The Dissertation Committee for Christopher George Snyder

certifies that this is the approved version of the following dissertation:

# Deep Learning and Representation: Translating Deep Learning to Medicine

**Committee:**

Sriram Vishwanath, Supervisor

Constantine Caramanis

Mia Markey

Jonathan Valvano

# Deep Learning and Representation: Translating Deep Learning to Medicine

by

**Christopher George Snyder**

**Dissertation**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Doctor of Philosophy**

**The University of Texas at Austin**

May 2020

# Acknowledgments

I would like to recognize the invaluable support provided to bring this project to life.

First, I would like to thank my advisor, Dr. Sriram Vishwanath, for his support and encouragement over these years. I whole-heartedly appreciate his advice and guidance.

Dr. Constantine Caramanis, Dr. Mia Markey, and Dr. Jonathan Valvano - all part of my graduate committee and Their support and their review of my graduate work is sincerely appreciated.

The hard work and camaraderie put forth by those I have been able to work with – Murat Kocaoglu, Ajil Jalal, Jared Ucherek.

CHRISTOPHER SNYDER

*The University of Texas at Austin*

*May 2020*

**Abstract**

## Deep Learning and Representation: Translating Deep Learning to Medicine

Christopher George Snyder, Ph.D.

The University of Texas at Austin, 2020

Supervisor: Sriram Vishwanath

Deep learning is a powerful method using neural networks to learn functional representations that relate variables of interest. This paper examines the manner of representation of those variables by neural networks and of neural networks by humans. In the first section, we examine causal relations among variables with CausalGAN. The following section will explore a theoretical connection between neural networks and support vector machines (SVMs) representing neural network functions through a sample compression scheme. The third section reparamaterizes neural networks using *Min* and *Max* combinations of linear functions and examines the connection with generalization and interpretation. The final section explores applications of this method to ECG model interpretation.

# Table of Contents

# Chapter 1

# Introduction

The broadening use of Neural Networks requires strategies that provide transparency with regard to what and whether or not a network actually learned. This is particularly crucial for a range of applications including areas requiring a high level of confidence, as well as a low risk tolerance for providing inaccurate output determinations.

Most importantly, Neural Networks, in spite of their growing utilization, are incomprehensible functions. While they are capable of processing complex data, the functions behave the way you want them to but we do not know how they achieve that behavior

This research thus targets a foundational and urgent gap in deep learning: generating useful descriptions of neural network functions. New representations of both conceptual and analytical problem solving frameworks are introduced. Improved theoretical understanding of deep neural networks (DNNs) will aid in structured, principled approaches to the design, analysis and use of such networks. First, we develop a representation of (ReLU) deep neural networks through compression strings encoded by support vector-like sample subsets. With this, we develop measures of model capacity in bits as a promising framework for generalization. We also devise an interpretable hierarchical, equivalent of the DNN discrete

classification map through recursive AND/OR operations on latent variables, which in the first iteration are interpretable linear classifier inputs. In doing this, we develop an understanding of black box DNN models through latent variable interpretations. Finally, we demonstrate an application of these concepts using electrocardiogram data (ECGs) through improving DNN output data interpretability.

## 1.1 Representation of Causal Implicit Generative Models

The problem of how adversarial training can use a two stage procedure for learning where data labels are relevant was explored. Neural networks are powerful tools for leveraging statistical relationships in the data to model the dependents between input and output variables. However, in medicine, one is often interested in "what if" dependencies between variables that can predict effects of interventions. This requires causal, not statistical, modeling. In CausalGAN, we investigate representation and learning of the causal functional relations of a known causal directed graph, using neural network building blocks. On the CelebA dataset, we presuppose some reasonable causal relationships then show how the neural network predicts not only statistical, but also interventional distributions and sampling. To learn these distributions, we structure the neural network layers in a way according to the causal graph and introduce adversarial learning as a technique for adjusting the parameters of the network.

## 1.2 Representation with Sample Compression

This section provides research into an approach to Neural Learning Networks, which operate without dependency on weighting of input data. Instead, the NN is transformed into a related support vector machine (SVM) problem, then the network function is recovered using support

vectors. Support for just such an approach are discussed at Section 3.3. This approach seeks to make analogy between NN and the more intuitable SVM learning frameworks. The result of this analogy is an information theoretic learning bound based on sample compression with side information. Specifically, the bound relates to the number of samples which in some sense are support vectors combined with the number of bits of freedom defining their neuron activation. A key assumption explored experimentally is the degree to which these define the model itself, i.e. a max margin assumption.

## 1.3 Generalization through Representation as Logical Circuits

Here we explore the generalization and interpretation of deep logical circuits as neural networks. Models which avoid over-fitting find a simple representation of the output in terms of the input. The difficulty of generalization theory for NN is identifying when the neural network has learned a simple function. That is, the functions themselves are represented by millions of parameters across tens of layers, cannot readily be identified as simple, even when they are not too different from linear. This work reparameterizes NN or representation theory using components of linear functions. This representation is designed to leverage dependencies in neuron states across layers to make the expression more simple. The precise connections to generalization theory and interpretation are explored.

## 1.4 A

proper representation of a neural network model should also help us interpret what that model does. Nowhere is this more important than in the medical field. One important application of neural networks to clinical studies is in the automated diagnoses of electrocardiogram (ECG) signals. There the potential to impact the medical field lies not only in the ability to

automatically diagnose ECG data, but also in our ability to understand the origin of these diagnoses. We want to be able to understand, disagree with, and learn from our models. This requires an interface which is a human-friendly representation of the mathematical model.

In this work, we adapt the model from the prior chapter, exploring Deep Logical Circuits, to ECG waveform classification. This adaptation seeks to represent the NN as acting through certain principle moods whose mechanism can be understood through the representation of corresponding wave forms from which they act. We observe that each of these modes is responsible for different types of waveforms, as if the model applies different rules for deciding diagnoses of ECGs of different character and kind. We present a composite figure representing the coarse scale representation of these rules for one particular example.

# Chapter 2

# CausalGAN: Learning Causal Implicit Generative Models with Adversarial Training [1]

We introduce causal implicit generative models (CiGMs): models that allow sampling from not only the true observational but also the true interventional distributions. We show that adversarial training can be used to learn a CiGM, if the generator architecture is structured based on a given causal graph. We consider the application of conditional and interventional sampling of face images with binary feature labels, such as *mustache, young*. We preserve the dependency structure between the labels with a given causal graph. We devise a two-stage procedure for learning a CiGM over the labels and the image. First we train a CiGM over

[1] **This chapter is based on material from the publication "CausalGAN: Learning Causal Implicit Generative Models with Adversarial Training" published as proceedings of the International Conference on Learning Representations (ICLR), May 2018 (Kocaoglu et al. [2018]). The author of this dissertation is an equal first author. His contributions were toward the research problem's conception, theoretical developments, and experimental validation.**

the binary labels using a Wasserstein GAN where the generator neural network is consistent with the causal graph between the labels. Later, we combine this with a conditional GAN to generate images conditioned on the binary labels. We propose two new conditional GAN architectures: CausalGAN and CausalBEGAN. We show that the optimal generator of the CausalGAN, given the labels, samples from the image distributions conditioned on these labels. The conditional GAN combined with a trained CiGM for the labels is then a CiGM over the labels and the generated image. We show that the proposed architectures can be used to sample from observational and interventional image distributions, even for interventions which do not naturally occur in the dataset.

## 2.1 Introduction

An implicit generative model (Mohamed and Lakshminarayanan [2016]) is a mechanism that can sample from a probability distribution without an explicit parameterization of the likelihood. Generative adversarial networks (GANs) arguably provide one of the most successful ways to train implicit generative models. GANs are neural generative models that can be trained using backpropagation to sample from very high dimensional nonparametric distributions (Goodfellow et al. [2014]). A *generator* network models the sampling process through feedforward computation given a noise vector. The generator output is constrained and refined through feedback by a competitive adversary network, called the discriminator, that attempts to distinguish between the generated and real samples. The objective of the generator is to maximize the loss of the discriminator (convince the discriminator that it outputs samples from the real data distribution). GANs have shown tremendous success in generating samples from distributions such as image and video (Vondrick et al. [2016]).

An extension of GANs is to enable sampling from the class conditional data distributions by feeding class labels to the generator alongside the noise vectors. Various neural

network architectures have been proposed for solving this problem (Mirza and Osindero [2016], Odena et al. [2017], Antipov et al. [2017]). However, these architectures do not capture the dependence between the labels. Therefore, they do not have a mechanism to sample images *given a subset of the labels*, since they cannot sample the remaining labels. In this paper, we are interested in extending the previous work on conditional image generation by *i)* capturing the dependence between labels and *ii)* capturing *the causal effect* between labels. We can think of conditional image generation as a causal process: Labels determine the image distribution. The generator is a non-deterministic mapping from labels to images. This is consistent with the causal graph *"Labels cause the Image"*, denoted by $L \rightarrow I$, where $L$ is the random vector for labels and $I$ is the image random variable. Using a finer model, we can also include the causal graph between the labels, if available.

As an example, consider the causal graph between *Gender* ($G$) and *Mustache* ($M$) labels. The causal relation is clearly *Gender causes Mustache*, denoted by the graph $G \rightarrow M$. Conditioning on *Gender = male*, we expect to see males with or without mustaches, based on the fraction of males with mustaches in the population. When we condition on *Mustache = 1*, we expect to sample from males only since the population does not contain females with mustaches. In addition to sampling from conditional distributions, causal models allow us to sample from various different distributions called *interventional distributions*. An intervention is an experiment that fixes the value of a variable in a causal graph. This affects the distributions of the descendants of the intervened variable in the graph. But unlike conditioning, it does not affect the distribution of its ancestors. For the same causal graph, intervening on *Mustache = 1* would not change the distribution of *Gender*. Accordingly, the label combination (*Gender = female, Mustache = 1*) would appear as often as *Gender = female* after the intervention. Please see Figure 2.1 for some of our conditional and interventional samples, which illustrate this concept on the *Bald* and *Mustache* variables.

(a) Top: Intervened on Bald=1. Bottom: Conditioned on Bald = 1. $Male \rightarrow Bald$.

(b) Top: Intervened on Mustache=1. Bottom: Conditioned on Mustache = 1. $Male \rightarrow Mustache$.

Figure 2.1: Observational and interventional samples from CausalBEGAN. Our architecture can be used to sample not only from the joint distribution (conditioned on a label) but also from the interventional distribution, e.g., under the intervention do$(Mustache = 1)$. The two distributions are clearly different since $\mathbb{P}(Male = 1|Mustache = 1) = 1$ and $\mathbb{P}(Bald = 1|Male = 0) = 0$ in the data distribution $\mathbb{P}$.

In this work we propose *causal implicit* generative models (CiGM): mechanisms that can sample not only from the correct joint probability distributions but also from the correct *conditional* and *interventional* probability distributions. Our objective is not to learn the causal graph: we assume that the true causal graph is given to us. We show that when the generator structure inherits its neural connections from the causal graph, GANs can be used to train causal implicit generative models. We use Wasserstein GAN (WGAN) (Arjovsky et al. [2017]) to train a CiGM for binary image labels, as the first step of a two-step procedure for training a CiGM for the images and image labels. For the second step, we propose two novel conditional GANs called CausalGAN and CausalBEGAN. We show that the optimal generator of CausalGAN can sample from the true conditional distributions (see Theorem 1).

We show that combining CausalGAN with a CiGM on the labels yields a CiGM on the labels and the image, which is formalized in Corollary 1.1 in Section 2.5. Our contributions are as follows:

- We observe that adversarial training can be used after structuring the generator architecture based on the causal graph to train a CiGM. We empirically show that WGAN can be used to learn a CiGM that outputs *essentially discrete*[2] labels, creating a CiGM

[2]Each of the generated labels is sharply concentrated around 0 or 1 (Please see Figure A.4a in the Appendix).

for binary labels.

- We consider the problem of conditional and interventional sampling of images given a causal graph over binary labels. We propose a two-stage procedure to train a CiGM over the binary labels and the image. As part of this procedure, we propose a novel conditional GAN architecture and loss function. We show that the global optimal generator provably samples from the class conditional distributions.
- We propose a natural but nontrivial extension of BEGAN to accept labels: using the same motivations for margins as in BEGAN (Berthelot et al. [2017]), we arrive at a "margin of margins" term. We show empirically that this model, which we call CausalBEGAN, produces high quality images that capture the image labels.
- We evaluate our CiGM training framework on the labeled CelebA data (Liu et al. [2015]). We empirically show that CausalGAN and CausalBEGAN can produce label-consistent images *even for label combinations realized under interventions that never occur during training*, e.g., "woman with mustache"[3].

## 2.2 Related Work

Using a GAN conditioned on the image labels has been proposed before: In Mirza and Osindero [2016], authors propose conditional GAN (CGAN): They extend generative adversarial networks to the setting where there is extra information, such as labels. Image labels are given to both the generator and the discriminator. In Odena et al. [2017], authors propose ACGAN: Instead of receiving the labels as input, the discriminator is now tasked with estimating the label. In Sricharan et al. [2017], the authors compare the performance of CGAN and ACGAN and propose an extension to the semi-supervised setting. In Chen Xi Duan et al. [2016], authors propose a new architecture called InfoGAN, which attempts

[3]This observation is not supported by theory since the distribution over the labels is not strictly positive.

to maximize a variational lower bound of mutual information between the inputs given to the generator and the image. To the best of our knowledge, the existing conditional GANs do not allow sampling from label combinations that do not appear in the dataset (Sricharan [2017]).

