3) behave similarly for large β, and converge to linear maps for small β. However, softplus diverges while centered-softplus stays close to the origin.

the Lipschitz constant of convolutional (Section 5.3.4) and batch normalization (Section 5.3.4) layers.

4. Section 5.4 proves that controlling normalized curvature controls relative gradient robustness and adversarial robustness.

5. Section 5.5 demonstrates that these innovations are successful in training low-curvature models without sacrificing training accuracy, and that such models have robust gradients and are more adversarially robust.

5.2 Related Work

5.2.1 Parameter Curvature and the Loss Landscape

A large literature examines the sensitivity of the loss with respect to the parameters in order to improve the training dynamics and thereby find more stable optima. High loss sensitivity to parameter choice around a fitted value indicates (by construction) either a high slope or curvature. However, more linear loss landscapes are easier to optimize so stability and extreme suboptimality are more expected in the presence of significant parameter nonlinearity. In this section, let "parameter Hessian" refer to the matrix of second derivatives of a DNN loss with respect to the parameters. Keskar et al. [88] coin

the term "sharpness" as an approximate spectral norm of the parameter Hessian, scaled by one plus the function value – observing that larger batch sizes tend to lead to sharper optima. Dinh et al. [42] builds on this work, showing that a simple reparamaterization can change the sharpness of a network arbitrarily without affecting its prediction and showing that methodological decisions in modelling nonlinearity can be pivotal. Ghorbani, Krishnan, and Xiao [62] give a method to efficiently and accurately estimate the entire spectrum of the parameter Hessian, and isolate difficulties in training to outliers in parameter Hessian eigenvalues. Foret et al. [51] find that penalizing a parameter Hessian inner product has significant empirical and theoretical advantages. Indeed, Andriushchenko and Flammarion [4] prove that models trained via sharpness-regularized optimization provably generalize better than standard SGD.

Section 5.E examines further the relationship between the input-loss Hessian, considered here, and the parameter-loss Hessian that features in work on the geometry of the loss landscape. In particular, it shows a duality between the Hessians with respect to the input and the first layer weights.

5.2.2 Adversarial Robustness of Neural Networks

Adding imperceptible noise can cause deep neural networks to misclassify points with high confidence (Szegedy et al. [152] and Goodfellow, Shlens, and Szegedy [65]). The canonical method to defend against this vulnerability is adversarial training (Madry et al. [105]) which trains models to accurately classify adversarial examples generated via an attack such as projected gradient descent (PGD). However, this approach is computationally expensive and provides no formal guarantees on robustness. Cohen, Rosenfeld, and Kolter [31] proposed randomized smoothing, which provides a formal guarantee on robustness by generating a smooth classifer from any black-box classifier. Hein and Andriushchenko [75] identified the Lipschitz constant as critical quantity to prove formal robustness guarantees. Moosavi-Dezfooli et al. [115] penalize the Frobenius norm of the Hessian, and show that they performs similarly to models trained via adversarial training. Qin et al. [126] introduce a local linearity regularizer, which also implicitly penalizes the Hessian.

5.2.3 Unreliable Gradient Interpretations in Neural Networks

Gradient explanations in neural networks can be unreliable. Ghorbani, Abid, and Zou [61] and Zhang et al. [177] showed that for any input, there are nearby inputs with highly dissimilar gradient explanations. Srinivas and Fleuret [148] showed that pre-

softmax logit gradients are independent of model behaviour, and as a result we focus on post-softmax loss gradients in this work. Ros and Doshi-Velez [131] showed empirically that robustness can be improved by gradient regularization, however Dombrowski et al. [44] showed that gradient instability is primarily due to large Hessian norms. This suggests that the gradient penalization in Ros and Doshi-Velez [131] performed unintentional Hessian regularization, which is consistent with our experimental results. To alleviate this, Dombrowski et al. [45] proposed to train low curvature models via softplus activations and weight decay, similar to our approach.

5.2.4 Lipschitz Layers in Neural Networks

There has been extensive work on methods to bound the Lipschitz constant of DNNs. Cisse et al. [30] introduced Parseval networks, which penalizes the deviation of linear layers from orthonormality – since an orthonormal linear operator evidently has a Lipschitz constant of one, this shrinks the Lipschitz constant of a layer towards one. Trockman and Kolter [159] use a reparameterization of the weight matrix, called the Cayley Transform, that is orthogonal by construction. Miyato et al. [112] and Ryu et al. [134] proposed spectral normalization, where linear layers are divided by their spectral norm, ensuring that the overall spectral norm of the layer is one.

Other works, such as Loukas, Poiitis, and Jegelka [103], have directly posited the Lipschitz constants at the data as a useful diagnostic on training, with lower empirical Lipschitz parameter trajectories resulting in less complex models.

5.3 Training Low-Curvature Neural Networks

This section introduces an approach to train low curvature neural networks (LCNNs).

5.3.1 Measuring Model Non-Linearity via Normalized Curvature

A linear function should have zero curvature, and the higher the curvature, the further from linear the function is. Moosavi-Dezfooli et al. [115] and Dombrowski et al. [45] proposed measuring the curvature of a DNN via the Frobenius norm of the Hessian. This approach is sensitive to gradient scaling, which is undesirable: if two functions are scaled versions of each other, then they should have similar curvatures. It is easy to see that Hessian norms do not have this property, as scaling the function also scales

the Hessian. Low curvature models with steep gradients may be overly sensitive, but are close to linear, however Hessian norms alone cannot adequately distinguish such models from those with high-curvature models and small gradients. A definition of curvature that is approximately *normalized* can disentangle the two. For a function f let

$$C_f(x) = \|\nabla^2 f(x)\|_2 / (\|\nabla f(x)\|_2 + \varepsilon) \tag{5.1}$$

be called the curvature of f at x. $\|\nabla f(x)\|_2$ and $\|\nabla^2 f(x)\|_2$ are the L_2 norm of the gradient and the Hessian, respectively, and $\varepsilon > 0$ is a small constant to ensure well-behavedness of the measure when the denominator approaches zero. This measure joins other approaches normalizing the second derviative in order to achieve desired scaling properties, such as from geometry, where a term of $(1 + f'(x))^{3/2}$ normalizes by the arclength of a curve in one dimension.

Equation (5.1) measures Hessian norm *relative* to the gradient norm, and captures relative local linearity, which can be seen via Taylor's theorem:

$$\underbrace{\frac{\|f(x+\delta) - f(x) - \nabla f(x)^\top \delta\|_2}{\|\nabla f(x)\|_2}}_{\text{relative local linearity}} \leq \frac{1}{2} \underbrace{\max_x C_f(x)}_{\text{max normalized curvature}} \times \|\delta\|_2. \tag{5.2}$$

5.3.2 A Data-Free Upper Bound on Curvature

Directly penalizing the loss curvature requires backpropagating an estimate of the Hessian norm, which itself requires backpropagating gradient-vector products. This requires chaining the entire computational graph of the model at least three times. Moosavi-Dezfooli et al. [115] reduce the complexity of this operation by computing a finite-difference approximation to the Hessian from gradients, but even this double-backpropagation is expensive. Ideally, a penalization procedure would take a single backpropagation step. A measure that is a function of model parameters alone (and not training data) satisfies this property.

To illustrate the idea, Lemma 6 shows this upper bound for the simplified case of the composition of one-dimensional functions.

Lemma 6. *For a function $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ with $f_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, 2, \ldots, L$, the normalized curvature $C_f = |f''/f'|$, is bounded by $\left|\frac{f''}{f'}\right| \leq \sum_{i=1}^{L} \left|\frac{f_i''}{f_i'}\right| \prod_{j=1}^{i} |f_j'|$ where f', f'' are the first*

and second order derivatives.

Proof. The chain rule for the first derivative gives $f' = \prod_{i=1}^{L} f_i'$. Differentiating this expression gives $f'' = \sum_{i=1}^{L} f_i'' \prod_{j=1}^{i} f_j' \prod_{k=1,k\neq i}^{L} f_k'$. Dividing by f', taking absolute value of both sides, and using the triangle inequality, delivers the intended result. □

Section 5.A derives the expression for general layer widths using a result from Wang et al. [167] that connects the spectral norms of order-n tensors to the spectral norm of their matrix "unfoldings". The simplified result is Theorem 8.

Theorem 8. *Given a function $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ with $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, the curvature C_f can be bounded as a function of the curvatures of individual layers $C_{f_i}(x)$, i.e.,*

$$C_f(x) \leq \sum_{i=1}^{L} n_i \times C_{f_i}(x) \times \prod_{j=1}^{i} \|\nabla f_j(F_{j-1}(x))\|_2 \leq \sum_{i=1}^{L} n_i \times \max_{x'} C_{f_i}(x') \times \prod_{j=1}^{i} \max_{x'} \|\nabla f_j(F_{j-1}(x'))\|_2, \tag{5.3}$$

where $F_{j-1}(x) = (f_{j-1} \circ f_{j-2} \circ \ldots \circ f_1)(x)$.

The rightmost term is independent of x and thus holds uniformly across all data points. This bound shows that controlling the curvature and Lipschitz constant of each layer of a neural network controls the overall curvature of the model. A similar bound is constructed recursively in Singla and Feizi [145]. Dombrowski et al. [45] gives a similar formula, albeit for the Frobenius norm, which is simpler, because the sum of squared entries is independent of the layout of the data. To our knowledge Equation (5.3) is the first explicit, easily-interpreted formula of its type.

Neural networks typically consist of linear maps such as convolutions, fully connected layers, and batch normalization layers, along with non-linear activation functions. Linear maps have zero curvature by definition, and non-linear layers often have bounded gradients (= 1), which simplifies computations. The next section analyzes the penalization of the remaining terms, i.e., the curvature of the non-linear activations, and the Lipschitz constant of the linear layers.

5.3.3 Centered-Softplus: An Activation with Trainable Curvature

Theorem 8 shows that the curvature of a neural network depends on the curvature of its constituent activation functions. We thus propose to use activation functions with minimal curvature.

A smooth activation such as the softplus function, $s(x;\beta) = \log(1+\exp(\beta x))/\beta$ is well suited to analyzing questions of curvature. Although not common, softplus is used, especially where its smoothness facilitates analysis, such as in Grathwohl et al. [67]. The curvature of the softplus function is

$$C_{s(\cdot;\beta)}(x) = \beta \times \left(1 - \frac{\mathrm{d}s(x;\beta)}{\mathrm{d}x}\right) \leq \beta. \tag{5.4}$$

Thus using softplus with small β values ensures low curvature. However, there are two drawbacks of softplus with small β: (1) $\lim_{\beta \downarrow 0} s(x;\beta) = \infty$, meaning that well-behaved low curvature maps cannot be recovered and, (2) instability around the origin upon composition, $s^n(0;\beta) = \underbrace{(s \circ s \circ \ldots \circ s)}_{n \text{ times}}(0;\beta) = \frac{\log(n+1)}{\beta}$. Thus $s^n(0;\beta) \to \infty$ as $n \to \infty$, which is problematic for deep networks. The *centered-softplus* $s_0(x;\beta)$ solves this by introducing a normalizing term as follows:

$$s_0(x;\beta) = s(x;\beta) - \frac{\log 2}{\beta} = \frac{1}{\beta}\log\left(\frac{1+\exp(\beta x)}{2}\right). \tag{5.5}$$

$s(0;\beta) = s^n(0;\beta) = 0$ for any positive integer n, thus s_0 is stable under composition. More importantly, $\lim_{\beta \downarrow 0} s_0(x;\beta) = x/2$, while still retaining $\lim_{\beta \uparrow \infty} s_0(x;\beta) = \mathrm{ReLU}(x)$. This ensures that s_0 can learn both well-behaved linear maps, as well as highly non-linear ReLU-like maps if required.

β is a learnable parameter with its value penalized, thereby penalizing the curvature of that layer. Having accounted for the curvature of the non-linearities, the next section discusses controlling the gradients of the linear layers.

5.3.4 Lipschitz Linear Layers

Theorem 8 shows that penalizing the Lipschitz constants of the constituent linear layers in a model penalizes overall model curvature. There are three classes of linear layers we consider: convolutions, fully connected layers, and batch normalization.

Spectrally Normalized Convolutions and Fully Connected Layers

Existing spectral normalization techniques suffice to penalize the Lipschitz constant of convolutions and fully connected layers. For fully connected layers, ordinary spectral normalization from Miyato et al. [112] which controls the spectral norm of a fully connected layer by reparameterization (replacing a weight matrix W, by $W/||W||_2$ and using a small number of power iterations to approximate $||W||_2$) works. Ryu et al. [134] generalize this normalization to convolutions with a power iteration method that works directly on the linear mapping implicit in a 2D convolution: maintaining 3D left and right singular "vectors" of the 4D tensor of convolutional filters and developing the corresponding update. Ryu et al. [134] call this "real" spectral normalization to distinguish it from approximations based on flattening higher order tensors and applying Miyato et al. [112]. Spectral normalization on fully connected layers and "real" spectral normalization on convolutional layers ensures that the spectral norm of these layers is exactly equal to one, simplifying the bound in Theorem 8.

γ-Lipschitz Batch Normalization

Ignoring the learnable parameters of batch normalization, at inference time batch normalization is multiplication by the diagonal matrix of inverse running standard deviation estimates. Thus clipping the reciprocal of the smallest running standard deviation value across dimensions at one normalizes the spectral norm. Experimentally, models with spectrally normalized batch norm layers failed to train well, indicating that the scaling is necessary for training (Ghorbani, Krishnan, and Xiao [62] offer a compelling explanation of this phenomena, since eliminating the scaling aspect of batch normalization reduces it to de-meaning – essentially removing it). To remedy this, let "BN" denote the normal batchnorm operation,

$$\begin{aligned}
&1\text{-Lipschitz-BN}(x) \triangleq \text{BN}(x)/||\text{BN}||_2 \\
&\gamma\text{-Lipschitz-BN}(x) \triangleq \underbrace{\min(\gamma, ||\text{BN}||_2)}_{\text{scaling factor} \leq \gamma} \times 1\text{-Lipschitz-BN}(x).
\end{aligned}$$

Clipping the scaling above at γ (equivalently, the running standard deviation below, at $1/\gamma$) ensures that the Lipschitz constant of a batch normalization layer is at most equal to γ. As with β, described in Section 5.3.3, we cast γ as a learnable parameter in order to penalize it during training. Gouk et al. [66] proposed a similar approach, though

restricted to a common γ for all batch norm layers, while here γ can vary by layer.

5.3.5 Penalizing Curvature

Section 5.3.4 discussed three architectural innovations – centered-softplus activations with a trainable β, spectrally normalized linear and convolution layers and a γ-Lipschitz batch normalization layer. This section discusses methods to penalize the overall curvature of a model built with these layers.

Spectrally-normalized convolutional and linear layers have spectral norms equal to one by construction, thus only batch normalization and activation layers contribute to curvature. We refer to these layers as γBN and βSP. For models with batch normalization, naïvely using the upper bound in Theorem 8 is problematic due to the exponential growth in the product of Lipschitz constants of batch normalization layers. We thus use a simpler penalization where $\log \gamma_i$ is aggregated additively across batch normalization layers, and independent of β_i in the following manner:

$$\lambda_\beta \times \sum_{i \in \beta\text{SP}} \beta_i + \lambda_\gamma \times \sum_{j \in \gamma\text{BN}} \log \gamma_j. \tag{5.6}$$

Additive aggregation ensures that the penalization does not grow exponentially. Note that the model is necessarily linear if the penalization term is zero, thus making it an appropriate measure of model non-linearity. γ and β are parameterized in terms of their log value, to ensure they remain positive during training.

The term "LCNN" refers to a model trained with the proposed architectural components (centered-softplus, spectral normalization, and γ-Lipschitz batch normalization) and associated regularization terms on β, γ. The next section discusses the robustness and interpretability benefits of LCNNs.

5.4 Why Train Low-Curvature Models?

This section discusses the advantages of low-curvature models, particularly as it pertains to robustness and gradient stability. These statements apply not just to LCNNs, but low-curvature models in general.

5.4.1 Low Curvature Models have Stable Gradients

Recent work such as Ghorbani, Abid, and Zou [61] and Zhang et al. [177] has shown that gradient explanations are manipulable, and that inputs whose explanations differ maximally from those at the original inputs are easily found, making them unreliable in practice for identifying important features. This amounts to models having a large curvature. In particular, the relative gradient distance is upper bounded by the normalized curvature C_f, as given below.

Theorem 9. *Consider a model f with $\max_x C_f(x) \leq \nu$ and two inputs x and $x+\delta$. The relative distance between gradients at these points is bounded by*

$$\frac{\|\nabla f(x+\delta) - \nabla f(x)\|_2}{\|\nabla f(x)\|_2} \leq \|\delta\|_2 \nu \times \exp(\|\delta\|_2 \nu).$$

If f is quadratic, then we obtain the tighter bound $\|\delta\|_2 C_f(x)$.