BiGAN (Donahue et al. [2017]) and ALI (Dumoulin et al. [2017]) extend the standard GAN framework by also learning a mapping from the image space to a latent space. In CoGAN (Liu and Wang [2016]) the authors learn a joint distribution over an image and its binary label by enforcing weight sharing between generators and discriminators. SD-GAN (Donahue et al. [2018]) is a similar architecture which splits the latent space into "Identity" and "Observation" portions. To generate faces of the same person, one can then fix the identity portion of the latent code. If we consider the "Identity" and "Observation" codes to be the labels then SD-GAN can be seen as an extension of BEGAN to labels. This is, to the best of our knowledge, the only extension of BEGAN to accept labels before CausalBEGAN. It is not trivial to extend CoGAN and SD-GAN to more than two labels. Authors in Antipov et al. [2017] use CGAN of Mirza and Osindero [2016] with a one-hot encoded vector that encodes the age interval. A generator conditioned on this one-hot vector can then be used for changing the age attribute of a face image. Another application of generative models is in compressed sensing: Authors in Bora et al. [2017] give compressed sensing guarantees for recovering a vector, if the data lies close to the output of a trained generative model.

Using causal principles for deep learning and using deep learning techniques for causal inference has been recently gaining attention. In Lopez-Paz and Oquab [2016], the authors observe the connection between GAN layers, and structural equation models. Based on this observation, they use CGAN (Mirza and Osindero [2016]) to learn the causal direction between two variables from a dataset. In Lopez-Paz et al. [2016], the authors propose using a neural network in order to discover the causal relation between image class labels based on

static images. In Bahadori et al. [2017], authors propose a new regularization for training a neural network, which they call causal regularization, in order to assure that the model is predictive in a causal sense. In a very recent work Besserve et al. [2018], authors point out the connection of GANs to causal generative models. However they see image as a cause of the neural net weights, and do not use labels. In an independent parallel work, authors in Goudet et al. [2018] propose using neural networks for learning causal graphs. Similar to us, they also use neural connections to mimic structural equations, but for learning the causal graph.

## 2.3 Causality Background

In this section, we give a brief introduction to causality. Specifically, we use Pearl's framework (Pearl [2009]), i.e., structural causal models (SCMs), which uses structural equations and directed acyclic graphs between random variables to represent a causal model.

Consider two random variables $X, Y$. Within the SCM framework and under the causal sufficiency assumption[4], $X$ *causes* $Y$ means that there exists a function $f$ and some unobserved random variable $E$, independent from $X$, such that *the value of* $Y$ *is determined based on the values of* $X$ *and* $E$ *through the function* $f$, i.e., $Y = f(X, E)$. Unobserved variables are also called *exogenous*. The causal graph that represents this relation is $X \to Y$. In general, a causal graph is a directed acyclic graph implied by the structural equations: The parents of a node $X_i$ in the causal graph, shown by $Pa_i$, represent the *causes* of that variable. The causal graph can be constructed from the structural equations as follows: The parents of a variable are those that appear in the structural equation that determines the value of that variable.

Formally, a structural causal model is a tuple $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathbb{P}_{\mathcal{E}}(.))$ that contains a

[4]In a causally sufficient system, every unobserved variable affects not more than a single observed variable.

set of functions $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, a set of random variables $V = \{X_1, X_2, \ldots, X_n\}$, a set of exogenous random variables $\mathcal{E} = \{E_1, E_2, \ldots, E_n\}$, and a product probability distribution over the exogenous variables $\mathbb{P}_{\mathcal{E}}$. The set of observable variables $\mathcal{V}$ has a joint distribution implied by the distribution of $\mathcal{E}$, and the functional relations $\mathcal{F}$. The causal graph $D$ is then the directed acyclic graph on the nodes $\mathcal{V}$, such that a node $X_j$ is a parent of node $X_i$ if and only if $X_j$ is in the domain of $f_i$, i.e., $X_i = f_i(X_j, S, E_i)$, for some $S \subset V$. See the Appendix for more details.

An *intervention* is an operation that changes the underlying causal mechanism, hence the corresponding causal graph. An intervention on $X_i$ is denoted as $do(X_i = x_i)$. It is different from conditioning on $X_i$ in the following way: An intervention removes the connections of node $X_i$ to its parents, whereas conditioning does not change the causal graph from which data is sampled. The interpretation is that, for example, if we set the value of $X_i$ to 1, then it is no longer determined through the function $f_i(Pa_i, E_i)$. An intervention on a set of nodes is defined similarly. The joint distribution over the variables after an intervention (post-interventional distribution) can be calculated as follows: Since $D$ is a Bayesian network for the joint distribution, the observational distribution can be factorized as $\mathbb{P}(x_1, x_2, \ldots x_n) = \prod_{i \in [n]} \mathbb{P}(x_i | Pa_i)$, where the nodes in $Pa_i$ are assigned to the corresponding values in $\{x_i\}_{i \in [n]}$. After an intervention on a set of nodes $X_S \coloneqq \{X_i\}_{i \in S}$, i.e., $do(X_S = \mathbf{s})$, the post-interventional distribution is given by $\prod_{i \in [n] \setminus S} \mathbb{P}(x_i | Pa_i^S)$, where $Pa_i^S$ represents the following assignment: $X_j = x_j$ for $X_j \in Pa_i$ if $j \notin S$ and $X_j = \mathbf{s}(j)$ if $j \in S$[5].

In general it is not possible to identify the true causal graph for a set of variables without performing experiments or making additional assumptions. This is because there are multiple causal graphs that allow the same joint probability distribution even for two

[5]With slight abuse of notation, we use $\mathbf{s}(j)$ to represent the value assigned to variable $X_j$ by the intervention rather than the $j$th coordinate of $\mathbf{s}$.

variables (Spirtes et al. [2001]). This paper does not address the problem of learning the causal graph: We assume that the causal graph is given to us, and we learn a causal model, i.e., the functions comprising the structural equations for some choice of exogenous variables[6]. There is significant prior work on learning causal graphs that could be used before our method (Spirtes et al. [2001], Heckerman [1995], Chickering [2002], Hoyer et al. [2008], Hyttinen et al. [2013], Hauser and Bühlmann [2014], Shanmugam et al. [2015], Lopez-Paz et al. [2015], Peters et al. [2016], Etesami and Kiyavash [2016], Quinn et al. [2015], Kocaoglu et al. [2017a,b]). When the true causal graph is unknown using a Bayesian network that respects the conditional independences in the data allows us to sample from the correct observational distributions. We explore the effect of the used Bayesian network in Section A.1.10, A.1.11.

## 2.4 Causal Implicit Generative Models

Implicit generative models can sample from the data distribution. However they do not provide the functionality to sample from interventional distributions. We propose *causal implicit generative models*, which provide a way to sample from both observational and interventional distributions.

We show that generative adversarial networks can also be used for training causal implicit generative models. Consider the simple causal graph $X \rightarrow Z \leftarrow Y$. Under the causal sufficiency assumption, this model can be written as $X = f_X(E_X), Y = f_Y(E_Y), Z = f_Z(X, Y, E_Z)$, where $f_X, f_Y, f_Z$ are some functions and $E_X, E_Y, E_Z$ are jointly independent variables. The following simple observation is useful: *In the GAN training framework, generator neural network connections can be arranged to reflect the causal graph structure*.

[6]Even when the causal graph is given, there will be many different sets of functions and exogenous noise distributions that explain the observed joint distribution for that causal graph. We are learning one such model.

Please see Figure 2.2b for this architecture. The feedforward neural networks can be used to represent the functions $f_X, f_Y, f_Z$. The noise terms $(N_X, N_Y, N_Z)$ can be chosen as independent, complying with the condition that $(E_X, E_Y, E_Z)$ are jointly independent. Note that although we do not know the distributions of the exogenous variables, for a rich enough function class, we can use Gaussian distributed variables (Mooij et al. [2010]) $N_X, N_Y, N_Z$. Hence this feedforward neural network can be used to represents the causal models with graph $X \to Z \leftarrow Y$.

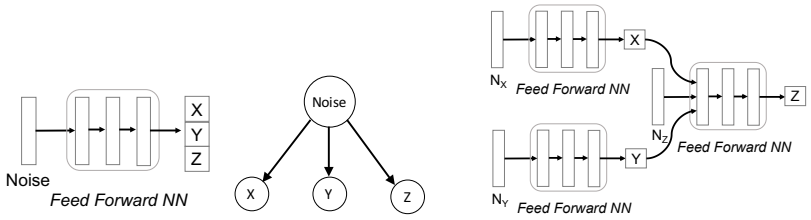
The following proposition is well known in the causality literature. It shows that given the true causal graph, two causal models that have the same observational distribution have the same interventional distributions for any intervention. $\mathbb{P}_V$ and $\mathbb{Q}_V$ stands for the distributions induced on the set of variables in $V$ by $\mathbb{P}_{N_1}$ and $\mathbb{Q}_{N_2}$, respectively.

**Proposition 1.** *Let $\mathcal{M}_1 = (D_1 = (V, E), N_1, \mathcal{F}_1, \mathbb{P}_{N_1}(.)), \mathcal{M}_2 = (D_2 = (V, E), N_2, \mathcal{F}_2, \mathbb{Q}_{N_2}(.))$ be two causal models, where $\mathbb{P}_{N_1}(.), \mathbb{Q}_{N_2}(.)$ are strictly positive densities. If $\mathbb{P}_V(.) = \mathbb{Q}_V(.)$, then $\mathbb{P}_V(.|do(S)) = \mathbb{Q}_V(.|do(S))$*

We have the following definition, which ties a feedforward neural network with a causal graph:

**Definition 1.** *Let $Z = \{Z_1, Z_2, \ldots, Z_m\}$ be a set of mutually independent random variables. A feedforward neural network $G$ that outputs the vector $G(Z) = [G_1(Z), G_2(Z), \ldots, G_n(Z)]$ is called **consistent** with a causal graph $D = ([n], E)$, if $\forall i \in [n]$, $\exists$ a set of feedforward layers $f_i$ such that $G_i(Z)$ can be written as $G_i(Z) = f_i(\{G_j(Z)\}_{j \in Pa_i}, Z_{S_i})$, where $Pa_i$ are the set of parents of $i$ in $D$, and $Z_{S_i} \coloneqq \{Z_j \ : j \in S_i\}$ are collections of subsets of $Z$ such that $\{S_i : i \in [n]\}$ is a partition of [m].*

Based on the definition, we can define causal implicit generative models as follows:




(a) Naive feedforward generator architecture and the causal graph it represents.

(b) Generator neural network architecture that represent the causal graph $X \rightarrow Z \leftarrow Y$.

Figure 2.2: (a) The causal graph implied by the naive feedforward generator architecture. (b) A neural network implementation of the causal graph $X \rightarrow Z \leftarrow Y$: Each feed forward neural net captures the function $f$ in the structural equation model $V = f(Pa_V, E)$.

**Definition 2** (CiGM)**.** *A feedforward neural network $G$ with output*

$$G(Z) = [G_1(Z), G_2(Z), \ldots, G_n(Z)], \tag{2.1}$$

*is called a causal implicit generative model for the causal model $\mathcal{M} = (D = ([n], E), N, \mathcal{F}, \mathbb{P}_N(.))$ if $G$ is consistent with the causal graph $D$ and $\mathbb{P}(G(Z) = \mathbf{x}) = \mathbb{P}_{[n]}(\mathbf{x}) > 0, \forall \mathbf{x}$.*

We propose using adversarial training where the generator neural network is consistent with the causal graph according to Definition 1, which is explained in the next section.

## 2.5 Causal Generative Adversarial Networks

CiGMs can be trained with samples from a joint distribution given the causal graph between the variables. However, for the application of image generation with binary labels, we found it difficult to simultaneously learn the joint label and image distribution[7]. For this application, we focus on dividing the task of learning a CiGM into two subtasks: First,

[7]Please see the Section A.1.16 in the Appendix for our primitive result using this naive attempt.

we train a generative model over the labels, then train a generative model for the images conditioned on the labels. For this training to be consistent with the causal structure, we assume that the image node is always the sink node of the causal graph for image generation problems (Please see Figure A.1 in Appendix). As we show next, our new architecture and loss function (CausalGAN) assures that the optimum generator outputs the label conditioned image distributions, under the assumption that the joint probability distribution over the labels is strictly positive[8]. Then for a strictly positive joint distribution between labels and the image, combining CiGM for only the labels with a label-conditioned image generator gives a CiGM for images and labels (see Corollary 1.1).

### 2.5.1 Causal Controller

First we describe the adversarial training of a CiGM for binary labels. This generative model, which we call the *Causal Controller*, will be used for controlling which distribution the images will be sampled from when intervened or conditioned on a set of labels. As in Section 2.4, we structure the Causal Controller network to sequentially produce labels according to the causal graph. Since our theoretical results hold for binary labels, we prefer a generator which can sample from an essentially discrete label distribution[9]. However, the standard GAN training is not suited for learning a discrete distribution, since Jensen-Shannon divergence requires the support to be the same for giving meaningful gradients, which is harder with discrete data distributions. To be able to sample from a discrete distribution, we employ WGAN (Arjovsky et al. [2017]). We used the model of Gulrajani et al. [2017], where the Lipschitz constraint on the gradient is replaced by a penalty term in the loss.

[8]This assumption does not hold in the CelebA dataset: $\mathbb{P}(Male = 0, Mustache = 1) = 0$. However, we will see that the trained model is able to extrapolate to these interventional distributions.

[9]Ignoring the theoretical considerations, adding noise to transform the labels artificially into continuous targets also works. However we observed better empirical convergence with this technique.

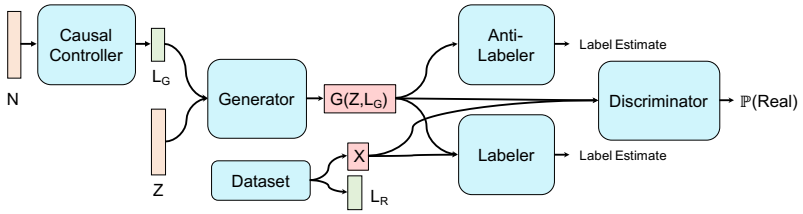



Figure 2.3: CausalGAN architecture: Causal controller is a pretrained causal implicit generative model for the image labels. Labeler is trained on the real data, Anti-Labeler is trained on generated data. Generator minimizes Labeler loss and maximizes Anti-Labeler loss.