The proof (Section 5.B.1) expands $f(x)$ in a Taylor expansion around $x+\delta$ and bounds the magnitude of the second and higher order terms over the neighborhood of x. Thus the smaller the model curvature, the more locally stable are the gradients.

5.4.2 Low Curvature is Necessary for L_2 Robustness

A small gradient norm is known to be an important aspect of adversarial robustness (see Hein and Andriushchenko [75]). However, small gradients alone are not sufficient, and low curvature is also necessary to achieve robustness. This is easy to see intuitively - a model may have low gradients at a point leading to robustness for small noise values, but if the curvature is large, then gradient norms at neighboring points can quickly increase, leading to misclassification for even slightly larger noise levels.

Theorem 10 formalizes this intuition establishing an upper bound on the distance between two nearby points, which depends on both the gradient norm (as was known previously) and well as the max curvature of the underlying model.

Theorem 10. *Consider a model f with $\max_x C_f(x) \leq \nu$, then for two inputs x and $x+\delta$, we have the following expression for robustness*

$$\|f(x+\delta) - f(x)\|_2 \leq \|\delta\|_2 \|\nabla f(x)\|_2 \left(1 + \|\delta\|_2 \nu \times \exp(\|\delta\|_2 \nu)/2\right).$$

If f is quadratic, then we obtain the tigher bound $\|\delta\|_2 \|\nabla f(x)\|_2 \left(1 + \|\delta\|_2 C_f(x)/2\right)$.

The proof (Section 5.B.2) uses similar techniques to the proof of Theorem 9. This result shows that between two models with equal gradients at data points, greater robustness will be achieved by the model with the smaller curvature.

5.5 Experiments

This section presents experiments to (1) evaluate the effectiveness of the proposed method to achieve low curvature as intended, (2) evaluate whether low curvature models have robust gradients, and (3) evaluate whether low-curvature models are adversarially robust. These experiments are primarily conducted on a base ResNet-18 architecture from He et al. [73] using the CIFAR10 and CIFAR100 datasets (cf. Krizhevsky [97]), and using the PyTorch (Paszke et al. [121]) framework.

5.5.1 Loss curvature and logit curvature

Section 5.4.1 and Section 5.4.2 interpreted f as the logits of a deep neural network. This is advantageous in permitting conclusions about adversarial robustness (Theorem 10) and notationally convenient in relying on only the input x and not the y.

For consistency with earlier papers, such as Moosavi-Dezfooli et al. [115], here we consider the *loss curvature*, where the loss function is cross-entropy loss. The two are connected via $\text{loss}(x, y) = -\text{logsoftmax}(f(x))_y$. This is the notion tabulated below, though the penalty is Equation (5.6). The empirical analogues tabulated here are computed exactly using PyTorch's native autograd functionality.

Recognizing that $\text{loss}(x, y) \leq ||\text{logsoftmax}(f(x))||$ we have this informative relationship between the loss and logit derivatives

$$\nabla(\text{logsoftmax} \circ f)(x) = \nabla \text{logsoftmax}(f(x)) \nabla f(x) \text{ and} \tag{5.7}$$

$$\nabla^2(\text{logsoftmax} \circ f)(x) = \nabla f(x) \nabla^2 \text{logsoftmax}(f(x)) \nabla f(x)^\top + \nabla \text{logsoftmax}(f(x)) \nabla^2 f(x). \tag{5.8}$$

Minimizing logit curvature also tends to decrease the loss curvature via the second term. Moreover, reducing the loss gradient norm tends to reduce logit gradient norm which tends to separately decrease the loss curvature. Though loss depends on x and y, in the empirical results, all derivatives are with respect to x only.

5.5.2 Baselines and Our Methods

The main baseline is a ResNet-18 model with softplus activation with $\beta = 10^3$ to mimic ReLU, and yet have well-defined curvatures. Another baseline is gradient norm regularization (henceforth called 'GradReg', Drucker and Le Cun [47]), trained with an additional penalty on the gradient norm. We train two variants of our approach - a base LCNN, which penalizes the curvature, and another variant combining LCNN penalty and gradient norm regularization (LCNN + GradReg), which controls both the curvature and gradient norm. Our theory indicates that the LCNN + GradReg variant is likely to produce more robust models, which is verified experimentally. We also compare with Curvature Regularization (CURE) from Moosavi-Dezfooli et al. [115], softplus with weight decay from Dombrowski et al. [45] and adversarial training with L_2 PGD from Madry et al. [105] with noise magnitude of 0.1 and 3 iterations of PGD. Further experimental details are in the appendix.

5.5.3 Parameter Settings

All models are trained for 200 epochs with an SGD + momentum optimizer, with a momentum of 0.9 and an initial learning rate of 0.1 which decays by a factor of 10 at 150 and 175 epochs, and a weight decay of 5×10^{-4}.

5.5.4 Evaluating the Efficacy of Curvature Penalization

This section evaluates whether LCNNs reduce model curvature in practice. Table 5.1 contains the results, from which we observe: (1) most baselines except CURE and

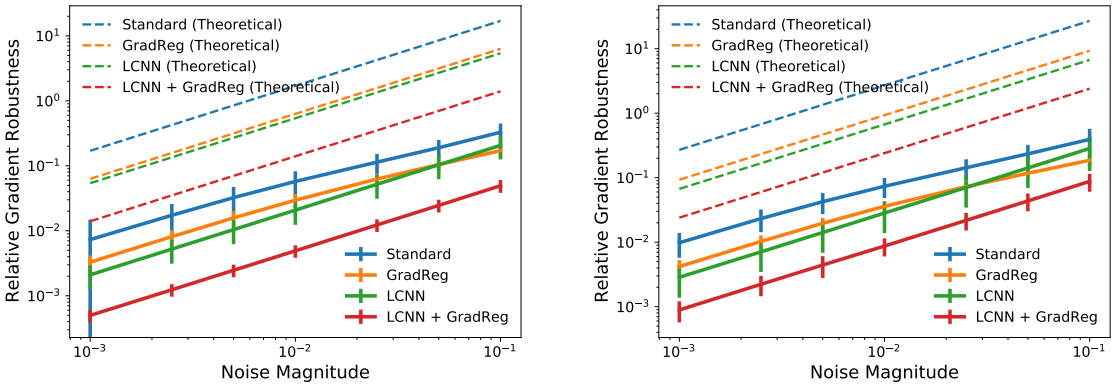
adversarial training do not meaningfully lose predictive performance (2) GradReg and adversarially trained models are best at reducing gradient norm while LCNN-based models are best at penalizing curvature. Overall, these experimental results show that LCNN-based models indeed minimize curvature as intended.

Table 5.1 also shows that GradReg has a regularizing effect on the Hessian and curvature above and beyond penalizing the curvature. This can be seen from Equation (5.8), as setting aside the logsoftmax terms $\nabla \text{loss} \propto \nabla f$, and $\nabla^2 \text{loss} = c_1(\nabla f)(\nabla f)^\top + c_2 \nabla^2 f$. Thus, penalizing both gradient norm and curvature penalizes both curvature directly, and a component of curvature arising from the gradient norm. Lower gradient norms generically benefit model stability, for example adversarial robustness. And penalizing curvature alone does not directly reduce the gradient norm (unlike existing methods), thus penalizing both gradient norm and gradient norm-scaled curvature is anticipated to have the best overall properties and the most direct competitor to existing approaches.

The average per-epoch training times on a GTX 1080Ti are: standard models and softplus + weight decay (100 seconds), LCNN (160 seconds), GradReg (270 seconds), LCNN + GradReg (350 seconds), CURE and Adversarial Training (500 seconds). Note that the increase in computation for LCNN is primarily due to the use of spectral normalization layers. The results show that LCNNs are able to penalize curvature while only marginally (1.6 ×) increasing training time, and using LCNN+GradReg only increases time 1.3 × over GradReg while reducing curvature.

Table 5.1: Model geometry of ResNet-18 models trained with various regularizers on the CIFAR100 test dataset. Gradient norm regularized models ('GradReg') are best at reducing gradient norms, while LCNN-based models are best at reducing curvature, leaving gradients unpenalized. We obtain the benefits of both by combining these penalties. All models use softplus (or centered softplus). Results are averaged across two runs. All statistics are averaged over the test data.

Model	$\|\nabla \text{loss}\|_2$	$\|\nabla^2 \text{loss}\|_2$	C_{loss}	Acc. (%)
Standard	19.66 ±0.33	6061.96 ±968.05	270.89 ±75.04	77.42 ±0.11
LCNN	22.04 ±1.41	1143.62 ±99.38	69.50 ±2.41	77.30 ±0.11
GradReg	**8.86** ±0.12	776.56 ±63.62	89.47 ±5.86	77.20 ±0.26
LCNN + GradReg	9.87 ±0.27	**154.36** ±0.22	**25.30** ±0.09	77.29 ±0.07
CURE	**8.86** ±0.01	979.45 ±14.05	116.31 ±4.58	76.48 ±0.07
Weight Decay	18.08 ±0.05	1052.84 ±7.27	70.39 ±0.88	77.44 ±0.28
Adversarial Training	**7.99** ±0.03	501.43 ±18.64	63.79 ±1.65	76.96 ±0.26



(a) Gradient Robustness on CIFAR10

(b) Gradient Robustness on CIFAR100

Figure 5.3: Relative gradient robustness $\frac{\|\nabla f(x+\delta)-\nabla f(x)\|_2}{\|\nabla f(x)\|_2}$ as a function of noise magnitude $\|\delta\|_2$ on (a) CIFAR10 and (b) CIFAR100 with a ResNet-18 model. Low-curvature models show an order of magnitude improvement in gradient robustness, which closely follows the trend predicted by the theoretical upper bound in Section 5.4.

5.5.5 Impact of Curvature on Gradient Robustness

Section 5.4.1 showed that low-curvature models tend to have more robust gradients. This section measures the relative gradient robustness for the models with various ranges of curvature and noise levels. Figure 5.3 plots results of measuring robustness to random noise at fixed magnitudes ranging logarithmically from 1×10^{-3} to 1×10^{-1}. Theresults match the quadratic approximation in Section 5.4.1 quite closely in terms of the overall trends, and low curvature models have an *order of magnitude* improvement in robustness over standard models.

5.5.6 Impact of Curvature on Adversarial Robustness

Theorem 10 shows that having low curvature is necessary for robustness, along with having small gradient norms. This section evaluates this claim empirically, by evaluating adversarial examples via L_2 PGD adversaries with various noise magnitudes. PGD is implemented via the Cleverhans library (Papernot et al. [120]). The results in Table 5.2 show that LCNN+GradReg models perform on par with adversarial training, with no accuracy loss.

Table 5.2: Results indicating off-the-shelf model accuracies (%) against L_2 PGD adversarial examples across various noise magnitudes. Adversarial training performs the best overall, however sacrifices clean accuracy. LCNN+GradReg models perform similarly but without significant loss of clean accuracy. All models use softplus (or centered softplus). Results are averaged across two runs.

Model	Acc. (%)	$\|\delta\|_2 = 0.05$	$\|\delta\|_2 = 0.1$	$\|\delta\|_2 = 0.15$	$\|\delta\|_2 = 0.2$
Standard	77.42 ±.10	59.97 ±.11	37.55 ±.13	23.41 ±.08	16.11 ±.21
LCNN	77.16 ±.07	61.17 ±.53	39.72 ±.17	25.60 ±.32	17.66 ±.18
GradReg	77.20 ±.26	71.90 ±.11	61.06 ±.03	49.19 ±.12	38.09 ±.47
LCNN + GradReg	77.29 ±.26	**72.68** ±.52	**63.36** ±.39	**52.96** ±.76	**42.70** ±.77
CURE	76.48 ±.07	71.39 ±.12	61.28 ±.32	49.60 ±.09	39.04 ±.16
Weight Decay	77.44 ±.28	60.86 ±.36	38.04 ±.43	23.85 ±.33	16.20 ±.01
Adversarial Training	76.96 ±.26	**72.76** ±.15	**64.70** ±.20	**54.80** ±.25	**44.98** ±.57

5.5.7 Train-Test Discrepancy in Model Geometry

Table 5.3 shows that the empirical gradient norm and Hessian norms are much larger on train than on test data, hinting at a form of overfitting with regards to these quantities. We term this phenomenon the *train-test discrepancy* in model geometry. Interestingly, no such discrepancy appears for our curvature measure, indicating that it may be a more reliable measure of model geometry. We leave further investigation of this phenomenon as a topic for future work.

Table 5.3: Train-test discrepancy in model geometry. For a scalar-valued function g of a dataset x_{train} or x_{test}, let ttd-g denote $|1 - g(x_{\text{train}})/g(x_{\text{test}})|$. We present ttd-$g$ for g being the gradient norm, Hessian norm, and curvature. There is a large train-test discrepancy for gradient and Hessian norm, but almost none for curvature, indicating that it may be a stable model property.

Model	ttd-$\|\nabla \text{loss}\|_2$	ttd-$\|\nabla^2 \text{loss}\|_2$	ttd-C_{loss}
Standard	11.75	12.28	**0.025**
GradReg	11.33	11.22	**0.017**
LCNN	19.99	11.33	**0.129**
LCNNs + GradReg	21.82	10.43	**0.146**

Summary of Experimental Results Overall, our experiments show that:

(1) LCNNs have lower curvature than standard models, and combining them with

gradient norm regularization further decreases curvature (see Table 5.1). The latter phenomenon is unexpected, as our curvature measure ignores gradient scaling.

(2) LCNNs combined with gradient norm regularization achieve an order of magnitude improved gradient robustness over standard models (see Figure 5.3).

(3) LCNNs combined with gradient norm regularization outperform adversarial training with (1) better predictive accuracy at a lower curvature (see Table 5.1), and (2) 1.4 × faster training, with comparable adversarial robustness (see Table 5.2).

(4) We observe that there exists a train-test discrepancy for standard geometric quantities like the gradient and Hessian norm, and this discrepancy disappears for our proposed curvature measure (see Table 5.3).

Section 5.D presents ablation experiments, additional adversarial attacks, and evaluations on more datasets and architectures.

5.6 Conclusion

This chapter presented a modular approach to remove excess curvature in neural network models. Importantly, we found that combining LCNNs with gradient norm regularization resulted in models with the smallest curvature, the most stable gradients as well as those that are the most adversarially robust.

A limitation of our approach is that we only consider convolutional and fully connected layers, and not self-attention or recurrent layers. We also do not investigate the learning-theoretic benefits (or harms) of low-curvature models, or characterize how they may affect generalization for small number of training samples, or robustness to label noise. Investigating these topics would be important future work.

5.A Proof of Theorem 8

This section proves Theorem 8, which decomposes overall curvature into curvatures and slopes of constituent layers. We restate it here for reference. Section 5.A.1, Section 5.A.2, and Section 5.A.3 given necessary preliminaries, and the actual bound is in Section 5.A.4.

5.A.1 Derivatives of Compositional Functions

For a function $f : \mathbb{R}^d \to \mathbb{R}^r$, let $\nabla f : \mathbb{R}^d \to \mathbb{R}^{d\times r}$ denote its gradient, and $\nabla^2 f : \mathbb{R}^d \to \mathbb{R}^{d\times r\times d}$ denote its Hessian. Given functions $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, i = 1\ldots, L$ let $f_{k,k+j} = f_{k+j} \circ f_{k+j-1} \circ \ldots \circ f_k : \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_{k+j}}$ for $1 \le k \le k+j \le L$. If each f_i is continuously differentiable, then

$$\nabla f_{k,k+j} = \prod_{i=1}^{j} \nabla f_{k+j-i+1} \in \mathbb{R}^{n_{k+j}\times n_k}. \tag{5.9}$$

with the convention that $f_{j,j}(x) = x$. The product begins at the end, with ∇f_{k+j}, and progresses forward through the indices via the chain rule of differentiation. Supposing that each f_i is twice-differentiable, the second derivative of $f_{1,k}$ is given by:

$$\begin{aligned} \nabla^2 f_{1,k} &= \sum_{i=1}^{k} \nabla^2 f_{1,i} - \nabla^2 f_{1,i-1} \\ \text{where } (\nabla^2 f_{1,i} - \nabla^2 f_{1,i-1}) &= (\nabla^2 f_i)\left(\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \nabla f_{1,i-1}\right) \in \mathbb{R}^{n_0\times n_k\times n_0}. \end{aligned} \tag{5.10}$$

This equation uses the *covariant multilinear matrix multiplication* notation: $\nabla^2 f_i$ is an order-three tensor $\in \mathbb{R}^{n_i\times n_i\times n_i}$, with the first and third modes multiplied by $\nabla f_{1,i-1} \in \mathbb{R}^{n_i\times n_0}$ and the second mode multiplied by $\nabla f_{i+1,k}^\top \in \mathbb{R}^{n_i\times n_k}$.

5.A.2 Tensor Calculus

This section presents a simplifed version of the notation from Wang et al. [167]. A k-linear map is a function of k variables such that if any $k-1$ variables are held constant, the map is linear in the remaining variable. A k-linear function can be represented by an order-k tensor $\mathscr{A}$ given elementwise by $\mathscr{A} = [\![a_{j_1\ldots j_k}]\!] \in \mathbb{R}^{d_1\times\ldots\times d_k}$.