### 2.5.2 CausalGAN

#### Architecture

As part of the two-step process proposed in Section 2.4 for learning a CiGM over the labels *and* the image variables, we design a new conditional GAN architecture to generate the images based on the labels of the Causal Controller. Unlike previous work, our new architecture and loss function assures that the optimum generator outputs the label conditioned image distributions. We use a pretrained Causal Controller which is not further updated.

**Labeler and Anti-Labeler:** We have two separate labeler neural networks. *The Labeler* is trained to estimate the labels of images in the dataset. *The Anti-Labeler* is trained to estimate the labels of the images sampled from the generator, where image labels are those produced by the Causal Controller.

**Generator:** The objective of the generator is 3-fold: producing realistic images by competing with the discriminator, producing images consistent with the labels by minimizing the Labeler loss and avoiding unrealistic image distributions that are easy to label by maximizing the Anti-Labeler loss.

The most important distinction of CausalGAN with the existing conditional GAN architectures is that it uses an Anti-Labeler network in addition to a Labeler network. Notice

that the theoretical guarantee we develop in Section 2.5.2 does not hold without the Anti-Labeler. Intuitively, the Anti-Labeler loss discourages the generator network to output only few typical faces for a fixed label combination. This is a phenomenon that we call *label-conditioned mode collapse*. Minibatch-features are one of the most popular techniques used to avoid mode-collapse (Salimans et al. [2016]). However, the diversity within a batch of images due to different label combinations can make this approach ineffective for combating label-conditioned mode collapse. This effect is most prominent for rare label combinations. In general, using Anti-Labeler helps with faster convergence. Please see Section A.1.17 in the Appendix for results.

**Loss Functions**

We present the results for a single binary label $l$. The results can be extended to more labels. For a single binary label $l$ and the image $x$, we use $\mathbb{P}_r(l, x)$ for the data distribution between the image and the labels. Similarly $\mathbb{P}_g(l, x)$ denotes the joint distribution between the labels given to the generator and the generated images. In our analysis we assume a perfect Causal Controller[10] and use the shorthand $\mathbb{P}_g(l = 1) = \mathbb{P}_r(l = 1) = \rho = 1 - \bar{\rho}$. Let $G(.), D(.), D_{LR}(.)$, and $D_{LG}(.)$ are the mappings due to generator, discriminator, Labeler, and Anti-Labeler respectively.

The generator loss function of CausalGAN contains label loss terms, the GAN loss in Goodfellow et al. [2014], and an added loss term due to the discriminator. With the addition of this term to the generator loss, we are able to prove that the optimal generator outputs the class conditional image distribution. This result is also true for multiple binary labels, which is shown in the Appendix.

[10]Even for multiple labels, we observe convergence in total variation distance. Please see Figure A.4b.

For a fixed generator, Anti-Labeler solves the following optimization problem:

$$\max_{D_{LG}} \rho \mathbb{E}_{x\sim\mathbb{P}_g(x|l=1)} \left[\log(D_{LG}(x))\right] + \bar{\rho}\mathbb{E}_{x\sim\mathbb{P}_g(x|l=0)} \left[\log(1 - D_{LG}(x)\right]. \tag{2.2}$$

The Labeler solves the following optimization problem:

$$\max_{D_{LR}} \rho \mathbb{E}_{x\sim\mathbb{P}_r(x|l=1)} \left[\log(D_{LR}(x))\right] + \bar{\rho}\mathbb{E}_{x\sim\mathbb{P}_r(x|l=0)} \left[\log(1 - D_{LR}(x)\right]. \tag{2.3}$$

For a fixed generator, the discriminator solves the following optimization problem:

$$\max_{D} \mathbb{E}_{(l,x)\sim\mathbb{P}_r(l,x)} \left[\log(D(x))\right] + \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)} \left[\log\left(1 - D(x)\right)\right]. \tag{2.4}$$

For a fixed discriminator, Labeler and Anti-Labeler, the generator solves the following problem:

$$\begin{aligned}
&\min_{G} \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)} \left[\log\left(\frac{1 - D(x)}{D(x)}\right)\right] - \rho\mathbb{E}_{x\sim\mathbb{P}_g(x|l=1)} \left[\log(D_{LR}(X))\right] \\
&- \bar{\rho}\mathbb{E}_{x\sim\mathbb{P}_g(x|l=0)} \left[\log(1 - D_{LR}(X))\right] + \rho\mathbb{E}_{x\sim\mathbb{P}_g(x|l=1)} \left[\log(D_{LG}(X))\right] \\
&+ \bar{\rho}\mathbb{E}_{x\sim\mathbb{P}_g(x|l=0)} \left[\log(1 - D_{LG}(X))\right].
\end{aligned} \tag{2.5}$$

**Theoretical Guarantees**

We show that the best CausalGAN generator for the given loss function samples from the class conditional image distribution when Causal Controller samples from the true label distribution and the discriminator and labeler networks always operate at their optimum. We show this result for the case of a single binary label $l \in \{0, 1\}$. The proof can be extended to multiple binary variables, which is given in the Appendix. As far as we are aware of, this is the only conditional generative adversarial network architecture with this guarantee after

CGAN[11].

First, we find the optimal discriminator for a fixed generator. Note that in (2.4), the terms that the discriminator can optimize are the same as the GAN loss in Goodfellow et al. [2014]. Hence the optimal discriminator behaves the same. To characterize the optimum discriminator, labeler and anti-labeler, we have Proposition 3, Lemma 1 and Lemma 2 given in the appendix.

Let $C(G)$ be the generator loss for when the discriminator, Labeler and Anti-Labeler are at the optimum. Then the generator that minimizes $C(G)$ samples from the class conditional distributions:

**Theorem 1.** *Assume $\mathbb{P}_g(l) = \mathbb{P}_r(l)$. Then the global minimum of the virtual training criterion $C(G)$ is achieved if and only if $\mathbb{P}_g(l, x) = \mathbb{P}_r(l, x)$, i.e., if and only if given a label $l$, generator output $G(z, l)$ has the same distribution as the class conditional image distribution $\mathbb{P}_r(x|l)$.*

Now we can show that our two stage procedure can be used to train a causal implicit generative model for any causal graph where the *Image* variable is a sink node, captured by the following corollary. $\mathcal{L}, \mathcal{I}, \mathcal{Z}_1, \mathcal{Z}_2$ represent the space of labels, images, and noise variables, respectively.

**Corollary 1.1.** *Suppose $C : \mathcal{Z}_1 \to \mathcal{L}$ is a causal implicit generative model for the causal graph $D = (\mathcal{V}, E)$ where $\mathcal{V}$ is the set of image labels and the observational joint distribution over these labels are strictly positive. Let $G : \mathcal{L} \times \mathcal{Z}_2 \to \mathcal{I}$ be a generator that can sample from the image distribution conditioned on the given label combination $L \in \mathcal{L}$. Then $G(C(Z_1), Z_2)$ is a causal implicit generative model for the causal graph $D' = (\mathcal{V} \cup \{Image\}, E \cup \{(V_1, Image), (V_2, Image), \dots (V_n, Image)\})$.*

[11] CGAN (Mirza and Osindero [2016]) can be shown to have the same guarantee. The difference of our architecture is that we do not feed image labels to the discriminator.

In Theorem 1 we show that the optimum generator samples from the class conditional distributions given a single binary label. Our objective is to extend this result to the case with $d$ binary labels. First we show that if the Labeler and Anti-Labeler are trained to output $2^d$ scalars, each interpreted as the posterior probability of a particular label combination given the image, then the minimizer of $C(G)$ samples from the class conditional distributions *given d labels*. This result is shown in Theorem 8 in the appendix. However, when $d$ is large, this architecture may be hard to implement. To resolve this, we propose an alternative architecture, which we implement for our experiments: We extend the single binary label setup and use cross entropy loss terms for each label. This requires Labeler and Anti-Labeler to have only $d$ outputs. However, although we need the generator to capture the joint label posterior given the image, this only assures that the generator captures each label's posterior distribution, i.e., $\mathbb{P}_r(l_i|x) = \mathbb{P}_g(l_i|x)$ (Proposition 4). This, in general, does not guarantee that the class conditional distributions will be true to the data distribution. However, for many joint distributions of practical interest, where *the set of labels are completely determined by the image*[12], we show that this guarantee implies that the joint label posterior will be true to the data distribution, implying that the optimum generator samples from the class conditional distributions. Please see Section A.1.7 for the formal results and more details.

**Remark:** Note that the trained causal implicit generative models can also be used to sample from the counterfactual distributions if the exogenous noise terms are known. Counterfactual sampling require conditioning on an event and sampling from the push-forward of the posterior distributions of the exogenous noise terms under the interventional causal graph due to a possible intervention. This can be done through rejection sampling to observe the evidence, holding the exogenous noise terms consistent with the observed evidence and interventional sampling afterwards.

[12]The dataset we are using arguably satisfies this condition.

### 2.5.3 CausalBEGAN

In this section, we sketch a simple, but non-trivial extension of BEGAN where we feed image labels to the generator, leaving the details to the Appendix (Section A.1.8). To accommodate interventional sampling, we again use the Causal Controller to produce labels.

In terms of architecture modifications, we use a Labeler network with a dual purpose: to label real images well and generated images poorly. This network can be seen as both analogous to the original two-roled BEGAN discriminator and analogous to the CausalGAN Labeler and Anti-Labeler.

In terms of margin modifications, we are motivated by the following observations: (1) Just as a better trained BEGAN discriminator creates more useful gradients for image quality, (2) a better trained Labeler is a prerequisite for meaningful gradients for label quality. Finally, (3) label gradients are most informative when the image quality is high. Each observation suggests a margin term; the final observation suggests a (necessary) *margin of margins* term comparing the first two margins.

## 2.6 Results

In this section, we train CausalGAN and CausalBEGAN on the CelebA Causal Graph given in Figure A.1. For this, we first trained the Causal Controller (See Section A.1.11 for Causal Controller results.) on the image labels of CelebA Causal Graph. Please see Section A.1.17 for implementation details. The results are given in Figures 2.4, 2.5 for CausalGAN and Figures 2.6, 2.7 for CausalBEGAN. The difference between intervening and conditioning is clear through certain features. We implement conditioning through rejection sampling. See Naesseth et al. [2017], Graham and Storkey [2017] for other works on conditioning for implicit generative models.

Top: Intervene Mustache=1, Bottom: Condition Mustache=1

Figure 2.4: Intervening/Conditioning on Mustache label in CelebA Causal Graph with CausalGAN. Since $Male \rightarrow Mustache$ in CelebA Causal Graph, we do not expect $do(Mustache = 1)$ to affect the probability of $Male = 1$, i.e., $\mathbb{P}(Male = 1|do(Mustache = 1)) = \mathbb{P}(Male = 1) = 0.42$. Accordingly, the top row shows both males and females with mustaches, even though the generator never sees the label combination $\{Male = 0, Mustache = 1\}$ during training. The bottom row of images sampled from the conditional distribution $\mathbb{P}(.|Mustache = 1)$ shows only male images.

Top: Intervene Mouth Slightly Open=1, Bottom: Condition Mouth Slightly Open=1

Figure 2.5: Intervening/Conditioning on Mouth Slightly Open label in CelebA Causal Graph with CausalGAN. Since $Smiling \rightarrow MouthSlightlyOpen$ in CelebA Causal Graph, we do not expect $do(\textbf{\textit{Mouth Slightly Open}} = 1)$ to affect the probability of $Smiling = 1$, i.e., $\mathbb{P}(Smiling = 1|do(\textbf{\textit{Mouth Slightly Open}} = 1)) = \mathbb{P}(Smiling = 1) = 0.48$. However on the bottom row, conditioning on *Mouth Slightly Open* $= 1$ increases the proportion of smiling images (From $0.48$ to $0.76$ in the dataset), although 10 images may not be enough to show this difference statistically.

Top: Intervene Mustache=1, Bottom: Condition Mustache=1

Figure 2.6: Intervening/Conditioning on Mustache label in CelebA Causal Graph with CausalBEGAN. Since $Male \rightarrow Mustache$ in CelebA Causal Graph, we do not expect $do(Mustache = 1)$ to affect the probability of $Male = 1$, i.e., $\mathbb{P}(Male = 1|do(Mustache = 1)) = \mathbb{P}(Male = 1) = 0.42$. Accordingly, the top row shows both males and females with mustaches, even though the generator never sees the label combination $\{Male = 0, Mustache = 1\}$ during training. The bottom row of images sampled from the conditional distribution $\mathbb{P}(.|Mustache = 1)$ shows only male images.

Top: Intervene Narrow Eyes=1, Bottom: Condition Narrow Eyes=1

Figure 2.7: Intervening/Conditioning on Narrow Eyes label in CelebA Causal Graph with CausalBEGAN. Since $Smiling \rightarrow$ *Narrow Eyes* in CelebA Causal Graph, we do not expect $do($*Narrow Eyes* $= 1)$ to affect the probability of $Smiling = 1$, i.e., $\mathbb{P}(Smiling = 1|do($*Narrow Eyes* $= 1)) = \mathbb{P}(Smiling = 1) = 0.48$. However on the bottom row, conditioning on *Narrow Eyes* $= 1$ increases the proportion of smiling images (From $0.48$ to $0.59$ in the dataset), although 10 images may not be enough to show this difference statistically. As a rare artifact, in the dark image in the third column the generator appears to rule out the possibility of *Narrow Eyes* $= 0$ instead of demonstrating *Narrow Eyes* $= 1$.