The covariant multilinear matrix multiplication of a tensor with matrices $M_1 = (m^{(1)}_{i_1 j_1}) \in \mathbb{R}^{d_1\times s_1}, \ldots, M_k = (m^{(k)}_{i_k j_k}) \in \mathbb{R}^{d_k\times s_k}$ is

$$\mathscr{A}(M_1,\ldots,M_k) = \left[\!\!\left[\sum_{i_1=1}^{d_1} \ldots \sum_{i_k}^{d_k} a_{i_1\ldots i_k} m^{(1)}_{i_1 j_1} \ldots m^{(k)}_{i_k j_k} \right]\!\!\right] \in \mathbb{R}^{s_1\times\ldots\times s_k}.$$

For example, covariant multilinear matrix multiplication of an order two tensor is pre-

and post- multiplication by its arguments: $M_1^\top \mathscr{A} M_2 = \mathscr{A}(M_1, M_2)$. This operation can be implemented via iterated einsums as:

```
def covariant_multilinear_matmul(a: torch.Tensor,
                                 mlist: List[torch.Tensor]) -> torch.Tensor:
    order = a.ndim
    base_indices = string.ascii_letters
    indices = base_indices[:order]
    next_index = base_indices[order]
    val = a
    for idx in range(order):
        resp_str = indices[:idx] + next_index + indices[idx+1:]
        einsum_str = indices + f",{indices[idx]}{next_index}->{resp_str}"
        val = torch.einsum(einsum_str, val, mlist[idx])
    return val
```

The spectral norm of a general tensor is $||\mathscr{A}||_2 = \sup\{\mathscr{A}(x_1, \ldots, x_k) : ||x_i|| = 1, x_i \in \mathbb{R}^{d_i}, i = 1, 2, \ldots, k\}$. The computation of order-k operator norms is hard in theory, and also in practice (cf. Friedland and Lim [53]). Following the literature, we thus upper-bound the operator norm by the norm of one unfolding. For $\mathscr{A} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$, let $\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{A}) \in \mathbb{R}^{d_1 d_2 \times d_3}$ be the matrix with the jth column being the flattened jth (in the final index) $d_2 \times d_3$ matrix.[1] Unfolding is useful because it bounds an order-3 operator norm in terms of order-2 operator norms: Wang et al. [167, Theorem 4.8] shows that $||\mathscr{A}|| \leq ||\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{A})||$. The upper bound – the operator norm of a *matrix* – can computed with standard largest singular-value routines. A similar bound was used in Singla and Feizi [144] to give improved estimates on the spectral norm of convolution operators.

To facilitate the analysis of unfolded tensors, let $\text{pdiag2} : \mathbb{R}^d \mapsto \mathbb{R}^{d \times d}$ and $\text{pdiag3} : \mathbb{R}^d \mapsto \mathbb{R}^{d \times d \times d}$ be operations that *put to* the diagonal of tensors:

$$\text{pdiag2}(x)_{ij} = \begin{cases} x_j & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}, \text{ and } \text{pdiag3}(x)_{ijk} = \begin{cases} x_k & \text{if } i = j = k \\ 0 & \text{otherwise.} \end{cases}$$

Further, let $1_n \in \mathbb{R}^n$ be a vector of ones, $I_n = \text{pdiag2}(1_n) \in \mathbb{R}^{n \times n}$ be the n-dimensional identity matrix, and $\mathscr{I}_n = \text{pdiag3}(1_n) \in \mathbb{R}^{n \times n \times n}$. For two vectors $a \in \mathbb{R}^n, b \in \mathbb{R}^n$, let ab denote the elementwise product. $\otimes$ denotes the well-understood Kronecker product, so that, for example, $1_n^\top \otimes I_m$ is an $m \times nm$ matrix consisting of n copies of the $m \times m$ identity matrix stacked side by side.

[1] In PyTorch, $\text{unfold}_{\{\{1,2\},\{3\}\}}$(a) = `torch.flatten(a, end_dim=1)`.

These straightforwardly-verified facts are used subsequently:

1. $\mathscr{A} = \text{pdiag3}(ab) \implies \mathscr{A}(M_1, M_2, M_3) = \mathscr{I}(\text{pdiag2}(a)M_1, M_2, \text{pdiag2}(b)M_3)$,
2. $\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{I}(M_1, M_2, M_3)) = (M_1 \otimes I_{s_2})^\top \text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{I}(I_{d_1}, M_2, M_3))$, and
3. $||(1_{d_1}^\top \otimes I_{s_2})\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{I}(I_{d_1}, M_2, M_3))|| \leq ||M_2^\top M_3|| s_2$.

Taken together Fact #2 and #3 imply that

$$||\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{I}(M_1, M_2, M_3))|| \leq ||M_1|| \times ||M_2^\top M_3|| \times s_2. \tag{5.11}$$

5.A.3 Hessian Increment Bound

Let $\sigma(x) = \exp(x)/(1+\exp(x))$ denote the (elementwise) logistic function $\mathbb{R}^d \mapsto \mathbb{R}^d$. The derivatives of $s(x;\beta)$ can be written as

$$\nabla s(x;\beta) = \text{pdiag2}(\sigma(\beta x)) \in \mathbb{R}^{d \times d} \tag{5.12}$$
$$\nabla^2 s(x;\beta) = \text{pdiag3}(\beta\sigma(\beta x)(1-\sigma(\beta x))) \in \mathbb{R}^{d \times d \times d}. \tag{5.13}$$

Let the ith softplus layer have coefficient β_i, then the increment from Equation (5.10), can be bounded as follows:

$$||(\nabla^2 f_i)\left(\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \nabla f_{1,i-1}\right)|| \tag{5.14}$$
$$= ||\beta_i \mathscr{I}_{n_i}\left(\text{pdiag2}(1-\sigma(\beta_i x))\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \text{pdiag2}(\sigma(\beta_i x))\nabla f_{1,i-1}\right)|| \tag{5.15}$$
$$\leq ||\beta_i \text{unfold}_{\{\{1,2\},\{3\}\}}(\mathscr{I}_{n_i}(\text{pdiag2}(1-\sigma(\beta_i x))\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \text{pdiag2}(\sigma(\beta_i x))\nabla f_{1,i-1})|| \tag{5.16}$$
$$\leq ||\beta_i(\text{pdiag2}(1-\sigma(\beta_i x))\nabla f_{1,i-1}|| \times ||\nabla f_{i+1,k}^\top \text{pdiag2}(\sigma(\beta_i x))\nabla f_{1,i-1}|| \times n_i \tag{5.17}$$
$$\leq n_i \times ||\beta_i(\text{pdiag2}(1-\sigma(\beta_i x))|| \times ||\nabla f_{1,i-1}|| \times ||\nabla f_{i+1,k}^\top \nabla f_i \nabla f_{1,i-1}|| \tag{5.18}$$
$$= n_i \times C_{f_i} \times ||\nabla f_{1,i-1}|| \times ||\nabla f||. \tag{5.19}$$

Equation (5.15) follows by Fact #1 above, along with Equation (5.13). Equation (5.16) is the standard unfolding bound by Wang et al. [167]. Equation (5.17) is Equation (5.11). Equation (5.18) follows from the Cauchy-Schwartz inequality. The replacement in the last term of the product is Equation (5.12). Equation (5.19) rewrites Equation (5.18) using Equation (5.9): $f_{i+1,k}^{\top} \nabla f_i \nabla f_{1,i-1} = \nabla f_{1,k}$ and the definition of the curvature of f_i.

5.A.4 Putting it Together

Ignoring the ϵ term in the denominator,

$$C_f(x) = \frac{\|\nabla^2 f(x)\|}{\|\nabla f(x)\|} \tag{5.20}$$

$$= \frac{1}{\|\nabla f(x)\|} \left\| \sum_{i=1}^{L} \nabla^2 f_{1,i}(x) - \nabla^2 f_{1,i-1}(x) \right\| \tag{5.21}$$

$$\leq \frac{1}{\|\nabla f(x)\|} \sum_{i=1}^{L} ||\nabla^2 f_{1,i}(x) - \nabla^2 f_{1,i-1}(x)|| \tag{5.22}$$

$$\leq \sum_{i=1}^{L} n_i \times C_{f_i}(x) \times ||\nabla f_{1,i-1}(x)|| \tag{5.23}$$

$$\leq \sum_{i=1}^{L} n_i \times C_{f_i}(x) \times \prod_{i=1}^{j} ||\nabla f_i(F_{i-1}(x))|| \tag{5.24}$$

$$\leq \sum_{i=1}^{L} n_i \times \max_{x'} C_{f_i}(x') \times \prod_{i=1}^{j} \max_{x'} ||\nabla f_i(F_{i-1}(x'))||. \tag{5.25}$$

Equation (5.21) substitutes Equation (5.10). Equation (5.22) is the triangle inequality. Equation (5.23) is Equation (5.19), along with cancelling the term of $||\nabla f||$ top and bottom. Equation (5.24) are Equation (5.25) are obvious and a standard simplification in the literature on controlling the Lipschitz constant of neural networks. Because exactly computing the smallest Lipschitz constant of a general neural network is NP-complete, a widely-used baseline measure of Lipschitz-smoothness is rather the product of the Lipschitz constants of smaller components of the network, such as single layers (Scaman and Virmaux [139]). □

5.B Proofs of LCNN Properties

This section proves the properties linking LCNNs to gradient smoothness and adversarial robustness. Lemma 7, used in both these results, states that the ratio of gradient

norms at nearby points can rise at most exponentially with distance between those points in the worst case.

Lemma 7. *For a function f satisfying $\max_{x'} C_f(x') \leq v$, points x and $x+\delta$ that are $\|\delta\|_2 = r$ away,*

$$\frac{\|\nabla f(x+\delta)\|_2}{\|\nabla f(x)\|_2} \leq \exp(rv).$$

Proof. Let $g(x) = \log \|\nabla f(x)\|_2^2$. Applying a first order Taylor series expansion with Lagrange remainder, for some ξ:

$$\begin{aligned} g(x+\delta) - g(x) &= \nabla g(x+\xi)^\top \delta \implies \\ \log \frac{\|\nabla f(x+\delta)\|_2^2}{\|\nabla f(x)\|_2^2} &= 2\frac{\nabla f(x+\xi)\nabla^2 f(x+\xi)^\top \delta}{\|\nabla f(x+\xi)\|_2^2} \\ &\leq 2\frac{\|\nabla f(x+\xi)\|_2 \|\nabla^2 f(x+\xi)\|_2 \|\delta\|_2}{\|\nabla f(x+\xi)\|_2^2} \\ &= 2C_f(x)\|\delta\|_2 \\ &\leq 2rv. \end{aligned}$$

Dividing both sides by two and exponentiating both sides gives the intended result. □

5.B.1 Low Curvature $\implies$ Robust Gradients: Proof of Theorem 9

Proof. Use the Lagrange form of the Taylor error: there exists some ξ with $||\xi|| \leq ||\delta||$ such that

$$\begin{aligned} \nabla f(x+\delta) - \nabla f(x) &= \nabla^2 f(x+\xi)^\top \delta \implies \\ \|\nabla f(x+\delta) - \nabla f(x)\|_2 &\leq \|\nabla^2 f(x+\xi)\|_2 \|\delta\|_2 \implies \\ \frac{\|\nabla f(x+\delta) - \nabla f(x)\|_2}{\|\nabla f(x)\|_2} &\leq \frac{C_f(x+\xi)\|\nabla f(x+\xi)\|_2}{\|\nabla f(x)\|_2}||\delta|| \implies \\ \frac{\|\nabla f(x+\delta) - \nabla f(x)\|_2}{\|\nabla f(x)\|_2} &\leq C_f(x+\xi)\exp(v||\xi||)||\delta||. \end{aligned}$$

To derive the simplified quadratic approximation, plug in $\xi = 0$, otherwise this quantity is bounded above by $||\delta||v\exp(||\delta||v)$ by Lemma 7. □

5.B.2 Curvature is Necessary for Robustness: Proof of Theorem 10

Proof. Now take a second order Taylor expansion with remainder: there exists some ξ with $||\xi|| \leq ||\delta||$ such that

$$
\begin{aligned}
f(x+\delta) - f(x) = \nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x+\xi)\delta \implies \\
\|f(x+\delta) - f(x)\|_2 \leq \|\nabla f(x)^\top \delta\|_2 + \frac{1}{2}\|\delta^\top \nabla^2 f(x+\xi)\delta\|_2 \\
\leq \|\nabla f(x)\|_2 \|\delta\|_2 + \frac{1}{2}||\nabla^2 f(x+\xi)||_2 \times \|\delta\|_2^2 \\
= \|\nabla f(x)\|_2 \|\delta\|_2 \left(1 + \frac{1}{2}\frac{||\nabla^2 f(x+\xi)||_2}{||\nabla f(x)||_2}\|\delta\|_2\right) \\
= \|\nabla f(x)\|_2 \|\delta\|_2 \left(1 + C_f(x+\xi)\frac{||\nabla f(x+\xi)||_2}{||\nabla f(x)||_2}\|\delta\|_2/2\right).
\end{aligned}
$$

To derive the simplified quadratic approximation, plug in $\xi = 0$, otherwise this quantity is bounded above by $\|\delta\|_2\|\nabla f(x)\|_2\left(1 + \|\delta\|_2 v \times \exp(\|\delta\|_2 v)/2\right)$ by Lemma 7.

□

5.C Experimental Settings

This section elaborates the hyper-parameter settings used for our tuning our models. The standard ResNet-18 used standard hyper-parameter settings as indicated above, and we do not modify this for the other variants. For LCNNs, the regularizing constants are $\lambda_\beta = 10^{-4}$ and $\lambda_\gamma = 10^{-5}$. For GradReg, $\lambda_{\text{grad}} = 10^{-3}$, and for LCNNs + GradReg, $\lambda_\beta = 10^{-4}$, $\lambda_\gamma = 10^{-5}$, $\lambda_{\text{grad}} = 10^{-3}$. We performed a coarse grid search and chose the largest regularizing constants that did not affect predictive performance.

5.D Additional Experiments

5.D.1 Ablation Experiments

This section presents ablation studies that introduce the proposed modifications separately: models trained with only spectral norm for the convolution layers, γ-Lipschitz Batchnorm or centered softplus. Table 5.4 shows that for the ResNet-18 architecture,

(1) spectral normalization had no effect of curvature as presumably the batchnorm layers are able to compensate for the lost flexibility, and

(2) either penalizing the batchnorm alone or the softplus alone performs almost as well as LCNN, or sometimes even better in terms of curvature reduction. The most expensive computational step is the spectral norm of the convolutional layers, so removing this operation may yield speedups without substantially different results.

Table 5.4: Ablation experiments to study effect of individual modifications to LCNN architectures. Using only centered softplus or γ-BN suffices in practice to minimize curvature, while spectral norm on the convolutional layers may not be necessary.

Model	C_{loss}	$\|\nabla^2\text{loss}\|_2$	$\|\nabla\text{loss}\|_2$	Accuracy (%)
ConvSpectralNorm only	358.86	8380.55	23.92	77.55
γ-BN only	65.78	1086.95	20.86	77.33
c-Softplus only	57.49	734.05	16.99	77.31
Standard	270.89	6061.96	19.66	77.42
LCNN	69.40	1143.62	22.04	77.30

While it appears that for ResNet-18 γ-Lipschitz or centered softplus alone suffices for curvature reduction, it is necessary to regularize all components to avoid scale restrictions in one layer being undone in another, as dictated by the upper bound. In particular, penalizing only a subset of layers may not generalize to other architectures.

5.D.2 Robustness Evaluation on RobustBench / Autoattack

The attack presented in Section 5.5 was relatively "weak" – a network could be truly susceptible to stronger adversarial attacks. Short of a comprehensive verification (e.g. Katz et al. [87]), which is computationally intractable at scale, there is no fully

satisfactory way to guarantee robustness. However, one common method to develop confidence in a model is to demonstrate robustness in the face of a standard set of capable attackers. This is done in Table 5.5, which uses the Robustbench software (Croce et al. [36]) to evaluate both a white-box (having access to the internal details of the model), and a black-box (using only function evaluations) attacks.

Table 5.5: Adversarial accuracy to standard attacks accessed via Robustbench (Croce et al. [36]). "APGD-t" = the white box targetted auto PGD attack from Croce and Hein [35], "Square" = the black-box square attack from Andriushchenko et al. [5].

Model	APGD-t (%)	Square (%)	Clean (%)
Standard	22.122	52.874	76.721
LCNN	23.709	52.179	76.602
GradReg	50.294	64.678	76.394
LCNN+GradReg	52.477	64.678	76.622
CURE	50.096	63.488	75.928
Weight decay	23.907	53.766	76.622
Adv Training	55.155	66.861	75.521
CURE + GradReg	60.810	67.357	74.192
LCNN + GradReg + Adv Training	59.222	66.861	75.382

These experimental results show overall that:

(1) LCNN + GradReg is still on par with adversarial training even against stronger attacks such as APGD-t and Square, as they are with PGD.

(2) Combining LCNN + GradReg with adversarial training (in the last row) further improves robustness at the cost of predictive accuracy.

(3) Combining CURE with GradReg (in the penultimate row) improves robustness at the cost of further deteriorating predictive accuracy.

5.D.3 Additional Evaluations on other Architectures / Datasets

We present results on the following dataset - architecture pairs:

1. Table 5.6 presents results on SVHN dataset and VGG-11 model.
2. Table 5.7 presents results on SVHN dataset and ResNet-18 model.

3. Table 5.8 presents results on CIFAR-100 dataset and VGG-11 model.

The results for each similar to those for CIFAR-100 with ResNet-18.

Table 5.6: Model geometry of VGG-11 models trained on the SVHN test dataset.

Model	$\|\nabla \text{loss}\|_2$	$\|\nabla^2 \text{loss}\|_2$	C_{loss}	Accuracy (%)
Standard	2.87	158.29	54.24	96.01
LCNNs	4.04	83.34	30.05	95.61
GradReg	1.85	57.52	33.34	96.03
Adversarial Training	1.25	27.64	24.23	96.37

Table 5.7: Model geometry of ResNet-18 models trained on the SVHN test dataset.

Model	$\|\nabla \text{loss}\|_2$	$\|\nabla^2 \text{loss}\|_2$	C_{loss}	Accuracy (%)
Standard	2.64	204.38	78.22	96.41
LCNNs	2.91	77.78	25.36	96.35
GradReg	1.63	68.22	39.55	96.57
Adversarial Training	1.05	22.96	24.48	96.64

Table 5.8: Model geometry of VGG-11 models trained on the CIFAR-100 test dataset.

Model	$\|\nabla \text{loss}\|_2$	$\|\nabla^2 \text{loss}\|_2$	C_{loss}	Accuracy (%)
Standard	17.07	1482.16	85.81	73.33
LCNNs	15.88	282.06	41.14	73.76
GradReg	10.64	534.71	48.26	72.65
Adversarial Training	6.20	166.73	27.37	71.13

5.E Input-output Curvature and Parameter Curvature

This short section explores mathematically the relationship between input-output and parameter curvature, discussed in Section 5.2.1. For a scalar DNN, with one hidden unit at each layer, and no biases let $\ell_0, \ell_1, \ldots, \ell_k$ be linear layers that are of consistent dimension, and let $a_1, \ldots, a_k$ be elementwise nonlinearities. ℓ_i is parameterized by θ_i.

Without loss of generality, the network is composed of alternating linear and nonlinear layers: $f(x,\theta) = \ell_n(f_n(x,\theta))$ where $f_k = a_k \circ \ell_{k-1} \circ \ldots \circ a_1 \circ \ell_0$ for $k > 0$, where $\ell_j = x \mapsto \theta_j x$

(we adopt a "bilinear" notation, in which ℓ is a function of θ and x separately, thus $\nabla_x \ell_j = \theta_j$), a_j are the nonlinear activations, and $f_0(x,\theta) = x$.

Equation (5.9) and Equation (5.10) give expressions for $\nabla_x f$ and $\nabla_x^2 f$ (subscripts on the ∇ operation indicate with respect to which argument the derivative is being taken), $\nabla_x f = \theta_0 \prod_{k=1}^n \theta_k \times \prod_{k=1}^n \nabla a_k$, and

$$\nabla_x^2 f = \theta_0^2 \left(\prod_{k=1}^n \theta_k \right) \times \left(\prod_{k=1}^n \nabla a_k \right) \times \left(\sum_{k=1}^n \frac{\nabla^2 a_k}{\nabla a_k} \prod_{j=1}^{k-1} \theta_j \nabla a_j \right). \tag{5.26}$$

The derivatives with respect to the parameters are

$$\nabla_\theta f = \begin{pmatrix} \nabla_{\theta_0} f \\ \vdots \\ \nabla_{\theta_n} f \end{pmatrix} = \begin{pmatrix} \prod_{k=1}^n \theta_k \times \prod_{k=1}^n \nabla a_k \times f_0 \\ \prod_{k=2}^n \theta_k \times \prod_{k=2}^n \nabla a_k \times f_1 \\ \prod_{k=3}^n \theta_k \times \prod_{k=3}^n \nabla a_k \times f_2 \\ \vdots \\ \prod_{k=i}^n \theta_k \times \prod_{k=i}^n \nabla a_k \times f_{i-1} \\ \vdots \\ \theta_n \times \nabla a_n \times f_{i-1} \\ f_n \end{pmatrix}.$$

Thus, for $r \le c$ without loss of generality (by symmetry)

$$\begin{gathered} \nabla_{\theta_c \theta_r} f = f_r \times \prod_{k=r+1}^n \nabla a_k \times \prod_{k=r+1}^n \theta_k \times h_{rc} \\ \text{where} \\ h_{rc} = \begin{cases} 0 & \text{if } r = c = n+1 \\ \xi_c & \text{if } r = c \text{ and } r < n+1 \\ \xi_c + 1/\theta_c & \text{otherwise,} \end{cases} \\ \text{and } \xi_c = f_c \sum_{k=c+1}^n \frac{\nabla^2 a_k}{\nabla a_k} \prod_{j=1}^{k-1} \theta_j \nabla a_j. \end{gathered} \tag{5.27}$$

This equation favors simplicity over numerical stability – in finite precision $\prod_i \theta_i (a + b/\theta_j)$ can be very different to $\prod_i \theta_i a + \prod_{i \neq j} \theta_i b$.

Comparing Equation (5.26) and Equation (5.27) we see that

$$\nabla_{\theta_c\theta_r} f = \nabla_x^2 f \times \frac{f_r}{\theta_0^2} \times \left(\frac{1}{\prod_{k=1}^{r} \theta_k} \times \frac{1}{\prod_{k=1}^{r} \nabla a_k} \times \frac{h_{rc}}{h_{00}/f_0} \right). \tag{5.28}$$

A similar expression holds for $\nabla_{\theta_c\theta_r} f / \nabla_{\theta_c} f / \nabla_{\theta_r} f$. To the best of our knowledge, such a relationship connecting the second derivative of the input-output mapping to the logit Hessian with respect to the parameters is novel. For example, $\theta_0^2 \nabla_{\theta_0\theta_0} f = x^2 \nabla_x^2 f$, which is intuitive enough since $x\theta_0$ is the input to the network, and a change in one term can be straightforwardly adapted to another.

Equation (5.28) gives a sense in which lowering the input-output curvature reduces parameter curvature, if the other terms do not more than offset the effect.

Chapter 6

Future Developments and Conclusion

Deep learning is here to stay: if progress stopped today, it would be decades before mankind could fully digest all of the marvelous and scary things that such technology enables. This final short chapter applies the ideas developed in this thesis to look at what the future may hold.

One very apparent trend is the increased capability of DNNs outside of engineering and scientific domains. Natural language modelling, for instance, has been completely revolutionized by DNNs. For large language models (LLMs), it seems that wholly different approaches to safety will need to be developed. This is partly pragmatic – these models are large and implement mathematically diverse logic – thus a purely computation-based approach faces considerable headwinds. But mostly, it reflects different priorities and standards. Humans themselves are highly fallible users of natural language, some degree of unsafe output may be acceptable in such systems, and language is a dynamic societal contstruct. Thus it seems like a more empirical, rather than almost-certain, standard of safety is more compatible as an acceptable tradeoff of risk and benefit. The correct notions of control and understanding are not purely technical issues and must be codetermined with the goals and attitudes of the civilization with which it coexists. A further comparison of such systems to engineering DNNs is in Appendix A.2.

Section 1.5 compared the historical development of the probability to current progress in deep learning. By continuing into the present of probability theory, we can attempt to glean some insights about the future of deep learning.

Hard takeoff? The recent trajectory of deep learning is of rapid (and arguably accelerating) progress. Following considerable advances in the 1800s, progress in probability theory did eventually plateau – the basic framework appears to have changed little since Kolmogorov in the 1930s. The level to which progress on DNNs will begin converging remains to be seen, but knowing that progress should decelerate eventually is itself comforting.

Derivative fields The tempering of progress on probability is partially explained by observing that probabilitistic thinking birthed entirely new fields. For example, one can productively navigate the statistical analysis of data or monte carlo methods for simulation with a fairly shallow understanding of the foundational field. New fields are already being created, such as natural language methods for interacting with LLMs, image and video synthesis, and DNN-based combinatorial optimization.

Human institutions The gambling analogy shows how theoretical possibility can transpose itself to human institutions. Blackjack survived the widespread diffusion of card counting, despite making it theoretically unprofitable. Online poker is a healthy industry despite it being simple to develop a poker engine that is sufficiently superhuman to overcome the "rake" and automate its gameplay. Simple countermeasures can suffice practically to give a sustainable equilibrium. This is to say that the ability and willingness of human institutions to subvert mathematical rationality should not be underestimated.

Education My academically unremarkable high school in mid 2000s rural America required all students to learn the rudiments of probability. Japanese school children today attend compulsory courses in computer programming ([137]). It appears that future children might be obliged to learn the basics of deep neural networks in a direct continuation of this trend.

The primary forecastable sources of uncertainty about the future progress of deep learning are around scaling data and compute. Humanity has an impressive record of both increasing computational resources and reducing the necessary complexity of large computations. And observations about continual improvements in computing are supported by a *most* necessary items becoming more economical over time, through conservation, increased production, or substitution of related inputs. Since at least Malthus, predictions of unavoidable scarcity have been repeatedly shown wrong. Thus, I am optimistic about the future of the compute dimension, however continued growth in data and data-efficiency seems more challenging. There are fundamental – hopefully immutable – limits arising from privacy and ethical considerations on

how far data collection about humans might progress. In this respect, it warrants noticing that engineering DNNs tend not to "run out" of data, thus it is my belief that the engineering DNNs will continue to become a more relevant class of models.

The famous "AI Effect" was pithily summarized by John McCarthy as "As soon as it works, no one calls it AI anymore" (Vardi [164]) with reference to earlier breakthroughs such as superhuman chess playing. With deep neural networks, this paradigm seems to have been convincingly broken: systems that underlie our daily lives "work", but we still cannot guarantee many elementary properties about its behavior. With research of the sort presented in this thesis, I believe that there is some hope of engineering DNNs no longer being called AI.

The future is bright, if we can make it there. But: it starts with understanding and controlling small, simple DNNs.

Appendix A

Further Discussion of the Role of Engineering DNNs in Society

A.1 Examples

The application of DNNs to science and engineering has not captured the public's attention like work in natural language or image generation. But away from the spotlight, significant successes can be seen using DNNs as learned, general function approximators. For instance, in 2019 more than half of 73 top European manufacturers surveyed by CapGemini were using at least one AI model in their production process (see CapGemini Research Institute [23]). This progress is welcome because in the physical world, although progress has also been significant, robots still struggle to navigate uneven terrain or smoothly open a door.

A quick aside on computer vision: Most DNNs taking images as inputs should probably not be considered engineering DNNs. Their inputs are generally not interpretable to people, performance was not typically very good prior to the adoption of deep learning, and architectures can be complicated and large. However, a DNN that takes homogeneous inputs from a controlled environment (fixed lighting, background, resolution, etc.) to predict very narrow outcomes, has flavors of an engineering DNN. A related modelling problem are computer vision models that operate on synthetic data – say generated using a video game graphics engine. DNNs used in optical character recognition, fault detection of manufactured items, or some medical imaging applications, have flavours of engineering DNNs.

A.1.1 Example #1: Quadripedal robot locomotion

Tan et al. [154] describes an application of DNNs to optimize the efficiency of a quadripedal robot. By utilizing the technical specifications and actual observations of the component's material properties the authors are able to develop a bottom-up software simulation of the actual hardware. The model is parameterized by physical constants such as coefficients of friction, mass, battery voltage, and response latency. Within each such model of the robot's operations, they are able to understand how sensor readings such as the roll, pitch, and velocities of each component map into the actuators controlling the position of each leg.

By formulating a reward function that favors fast and energy-efficient locomotion, reinforcement learning can be used to optimize this mapping, where the policy function is a two hidden layer neural network. And finally, by randomizing over many configurations of the assumed physical constants that govern the in-silico simulation, they are able to develop superior trotting and galloping algorithms to those furnished by the manufacturer, when transposed to the actual hardware.

A.1.2 Example #2: AlphaDogfight Trials

Pope et al. [125] presents the AlphaDogfight Trials: a competition hosted by the United States' military research arm, DARPA. The competition entailed quantitative modellers and human pilots competing in a series of military "dogfights" (singular air to air combat) simulated with high fidelity in software.

Agents, human or silicone, had access to the state of the system such as the fuel, health, position and velocity, of their own and opposing planes, and emitted actions 50 times / second to a simulated F-16 (US military fighter jet)'s aileron, elevator, rudder, and throttle controls, with a goal of reaching a favorable attack configuration. Pope et al. [125] and others were able to solve this as a reinforcement learning problem and develop DNN agents that were able to best highly-trained human fighter pilots.

A.1.3 Anti-Example #1: Limit Order Book models

Financial markets are where financial assets such as stocks and futures are traded. The most important markets are centralized around a double auction mechanism, where participants continually post prices at which they are willing to buy and sell the asset, and whenver someone is willing to purchase for more than someone is willing

to sell (or sell for less than someone is willing to buy), a trade occurs. Some market participants attempt to profit by forecasting the future direction of prices by looking at summaries of the *limit order book*, which contains a full tabulation of the prices ("level") and quantities ("depths") that all participants have indicated a willingness to transact at. For example, if there are 1000 shares of a stock for sale at 12.34 CHF and an offer to purchase only one share at 12.33 CHF, then we would suspect that the next distinct price would be 12.33 since otherwise 1000 shares would have to be removed from the sell side before 10 shares were removed from the buy side. Sirignano [146] posits predicting one second ahead price changes of large American equiities using the book depth at many relative levels.

Interestingly, using only order book data does imply a simple relationship: putting aside spoofing, data errors, hidden orders, or other exceptional situations, more offers to sell at any level must – by the construction of a market equilibrium – be bearish for prices, and more offers to buy bullish. *A priori* monotonicity relationships like this are exactly what is meant in Figure 1.1. However, forecasts can be improved by including more inputs in the prediction, such as the price changes of related assets and derivatives, trade volume, and order cancellations, and the direction of the predictability of these features must be estimated.

Another difference between limit order book modelling and engineering DNNs is data scarcity. Market participant's efforts to exploit limited inherent predictability over short horizons (an oracle able to forecast which assets will rise in price soon would purchase those assets now and eliminate the anticipated price rise, making the markets even less forecastable to others) make signals very weak and time-varying in nature. Not overfitting limited data is *the* fundamental problem in this type of modelling, and to a well-resourced participant in financial markets, new data can *only* be acquired with the passage of time. This is the opposite of problems in engineering, where data can be acquired with a modest marginal cost.

A.2 Some Asides on Engineering DNNs in Society

Chapter 1 has shown a top-down view of how science progresses human welfare, and Appendix A.1 gave bottom-up examples of engineering DNNs. This section fleshes out the middle, with further justification for research on engineering DNNs informed by real-world events affecting our lives in 2023. Appendix A.2.1 shows that DNNs as weapons systems is not an abstract possibility, but is a reality today. The essence of Appendix A.2.2 is that engineering DNNs have some claim on the moral or ethical high

ground within the space of DNNs.

A.2.1 Military Applications

As shown in Appendix A.1, engineering DNNs clearly *could* impact the battlefield. This is not hypothetical, but rather a current and ongoing reality.

Scharre [140] describes the Harpy, a "loitering munition" that can be launched and stay airborne for hours before deciding autonomously (without a human in the loop) when and where to strike. Put simply: a Harpy could be launched, and eight hours later someone could be violently killed by the automated decisionmaking of a machine learning model (the Harpy is an "anti-radiation" weapon designed to target radar installations, so this is not the primary anticipated target). The legal and ethical principles demonstrated by the Harpy are profound. For a readable introduction to autonomy on the battlefield, see Kott and Stump [91]. A more recent anecdote comes from a 2022 SEC filing by Nvidia Corporation [118]

> On August 26, 2022, the U.S. government, or USG, informed NVIDIA Corporation, or the Company, that the USG has imposed a new license requirement, effective immediately, for any future export to China (including Hong Kong) and Russia of the Company's A100 and forthcoming H100 integrated circuits. DGX or any other systems which incorporate A100 or H100 integrated circuits and the A100X are also covered by the new license requirement. The license requirement also includes any future NVIDIA integrated circuit achieving both peak performance and chip-to-chip I/O performance equal to or greater than thresholds that are roughly equivalent to the A100, as well as any system that includes those circuits. A license is required to export technology to support or develop covered products. The USG indicated that the new license requirement will address the risk that the covered products may be used in, or diverted to, a 'military end use' or 'military end user' in China and Russia.