## 2.7 Conclusion

We proposed a novel generative model with label inputs. In addition to being able to create samples *conditioned* on labels, our generative model can also sample from the *interventional* distributions. Our theoretical analysis provides provable guarantees about correct sampling under such interventions. Causality leads to generative models that are more creative since they can produce samples that are different from their training samples in multiple ways. We have illustrated this point for two models (CausalGAN and CausalBEGAN).

# Chapter 3

# Sample Compression, Support Vectors, and Generalization in Deep Learning [1]

Even though Deep Neural Networks (DNNs) are widely celebrated for their practical performance, they possess many intriguing properties related to depth that are difficult to explain both theoretically and intuitively. Understanding how weights in deep networks coordinate together across layers to form useful learners has proven challenging, in part because the repeated composition of nonlinearities has proved intractable. This paper presents a reparameterization of DNNs as a linear function of a feature map that is *locally independent* of the weights. This feature map transforms depth-dependencies into simple *tensor* products and maps each input to a discrete subset of the feature space. Then, using a max-margin assumption, the paper develops a *sample compression* representation of the neural network

[1] **This chapter is based on material from the publication (Snyder and Vishwanath [2020a]), which is in press to be published with the IEEE Journal on Selected Areas in Information Theory.**

in terms of the discrete activation state of neurons induced by $s$ "support vectors". The paper shows that the number of support vectors $s$ relates with learning guarantees for neural networks through sample compression bounds, yielding a sample complexity of $\mathcal{O}(ns/\epsilon)$ for networks with $n$ neurons. Finally, the number of support vectors $s$ is found to monotonically increase with width and label noise but decrease with depth.

## 3.1 Introduction

Neural networks represent an intriguing class of models that have achieved state-of-the-art performance results for many machine learning tasks. Although neural networks have been studied for over half a century McCulloch and Pitts [1943], the variations which have recently garnered interest are called "deep" neural networks (DNNs). Deep learning is characterized by stacking one layer after another and using the computational power of modern graphical processor units (GPUs) or custom processors/ASICs to train them. It is shown experimentally that such networks with more layers tend to generalize better Novak et al. [2018] Neyshabur et al. [2019].

Thus, deep learning presents a scenario where the best performing models are also generally poorly understood. Improvements to our theoretical understanding of deep neural networks will aid in structured, principled approaches to the design, analysis, and use of such networks. An important step in understanding a model is to prove generalization bounds that agree with performance in practice. For DNNs, several attempts have been made in this direction based on margin, perturbation, PAC-Bayes, or complexity type approaches (see related work section for further details).

Proving generalization bounds for our existing models helps us to build better models in the future by forcing us to articulate and analyze what exactly it is about DNNs that makes them work. From classical learning theory, our intuitions is to associate regularity with

small parameter number or norm penalization. We expect that discussion of the interplay between depth and generalization of unregularized networks will be particularly fruitful exactly because it runs contrary to these intuitions. We seek to add an alternative perspective by compressing the DNN classifier as one of the solution to one of a small number of optimization problems, side-stepping the need to discuss the norm of various weight matrices or the architecture dimensions explicitly.

In this paper, under assumptions presented in Section 3.3.3, we recast Leaky-ReLU type networks as an equivalent support vector machine (SVM) problem where the features correspond to paths through the network and the embedding map $\phi$ has local invariance to perturbation of the weights of the DNN. Though this embedding is a non-trivial function of these weights, the induced kernel has a simple interpretation as the inner product in the input space scaled by the number of shared paths in the network.

Our main contributions can be summarized as:

1. We present a framework for recasting neural networks with two-piecewise-linear nonlinearities (such as ReLU) as an SVM problem where classification with the network is equivalent to linear classification in a particular tensor space. Here, the corresponding embedding of training points is insensitive to local perturbations of weights.
2. We introduce for study a particular type of DNN satisfying a max-margin assumption. These networks can be compressed as the solution to an optimization problem determined by a few samples (much like SVMs). We point through analogy with unregularized logistic regression (in the feature space) that we may expect DNNs trained with unregularized gradient to satisfy this assumption in the training limit. We show empirically that modifying trained DNNs to satisfy this max-margin assumption essentially leaves their predictions unchanged.

3. Under a max-margin assumption on the network within this new feature space, we define "network support vectors": those training samples that are mapped to support vectors under the learned embedding map. An important consequence of this max-margin assumption is that only *finitely many* neural network classifiers correspond to a set of network support vectors.
4. We show that the number of network support vectors, $s$, can be related theoretically to generalization and experimentally to network architecture. We use a sample compression variant of PAC-Bayes to prove in Theorem 3.3.5 a bound on the sample complexity of max-margin networks with $n$ neurons of $\mathcal{O}(ns)$. We show that the quantity, $ns$, experimentally decreases with depth (despite $n$ increasing with depth). We expect the quantity $n$ to be subject to further improvement.

## 3.2 Related Work

There have been multiple, well-thought-out efforts to model, characterize and understand generalization error in DNNs. One well-studied direction is to impose a sufficiently small norm condition on the neural network weights Golowich et al. [2018]Neyshabur et al. [2015]. Since the weight norms induce a bound on the network's Lipschitz constant, one can connect this with insensitivity of the network output to input perturbations, either through a product of weight spectral norms Bartlett et al. [2017a] or through the norm of the network Jacobian itself Sokolic et al. [2017]. Instead of invariance to input perturbations, one can also consider the degree of invariance of the network dependence to weight perturbations Neyshabur et al. [2017a]. A natural way to concretely relate such perturbation schemes to generalization error is through the means of probably approximately correct PAC-Bayes analysis as in McAllester [1999] McAllester [2013].

The general principle underlying PAC-Bayes analysis is to characterize (in bits,

with respect to some prior) the degree of precision in specifying the final neural network weights in order to realize the observed training performance. Such PAC-Bayes based generalization bounds were applied successfully to the study of neural networks by Langford and Caruana [2002] and more recently by Dziugaite and Roy [2017], albeit for stochastic networks. Generally speaking, if multiple weights corresponding to a large neighborhood result in similar neural network behavior, then fewer bits are needed to specify these weights. Overall, insensitivity to weight perturbation is one potential manner to formalize the popular high-level idea that "flat minima generalize well" Neyshabur et al. [2017b], Hochreiter and Schmidhuber [1997].

In order to correctly reproduce the improvement in generalization observed in deep learning with each additional layer, the principle difficulty is that these approaches must make layer-wise considerations (either of each weight matrix or each layer-wise computation) that accumulate and grow the generalization bound as depth increases. Of course, it is possible to find suitable assumptions that control or mitigate this depth-dependent growth as in Golowich et al. [2018] or Kawaguchi et al. [2017]. Given this challenge, other "network compression" type approaches that characterize the network function without addressing every individual parameter are gaining interest. For example, Neyshabur et al. [2014] analyzes the number of nonzero weights as a form of capacity control, while others have studied approximating a deep network by a "compressed" version with fewer nonzero weights Arora et al. [2018a] Zhou et al. [2018].

In this paper, we use a *sample* compression Littlestone and Warmuth [1986] representation approach for understanding neural network depth-dependence. We transform the neural network into a related SVM problem, then recover the network function from (suitably defined) support vectors. Sample compression characterizes PAC learnable functions as those that can be recovered from a small enough subset of training samples Floyd and Warmuth

[1995]. This theory readily finds application in kernel modeling, for example Germain et al. [2011], since support vectors provide a natural correspondence between (max-margin) hypotheses data subsets. Our own work makes consistent use of machinery developed by Laviolette Laviolette and Marchand [2007] who significantly increased the applicability of sample-compression theory by allowing model reconstructions to depend additionally on "side channel" information.

## 3.3 On Neural Networks as Support Vector Machines

### 3.3.1 Notation Definitions and Setting

In this paper, we consider the family of nonlinearities $\rho(x) = \beta x \mathbb{1}_{\{x<0\}}(x) + \gamma x \mathbb{1}_{\{x\geq 0\}}(x)$ for $\beta, \gamma \in \mathbb{R}$ for the neural network, which encompasses ReLU, Leaky-ReLU, and absolute value as examples. We will refer to these nonlinearities collectively as "Leaky-ReLU". For vector arguments, $\rho$ is understood to be applied element-wise. We do not use biases. For integer $m$, we will use $[m]$ to mean the set $\{1, \ldots, m\}$.

Consider a neural network with $d$ (depth) hidden layers, width $\Omega$ neurons in layer $l$, $f$ input features, and $m$ training samples.

We will use $\mathcal{W} = \mathbb{R}^{\Omega} \times (\prod_{i=1}^{d-1} \mathbb{R}^{\Omega \times \Omega}) \times \mathbb{R}^{\Omega \times f}$ to denote the set of all possible weights within the neural network. Here $A^{(l)}_{i_{l+1}, i_l}$ refers to the scalar weight from neuron $i_l$ in $l$ to neuron $i_{l+1}$ in layer $l+1$. We use $w$ to refer to *all of the weights collectively*, with $w = (A^{(d)}, \ldots, A^{(1)}, A^{(0)}) \in \mathcal{W}$. Each $w$ corresponds to a neural network mapping from each $x \in \mathcal{X} \triangleq \mathbb{R}^f$ to $\mathbb{R}$ as follows:

$$\mathcal{N}(x, w) \triangleq \mathcal{N}_w(x) \triangleq A^{(d)} \rho(A^{(d-1)} \rho(\ldots (A^{(1)} \rho(A^{(0)} x) \ldots)) \tag{3.1}$$

We distinguish between $\mathcal{N}_w : \mathcal{X} \mapsto \mathbb{R}$, which returns scalar values, and the related

classifier returning labels, $\mathcal{N}_w^{sign} \triangleq sign \circ \mathcal{N}_w : \mathcal{X} \mapsto \mathcal{Y}$. Here $\mathcal{Y} \triangleq \{-1, +1\}$ and $sign(\cdot)$ is a function returning the sign of its argument (defaulting to $+1$ for $0$ input). For a data distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$, the goal is to use a training set $S^{(m)} = \{(x^j, y^j)\}_{j=1}^m \sim \mathcal{D}^m$ to learn a set of weights $w$ so that $\mathcal{N}_w^{sign}$ has small probability of misclassification on additional samples drawn from $\mathcal{D}$.

We define a *path* (in a neural network) to be an element of $(\prod_{i=1}^d [\Omega]) \times [f]$, corresponding to a choice of 1 neuron per hidden layer and 1 input feature. Sometimes it is convenient to refer to these input features as neurons in layer $l = 0$. Thus, one says that the path $i_d, \ldots, i_1, i_0$ traverses neuron $i_l$ in layer $l = 0, 1, \ldots, d$.

Given a set of weights $w$, we define $\Lambda(w)$ to be the path-indexed vector with the product of weights along path $p$ in position $p$. Often we use $\bar{w}$ to shorten $\Lambda(w)$, and we use $\bar{w}_p$ or $\bar{w}_{i_d,\ldots,i_1,i_0}$ when we want to specify the path.

### 3.3.2 A Reparameterization of the Network

Consider the set of all paths starting from some feature in the input and passing through one neuron per hidden layer of a ReLU neural network. Index these $f\Omega^d$ many paths by the coordinate tuple $(i_d, \ldots, i_1, i_0)$ to denote the path starting at feature $i_0$ in the input and passing through neuron $i_l$ in hidden layer $l$. Given a set of network weights $w$, we can define $\Lambda(w) = \bar{w} = \bar{w}_{i_d,\ldots,i_1,i_0}$, whose $(i_d, \ldots, i_1, i_0)^{th}$ coordinate is the product of weights along path $(i_d, \ldots, i_1, i_0)$. Inspired by Kawaguchi et al. [2017], (who used a similar factorization without exploring the connections with support vector machines) we note that the output of a neural network can be viewed as a sum of contributions over paths

$$\mathcal{N}(x, w) = \sum_{p=(i_d,\ldots,i_1,i_0)} \sigma^{(d)}(x, w)_{i_d} \cdots \sigma^{(1)}(x, w)_{i_1} x_{i_0} \bar{w}_p$$

where $\sigma^{(l)}(x, w)$ is an indicator vector for which neurons in layer $l$ are active for input $x$ with weights $w$. For convenience, we also define $\bar{\sigma}(x, w) = \bar{\sigma}(x, w)_{i_d,\ldots,i_1} = \sigma^{(d)}(x, w)_{i_d} \cdots \sigma^{(1)}(x, w)_{i_1}$, which is also an indicator but over paths instead of neurons[2]. The above summation over all tuples $(i_d, \ldots, i_1, i_0)$ can be interpreted as an inner product $\langle \phi(x, w), \bar{w} \rangle$ where

$$\phi(x, w)_{i_d,\ldots,i_1,i_0} = \sigma^{(d)}(x, w)_{i_d} \cdots \sigma^{(1)}(x, w)_{i_1} x_{i_0} \tag{3.2}$$

is a $w$-parameterized family of embedding maps from the input to a feature space we denote as the "Path Space" $\mathcal{F}$, i.e., the set of all tensors assigning some scalar to each path index-tuple $i_d, \ldots, i_1, i_0$ with $i_l \in [b]$ for $l \in [d]$ and $i_0 \in [f]$. The neural network then is *almost* a kernel classifier in that the model only interacts with the input through inner products with a feature map $\phi(x, w)$. Though unlike a SVM, the feature map has some dependence on $w$.

An important insight is that, over small regions of the weight space, our embedding $\phi(x_i, w)$ does not depend on $w$ for any of the finitely many training points. More precisely, suppose that none of the pre-nonlinearity activations of neurons in $\mathcal{N}$ are *identically* zero. Then for each training sample and each neuron pre-activation, we obtain an open ball about this pre-activation (excluding zero). Since the function from the weights to each pre-activation is continuous, the preimage of each ball in the weight space is open. The intersection of these (finitely many) preimages is an open set around the current network weights in which the feature space embedding of training samples (not necessarily test samples) is *independent* of our weights. Interestingly, this implies that over small, say $\epsilon > 0$ sized, regions of weights around $w$, say $B_\epsilon(w)$, we may parameterize our training outputs unambiguously by the product of weights over paths, $\bar{w} \in \Lambda(B_\epsilon(w))$, instead of the "usual"

[2]Leaky-ReLU units scale inputs by $\beta$ or $\gamma$ in place of 0 or 1, hence the $\sigma^{(l)}$ are no longer literally indicator functions

parameterization $w$. Note though that globally the relationship is not $1-1$.

Though the practical size of these regions may be very small, this local linearization in terms of $\bar{w}$ is analytically well suited for characterizing networks trained by local methods, e.g. gradient descent. Indeed, the gradient operator at every $w$ treats $\bar{\sigma}$ as a constant function of $w$, independently of the size of the containing region. If ultimately, the final training weights satisfy some condition in terms of the gradient operator then we should study this condition by considering local perturbations of a set of linear models parameterized by $\bar{w}$. We assume cross-entropy loss, in which case the local loss landscape of models in a small neighborhood of $\bar{w}$ is exactly that of the loss landscape of logistic regression models on $\mathcal{F}$ with the same training data and feature map $\phi(\cdot, w)$.

### 3.3.3 Assumptions Made

Prior to detailing the assumptions made in this paper, we first highlight a compelling recent work on *unregularized* logistic regression for linearly separable problems in Soudry et al. [2017]. Here, the authors prove that gradient descent yields a sequence of classifiers whose normalized versions converge to the max margin solution. For example, the authors provide a theoretical basis for the increase in test accuracy and test loss during training even after the training accuracy is 100%. Note that this peculiar behavior is also common to neural networks Shwartz-Ziv and Tishby [2017]. Inspired by this connection, we assume the following:

**Assumption 1.** *Zero Training Error*

*The weights $w$ obtained from training on $S^{(m)}$ ensure $\mathcal{N}_w^{sign}$ correctly classifies every sample in $S^{(m)}$. Equivalently:*

$$\forall (x, y) \in S^{(m)} \quad y\langle \Lambda(w), x \rangle \geq 0$$

Note that, for zero training error, linear separability[3] of our embedded data, $\{(\phi(x^j, w), y^j)\}_{j=1}^m$, is strictly necessary. Motivated by analogy with maximum margin classifiers in logistic regression, we make the following second assumption on the network weights obtained by training on $S^{(m)}$:

**Assumption 2.** *Max-Margin*

*The training procedure returns weights $w$ such that up to positive scaling, $\Lambda(w)$ is the maximum margin classifier for the $w$-parameterized embedding $\{(\phi(x^j, w), y^j) : j \in [m]\}$. Equivalently, $w$ must satisfy the relation*

$$\Lambda(w) \in \arg\max_{\bar{v} \in \mathcal{F}} \min_{(x,y) \in S^{(m)}} \frac{y\langle \bar{v}, \phi(x, w)\rangle}{\|\bar{v}\|}$$

### 3.3.4 Merit of Assumptions

Of the two assumptions made in the paper, note that Assumption 1, of zero training error, is not uncommon for neural networks in practice Soudry and Carmon [2016]. Therefore, we do not discuss Assumption 1 in greater detail in this subsection, focusing more on the second assumption in this paper.

The value of Assumption 2 is more nuanced, and we devote an entire section to discussing this, expanding with relevant experiments in Appendix A.2.2. In short, we show empirically that networks trained with gradient descent satisfying Assumption 1 are not too different from those satisfying Assumption 2. This is not unexpected given the comparison with unregularized logistic regression (in the feature space). The merit of an assumption lies not in whether it is strictly true but in whether it is *interesting*. This assumption concisely explains certain experimental phenomena and allows theoretical tractability. Experimentally, it also seems *relevant* to practical DNNs. Unregularized logistic regression only finds the

[3] In fact we are guaranteed a separating hyperplane containing the origin

max-margin classifier in the training limit as the number of iterations approaches infinity. Our experimental observations of DNNs trained with finitely many iterations are consistent with those that *approximately* satisfy the max-margin assumption.

Note that, without idealized assumptions such as Assumption 2, it is very difficult to build a framework that helps us gain an understanding of the problem, or the implications of its solution. In particular, Assumption 2 forms a starting point for a deeper theoretical understanding of neural networks, one that provides useful insights that can be employed towards a more general, overall theory for deep neural networks.

### 3.3.5 Network Support Vectors

In this section we use Assumption 2 to extend the definition of support vectors to neural networks with zero training error. By the Representer Theorem Schölkopf et al. [2001], the max-margin condition on $\bar{w}$ in Assumption 2 implies that for some nonnegative scalars $\alpha_1, \ldots, \alpha_m$,

$$\bar{w} = \sum_{k=1}^{m} \alpha_k y^k \phi(x^k, w). \tag{3.3}$$

Analogously to classical SVMs, for a fixed set of weights $w$ achieving Assumption 2, we define the subset $S^{(s)} \triangleq \{(x^k, y^k) : \alpha_k \neq 0\}$ of those training data points that correspond to nonzero $\alpha_k$ to be "network support vectors"(NSVs) or simply "support vectors" when context is clear. We also use $S^{(m-s)} = S^{(m)} - S^{(s)}$ to denote the $m - s$ data which are not support vectors.

To gain an experimental understanding of these "support vectors", we train neural networks on a 2-class MNIST variant formed by grouping labels $0 - 4$ and $5 - 9$. We show that many qualitative properties of SVMs continue to hold true in this case when the embedding map is learned. We first determine network weights obtained from minimizing the neural network loss. Then, we define an embedding map $\phi(\cdot, w)$ using those weights.

Finally, we train a SVM using the kernel as defined by $\langle \phi(x^i, w), \phi(x^j, w) \rangle$. The details of this experiment and all others in this section are presented in the Appendix A.2.3.

As noted in these experiments, we determine that the behavior of the number of NSVs is qualitatively similar to what we might find in a conventional SVM setting. For example, we typically find $s/m \approx 0.15$. We find that every time we increase the number of training samples, $m$, and retrain the network from scratch, the net effect is that $s$ increases but $s/m$ asymptotically decreases to $0.1$ (Figure 3.1). This is entirely expected in the simplified setting with a fixed embedding map: additional samples can only decrease the margin, reducing the fraction of volume within the margin of the hyperplane. Thus, additional randomly selected samples are increasingly unlikely to be support vectors.

Given that the SVM model (with fixed embedding) is determined entirely by $S^{(s)}$, the model is said to have "memorized" the sample $(x, y) \in S^{(m)}$ iff $(x, y)$ is a support vector. We find that a similar notion holds for network support vectors. In deep learning, the notion of memorizing a given individual sample is less clear, but we often describe a DNN with wildly divergent test and train accuracies as having "memorized the dataset". For example, DNNs will often achieve zero training error even when there is no relationship between inputs $\mathcal{X}$ and outputs $\mathcal{Y}$.

If we randomize each label of $S^{(m)}$ prior to training so that the training data is sampled from a product of marginal distributions instead, $S^{(m)} \sim \mathcal{D}_{\mathcal{X}} \times \mathcal{D}_{\mathcal{Y}}$, we observe experimentally that $s/m \approx 0.6$ (Figure A.22). This can be understood as follows: although the labels are independent of the inputs, there are natural clusters in the input that the model can use to fit these random labels in the training data. Each sample has a label consistent with at least half of the training set, since half of the training data have the correct label. Thus, the DNN is learning a pattern corresponding to the true labeling (or its reverse) and *building in exceptions* for the rest of the data by adding them as support vectors. Note that

learning this labeling on MNIST requires $0.1m$ support vectors (from before). The addition of $0.5m$ training samples that violate the first learned labeling results in the observed $0.6m$ total.

In a conventional SVM setting, models with fewer support vectors are thought of as more parsimonious. Furthermore, the fraction of training samples that are support vectors can be concretely linked to generalization bounds through sample compression techniques, as in Littlestone and Warmuth [1986]. An important observation is that the SVM solution can be reconstructed from the subset of support vectors $S^{(s)} \subset S^{(m)}$, so bounding $s = |S^{(s)}|$ controls the number of training samples the model can memorize. Similarly, in Section 3.4, we construct analogous bounds for deep neural networks that depend centrally on this number of NSVs.

We now turn to understanding how this number of NSVs varies with architecture parameters. We first study fully-connected networks on flattened MNIST images. There, we find that the fraction $s/m$ increases logarithmically as we increase the width $\Omega$ (Figure 3.2) but *decreases* linearly as we increase the depth $d$ (Figure 3.3). It is interesting to note that $s$ decreases with depth $d$ in these cases. Taking on faith for the moment that the next section (specifically, Theorem 3.3.5) will fashion a bound on the test error of the form $\mathcal{O}(ns/m)$, (as initially advertised in the abstract), we can understand the significance of this observation. Although $n$ increases linearly with $d$, we observe a net decrease in the generalization bound with depth, since the decrease in $s$ with depth is "superlinear" in the sense that doubling the number of layers from 3 to 6 more than halves $s$. As a result, the closely related generalization bound we will justify in the subsequent section seems to decrease with depth as well (Figure 3.5). While these experimental relationships are interesting, these are preliminary in nature, and additional study is required to make concrete claims on the relationships between parameters of the network.

Of particular interest is the inverse relationship between the number of NSVs and depth. In order to understand whether this relationship continues to hold in more general settings, we study binary classification of Frogs vs Ships on the CIFAR-10 dataset using convolutional networks with nonzero biases. These networks consist of initial convolutional and max pooling layers followed by a variable number of FC depth many fully-connected layers. We see that the relationship is more noisy, but still there is a clear trend that $s$ decreases significantly for larger depths (Figure 3.4). While we have extended the notion of network support vectors to convolution and nonzero bias networks (Appendix A.2.1), our generalization theory developed in the next section only supports fully-connected networks for now. Therefore we don't calculate a bound such as in Figure 3.5 for this data.

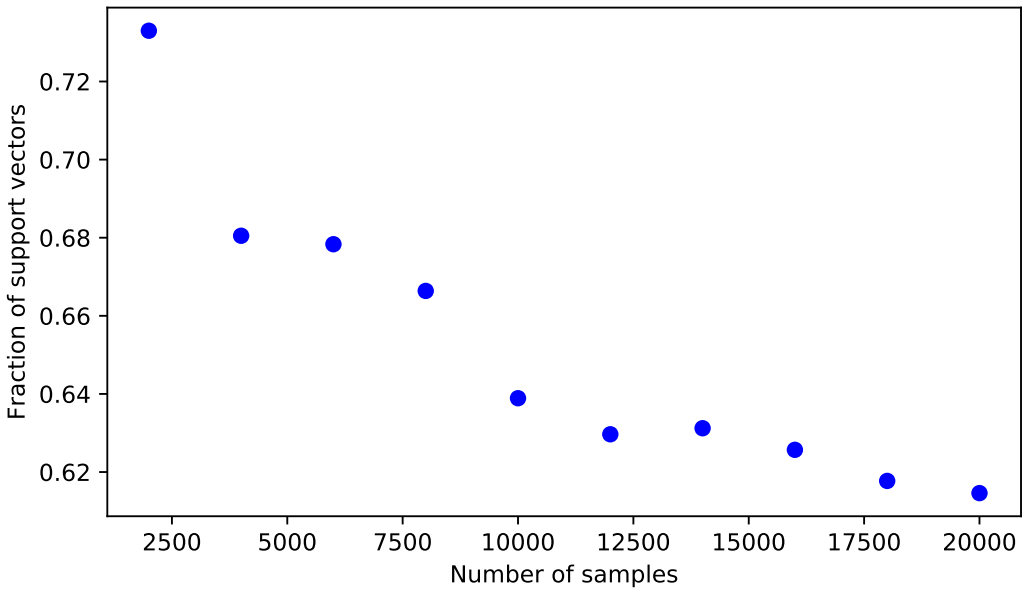



Figure 3.1: Support vector fraction of data $s/m$ vs Number of samples m: Increasing the size of the training set decreases asymptotically the fraction $s/m$ of support vectors.

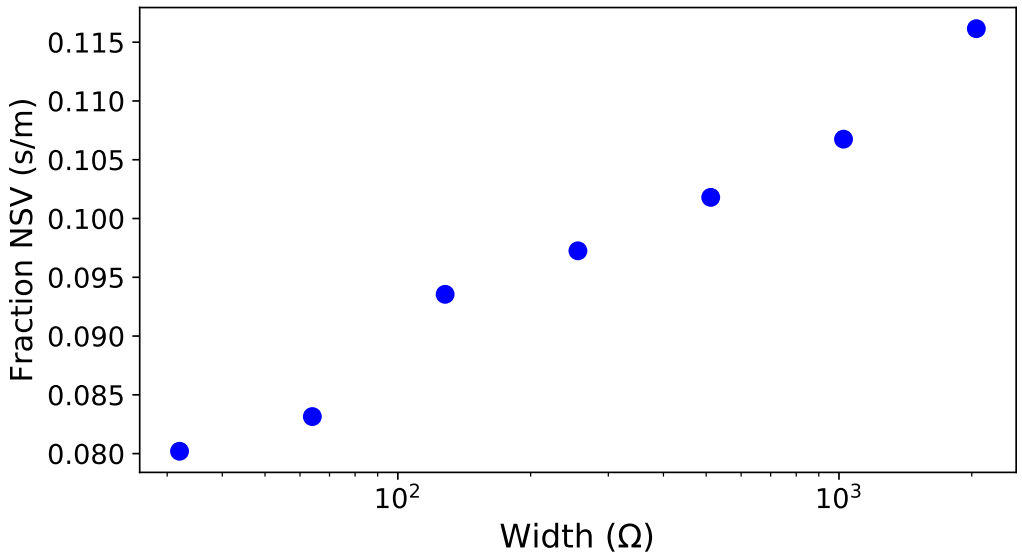



Figure 3.2: Fraction Network Support Vectors $(s/m)$ vs width $\Omega$: ReLU networks of varying width $\Omega$ are trained to classify MNIST images. Each width-dependent trained set of network weights, $w$, is used to define an embedding $\phi(\cdot, w)$. The number of support vectors, $s$, corresponding to the maximum margin classification of $(\phi(x^j, w), y^j)_{j=1}^m$ is measured ($m$ is constant). Each point represents an average of three runs. The results indicate that $s$ grows proportionally to $\log(\Omega)$.