This is not a minor technicality or quirk of policy. This is the government of the world's largest economy ordering one of the largest companies in the world not to export its flagship product to the world's second largest and eleventh largest economies because they do not want it to be used (to train DNNs, presumably) for military purposes. And this is clearly a substantive embargo: there has by now been several iterations of the export ban being updated and NVIDIA adapting their offerings (e.g. offering a detuned

version to comply with this regulation). It is not hyperbolic to say that the age of deep neural networks becoming literal weapons of war is already upon us.

A.2.2 The Human Harm of Deep Learning

Online platforms maximize revenue by maximizing engagement. And the consequences for the mental and physical health of individuals, and the quality of public discourse and governance of a populace, that is "extremely online" can be negative.

Dattani et al. [39], gives analytics on the spatial and temporal dynamics of mental health, and shows that anxiety disorder prevalence appears to increase with GDP per capita, and is getting generally worse for all countries over the 1990 to 2019 period. To the extent that deep learning enables the subjugation of people's preferences, it is harmful. And we know that deep learning is being used in this endeavor. Meta (formerly Facebook), for instance, have numerous large and well resourced teams working on deep learning. Similarly for alphabet (formerly Google), X (formerly Twitter), Netflix, and others, as well as the Chinese corollaries of these companies. The abstract modelling of social media, search engines, and the like, are decidedly incompatible with Figure 1.1.

A worrying trend is DNNs becoming comparatively better at relatively "good" tasks. Consider two important consumption categories of a modern resident of a developed city: (1) music, movies, literature, visual art, poetry, therapy, and philosophy, and (2) food delivery, cleaning services, agricultural products, garbage collection, and taxi services. Both classes of goods are being automated by DNNs, though only the first would be considered by most to be psychically interesting and satisfying to produce. It would be a utilitarian nightmare if large numbers of creative and empathic workers were replaced by DNNs, though this is what unfortunately appears to be occuring.

Put bleakly: a growing number of jobs amount to acting as dextrous actuators or capable classifiers for a computer program, for example independent food delivery, ride-hailing apps, data annotation, content moderation, and warehouse logistics. Amazon employs upwards of 1.5 million people ([151]) and it appears that the majority work in "fulfillment", doing the fine-grained tasks around packing, sorting, and manipulating orders that cannot be more directly automated. Any particular instance may be a fine organization, the simple principle of an algorithm having considerable control over people with minimal direct oversight is undeniably risky.

Historically, less desirable tasks were easier to automate, aligning the moral and eco-

nomic considerations, but in a world of highly competent non-engineering DNNs, this is becoming less true. Scientifically, it may be simpler and more economical to develop a computer that can diagnose illnesses in natural language than it is to design a program that can clean toilets, but – humanistically – I have a strong preference to see dangerous, dirty, and difficult tasks displaced first. Concern about DNN capabilities becoming "imbalanced" is another reason to prefer advances in engineering DNNs, if indifferent.

Appendix B

Polytope Calculations

B.1 Introduction

Polytopes are curious objects: their description is simple and their intuition is clear, but their mathematical analysis is quite complex and subtle. It is possible to have a long career doing applied math and yet remain clueless about polytopes (I am proof). This note collects some useful methods – some only found in recent research papers – from my work with polytopes. Some bits of this "cookbook" are modestly interesting or cool, but really *nothing here stands on its own scientifically for its novelty – I include it my thesis because I want a convenient, persistent collection of these facts.* One can think of this as an effort to make polytopes easier to work with "without thinking", in the words of Whitehead [170].

The overarching principle is to be useful and simple. This means being very concrete and simple, and not relying on abstract mathematics. This necessary comes at the expense of generality and elegance. Secondarily, the methods I propose largely work for unbounded polytopes in high dimensions, and ignores many results optimized for three or four dimensional results arising in computer graphics or physics simulations. I also provide pointers to software and discuss numerical stability.

A great resource (that I refer to often) is `https://people.inf.ethz.ch/~fukudak/polyfaq/`. Phrased as a "FAQ" (frequently asked questions) it is a better introductory resource, with more math, and written by a genuine master in the area. This section is complementary in that it is probably more useful to someone looking to model productively with polytopes and is willing to forego some of the deeper concepts.

B.2 H and V Representations of Polytopes

First, the most prominent aspect of computing with polytopes: two equivalent formulations as (1) a system of linear inequalities parameterized by A, b: $\{x : Ax \leq b\}$ and (2) a nonnegative combination of vertices parameterized by V, R

$$\left\{ V\mu + R\lambda : \lambda \geq 0, \mu \geq 0, \sum_i \mu_i = 1 \right\}. \tag{B.1}$$

Definition 1, as well as Lemma 3 and Lemma 4 worked with what is known as the *H representation* (or form) of a polytope. Equation (B.1) is called the *V representation*. The "Minkowski-Weyl Theorem" (Fukuda [54]) states that these are equivalent and demonstrates the idea for moving between them.

In words: A, b give the weight and limits in a formulation as the intersection of hyperplane (also called *facets*), and the columns of V, R in Equation (B.1) characterize the elements of the Minkowski sum of (1) a convex combination of vertices (columns of V), and (2) a conic combination of some rays (columns of R).

Some examples of things that are generally easier in V form:

- Compute the image of a polytope under a linear mapping
- Computing the dimension of a polytope
- Sampling from a polytope
- Proving uniformity of a sign (if all vertices have a positive coordinate, then the polytope does not cross the origin in that coordinate)
- Computing the volume of a polytope

Some examples of things that are generally easier in H form:

- Invoking optimization software
- Describing simple sets such as the d dimensional unit cube (as $2d$ inequalities)
- Compute the preimage of a polytope under a linear mapping

- Compute the distance between two polytopes
- Intersect two polytopes
- Checking if a point is contained in a polytope.

More examples of operations which are simple in one form and difficult in the other will be given subsequently.

The H and V formulations are equivalent in the sense that given one there are well-studied and nicely implemented methods for converting it to the other. Unfortunately, this is challenging, both theoretically and practically. Analyzing formally the computational complexity of the problem is complicated, because the complexity of the algorithm depends on both the size of the input and output, and small changes to the input can have very large effects on the output.

In general, there are no known algorithms that have polynomial running time in the input and output size. For some polytopes, there are algorithms that have polynomial time and space complexity in the input and output. However, but this is still exponentially large in the dimension of the polytopes, again because output size can be exponentially large in the input size.

More details are in Fukuda [54, Section 9].

B.3 Unary Operations

This section describes some useful operations on a single polytope.

B.3.1 Redundancy Removal

When modelling, it is very possible to have V, R or A, b pairs that are "too large", in the sense that there are other matrices of smaller size that represent the same polytopes. For the H representation, one ad hoc method is to compute the correlation matrix between the rows of $\begin{pmatrix} b & A \end{pmatrix}$ and remove any rows that are perfectly correlated to others. A more principled method is given by Fukuda [54, Section 8.2].

For the V representation, convex hull software, for example Barber et al. [10] or Fukuda and Prodon [55], may give a representation without redundancies, though this can be

slow (see Appendix B.3.3). A method akin to that used for the H representation should be quite doable, if a convex hull is not required and speed is a consideration.

B.3.2 Applying Linear Mappings

Applying a linear mapping to a polytope is a polytope, and this section gives equations for the linearly transformed polytope. Fundamentally, there are two problems:

- The application problem: $\{Wx + a : x \in P\}$.
- The preimage problem: $\{x : Wx + a \in P\}$.

For each of these problems, there are two basic cases: from and to the V and H representations. The easy cases are (1) given the V representation of a polytope and want the V representation of its image:

$$\begin{aligned}
&\{Wx + a : x \in P\} \\
&= \left\{ Wx + a : x = \sum_i v_i \lambda_i + \sum_j r_j \nu_j, \lambda_i \geq 0, \nu_j \geq 0, \sum_i \lambda_i = 1 \right\} \\
&= \left\{ W\left(\sum_i v_i \lambda_i + \sum_j r_j \nu_j \right) + a : \lambda_i \geq 0, \nu_j \geq 0, \sum_i \lambda_i = 1 \right\} \\
&= \left\{ \sum_i (Wv_i + a)\lambda_i + \sum_j Wr_j \nu_j : \lambda_i \geq 0, \nu_j \geq 0, \sum_i \lambda_i = 1 \right\}
\end{aligned}$$

and (2) given the H representation of a polytope and want the H representation of its preimage:

$$Wx + a \in P \iff A(Wx + a) \leq b \iff x \in \{x' : AWx' \leq b - Aa\}.$$

Next, we look at the preimage problem for V representations. We suppose that W is full (column)-rank, and let $W^\dagger$ be its pseudoinverse and $W^\perp$ be a basis for the nullspace (with kth column $W^\perp_k$) then, for $\lambda_i \geq 0, \nu_j \geq 0, \sum_i \lambda_i = 1$:

$$
\begin{aligned}
Wx + a = \sum_i v_i \lambda_i + \sum_j r_j \nu_j \iff \\
x = \sum_i W^\dagger (v_i - a)\lambda_i + \sum_j (W^\dagger r_j)\nu_j + \sum_k W_k^\perp \gamma_k \\
= \sum_i W^\dagger (v_i - a)\lambda_i + \sum_j (W^\dagger r_j)\nu_j + \sum_k W_k^\perp \gamma_k^+ + \sum_k (-1 \times W_k^\perp)\gamma_k^- \\
\text{where } \gamma_k^- \geq 0, \gamma_k^+ \geq 0.
\end{aligned}
\tag{B.2}
$$

So, $Wx + a \in P \iff x$ is contained in a polytope which has V representation with vertices $W^\dagger(v_i - a)$ and rays $(W^\dagger r_j), W_k^\perp, -1 \times W_k^\perp$.

Lastly, we investigate the H form of the application of a linear mapping to a polytope in H form. As before, we assume that W has full column rank, so that $W^\dagger W = I$, and

$$
\begin{aligned}
x \in P \iff Ax \leq b \iff AW^\dagger Wx \leq b \iff AW^\dagger (Wx + a) \leq b + AW^\dagger a \\
\iff Wx + a \in \left\{ x' : AW^\dagger x' \leq b + AW^\dagger a \right\}.
\end{aligned}
$$

B.3.3 Volume and Dimension

This section concerns operations on the volume and dimension of polytopes.

Volume is a stronger concept than dimension: having a positive volume necessarily implies full-dimension (and any full dimensional polytope must have positive volume) but, volume conveys more information besides.

Compute Polytope Dimension

The dimension of a polytope is the maximum number of affinely independent points it contains, minus one (Fukuda [54]). Given a polytope in the form of Equation (B.1), the dimension is

$$
\operatorname{rank} \begin{pmatrix} V & R \\ 1^\top & 0^\top \end{pmatrix} - 1. \tag{B.3}
$$

Given a polytope in H representation, $\{x : b - Ax \geq 0\}$, let $\varkappa^\star$ be the criterion achieved in the optimization problem

$$\text{maximize } \varkappa \text{ subject to } Ax + \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \varkappa \leq b, \varkappa \leq 1. \tag{B.4}$$

The conformable column vector of ones, representing the intuition that it is possible to loosen all inequality conditions by a strictly positive amount, meaning that there is some volume interior to the polytope. The ancillary condition that $\varkappa \leq 1$ is used to keep the problem well-conditioned.

If $\varkappa^\star > 0$, then the polytope has the dimension of x. If $\varkappa^\star < 0$ then $Ax \leq b$ is totally infeasbile, and the polytope is empty, thus its dimension is zero. If $0 = \varkappa^\star$, then we can eliminate some implicit equalities in the definition of the polytope and then reattempt to determine if the polytope is full-dimensional in a lower-dimensional space. Fukuda [54, Section 8.3] contains more details on this intuitive process.

Projected Polytopes: Form and Dimension

Projecting a polytope, or interpreting a polytope as a projection of a higher-dimensional space, can be quite useful. Unfortunately, the notation is quite heavy and not really justified by the limited degree to which we use it. Thus, I will just briefly introduce the concept and the idea, and describe a few of the results from Balas and Oosten [9], which is a quite thorough treatment of the concept.

For a polytope $P = \{(x_1, x_2) \in \mathbb{R}^{d_1+d_2} : A_1 x_1 + A_2 x_2 \leq b\}$, the *projection onto x_2 space* is the set $\{x_2 \in \mathbb{R}^{d_1} : (x_1, x_2) \in P \text{ for some } x_1 \in \mathbb{R}^{d_1}\}$.

Balas and Oosten [9] gives various useful facts about the projection of a polytope, including a formulation as a polytope itself:

$$\left\{x_2 \in \mathbb{R}^{d_1} : (x_1, x_2) \in P \text{ for some } x_1 \in \mathbb{R}^{d_1}\right\} = \left\{x_2 : (vB)x_2 \leq vb \text{ for all } v \geq 0 \text{ such that } vA = 0\right\}.$$

And moreover gives an equation for the dimension of a polytope projection in terms of

the parameters of the original polytope.

Computing Polytope Volume

Evaluating the volume of a polytope is difficult in general. Put simply, if a polytope $\subseteq \mathbb{R}^d$ is written in the form of Equation (B.1) with $R = 0$, no redundant columns in $V = \begin{pmatrix} V_1 & V_2 & \dots & V_{d+1} \end{pmatrix}$ (meaning that for all V' comprised of a strict subset of rows of V the implied polytope of V' is different than that implied by V), with $d+1$ vertices, then the volume of this polytope is

$$\det\begin{pmatrix} V_2 - V_1 & V_3 - V_1 & \dots & V_{d+1} - V_1 \end{pmatrix}. \tag{B.5}$$

As put by Büeler, Enge, and Fukuda [20], "All known algorithms for exact volume computation decompose a given polytope into simplices [the polytope described above], and thus they all rely, explicitly or implicitly, on [Equation (B.5)]."

The computation of volume is closely related to the convex hulls, and shares the same fundamental difficulty of representing a general polytope as the union of simplies, and computing the convex hull using the quickhull algorithm (Barber et al. [10]), which is built into scipy as `scipy.spatial.ConvexHull` gives the volume of the polytope as a byproduct of the computation. Computing either is intractable at scale (both in number of points and dimension) in general.

Maximum Volume Inner Box

Given a polytope full-dimensional polytope in H form $P = \{x : Ax \leq b\}$, suppose that want to compute the largest box $[\ell, u] \subseteq P$. For example, this would give a quick and dirty lower bound on the volume of a polytope.

Bemporad, Filippi, and Torrisi [13] show that for $[x^\star, x^\star + y^\star]$ is a maximum volume inner box for P, where

$$x^\star, y^\star = \underset{x,y}{\operatorname{argmax}} \sum_i \log y_i \text{ subject to } Ax + \operatorname{ReLU}(A)y \leq b. \tag{B.6}$$

Equation (B.6) is not a LP, but it is convex, and thus although more general and difficult to analyze, it is not more conceptually or computationally difficult.

Minimum Volume Outer Box

Suppose that you want to find a box that encloses a polytope $P = \{x : Ax \leq b\}$, one fast way to do this is to solve the $2n$ optimization problems:

$$\ell_j = \min \iota_j' x \text{ subject to } Ax \leq b \tag{B.7}$$

$$u_j = \max \iota_j' x \text{ subject to } Ax \leq b \tag{B.8}$$

Then by construction $P \subseteq [\ell, u]$. An encompassing box can be a useful diagnostic since it is trivially simple to reason about.

Similarly, the method lends itself to summarizing multiple polytopes: a box that encompasses all $P_i = \{x : A_i x \leq b_i\}$ for multiple i can be constructed by solving the same problem with all constraints applied simultaneously.

B.4 Binary Operations

B.4.1 Checking if a Point is in a Polytope

Determining when some x is in P is trivial if P is in its H representation: by construction $x \in P$ if $Ax \leq b$. This is a simple matter of a matrix multiplication and a comparison.

When P is given by its V representation, we can also reason directly from the definition of P. The simplest way that to do this is to solve the optimization problem

$$\min_{\mu,\lambda} ||V\mu + R\lambda - x|| \text{ subject to } \sum_j \mu_j = 1, \mu \geq 0, \lambda \geq 0. \tag{B.9}$$

And to conclude that that $x \in P$ iff this optimization program achieves a zero value. Which norm is chosen in the criterion should be dictated by the availability of software,

for example the one and infinity norm problems can be solved using a LP solver. If quadratic programming software is available, the L_2 norm may be preferable for some problems. This approach has the benefit of being fully constructive, in that it proves the inclusion with the weights that construct it.

`https://en.wikipedia.org/wiki/Point_in_polygon` details some methods that do not use a numerical solver.

B.4.2 Checking Polytope Containment

As a generalization of Appendix B.4.1, it is possible to see whether one polytope contains another as follows.

H Representation ⊆ H Representation?

If both polytopes are in H representation, we can use the approach described at [1]. For $P_1 = \{x : A_1 x \leq b_1\}, P_2 = \{x : A_2 x \leq b_2\}$ if

$$\max_x A_{1i} x - b_{1i} \text{ subject to } A_2 x \leq b_2 \text{ is } \leq 0 \text{ for all } i$$

then $P_1 \subseteq P_2$. Farkas' Lemma (e.g., Carver [26]) can also be used to put this problem into a dual form which requires only a single feasibility check.[1]

V Representation ⊆ H Representation?