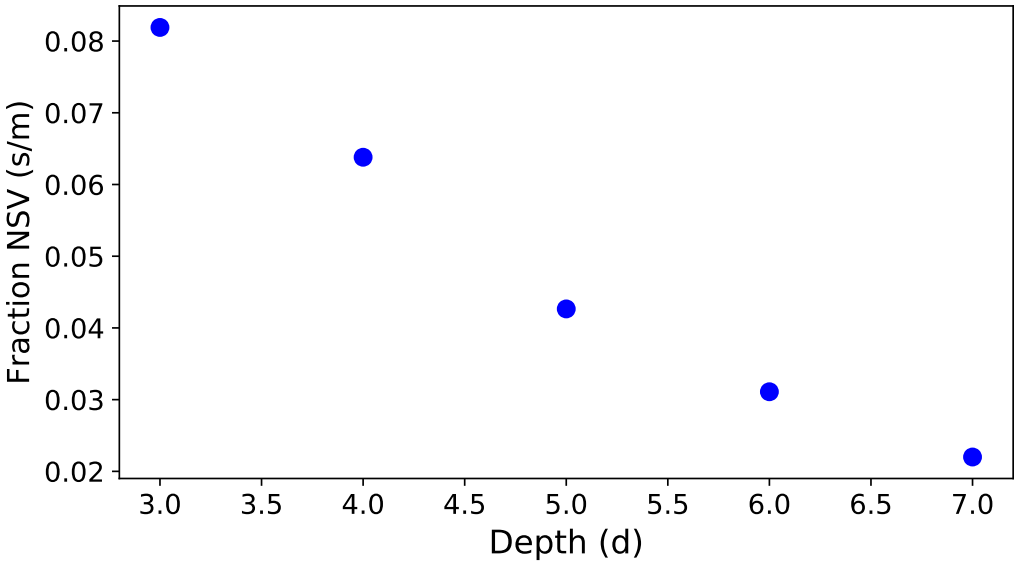



Figure 3.3: Fraction Network Support Vectors $(s/m)$ vs depth $d$: The depth, i.e., the number of hidden layers, is varied, resulting in a depth-dependent embedding of the training data, $(\phi(x^j, w), y^j)_{j=1}^m$, where $w$ is the set of weights obtained from training a DNN with $d$ layers to classify data in $S^{(m)}$. The number of support vectors $s$ decreases with depth over these finite ranges. Each point represents an average of three runs.

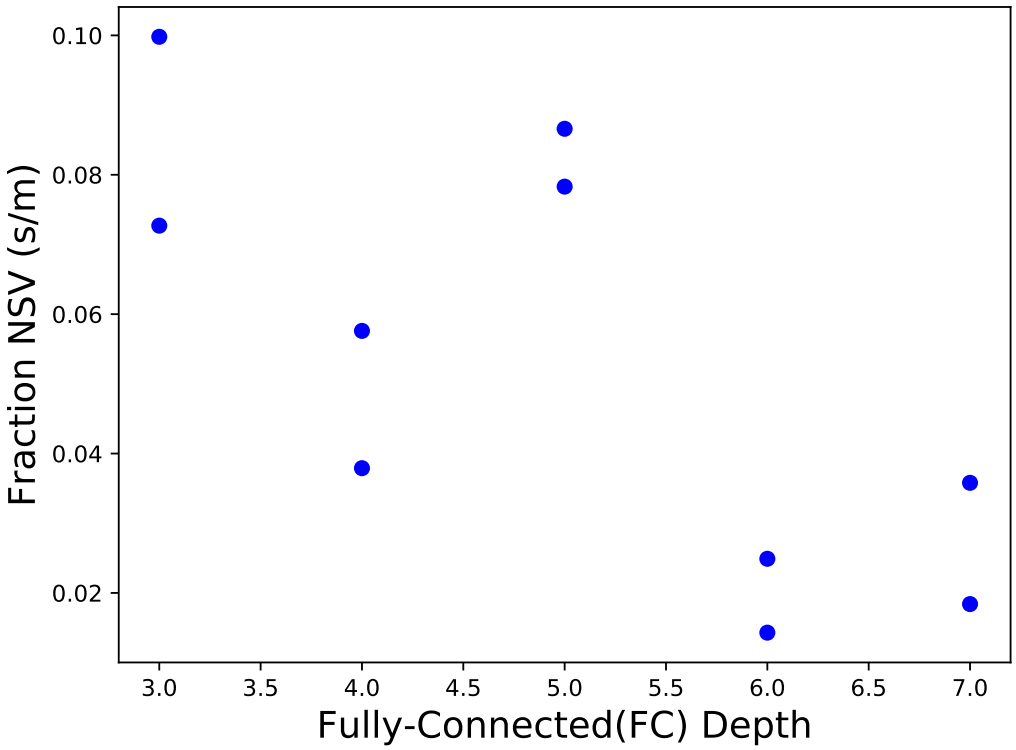



Figure 3.4: Fraction Network Support Vectors $(s/m)$ vs fully-connected (FC) depth on the CIFAR dataset. The first three layers learned are convolutional and are not counted toward the depth. We see that the max-margin classification of $(\phi(x^j, w), y^j)_{j=1}^m$ results in many fewer NSVs when then depth is made larger.

## 3.4 Sample Compression Bounds

In this section, we present a concrete theoretical relationship between the number of network support vectors and a bound of the test error of deep neural networks satisfying assumptions as outlined in Section 3.3.3. The setting for all theorems will be Leaky-ReLU (incl. ReLU) networks with arbitrary, fixed fully-connected architecture. Just as a SVM max-margin classifier is determined entirely by its cast of support vectors, *only finitely many* neural networks satisfying the max-margin assumption (Assumption 2) correspond to a given set of at most $s$ network support vectors. This is presented as the following theorem (proof given in Appendix A.2.7):

**Theorem 2.** *Let $\mathcal{N}$ refer to a Leaky-ReLU neural network with $d$ hidden layers each consisting of width $\Omega$ neurons so that we have $n = d\Omega$ neurons total. Let the weights $w$ be deterministic functions of $S^{(m)}$, which is a set of $m$ i.i.d. data samples from $\mathcal{D}$. Let $s < m$ be a fixed integer which does not depend on $S^{(m)}$. Supposing that:*

1. *Assumption 1 (Zero training error): $\mathcal{N}_w^{sign}(x) = y \; \forall (x, y) \in S^{(m)}$,*
2. *Assumption 2 (Max-margin): $\Lambda(w)$ is some positively scaled version of the max-margin classifier for $\{(\phi(x, w), y) : (x, y) \in S^{(m)}\}$, and*
3. *(At most $s$ support vectors): $\Lambda(w) = \sum_{k=1}^{m} \alpha_k y^k \phi(x^k, w)$ for some set of coefficients $\alpha_k$, at most $s$ of which are nonzero.*

*then we have, $\forall \delta \in (0, 1]$*

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} \left[ R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq \mathcal{F}(m, n, s, \delta) \right] \geq 1 - \delta$$

*where*

$$\mathcal{F}(m,n,s,\delta) = \frac{n + ns + s + s\ln\left(\frac{m}{s}\right) + \ln\left(\frac{1}{\delta}\right)}{m-s} \approx \frac{ns + \ln\left(\frac{1}{\delta}\right)}{m} \tag{3.4}$$

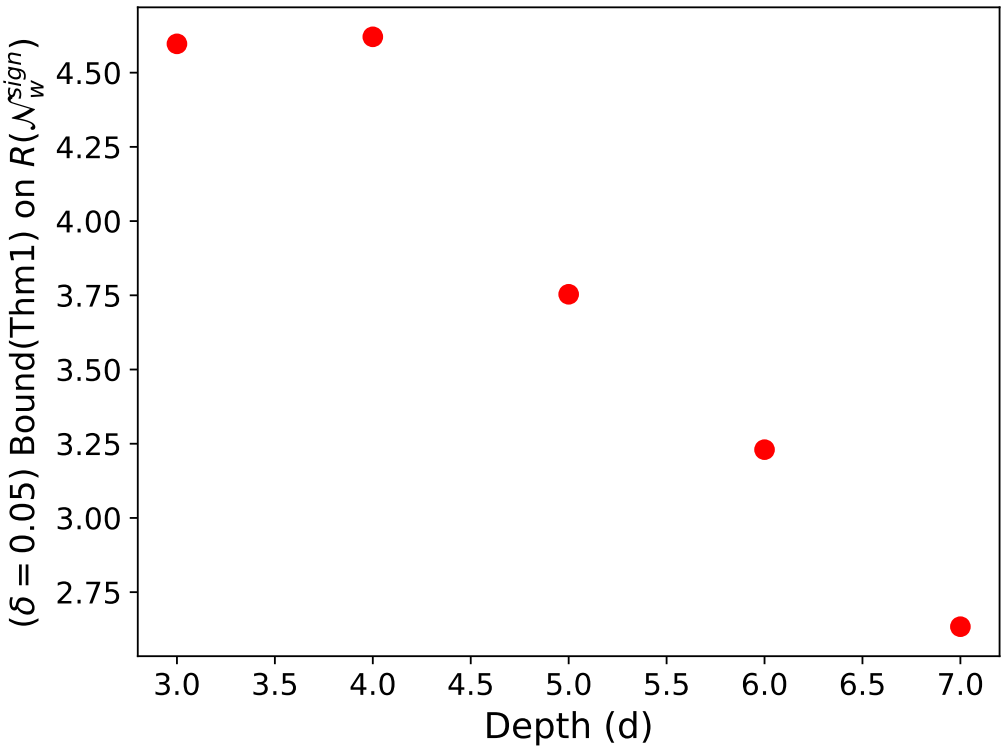



Figure 3.5: Numerical Value of the Risk Upper Bound, $\mathcal{F}(m,n,s,\delta)$, in Theorem 3.3.5 as Depth $d$ Varied: The outcomes and values of the experiments in the previous section (see Appendix A.2.3 for details) were used for the generalization bound in Theorem 3.3.5 *as if* the assumptions apply.

It is interesting take note of what the bound does *not* explicitly depend on. Very often, learning bounds for linear classifiers will depend on the norm of linear classifier. But one of the most striking properties of deep networks is their ability to generalize even

without penalization of weight norms. In an effort to comment on this, our bound by design does not depend on the norm of $\bar{w}$. Also, the architecture dimensions and input dimension, $f$ are not explicitly mentioned in the bound. This is because the effect of changing the input dimension (or architecture dimension) is captured by changes in the feature map definition, $\phi(x, w)$. In particular, the dimension of the embedding space $\mathcal{F}$ may change. But, just as the feature space dimension is often absent from SVM theory, the input dimension $f$ makes no appearance in our results. In fact, if the feature map $\phi(\cdot, w)$ were known *a priori*, (reducing to learning to an SVM problem) then also $n$ would not appear in our bound.

Sample compression bounds, as in Theorem 3.3.5, are based on the premise that each learned classifier is specified by some small enough subset of the training data. For example, a SVM model can always be identified by its set of $s$ support vectors. On the contrary, if $K > s$ training samples are "memorized" during learning, then the SVM model cannot be specified by $s < K$ samples. Suppose, *a priori*, that the SVM model has at most $s$ support vectors, then there are some $m - s$ training samples on which the learned model has minimal dependence. Thus, the risk on those $m - s$ samples should approximate the true risk. This intuitively explains why specifying a DNN by means of a subset of the training data is related to generalization.

A more general approach allows subsets of training samples to specify a sufficiently small set of $N$ models containing the learned model. The bound produced by this generalization is related to the previous $N = 1$ bound by an additive factor of $\ln(N)/m$. Note that, for any fixed $T \in (\mathcal{X} \times \mathcal{Y})^s$, at most $2^{s+ns+n}$ different DNN classifiers, $\{x \mapsto \mathcal{N}_w^{sign}(x) : w \in \mathcal{W}\}$, can simultaneously have weights $w$ that satisfy the maximum margin Assumption 2 for some set of network support vectors contained in $T$.

Conceptually, there are two steps to our argument:

1. Theorem 3 will show that for each $\bar{w} \in \mathcal{F}$, there are only $2^n$ many classifiers $\mathcal{N}_w^{sign}$

with $\bar{w} = \Lambda(w)$. [4] This is true even without Assumption 2. The DNN is entirely specified by $\bar{w}$ modulo at most $n$ bits needed to determine weight signs.

2. When also we have Assumption 2, we can avoid specifying $\bar{w}$ directly by instead supplying the image of $s$ support vectors under the feature map $\phi(\cdot, w)$. We can do this by supplying $s$ samples and at most $ns$ bits determining their image under $\bar{\sigma}(\cdot, w)$.

**Theorem 3.** *For $P \subset \mathcal{F}$, define $\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(P)) \triangleq \{\mathcal{N}^{sign}(\cdot, w) : w \in \mathcal{W}, \Lambda(w) \in P\}$. For $\bar{w} \in \mathcal{F}$, define $\mathbb{R}^+\bar{w} \triangleq \{\alpha\bar{w} : \alpha > 0\}$.*

*Then*

$$|\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(\mathbb{R}^+\bar{w}))| \leq 2^n \tag{3.5}$$

*where $n = d\Omega$ is the number of neurons in the Leaky-ReLU network.*

Though we primarily use Theorem 3 as a tool to prove Theorem 3.3.5, it has its own interesting interpretation as a characterization of the expressivity gap between SVMs and NNs, which we leave for Appendix A.2.5.