A formulation similar to Appendix B.4.1 can be used. If

$$\max_{\mu,\lambda,x} A_i x - b_i \text{ subject to } x = V\mu + R\lambda, \sum_j \mu_j = 1, \mu \geq 0, \lambda \geq 0 \text{ is } \leq 0 \text{ for all } i.$$

then the polytope with V representation characterized by (V, R) is a subset of the polytope with H representation characterized by (A, b).

[1] See `https://math.stackexchange.com/q/2097261`.

This requires the solution of only as many linear programs as there are hyperplanes defining the possibly enclosing polytope.

H Representation ⊆ V Representation?

A simple approach is to put V represntation into its H representation then apply Appendix B.4.2.

Freund and Orlin [52] shows that the original problem is NP-complete, thus this naïve approach will not render intractable an otherwise soluble problem.

B.4.3 Compute Distance Between Polytopes

The distance between polytopes P_1, P_2 is the minimum distance $||p_1 - p_2||, p_1 \in P_1, p_2 \in P_2$. This quantity can be computed directly from the definition for a variety of convex norms using the formulations of polytope containment from Equation (B.9).

B.4.4 Intersect Polytopes

Given the H representation of polytopes, computing the H representation is easy: stack them up.

Given the V representation of polytopes, computing the V representation is hard: `https://people.inf.ethz.ch/fukudak/polyfaq/node25.html`.

An important special case where more can be said is finding a single point in the intersection two affine spaces. This problem arises, for example, in the computation of the preimage of a point through a ReLU mapping via the algorithm presented by Carlsson, Azizpour, and Razavian [25].

Suppose that we want to find a point in the intersection of two affine spaces, $\{c_1 + V_1\mu : \sum_j \mu_j = 1, \mu_j \geq 0\}$ and $\{c_2 + V_2\mu : \sum_j \mu_j = 1, \mu_j \geq 0\}$. This means finding a w_1, w_2 so that $x = c_1 + V_1 w_1 = c_2 + V_2 w_2$. For this, observe that:

$$\begin{aligned} x = c_1 + V_1 w_1 = c_2 + V_2 w_2 &\iff 0 = c_1 - c_2 + V_1 w_1 - V_2 w_2 \\ &\iff c_1 - c_2 = \begin{pmatrix} -V_1 & +V_2 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \\ &\iff \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -V_1 & +V_2 \end{pmatrix}^{\dagger} (c_1 - c_2). \end{aligned}$$

B.4.5 Test Whether Two Polytopes in H Representation are Equal

Matrices are useful for representing polytopes, but they are imperfect in that many different matrices can represent the same polytope. In order to determine whether two polytopes are equal, reducing false negatives is useful.

The two frequent sources of indeterminacy are (1) multiplying rows of the inequality $Ax \le b$ by positive constants, and (2) reordering the rows of $Ax \le b$. And one method to resolve both is to first scale each row in a consistent convention, then second compare de-ordered rows for example as sets of rows (vectors) – since sets have no order all row permutations will be considered equal.

A complete approach can be performed at greater cost by applying Appendix B.4.2 in both directions.

B.4.6 Unioning Two Polytopes

Let $P_1 = \{x : A_1 x \le b_1\}$ and $P_2 = \{x : A_2 x \le b_2\}$. Is $P_1 \cup P_2$ a polytope? And can it be written as a function of A_1, A_2, b_1, and b_2 if so? This section summarizes Bemporad, Fukuda, and Torrisi [14].

Let A_{12}, b_{12} be the rows of A_1, b_1 that also respect the constraints of A_2, b_2, and similarly let A_{21}, b_{21} denote the rows of A_2, b_2 that also respect the constraints of A_1, b_1. And let $\text{ENV}(P_1, P_2) = \{x : A_{12} x \le b_{12}, A_{21} x \le b_{21}\}$ denote the "envelope" of P_1 and P_2. $\text{ENV}(P_1, P_2)$ is clearly a polytope, and $P_1 \cup P_2 \subseteq \text{ENV}(P_1, P_2)$. And moreover, $P_1 \cup P_2$ is convex if and only if $P_1 \cup P_2 = \text{ENV}(P_1, P_2)$. The idea is demonstrated schematically in Figure B.1. Bemporad, Fukuda, and Torrisi [14] uses this intuition to furthermore give an algorithm which solves an LP for each pair (one from each of P_1 and P_2) of excluded constraints in order to find out whether it is possible to violate both whilst staying within $\text{ENV}(P_1, P_2)$ as an algorithm for determining if $P_1 \cup P_2$ is convex.

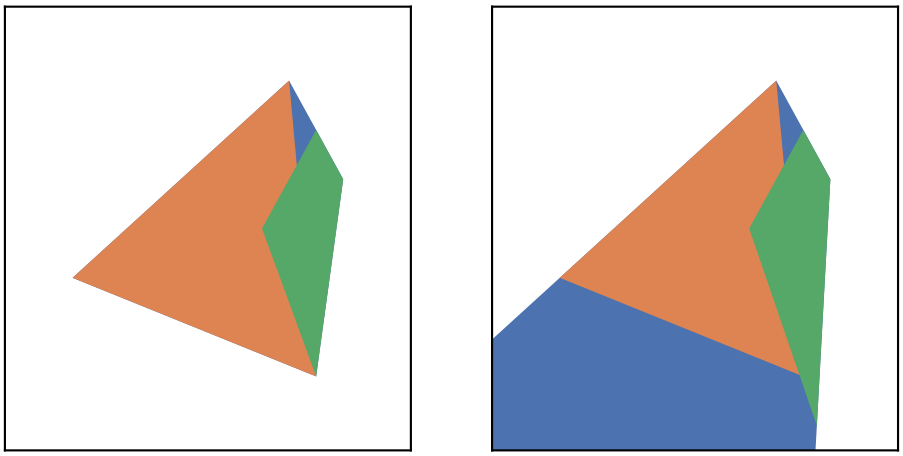


Figure B.1: Two polytopes in orange and green whose union is not convex, and their envelope given by the union of all three colored components. The plot on the left is inspired by Bemporad, Fukuda, and Torrisi [14, Figure 1] and gives a sense that the blue component $= \text{ENV}(P_1, P_2) \setminus (P_1 \cup P_2)$ is minimal, since the envelope is the convex hull of the union, however, this is not the case on the right, which slightly perturbs one vertex and results in an unbounded envelope.

For the V representation Bemporad, Fukuda, and Torrisi [14, Lemma 1] states: If P_1 has V representation given by V_1 and R_1 and P_2 has V representation given by V_2 and R_2, then if (and only if, though this is perhaps less useful) $P_1 \cup P_2$ is convex then it has V representation given by $\begin{pmatrix} V_1 & V_2 \end{pmatrix}$ and $\begin{pmatrix} R_1 & R_2 \end{pmatrix}$.

B.4.7 Minkowski Sum

The Minkowski sum of two polytopes can be written as a polytope by using Appendix B.3.2 with A blockwise diagonal, and $W = \begin{pmatrix} +I & -I \end{pmatrix}$. In turn, this can be used to compute the convex hull of a Minkowsi sum, via the fact that the convex hull of a Minkowski sum is the Minkowski sum of the convex hulls.

Appendix C

Software used

I wish to gratefully acknowledge the teams behind:

- cdd [55]
- git
- bokeh [15]
- biblatex, and biber [124]
- cleverhans [120]
- hydra [172]
- LaTeX
- matplotlib [82]
- pandas [111]
- pydev debugger `https://github.com/fabioz/PyDev.Debugger`
- python [163]
- pytorch [121]
- torchvision [106]
- seaborn [168]

- pytest [95]
- scikit-learn [21]
- TikZ and pgf [155]
- vim (Rest in Peace to Bram Moolenaar, who sadly passed away during the writing of this thesis)

When it is straightforward to cite a definite author, I have done so :). Not open source, but generously offering academic licenses are Gurobi [69] and Mosek [6].

Bibliography

[1] Michael Grant (https://math.stackexchange.com/users/52878/michael-grant). *How to check whether a convex polyhedron is contained in another convex polyhedron?* Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/1344131 (cit. on p. 131).

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. "Learning and Generalization in Overparameterized Neural Networks, Going beyond Two Layers". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (2019). URL: https://dl.acm.org/doi/10.5555/3454287.3454840 (cit. on p. 60).

[3] Dario Amodei et al. "Concrete problems in AI safety". In: *arXiv e-prints* (2016). URL: http://arxiv.org/abs/1606.06565 (cit. on pp. 13, 22).

[4] Maksym Andriushchenko and Nicolas Flammarion. "Towards Understanding Sharpness-Aware Minimization". In: *ICML*. Proceedings of Machine Learning Research 162 (July 2022). Ed. by Kamalika Chaudhuri et al., pp. 639–668. URL: https://proceedings.mlr.press/v162/andriushchenko22a.html (cit. on p. 87).

[5] Maksym Andriushchenko et al. "Square Attack: A Query-Efficient Black-Box Adversarial Attack via Random Search". In: *Computer Vision – ECCV 2020* (2020), pp. 484–501. URL: https://doi.org/10.1007/978-3-030-58592-1_29 (cit. on p. 108).

[6] MOSEK ApS. *MOSEK Optimizer API for Python 10.1.11*. 2022. URL: https://docs.mosek.com/latest/pythonapi/index.html (cit. on pp. 47, 136).

[7] Stuart Armstrong and Benjamin Levinstein. "Low impact artificial intelligences". In: *arXiv e-prints* (2017). URL: http://arxiv.org/abs/1705.10720 (cit. on pp. 12, 13, 18, 21, 22).

[8] Emanouil I. Atanassov. “On the Discrepancy of the Halton Sequences”. In: *Mathematica Balkanica* 18 (2004), pp. 15–32. URL: http://www.math.bas.bg/infres/MathBalk/MB-18/MB-18-015-032.pdf (cit. on p. 40).

[9] Egon Balas and Maarten Oosten. “On the dimension of projected polyhedra”. In: *Discrete Applied Mathematics* 87.1 (1998), pp. 1–9. URL: https://doi.org/10.1016/S0166-218X(98)00096-1 (cit. on p. 128).

[10] C. Bradford Barber et al. “The Quickhull Algorithm for Convex Hulls”. In: *ACM Trans. Math. Softw.* 22.4 (Dec. 1996), pp. 469–483. URL: http://doi.acm.org/10.1145/235815.235821 (cit. on pp. 125, 129).

[11] Kinjal Basu and Art B. Owen. “Transformations and Hardy–Krause Variation”. In: *SIAM Journal on Numerical Analysis* 54.3 (2016), pp. 1946–1966. URL: https://doi.org/10.1137/15M1052184 (cit. on p. 39).

[12] Jens Behrmann et al. “Analysis of Invariance and Robustness via Invertibility of ReLU-Networks”. In: *arXiv e-prints* (June 2018). URL: http://arxiv.org/abs/1806.09730 (cit. on p. 27).

[13] Alberto Bemporad, Carlo Filippi, and Fabio D. Torrisi. “Inner and outer approximations of polytopes using boxes”. In: *Computational Geometry* 27.2 (2004), pp. 151–178. URL: https://doi.org/10.1016/S0925-7721(03)00048-8 (cit. on pp. 31, 129).

[14] Alberto Bemporad, Komei Fukuda, and Fabio D. Torrisi. “Convexity recognition of the union of polyhedra”. In: *Computational Geometry* 18.3 (2001), pp. 141–154. URL: https://doi.org/10.1016/S0925-7721(01)00004-9 (cit. on pp. 133, 134).

[15] Bokeh Development Team. *Bokeh: Python library for interactive visualization*. 2018. URL: https://bokeh.pydata.org/en/latest/ (cit. on p. 135).

[16] Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. 1st. USA: Oxford University Press, Inc., 2014. ISBN: 0199678111 (cit. on p. 4).

[17] Y-Lan Boureau, Jean Ponce, and Yann LeCun. “A Theoretical Analysis of Feature Pooling in Visual Recognition”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10 (2010), pp. 111–118. URL: https://dl.acm.org/doi/10.5555/3104322.3104338 (cit. on p. 52).

[18] Steven Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf (cit. on p. 75).

[19] Greg Brockman et al. “OpenAI Gym”. In: *arXiv e-prints* (2016). URL: http://arxiv.org/abs/1606.01540 (cit. on pp. 30, 43, 44).

[20] Benno Büeler, Andreas Enge, and Komei Fukuda. "Exact Volume Computation for Polytopes: A Practical Study". In: (2000). Ed. by Gil Kalai and Günter M. Ziegler, pp. 131–154. URL: https://doi.org/10.1007/978-3-0348-8438-9_6 (cit. on p. 129).

[21] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122. URL: http://arxiv.org/abs/1309.0238 (cit. on pp. 43, 136).

[22] Alejandro Hernández Cano et al. *epfLLM Megatron-LM*. 2023. URL: https://github.com/epfLLM/Megatron-LM (cit. on p. ii).

[23] CapGemini Research Institute. *Scaling AI in Manufacturing Operations: A Practitioners' Perspective*. URL https://www.capgemini.com/wp-content/uploads/2019/12/AI-in-manufacturing-operations-2-1.pdf. Accessed 2023-10-01 (cit. on p. 117).

[24] Gerolamo Cardano. *The Book on Games of Chance: The 16th-Century Treatise on Probability*. Trans. by Sydney Henry Gould. Dover Publications (cit. on p. 8).

[25] Stefan Carlsson, Hossein Azizpour, and Ali Sharif Razavian. "The Preimage of Rectifier Network Activities". In: *Submitted to ICLR 2017* (2017). URL: https://openreview.net/pdf?id=HJcLcw9xg (cit. on pp. 26, 27, 132).

[26] Walter B. Carver. "Systems of Linear Inequalities". In: *Annals of Mathematics* 23.3 (1922), pp. 212–220. URL: http://www.jstor.org/stable/1967919 (cit. on pp. 69, 131).

[27] Zeming Chen et al. "MEDITRON-70B: Scaling Medical Pretraining for Large Language Models". In: (2023). URL: https://arxiv.org/abs/2311.16079 (cit. on p. ii).

[28] Aakanksha Chowdhery et al. "Palm: Scaling language modeling with pathways". In: *arXiv e-prints* (2022). URL: https://arxiv.org/abs/2204.02311 (cit. on p. 84).

[29] Paul F Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems* 30 (2017). Ed. by I. Guyon et al. URL: https://dl.acm.org/doi/10.5555/3294996.3295184 (cit. on p. 23).

[30] Moustapha Cisse et al. "Parseval Networks: Improving Robustness to Adversarial Examples". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17 (2017), pp. 854–863. URL: https://dl.acm.org/doi/10.5555/3305381.3305470 (cit. on p. 88).

[31] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. "Certified adversarial robustness via randomized smoothing". In: *International Conference on Machine Learning* (June 2019), pp. 1310–1320. URL: https://proceedings.mlr.press/v97/cohen19c.html (cit. on p. 87).

[32] Cody Coleman et al. "Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark". In: *arXiv e-prints* (June 2018). URL: http://arxiv.org/abs/1806.01427 (cit. on p. 51).

[33] Ronan Collobert. "Large Scale Machine Learning". PhD thesis. Université Paris VI, 2004 (cit. on p. 5).

[34] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *arXiv e-prints* (Nov. 2015). URL: https://doi.org/10.48550/arXiv.1511.00363 (cit. on p. 5).

[35] Francesco Croce and Matthias Hein. "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks". In: *Proceedings of the 37th International Conference on Machine Learning*. Proceedings of Machine Learning Research 119 (July 2020). Ed. by Hal Daumé III and Aarti Singh, pp. 2206–2216. URL: https://proceedings.mlr.press/v119/croce20b.html (cit. on p. 108).

[36] Francesco Croce et al. "RobustBench: a standardized adversarial robustness benchmark". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2021). URL: https://openreview.net/forum?id=SSKZPJCt7B (cit. on p. 108).

[37] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. URL: https://doi.org/10.1007/BF02551274 (cit. on pp. 6, 52).

[38] Alexander D'Amour et al. "Underspecification Presents Challenges for Credibility in Modern Machine Learning". In: *Journal of Machine Learning Research* 23.1 (Jan. 2022), pp. 1–61. URL: http://jmlr.org/papers/v23/20-1335.html (cit. on p. 84).

[39] Saloni Dattani et al. *Our World in Data: Mental Health*. URL https://ourworldindata.org/mental-health. Accessed 2023-9-15. 2023 (cit. on p. 121).

[40] Arthur P. Dempster and Robert M. Kleyle. "Distributions Determined by Cutting a Simplex with Hyperplanes". In: *The Annals of Mathematical Statistics* 39.5 (1968), pp. 1473–1478. URL: https://doi.org/10.1214/aoms/1177698126 (cit. on p. 74).

[41] Josef Dick. "Higher Order Scrambled Digital Nets Achieve The Optimal Rate of the Root Mean Square Error for Smooth Integrands". In: *The Annals of Statistics* 39.3 (2011), pp. 1372–1398. URL: http://www.jstor.org/stable/23033601 (cit. on p. 40).

[42] Laurent Dinh et al. "Sharp Minima Can Generalize For Deep Nets". In: *Proceedings of the 34th International Conference on Machine Learning*. Proceedings of Machine Learning Research 70 (Aug. 2017). Ed. by Doina Precup and Yee Whye Teh, pp. 1019–1028. URL: https://proceedings.mlr.press/v70/dinh17b.html (cit. on p. 87).

[43] Morganna Carmem Diniz, Edmundo de Souza e Silva, and H. Richard Gail. "Calculating the Distribution of a Linear Combination of Uniform Order Statistics". In: *INFORMS Journal on Computing* 14.2 (2002), pp. 124–131. URL: https://doi.org/10.1287/ijoc.14.2.124.121 (cit. on p. 74).

[44] Ann-Kathrin Dombrowski et al. "Explanations can be manipulated and geometry is to blame". In: *Advances in Neural Information Processing Systems* 32 (2019). Ed. by H. Wallach et al. URL: https://dl.acm.org/doi/10.5555/3454287.3455504 (cit. on p. 88).

[45] Ann-Kathrin Dombrowski et al. "Towards robust explanations for deep neural networks". In: *Pattern Recognition* 121 (2022), p. 108194. URL: https://doi.org/10.1016/j.patcog.2021.108194 (cit. on pp. 83, 85, 88, 90, 96).

[46] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations* (2021). URL: https://openreview.net/forum?id=YicbFdNTTy (cit. on pp. 50, 52).

[47] Harris Drucker and Yann Le Cun. "Improving generalization performance using double backpropagation". In: *IEEE Transactions on Neural Networks* 3.6 (1992), pp. 991–997. URL: http://dx.doi.org/10.1109/72.165600 (cit. on p. 96).

[48] Krishnamurthy Dvijotham et al. "A Dual Approach to Scalable Verification of Deep Networks". In: *Conference on Uncertainty in Artificial Intelligence* (2018). URL: https://api.semanticscholar.org/CorpusID:3972365 (cit. on p. 3).

[49] Tom Everitt et al. "Understanding Agent Incentives using Causal Influence Diagrams. Part I: Single Action Settings". In: *arXiv e-prints* (2019). URL: http://arxiv.org/abs/1902.09980 (cit. on p. 22).

[50] Benjamin Eysenbach et al. "Leave no trace: Learning to reset for safe and autonomous reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2018). URL: https://openreview.net/forum?id=S1vuO-bCW (cit. on p. 14).

[51] Pierre Foret et al. “Sharpness-aware Minimization for Efficiently Improving Generalization”. In: (2021). URL: https://openreview.net/forum?id=6Tm1mposlrM (cit. on p. 87).

[52] Robert Freund and James Orlin. “On the complexity of four polyhedral set containment problems”. In: *Mathematical Programming* 33 (Nov. 1985). URL: https://doi.org/10.1007/BF01582241 (cit. on p. 132).

[53] Shmuel Friedland and Lek-Heng Lim. “Nuclear norm of higher-order tensors”. In: *Mathematics of Computation* 87 (2018), pp. 1255–1281. DOI: https://doi.org/10.1090/mcom/3239 (cit. on p. 102).

[54] Komei Fukuda. *Lecture: Polyhedral Computation, Spring 2015*. Tech. rep. ETH Zürich, 2015. URL: https://people.inf.ethz.ch/fukudak/lect/pclect/notes2015/PolyComp2015.pdf (cit. on pp. 124, 125, 127, 128).

[55] Komei Fukuda and Alain Prodon. “Double description method revisited”. In: *Combinatorics and Computer Science* (1996). Ed. by Michel Deza, Reinhardt Euler, and Ioannis Manoussakis, pp. 91–111. URL: https://doi.org/10.1007/3-540-61576-8_77 (cit. on pp. 125, 135).

[56] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological Cybernetics* 36 (1980), pp. 193–202. URL: https://doi.org/10.1007/BF00344251 (cit. on p. 50).

[57] Javier Garcıa and Fernando Fernández. “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480. URL: https://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf (cit. on p. 12).

[58] Xavier Gastaldi. “Shake-Shake regularization”. In: *ICLR 2017 workshop* (2017). URL: https://arxiv.org/abs/1705.07485 (cit. on p. 5).

[59] Timon Gehr et al. “Ai^2: Safety and robustness certification of neural networks with abstract interpretation”. In: *2018 IEEE Symposium on Security and Privacy (SP)* (2018). URL: https://ieeexplore.ieee.org/document/8418593 (cit. on p. 37).

[60] Paul Gewirtz. “On "I Know It When I See It"”. In: *The Yale Law Journal* 105.4 (1996), pp. 1023–1047. URL: http://www.jstor.org/stable/797245 (cit. on p. 25).

[61] Amirata Ghorbani, Abubakar Abid, and James Zou. “Interpretation of neural networks is fragile”. In: *Proceedings of the AAAI conference on artificial intelligence* 33.01 (2019), pp. 3681–3688. URL: https://doi.org/10.1609/aaai.v33i01.33013681 (cit. on pp. 87, 94).

[62] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. "An Investigation into Neural Net Optimization via Hessian Eigenvalue Density". In: *ICML*. Proceedings of Machine Learning Research 97 (June 2019). Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov, pp. 2232–2241. URL: https://proceedings.mlr.press/v97/ghorbani19b.html (cit. on pp. 87, 92).

[63] Leilani H Gilpin et al. "Explaining explanations: An overview of interpretability of machine learning". In: *IEEE 5th International Conference on data science and advanced analytics (DSAA)* (2018), pp. 80–89. URL: https://arxiv.org/abs/1806.00069 (cit. on p. 23).

[64] Ian Goodfellow et al. "Maxout Networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Proceedings of Machine Learning Research 28.3 (June 2013). Ed. by Sanjoy Dasgupta and David McAllester, pp. 1319–1327. URL: http://proceedings.mlr.press/v28/goodfellow13.html (cit. on p. 51).

[65] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations* (Dec. 2015). URL: http://arxiv.org/abs/1412.6572 (cit. on p. 87).

[66] Henry Gouk et al. "Regularisation of neural networks by enforcing Lipschitz continuity". In: *Machine Learning* 110.2 (2021), pp. 393–416. URL: https://doi.org/10.1007/s10994-020-05929-w (cit. on p. 92).

[67] Will Grathwohl et al. "Scalable Reversible Generative Models with Free-form Continuous Dynamics". In: *International Conference on Learning Representations* (2019). URL: https://openreview.net/forum?id=rJxgknCcK7 (cit. on p. 91).

[68] Philipp Grüning and Erhardt Barth. "Bio-inspired Min-Nets Improve the Performance and Robustness of Deep Networks". In: *SVRHM 2021 Workshop @ NeurIPS* (Jan. 2022). URL: https://arxiv.org/abs/2201.02149 (cit. on p. 52).

[69] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com (cit. on pp. 47, 136).

[70] Dylan Hadfield-Menell et al. "The off-switch game". In: *International Joint Conference on Artificial Intelligence* (2017). URL: https://arxiv.org/abs/1611.08219 (cit. on p. 17).

[71] Boris Hanin and David Rolnick. "Complexity of Linear Regions in Deep Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Proceedings of Machine Learning Research 97 (June 2019). Ed. by Ka-

malika Chaudhuri and Ruslan Salakhutdinov, pp. 2596–2604. URL: https://proceedings.mlr.press/v97/hanin19a.html (cit. on p. 51).

[72] Boris Hanin and David Rolnick. "Deep ReLU Networks Have Surprisingly Few Activation Patterns". In: *International Conference on Neural Information Processing Systems* (Jan. 2019). URL: https://dl.acm.org/doi/abs/10.5555/3454287.3454320 (cit. on p. 35).

[73] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778. URL: https://doi.org/10.1109/CVPR.2016.90 (cit. on pp. 5, 49, 95).

[74] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf (cit. on p. 84).

[75] Matthias Hein and Maksym Andriushchenko. "Formal guarantees on the robustness of a classifier against adversarial manipulation". In: *Advances in neural information processing systems* 30 (2017). URL: https://dl.acm.org/doi/10.5555/3294771.3294987 (cit. on pp. 85, 87, 94).

[76] Christoph Hertrich et al. "Towards Lower Bounds on the Depth of ReLU Neural Networks". In: *Advances in Neural Information Processing Systems* 34 (2021). Ed. by M. Ranzato et al., pp. 3336–3348. URL: https://epubs.siam.org/doi/10.1137/22M1489332 (cit. on pp. 52, 54).

[77] Edmund Hlawka. "Funktionen von beschränkter Variatiou in der Theorie der Gleichverteilung". In: *Annali di Matematica Pura ed Applicata* 54 (1 1961), pp. 325–333. URL: https://doi.org/10.1007/BF02415361 (cit. on p. 40).

[78] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. URL: https://doi.org/10.1016/0893-6080(91)90009-T (cit. on p. 52).

[79] Andrew Howard et al. "Searching for MobileNetV3". In: *IEEE/CVF International Conference on Computer Vision* (2019). URL: https://doi.org/10.1109/ICCV.2019.00140 (cit. on p. 49).

[80] Xiaowei Huang et al. "Safety verification of deep neural networks". In: *International Conference on Computer Aided Verification* (2017), pp. 3–29. URL: https://link.springer.com/chapter/10.1007/978-3-319-63387-9_1 (cit. on p. 23).

[81] Christiani Hugeni. *Libellus de Ratiociniis in Ludo Aleae or The Value of all Chances in Games of Fortune*. S. Keimer for T. Woodward, 1657. URL: https://math.dartmouth.edu/~doyle/docs/huygens/huygens.pdf (cit. on p. 8).

[82] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. URL: https://doi.org/10.1109/MCSE.2007.55 (cit. on p. 135).

[83] Stephen Joe and Frances Y. Kuo. "Constructing Sobol Sequences with Better Two-Dimensional Projections". In: *SIAM Journal on Scientific Computing* 30.5 (2008), pp. 2635–2654. URL: https://doi.org/10.1137/070709359 (cit. on p. 40).

[84] Kyle D. Julian and Mykel J. Kochenderfer. "Guaranteeing safety for neural network-based aircraft collision avoidance systems". In: *IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)* (2019). URL: https://doi.org/10.1109/DASC43569.2019.9081748 (cit. on pp. 32–35, 44–46).

[85] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. "Deep Neural Network Compression for Aircraft Collision Avoidance Systems". In: *Journal of Guidance, Control, and Dynamics* 42.3 (2019), pp. 598–608. URL: https://doi.org/10.2514/1.G003724 (cit. on pp. 45, 46).

[86] Can Kanbak, Seyed-Moosavi Moosavi-Dezfooli, and Pascal Frossard. "Geometric Robustness of Deep Networks: Analysis and Improvement". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). URL: http://dx.doi.org/10.1109/cvpr.2018.00467 (cit. on p. 85).

[87] Guy Katz et al. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification* (2017). Ed. by Rupak Majumdar and Viktor Kunčak. URL: https://doi.org/10.1007/978-3-319-63387-9_5 (cit. on pp. 4, 32, 34, 35, 45, 107).

[88] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: (2017). URL: https://openreview.net/forum?id=H1oyRlYgg (cit. on p. 86).

[89] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv e-prints* (Dec. 2014). URL: http://arxiv.org/abs/1412.6980 (cit. on p. 43).

[90] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. "Robust Airborne Collision Avoidance through Dynamic Programming". In: *Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371* (2011). URL: https://www.ll.mit.edu/sites/default/files/publication/doc/2018-12/Kochenderfer_2011_ATC-371_WW-21458.pdf (cit. on pp. 32, 44, 45).

[91] Alexander Kott and Ethan Stump. "Intelligent Autonomous Things on the Battlefield". In: *Artificial Intelligence for the Internet of Everything* (Feb. 2019). URL: https://doi.org/10.1016/B978-0-12-817636-8.00003-X (cit. on p. 120).

[92] Victoria Krakovna et al. "Avoiding Side Effects By Considering Future Tasks". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2020). URL: https://dl.acm.org/doi/10.5555/3495724.3497324 (cit. on pp. 14, 15, 17, 19).

[93] Victoria Krakovna et al. "Penalizing side effects using stepwise relative reachability". In: *Workshop on Artificial Intelligence Safety at International Joint Conference on Artificial Intelligence* (2019). URL: https://arxiv.org/abs/1806.01186 (cit. on pp. 6, 12, 14, 16, 21).

[94] Victoria Krakovna et al. *Specification gaming: the flip side of AI ingenuity*. URL https://deepmind.google/discover/blog/specification-gaming-the-flip-side-of-ai-ingenuity/. 2020 (cit. on p. 12).

[95] Holger Krekel et al. *pytest*. 2004. URL: https://github.com/pytest-dev/pytest (cit. on p. 136).

[96] Alex Krizhevsky. "Convolutional Deep Belief Networks on CIFAR-10". In: (May 2012). URL: https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf (cit. on p. 5).

[97] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (cit. on p. 95).

[98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. URL: https://doi.org/10.1145/3065386 (cit. on pp. 49, 50).

[99] Stanford Vision Lab. *Large Scale Visual Recognition Challenge 2012 Leaderboard*. https://image-net.org/challenges/LSVRC/2012/results.html. 2012 (cit. on p. 51).

[100] Jan Leike et al. "Scalable agent alignment via reward modeling: a research direction". In: *arXiv e-prints* (2018). URL: http://arxiv.org/abs/1811.07871 (cit. on p. 22).

[101] Zongyi Li et al. "Fourier Neural Operator for Parametric Partial Differential Equations". In: *arXiv* (2021). URL: https://arxiv.org/abs/2010.08895 (cit. on p. 5).

[102] David Lindner, Kyle Matoba, and Alexander Meulemans. "Challenges for Using Impact Regularizers to Avoid Negative Side Effects". In: *SafeAI 2021 - AAAI's Workshop on Artificial Intelligence Safety* (2021). URL: https://ceur-ws.org/Vol-2808/Paper_20.pdf (cit. on p. 11).

[103] Andreas Loukas, Marinos Poiitis, and Stefanie Jegelka. "What training reveals about neural network complexity". In: 34 (2021). Ed. by M. Ranzato et al., pp. 494–508. URL: https://openreview.net/forum?id=RcjW7p7z8aJ (cit. on p. 88).

[104] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing* (2013). URL: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf (cit. on pp. 5, 83).

[105] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=rJzIBfZAb (cit. on pp. 6, 87, 96).

[106] Sébastien Marcel and Yann Rodriguez. "Torchvision the Machine-Vision Package of Torch". In: *Proceedings of the 18th ACM International Conference on Multimedia*. MM '10 (2010), pp. 1485–1488. URL: https://doi.org/10.1145/1873951.1874254 (cit. on pp. 49, 135).

[107] Kyle Matoba. "Deep neural network robustness evaluation with low discrepancy sequences". In: *Working paper* (2023) (cit. on p. 25).

[108] Kyle Matoba, Nikolaos Dimitriadis, and François Fleuret. "Benefits of Max Pooling in Neural Networks: Theoretical and Experimental Evidence". In: *Transactions on Machine Learning Research* (2023). URL: https://openreview.net/forum?id=YgeXqrH7gA (cit. on p. 49).

[109] Kyle Matoba and François Fleuret. "Exact Preimages of Neural Network Aircraft Collision Avoidance Systems". In: *Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems 2020* (Nov. 2020). URL: https://ml4eng.github.io/camera_readys/24.pdf (cit. on p. 25).

[110] Tadashi Matsunawa. "The exact and approximate distributions of linear combinations of selected order statistics from a uniform distribution". In: *Annals of the Institute of Statistical Mathematics* 37 (1 Dec. 1985), pp. 1572–9052. URL: https://doi.org/10.1007/BF02481076 (cit. on p. 74).

[111] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference* (2010). Ed. by Stéfan van der Walt and Jarrod Millman, pp. 56–61. URL: https://doi.org/10.25080/Majora-92bf1922-00a (cit. on p. 135).

[112] Takeru Miyato et al. “Spectral Normalization for Generative Adversarial Networks”. In: *ICLR* (Feb. 2018). URL: https://openreview.net/forum?id=B1QRgziT- (cit. on pp. 5, 88, 92).

[113] Guido Montúfar, Yue Ren, and Leon Zhang. “Sharp Bounds for the Number of Regions of Maxout Networks and Vertices of Minkowski Sums”. In: *SIAM Journal on Applied Algebra and Geometry* 6.4 (2022), pp. 618–649. URL: https://doi.org/10.1137/21M1413699 (cit. on p. 51).

[114] Guido F Montúfar et al. “On the Number of Linear Regions of Deep Neural Networks”. In: *Advances in Neural Information Processing Systems* 27 (2014). Ed. by Z. Ghahramani et al. URL: https://dl.acm.org/doi/10.5555/2969033.2969153 (cit. on pp. 51, 61).