There are two main ideas underlying Theorem 3 (Proof in Appendix A.2.4). Note that $\Lambda(w)$ only describes products of weights, which creates ambiguity in the scale of individual weight parameters. For example, replacing entries of $w$, $(A^{(l+1)}, A^{(l)})$, with $(\alpha A^{(l+1)}, \alpha^{-1} A^{(l)})$, does not change $\Lambda(w)$ for any choice of $\alpha > 0$. This implies $|\Lambda^{(-1)}(\bar{w})| = \infty$. However, the nonlinearity $\rho$ commutes with positive diagonal matrices, and class predictions are obtained as the sign of the network outputs, $sign \circ \mathcal{N}$. Theorem 3 implies that replacing $w$ with $\mathcal{N}_w^{sign}$ eliminates scale information that causes ambiguity in $w$ given $\Lambda(w)$ alone. In other words, the set $\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(\mathbb{R}^+\bar{w}))$ can potentially be finite as its elements cannot be indexed by a continuously-valued positive-scale parameter.

Given only $\Lambda(w)$, the second type of ambiguity in the weights $w$ is that of sign

[4] Though quantitatively $2^n$ is *also* the number of neuron "on"/"off" configurations, this similarity seems largely coincidental as we arrive at $2^n$ by counting allowable weight sign configurations.

parity. Overall, $\bar{w} = \Lambda(w)$ forms a system of equations (one per path) involving products of the variables $A_{i,j}^{(l)}$ that cannot be solved without additional information. If the sign of each network weight was known, we could determine the network weights by solving a system of linear equations $\ln(|\bar{w}|) = \ln(|\Lambda(w)|)$ in the variables $\ln(|A_{i,j}^{(l)}|)$. This provides a bound of $2^P \approx 2^{d\Omega^2}$ over the number of possibilities of $sign(w)$, where $P$ is the number of parameters. However, this would translate to a bound governed by the ratio of the number of parameters to samples. Such a bound is slightly unexciting in the context of deep learning, where often $P >> m$. Another idea contained in Theorem 3 is that one can replace the number of parameters with the number of neurons. The knowledge of $sign(\bar{w})$ can be used to reduce the bound to $2^n = 2^{d\Omega}$, where $n$ is the total number of neurons. In fact, it is an interesting intermediate result that given $\bar{w}$, $w$ is determined entirely by the sign of just $n$ weights in a particular geometric configuration (see Figure A.23). (Interestingly, the *sign of the weights*, which featured prominently in earlier experiments (Figure A.20), reappears as relevant theoretical quantity). Consequently, we arrive at an improved bound governed by $n/m$.

The bound on the true risk, $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$, depends on bounding the log of the number of classifiers consistent with any given training set. To summarize which steps in our bound over classifiers feature most prominently in our bound on $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$, we tabulate the results from previous discussion in Table 3.1. As each step in our argument has an additive effect on the bound, we can speak of the "contribution of each step" to the bound on $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$.

### 3.4.1 On Improvements and Further Research

A significant reduction in the generalization bound of Theorem 3.3.5 to well below $\mathcal{O}(ns/m)$ may be possible in practical settings. Specifically, the largest term in the numerator of the

Table 3.1: Additive Effect on Sample Complexity

| **STEP** | **#WAYS** | (m-s)$\mathcal{F}(m, n, s, \delta)$ |
|---|---|---|
| $S^{(m)} \rightarrow$NSVs | $2^s\binom{m}{s}$ | $s \ln\left(\frac{m}{s}\right) + O(s)$ |
| NSVs$\rightarrow \Lambda(\mathcal{W})$ | $2^{ns}$ | $ns$ |
| $\Lambda(\mathcal{W}) \rightarrow \mathcal{W} \rightarrow \mathcal{Y}^{\mathcal{X}}$ | $2^n$ | $n$ |

bound, $ns$, arises due to a bound over path activations on NSVs that allows each sample, $x$, to choose its embedding $\bar{\sigma}_w(x)$ independently. Experimentation, however, suggests that this bound is pessimistic under practical circumstances, and training samples are instead embedded in a co-dependent manner.

To understand the dispersion of $\{\bar{\sigma}(x, w) : (x, ) \in S^{(m)}\}$, we train a ReLU network with depth $d = 3$ and width $\Omega = 10$ for $50,000$ iterations on MNIST. As an output, we measure the number of unique patterns of path activation in the network, $|\{\bar{\sigma}(x, w) : (x, ) \in S^{(m)}\}|$, over either training or test data as the number of training samples $m$ varied (Table 3.6). For emphasis, we count $\bar{\sigma}(x^i, w) \neq \bar{\sigma}(x^j, w)$ as distinct patterns if even a single neuron, say $i_l$ in layer $l$, behaves differently on $x^i$ and $x^j$, i.e., $\sigma^{(l)}(x, w)_{i_l} \neq \sigma^{(l)}(x, w)_{i_l}$.

Based on previous experiments (see Figure 3.1), a reasonable guess for the number of support vectors is $s = |S^{(s)}| \approx 0.1m$. If, in practice, for each $j \in [m]$, the embedding of the $j^{\text{th}}$ NSV, $\bar{\sigma}(x^j, w)$, was unconstrained by that of the others, $\{\bar{\sigma}(x, w) : x \in S^{(s)} - x^j\}$, then with high likelihood we would expect to see around $0.1m$ unique path activations counted among support vectors. Although we do not measure this directly, we measure a relatively pessimistic upper bound instead by counting the number of unique path activations over *the entire training set*. We observe that $|\{\bar{\sigma}(x, w) : x \in S^{(s)}\}| \leq |\{\bar{\sigma}(x, w) : x \in S^{(m)}\}| \approx 0.01m$ (Table 3.6). The number of unique test embeddings of the $10k$ test samples are also relatively few (second row). This suggests that the embeddings, $x \mapsto \phi(x, w) = \bar{\sigma}(x, w) \otimes x$,

Figure 3.6: Unique Sets of Active Paths Over Inputs

| $m = \lvert S^{(m)} \rvert$ | 100 | 500 | 5000 | 20000 | 50000 |
|---|---|---|---|---|---|
| $\lvert\{\bar{\sigma}(x_{train})\}\rvert$ | 49 | 75 | 210 | 282 | 711 |
| $\lvert\{\bar{\sigma}(x_{test})\}\rvert$ | 75 | 153 | 240 | 265 | 468 |

over training and test data are actually tightly coordinated, which may help further bound the number of possible embeddings of a given set of support vectors.

Future research: We recognize considerable further experimentation is needed, particularly one would like to know "under what circumstances does Assumption 2 hold?". We point out that to even suspect that this is an interesting question to ask requires the experimental and theoretical contributions of this paper–sometimes finding the right question is difficult in and of itself. These contributions are themselves starting points: The existence of a relationship between the number of support vectors and the architecture parameters is intriguing but warrants further exploration. And, the theoretical generalization bounds we present that depend on the number of support vectors are notable for being the only sample-compression based bounds for neural networks, but by no means do they represent the most sharpened bounds possible. Our future goal is to develop improved bounds by continuing this line of thought in the future.

## 3.5 Conclusion

In this paper, we motivate and develop the study of Leaky-ReLU type deep neural networks as SVM models with embedding maps locally independent of the weights. Towards this end, we make an idealized assumption, that the neural network results in a “max-margin" classifier. We provide an example of an experimental observation involving the configuration of the signs of the weights that is difficult to reconcile without the lens of this max-margin

assumption.

Exploring the implications of this assumption, we demonstrate the experimental behavior and theoretical relevance of resulting “network support vectors", and draw parallels between conventional support vectors and NSVs. Subsequently, we develop a generalization bound for deep neural networks that are depth-dependent in Theorem 3.3.5. The conceptual shift underlying the concrete ideas in the paper is to *parameterize* the neural network not by the weights, but as the solution to one of a small number of optimization problems.

# Chapter 4

# Deep Logical Circuits as Neural Networks: Generalization and Interpretation [1]

Not only are Deep Neural Networks (DNNs) black box models, but also we frequently conceptualize them as such. We lack good interpretations of the mechanisms linking inputs to outputs. Therefore, we find it difficult to analyze in human-meaningful terms (1) what the network learned and (2) whether the network learned. We present a hierarchical decomposition of the DNN discrete classification map into logical (AND/OR) combinations of intermediate (True/False) classifiers of the input. Those classifiers that can not be further decomposed, called atoms, are (interpretable) linear classifiers. Taken together, we obtain a logical circuit with linear classifier inputs that computes the same label as the DNN. This circuit does not structurally resemble the network architecture, and it may require many fewer parameters, depending on the configuration of weights. In these cases, we obtain

[1] **This chapter is based on material from the publication (Snyder and Vishwanath [2020a]).**

simultaneously an interpretation and generalization bound (for the original DNN), connecting two fronts which have historically been investigated separately. Unlike compression techniques, our representation is *exact*. We motivate the utility of this perspective by studying DNNs in simple, controlled settings, where we obtain superior generalization bounds despite using only combinatorial information (e.g. no margin information). We demonstrate how to "open the black box" on the MNIST dataset. We show that the learned, internal, logical computations correspond to semantically meaningful (unlabeled) categories that allow DNN descriptions in plain English. We improve the generalization of an already trained network by interpreting, diagnosing, and replacing components *within* the logical circuit that is the DNN.

## 4.1 Introduction

Deep Neural Networks (DNNs) are among the most widely studied and applied models, in part because they are able to achieve state-of-the-art performance on a variety of tasks such as predicting protein folding, object recognition, and playing chess. Each of these domains was previously the realm of many disparate, setting-specific, algorithms. The underlying paradigm of Deep Learning (DL) is, by contrast, relatively similar across these varied domains. This suggests that the advantages of DL may be relevant in a variety of future learning applications rather than being restricted to currently-known settings.

The philosophy of investigating deep learning has typically focused upon keeping experimental parameters as realistic as possible. A key advantage enabled by this realism is that the insights from each experiment are immediately transferable to settings of interest. However, this approach comes with an important disadvantage: Endpoints from realistic experiments can be extremely noisy and complicated functions of variables of interest, even for systems with simple underlying rules. Newton's laws are simple, but difficult to discover

except in the most controlled of settings.

The goal of our study is to understand the relationship between generalization error and network size. We seek to clarify why DNN architectures that can potentially fit all possible training labels are able to generalize to unseen data. Specifically, we would like to understand why increasing the capacity of a DNN (through increasing the number of layers and parameters) is not always accompanied by an increase in test error. To this end we study fully-connected, Gaussian-initialized, unregularized, binary classification DNNs trained with gradient descent to minimize cross-entropy loss on 2-dimensional data [2]. Even in such simple settings, generalization is not yet well-understood (as bounds can be quite large for deep networks), and our goal is to take an important step in that direction.

In adopting a minimalist study of this generalization phenomenon, the view taken in this paper is aligned with that expressed by Ali Rahimi in the NIPS2017 "Test of Time Award" talk: "This is how we build knowledge. We apply our tools on simple, easy to analyze setups; we learn; and, we work our way up in complexity... Simple experiments — simple theorems are the building blocks that help us understand more complicated systems."

—Rahimi [2017]

Our contributions are:

1. We give an intuitive, visual explanation for generalization using experiments on simple data. We show that prior knowledge about the training data can imply regularizing constraints on the image of gradient descent independently of the architecture. We observe this effect is most pronounced at the decision boundary.
2. We represent exactly a DNN classification map as a logical circuit with many times fewer parameters, depending on the data complexity.
3. We demonstrate that our logical transformation is useful both for interpretation and

[2] Though the generalization of DNNs has been attributed in part to SGD, dropout, batch normalization, weight sharing (e.g. CNNs), etc., none of these are strictly necessarily to exhibit the apparent paradox we describe.

improvement of trained DNNs. On the MNIST dataset we translate a network "into plain English". We improve the test accuracy of an already trained DNN by debugging and replacing *within the logical circuit* of the DNN a particular intermediate computation that had failed to generalize.

4. We give a formal explanation for generalization of deep networks on simple data using classical VC bounds for learning Boolean formulae. Our bound is favorable to state of the art bounds that use more information (e.g. margin). Our bounds are extremely robust to increasing depth.

## 4.2 Setting and Notation

In this paper we study binary classification ReLU fully connected deep neural networks, $\mathcal{N} : \mathbb{R}^{\Omega 0} \mapsto \mathbb{R}$, that assign input $x$, label $y \in \{False, True\}$ according to the value $[\mathcal{N}(x) \geq 0]$. This network has $d$ hidden layers, each of width $\Omega l$, indexed by $l = 1, \ldots, d$. We reserve the index $0[d+1]$ for the input[output] space, so that $\Omega d + 1 = 1$. Our ReLU nonlinearities, $R(x)_i = \max\{0, x_i\}$, are applied coordinate-wise, interleaving the affine maps defined by weights $A^{(l)} \in \mathbb{R}^{\Omega l+1 \times \Omega l}, b^{(l)} \in \mathbb{R}^{\Omega l+1}$. These layers compute recursively

$$\mathcal{N}^{(l+1)}(x) \triangleq b^{(l+1)} + A^{(l)} R(\mathcal{N}^{(l)}(x)).$$

Here, we include the non-layer indices $0$ and $d+1$ to address the input, $x = \mathcal{N}^{(0)}(x)$, and the output, $\mathcal{N}(x) = \mathcal{N}^{(d+1)}(x)$, respectively.