[115] Seyed-Mohsen Moosavi-Dezfooli et al. “Robustness via Curvature Regularization, and Vice Versa”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 9070–9078. URL: http://dx.doi.org/10.1109/CVPR.2019.00929 (cit. on pp. 6, 85, 87–89, 95, 96).

[116] Anirbit Mukherjee and Amitabh Basu. “Lower bounds over Boolean inputs for deep neural networks with ReLU gates”. In: *CoRR* abs/1711.03073 (2017). arXiv: 1711.03073. URL: http://arxiv.org/abs/1711.03073 (cit. on p. 54).

[117] Yuval Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011* (2011). URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf (cit. on p. 6).

[118] Nvidia Corporation. *Form 8-K: Item 8.01, Other Events*. URL https://www.sec.gov/Archives/edgar/data/1045810/000104581022000146/nvda-20220826.htm. Accessed 2023-10-01 (cit. on p. 120).

[119] Art B. Owen. “Multidimensional Variation for Quasi-Monte Carlo”. In: *Contemporary Multivariate Analysis and Design of Experiments* (2005), pp. 49–74. URL: https://doi.org/10.1142/9789812567765_0004 (cit. on p. 39).

[120] Nicolas Papernot et al. “Technical Report on the CleverHans v2.1.0 Adversarial Examples Library”. In: *arXiv e-prints* (2018). URL: https://arxiv.org/abs/1610.00768 (cit. on pp. 98, 135).

[121] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32* (2019), pp. 8024–8035. URL: https://dl.acm.org/doi/10.5555/3454287.3455008 (cit. on pp. 95, 135).

[122] Florian Pausinger and Anne Marie Svane. “A Koksma-Hlawka Inequality for General Discrepancy Systems”. In: *Journal of Complex.* 31.6 (Dec. 2015), pp. 773–797. ISSN: 0885-064X. URL: https://doi.org/10.1016/j.jco.2015.06.002 (cit. on p. 39).

[123] Judea Pearl. *Causality*. Cambridge university press, 2009 (cit. on p. 22).

[124] Moritz Wemheuer Philip Kime and Philipp Lehman. *The biblatex package: Programmable Bibliographies and Citations*. Tech. rep. Version 3.19. 2023. URL: https://mirrors.ibiblio.org/CTAN/macros/latex/contrib/biblatex/doc/biblatex.pdf (cit. on p. 135).

[125] Adrian P. Pope et al. “Hierarchical Reinforcement Learning for Air Combat At DARPA’s AlphaDogfight Trials”. In: *IEEE Transactions on Artificial Intelligence* (2022), pp. 1–15. URL: https://doi.org/10.1109/TAI.2022.3222143 (cit. on p. 118).

[126] Chongli Qin et al. “Adversarial robustness through local linearization”. In: *Advances in Neural Information Processing Systems* 32 (2019). URL: https://dl.acm.org/doi/10.5555/3454287.3455527 (cit. on pp. 85, 87).

[127] Nasim Rahaman et al. “Learning the Arrow of Time for Problems in Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR)* (2020). URL: https://openreview.net/forum?id=rylJkpEtwS (cit. on p. 14).

[128] Aditya Ramesh et al. “Hierarchical text-conditional image generation with clip latents”. In: *arXiv e-prints* (2022). URL: https://arxiv.org/abs/2204.06125 (cit. on p. 84).

[129] Alex Ray, Joshua Achiam, and Dario Amodei. “Benchmarking safe exploration in deep reinforcement learning”. In: (2019). URL: https://cdn.openai.com/safexp-short.pdf (cit. on p. 21).

[130] Maximiliam Riesenhuber and Tomaso Poggio. “Hierarchical models of object recognition in cortex”. In: *Nat. Neurosci.* 2.11 (1999), pp. 1019–1025. URL: https://doi.org/10.1038/14819 (cit. on p. 50).

[131] Andrew Slavin Ros and Finale Doshi-Velez. "Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18 (2018). URL: https://dl.acm.org/doi/10.5555/3504035.3504238 (cit. on p. 88).

[132] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. URL: https://doi.org/10.1007/s11263-015-0816-y (cit. on p. 50).

[133] Stuart Russell. *Human Compatible: AI and the Problem of Control*. First. Penguin, Nov. 2019 (cit. on p. 1).

[134] Ernest Ryu et al. "Plug-and-Play Methods Provably Converge with Properly Trained Denoisers". In: *Proceedings of the 36th International Conference on Machine Learning*. Proceedings of Machine Learning Research 97 (June 2019). Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov, pp. 5546–5557. URL: https://proceedings.mlr.press/v97/ryu19a.html (cit. on pp. 88, 92).

[135] Sandhya Saisubramanian, Ece Kamar, and Shlomo Zilberstein. "A Multi-Objective Approach to Mitigate Negative Side Effects". In: *International Joint Conference on Artificial Intelligence* (2020). URL: https://doi.org/10.24963/ijcai.2020/50 (cit. on p. 15).

[136] Sandhya Saisubramanian, Shlomo Zilberstein, and Ece Kamar. "Avoiding negative side effects due to incomplete knowledge of AI systems". In: *arXiv e-prints* (2020). URL: https://doi.org/10.1609/aaai.12028 (cit. on p. 15).

[137] Atsuko Sano. *Coding will be mandatory in Japan's primary schools from 2020*. https://asia.nikkei.com/Economy/Coding-will-be-mandatory-in-Japan-s-primary-schools-from-2020. 2012 (cit. on p. 114).

[138] William Saunders et al. "Trial without Error: Towards Safe Reinforcement Learning via Human Intervention". In: *Proceedings of International Conference on Autonomous Agents and MultiAgent Systems* (2018). URL: https://dl.acm.org/doi/10.5555/3237383.3238074 (cit. on p. 12).

[139] Kevin Scaman and Aladin Virmaux. "Lipschitz Regularity of Deep Neural Networks: Analysis and Efficient Estimation". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18 (2018), pp. 3839–3848. URL: https://dl.acm.org/doi/10.5555/3327144.3327299 (cit. on p. 104).

[140] Paul Scharre. *Army of None: Autonomous Weapons and the Future of War*. W. W. Norton and Company, Apr. 2018 (cit. on p. 120).

[141] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. "Bounding and Counting Linear Regions of Deep Neural Networks". In: *Proceedings of the 35th International Conference on Machine Learning*. Proceedings of Machine Learning Research 80 (July 2018). Ed. by Jennifer Dy and Andreas Krause, pp. 4558–4566. URL: http://proceedings.mlr.press/v80/serra18b.html (cit. on pp. 35, 51).

[142] Rohin Shah et al. "Preferences Implicit in the State of the World". In: *International Conference on Learning Representations (ICLR)* (2019). URL: https://openreview.net/forum?id=rkevMnRqYQ (cit. on pp. 12, 14, 19).

[143] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations* (2015). Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1409.1556 (cit. on p. 49).

[144] Sahil Singla and Soheil Feizi. "Fantastic Four: Differentiable Bounds on Singular Values of Convolution Layers". In: *arXiv e-prints* (2019). URL: http://arxiv.org/abs/1911.10258 (cit. on p. 102).

[145] Sahil Singla and Soheil Feizi. "Second-Order Provable Defenses against Adversarial Attacks". In: *Proceedings of the 37th International Conference on Machine Learning*. Proceedings of Machine Learning Research 119 (July 2020). Ed. by Hal Daumé III and Aarti Singh, pp. 8981–8991. URL: https://proceedings.mlr.press/v119/singla20a.html (cit. on p. 90).

[146] Justin A. Sirignano. "Deep learning for limit order books". In: *Quantitative Finance* 19.4 (2019), pp. 549–570. URL: https://doi.org/10.1080/14697688.2018.1546053 (cit. on p. 119).

[147] Jost Tobias Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings* (2015). Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1412.6806 (cit. on pp. 5, 49, 51).

[148] Suraj Srinivas and François Fleuret. "Rethinking the Role of Gradient-based Attribution Methods for Model Interpretability". In: *International Conference on Learning Representations* (2021). URL: https://openreview.net/forum?id=dYeAHXnpWJ4 (cit. on p. 87).

[149] Suraj Srinivas et al. “Efficient Training of Low-Curvature Neural Networks”. In: *Advances in Neural Information Processing Systems* (2022). Ed. by Alice H. Oh et al. URL: https://openreview.net/forum?id=2B2xIJ299rx (cit. on pp. 6, 83).

[150] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 5).

[151] statista. *Number of Amazon.com employees from 2007 to 2022*. URL https://www.statista.com/statistics/234488/number-of-amazon-employees/. Accessed 2023-10-01. 2023 (cit. on p. 121).

[152] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *International Conference on Learning Representations* (2014). URL: http://arxiv.org/abs/1312.6199 (cit. on pp. 26, 87).

[153] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2016). URL: https://doi.org/10.1109/CVPR.2016.308 (cit. on p. 49).

[154] Jie Tan et al. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots”. In: *arXiv e-prints* (Apr. 2018). URL: https://www.roboticsproceedings.org/rss14/p10.pdf (cit. on p. 118).

[155] Till Tantau. *The TikZ and PGF Packages*. 2023. URL: https://pgf-tikz.github.io/pgf/pgfmanual.pdf (cit. on p. 136).

[156] Matus Telgarsky. “Benefits of depth in neural networks”. In: *29th Annual Conference on Learning Theory* (2016). URL: https://proceedings.mlr.press/v49/telgarsky16.html (cit. on p. 53).

[157] Edward O. Thorp. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. Blaisdell Publishing Company, 1962 (cit. on p. 9).

[158] Vincent Tjeng, Kai Xiao, and Russ Tedrake. “Evaluating Robustness of Neural Networks with Mixed Integer Programming”. In: *International Conference on Learning Representations* (Nov. 2019). URL: https://openreview.net/forum?id=HyGIdiRqtm (cit. on pp. 26, 35).

[159] Asher Trockman and J. Zico Kolter. “Orthogonalizing Convolutional Layers with the Cayley Transform”. In: *International Conference on Learning Representations* (2021). URL: https://openreview.net/forum?id=Pbj8H_jEHYv (cit. on p. 88).

[160] Hanna Tseran and Guido F Montúfar. “On the Expected Complexity of Maxout Networks”. In: *Advances in Neural Information Processing Systems* 34 (2021). Ed. by M. Ranzato et al., pp. 28995–29008. URL: https://openreview.net/forum?id=9dZ4oIjkv76 (cit. on pp. 51, 59, 61).

[161] Alexander Matt Turner, Dylan Hadfield-Menell, and Prasad Tadepalli. "Conservative Agency via Attainable Utility Preservation". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. AIES '20 (2020), pp. 385–391. URL: https://doi.org/10.1145/3375627.3375851 (cit. on pp. 6, 12, 14–16, 20, 21).

[162] Alexander Matt Turner, Neale Ratzlaff, and Prasad Tadepalli. "Avoiding Side Effects in Complex Environments". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20 (2020). URL: https://dl.acm.org/doi/10.5555/3495724.3497521 (cit. on p. 16).

[163] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995 (cit. on p. 135).

[164] Moshe Y. Vardi. "Artificial Intelligence: Past and Future". In: *Communications of the ACM* 55.1 (Jan. 2012), p. 5. URL: https://doi.org/10.1145/2063176.2063177 (cit. on p. 115).

[165] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems* 30 (2017). Ed. by I. Guyon et al. URL: https://dl.acm.org/doi/10.5555/3295222.3295349 (cit. on p. 51).

[166] Aladin Virmaux and Kevin Scaman. "Lipschitz regularity of deep neural networks: analysis and efficient estimation". In: *Advances in Neural Information Processing Systems* 31 (2018). Ed. by S. Bengio et al. URL: https://dl.acm.org/doi/10.5555/3327144.3327299 (cit. on pp. 37, 38).

[167] Miaoyan Wang et al. "Operator norm inequalities between tensor unfoldings on the partition lattice". In: *Linear Algebra and its Applications* 520 (2017), pp. 44–66. URL: https://doi.org/10.1016/j.laa.2017.01.017 (cit. on pp. 90, 101, 102, 104).

[168] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. URL: https://doi.org/10.21105/joss.03021 (cit. on p. 135).

[169] Herbert Weisberg. "The Distribution of Linear Combinations of Order Statistics from the Uniform Distribution". In: *The Annals of Mathematical Statistics* 42.2 (1971), pp. 704–709. URL: http://www.jstor.org/stable/2239815 (cit. on p. 73).

[170] Alfred North Whitehead. *An Introduction To Mathematics*. First. Williams and Norgate, 1911. URL: https://www.gutenberg.org/files/41568/41568-pdf.pdf (cit. on pp. 1, 123).

[171] Eric Wong and Zico Kolter. “Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope”. In: *Proceedings of the 35th International Conference on Machine Learning*. Proceedings of Machine Learning Research 80 (July 2018). Ed. by Jennifer Dy and Andreas Krause, pp. 5286–5295. URL: https://proceedings.mlr.press/v80/wong18a.html (cit. on pp. 26, 37).

[172] Omry Yadan. *Hydra - A framework for elegantly configuring complex applications*. Github. 2019. URL: https://github.com/facebookresearch/hydra (cit. on p. 135).

[173] Xiaodong Yang et al. “Reachability Analysis for Feed-Forward Neural Networks using Face Lattices”. In: *arXiv e-prints* (2020). URL: https://arxiv.org/abs/2003.01226 (cit. on p. 26).

[174] Huan Zhang et al. “Efficient Neural Network Robustness Certification with General Activation Functions”. In: *Advances in Neural Information Processing Systems* (2018). Ed. by S. Bengio et al., pp. 4939–4948. URL: https://dl.acm.org/doi/abs/10.5555/3327345.3327402 (cit. on pp. 5, 37, 38, 51).

[175] Shun Zhang, Edmund Durfee, and Satinder Singh. “Querying to Find a Safe Policy under Uncertain Safety Constraints in Markov Decision Processes”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03 (Apr. 2020), pp. 2552–2559. URL: https://doi.org/10.1609/aaai.v34i03.5638 (cit. on p. 14).

[176] Shun Zhang, Edmund H. Durfee, and Satinder Singh. “Minimax-Regret Querying on Side Effects for Safe Optimality in Factored Markov Decision Processes”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI’18 (2018), pp. 4867–4873. URL: https://dl.acm.org/doi/10.5555/3304652.3304685 (cit. on pp. 14, 23).

[177] Xinyang Zhang et al. “Interpretable deep learning under fire”. In: *29th USENIX Security Symposium (USENIX Security 20)* (2020). URL: https://dl.acm.org/doi/10.5555/3489212.3489306 (cit. on pp. 87, 94).

Kyle Matoba
Email: kylematoba@gmail.com Telephone: +41 078 215 5271

Education

EPFL 2019 - Present
PhD student in computer science (deep learning)
Stanford University 2010
MS in statistics (emphasis in probability and statistical learning)
University of California, Davis 2008
BS in statistics, BA in economics, minor in Japanese

Employment

GSA Capital Partners (London, UK) 2013 - 2019
Researching and running high Sharpe strategies in global equities
Researching and running scalable strategies across asset classes
Management of independent portfolio managers in the platform business
University of California (Los Angeles, USA) 2010 - 2013
Finance department: PhD student
Masters of Financial Engineering program: Instructor and teaching assistant
Institute for Pure and Applied Mathematics: Financial math mentor
REvolution Computing (Palo Alto, USA) 2010
Development consultant
Stanford University (Stanford, USA) 2009 - 2010
Department of Economics: Teaching assistant
Federal Reserve Bank of San Francisco (San Francisco, USA) 2008 - 2009
Macroeconomic research: Research associate
University of California (Davis, USA) 2007 - 2008
Department of Anthropology: Statistical programmer

Publications

Suraj Srinivas, Kyle Matoba, Himabindu Lakkaraju, and François Fleuret. Efficient training of low-curvature neural networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022

Kyle Matoba and François Fleuret. Exact preimages of neural network aircraft collision avoidance systems. In *Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems 2020*, November 2020

Kyle Matoba, Nikolaos Dimitriadis, and François Fleuret. Benefits of max pooling in neural networks: Theoretical and experimental evidence. *Transactions on Machine Learning Research*, 2023. Accepted for publication

David Lindner, Kyle Matoba, and Alexander Meulemans. Challenges for using impact regularizers to avoid negative side effects. In *SafeAI 2021 - AAAI's Workshop on Artificial Intelligence Safety*, 2021

Bruce I. Carlin, Francis A. Longstaff, and Kyle Matoba. Disagreement and asset prices. *Journal of Financial Economics*, 114(2):226 – 238, 2014

Makoto Matsumoto, Mutsuo Saito, and Kyle Matoba. A computable figure of merit for quasi-monte carlo point sets. *Mathematics of Computation*, 62(5):2201–2234, 2013

Ayşe İmrohoroğlu, Kyle Matoba, and Şelale Tüzel. Proposition 13: An equilibrium analysis. *American Economic Journal: Macroeconomics*, 10(2):24–51, April 2018