For a particular input, $x$, each neuron occupies a binary "state" according to the sign of its activation. The set of inputs for which the activation of a given neuron is identically $0$ comprises a "neuron state boundary"(NSB), of which we consider the decision boundary to be a special case by convention. We can either group these states by layer

or all together to designate either the layer state, $\sigma^{(l)}l(x) \in \{0,1\}^{\Omega l}$, or the network state, $\bar{\sigma}(x) = (\sigma^{(l)}1(x), \ldots, \sigma^{(l)}d(x))$, respectively.

We consider our training set, $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, to represent samples from some distribution, $\mathcal{D}$. We define generalization error of the network to be the difference between the fraction correctly classified in the finite training set and in the overall distribution. We define training as the process that assigns parameters the final value of unregularized gradient descent on cross-entropy loss.

## 4.3 Insights From a Controlled Setting

### 4.3.1 Finding the Right Question

Note that one is *not* always guaranteed small generalization error. There are many settings where DNNs under-perform and have high generalization error. For our purposes, it suffices to recall that when the inputs and outputs $(X, Y) \sim \mathcal{D}$ are actually independent, e.g., $Y|X \sim Bernoulli(1/2)$, neural networks still obtain zero empirical risk, which implies the generalization error can be arbitrarily bad in the worst case [Zhang et al., 2017]. From this, we can conclude that making either an explicit or implicit assumption about the dataset, the data, or both, is *strictly necessary* and unavoidable. At the very least, one must make an assumption which rules out random labels with high probability.

Notice that the only procedural distinction between a DNN that will generalize and one that will memorize is the dataset. Those network properties capable of distinguishing learning from memorizing, e.g., Lipschitz constant or margin, must therefore arise as secondary characteristics. They are functions of the dataset the network is trained on.

We want clean descriptions of DNN functions that generalize. By the above discussion, these are the DNNs that inherit some regularizing property from the training data through the gradient descent process. What sort of architecture agnostic language allows for

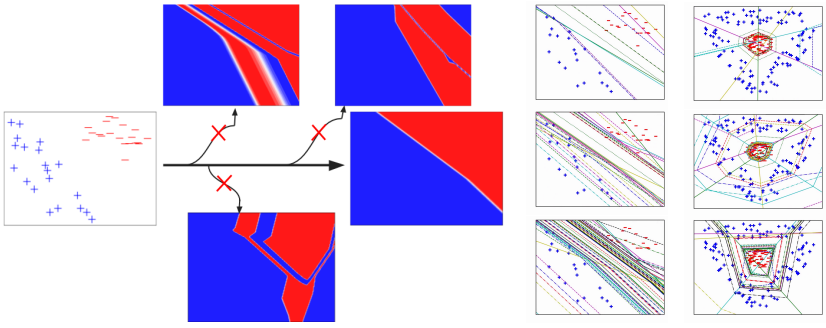
succinct descriptions of trained DNNs exactly when we make some strong assumption about the training set?

### 4.3.2 A Deep Think on Simple Observations

> We find in our experiments that DNNs of any architecture trained on linearly classifiable data are almost always linear classifiers (Fig 4.1).

Is this interesting? Let us consider: though our network has enormous capacity, in this fixed setting of linearly separable data, the deep network behaves as though it has no more capacity than a linear model. When we discuss capacity of a class a functions, we ordinarily consider a hypothesis class consisting of networks indexed over all possible values of weights (or perhaps in a unit ball), since no such restrictions are explicitly built into in the learning algorithm. For a large architecture, such as ArchIII, this hypothesis class consists of a tremendous diversity of decision boundaries that fit the data. However, here we observe only a subset of learners: Not every configuration of weights nor every hypothesis is reachable by training with gradient descent on linearly classifiable data. Consider a learning the DNN weights corresponding to the $9$ layer network, ArchIII. The VCDim of such hypotheses indexed by every possible weight assignment is $1e6$, which is unhelpfully large. But, have we measured the capacity of the correct class? If we instead use the class reachable by gradient descent, then data assumptions, which are in some form necessary, by constraining the inputs to our learning algorithm in turn restrict our hypothesis class. Linear separability is a particularly strong data assumption which reduces our the VC dimension of our hypothesis class from $1e6$ to $3$. We conclude:

To ensure generalization of unregularized DNN learners, not only are data assumptions *necessary*, but also strong enough assumptions on the training data are themselves *sufficient* for generalization.



(a) Regularizing Effect of Linearly Separable Data (b) DataI (c) DataII

Figure 4.1: Structural organization of the decision boundary(DB) and NSBs (where each neuron changes from "on" to "off") of trained DNNs as the data (Fig. 4.1a) and architecture complexity varies (Figs.4.1b, 4.1c). In Fig. 4.1a, *if and only if* we include additional noisy training data to the linearly classifiable DataI can we avoid learning an (essentially) linear classifier. Regularity is not tied to data fit but data structure: all 4 ArchIII classifiers have 0 training error and vanishing loss on the same original data. In columns (4.1b,4.1c) we plot all NSBs (different linestyle [color] distinguishes NSBs of neurons in different [the same] hidden layer) and the DB (dotted). We see that for fixed dataset, increasing the architecture size (moving down a column) does not qualitatively change the learned DB. Additional layers may add more NSBs, but these organize during training in redundant, parallel shells that do not make the DB more complex. Only those NSBs that intersect the DB influence the DB and cause it to bend. Not only is the number of intersections between the DB and NSBs minimized, but also they separate from one another during training, as if by some (regularizing) "repulsive force", most readily apparent in row 2 col 4.1b (and in the Supplemental animations), that repels the NSBs from the decision boundary. There are several sources of relevant additional information for this figure. The Appendix contains Figures A.27 and A.28, which are useful to quickly visually appreciate the architectures, ArchI,II,III, and training data, DataI,II,III, that we use throughout. It also contains Figure A.24, which along with the Supplemental Animations, elaborate on these NSB diagrams in number, kind, and size.

In Figures 4.1b,4.1c, we see that a DNN with more parameters learns a more complicated *function* but not a more complicated *classifier*. For example, the number of linear regions does seem to scale with depth for fixed dataset. However, instead of intersecting the decision boundary or one another, these additional NSBs form redundant onion-like structures parallel to the decision boundary.

Since we have argued that learning guarantees in this setting are essentially equivalent to training data guarantees, capacity measures on the learned network $\mathcal{N}$ that imply generalization must somehow reflect the regularity of the data that was originally trained on. Conversely, the factors not determined by the training data structure should *not* factor into the capacity measure. For example, we desire bounds which do not grow with depth.

A capacity measure on $\mathcal{N}(x)$ that is determined entirely by restricting $\mathcal{N}$ to a neighborhood of its decision boundary accomplishes both such goals. The effect of the data is captured because the geometry of this boundary closely mirrors the that of the training data in arrangement and complexity. Consider also that behavior of $\mathcal{N}$ at the decision boundary is still is sufficient to determine each input classification and therefore the generalization analysis is unchanged. We claim restricting $\mathcal{N}$ to near the decision boundary destroys the architecture information used to parameterize $\mathcal{N}$. More specifically, only the existence of neurons whose NSB intersects the decision boundary can be inferred from observation of inputs $x$ and outputs $\mathcal{N}(x)$ near the boundary. For example, this restriction is the same both for a linear classifier and for a deep network that learns a linear classifier with $50$ linear regions (as in Fig. 4.1b).

## 4.4 Opening the Black Box through Deep Logical Circuits

The key idea is to characterize the decision boundary of the DNN by writing the discrete valued classifier $x \mapsto \mathcal{N}(x) \geq 0$ as a *logical combination* of a hierarchy of intermediate

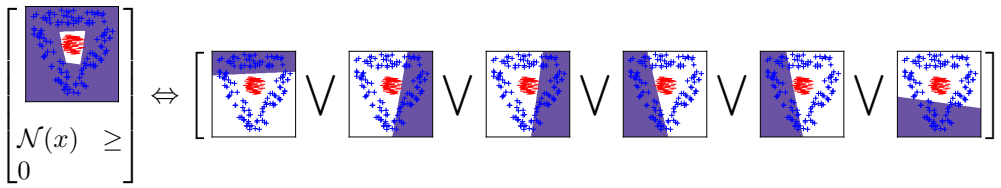
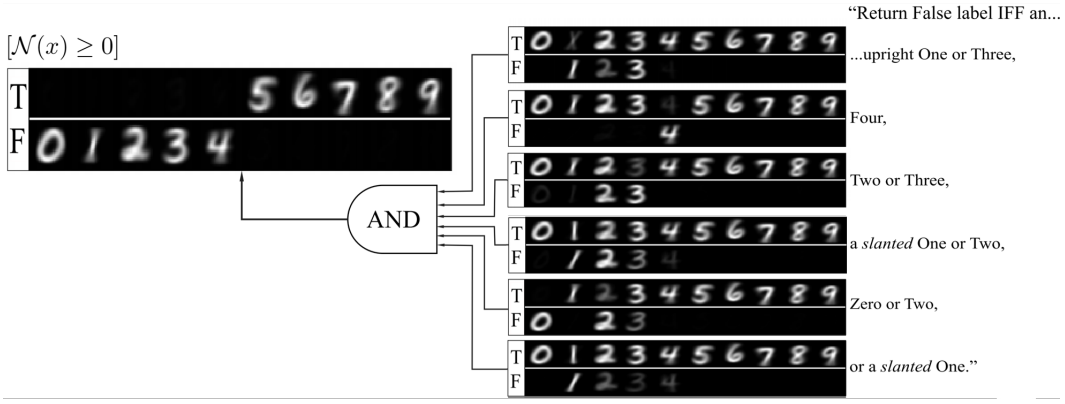



Figure 4.2: The logical representation of the classifier learned by the ArchIII network to classify the DataII data (shaded region classified True). Our algorithm outputs the RHS: the rules the DNN uses to assign a positive label. For at least one of the images on the right, the input must lie in the blue region. These rules are not apparent by inspection of the $\sim 7e5$ network parameters.

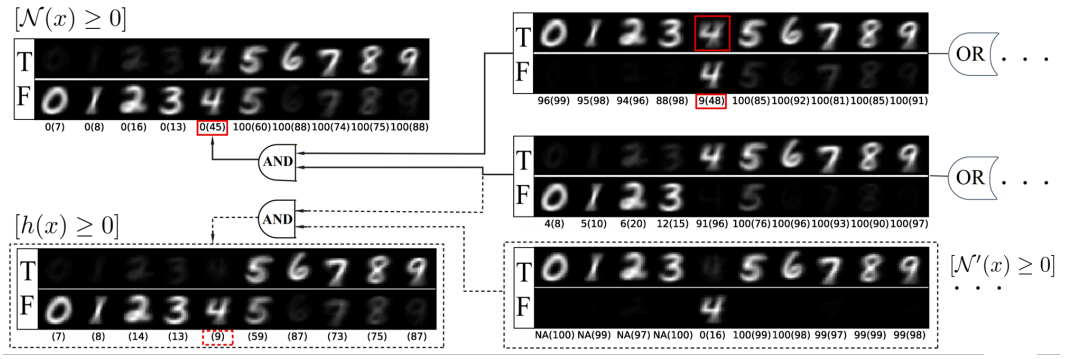
classifiers. These intermediate classifiers identify higher order features useful for the learned task. The final result will be a logical circuit which produces the same binary label as the DNN classifier on all inputs. Finding a circuit that is "simple" is our key to both interpretability and generalization bounds.

One such example is shown in Figure 4.2. We show that our method translates a $1e^6$ DNN classification map into an OR combination of just $6$ linear classifiers. To functionally emphasize, our representation *is* the DNN. It applies to all inputs: training, test, and adversarial alike.

When we train networks on the MNIST dataset, the learned circuit is more complicated, but we can still understand "role" of the intermediate classifiers within the circuit. By probing the internal circuitry with training and validation inputs, we can interpret the role of the components by cross-referencing with semantic categories (perhaps provided by a domain expert). *A priori*, there is no reason why this should be possible: The high level features a DNN learns as useful for this task are not obliged to be those that humans identify. However we see experimentally extremely encouraging evidence for this. When we group digits $0-4$ and $5-9$ into binary targets for classification, the DNN virtually always learns individual digits as intermediate steps within the logical circuit (Figure 4.3). For space, only




(a) Trained DNN with a Concise "English" Description




(b) A circuit with localized memorization

Figure 4.3: Selected subsets of the logical circuits corresponding to binary classification DNNs trained on the MNIST dataset (caption continued on next page.)

Figure 4.3: (Caption continued from previous page) Selected subsets of the logical circuits corresponding to binary classification DNNs trained on the MNIST dataset. Each $2 \times 10$ array represents a different binary classifier within the network circuit, which assigns True or False to every input image. In Fig. 4.3a[Fig. 4.3b] training[test] images of number $l$ contribute in the $l^{th}$ column to the brightness of either top or the bottom row of every array. The choice of row corresponds to whether corresponding classifier outputs True or False. The diagram reads right to left along solid arrows terminating in the leftmost array corresponding to the DNN binary output $[\mathcal{N}(x) \geq 0]$. The training objective only explicitly distinguishes $0-4$ from $5-9$. Yet, we see that the intermediate logical computations the network learns delineate semantically meaningful subcategories. In Fig. 4.3a, the DNN internal logic even admits a description *in plain English*. We show in Figure 4.3b how the internal logical circuitry *within the DNN* can be tweaked to improve generalization. Connected with solid lines, we see a network that has overfit badly ($1.0, 0.78$ train and test accuracy). The percentage [in parenthesis] under each column indicates how that training[test] digit is assigned True. We see that the intermediate classifier (middle right) struggles to separate Four from the positive labels. A second classifier (top right) is dedicated to learning to identify Fours as False, allowing the network to fit the training data. However, by comparing training and test performance, we can see that these Fours are not learned but memorized: As shown in solid rectangles, the intermediate and final classifier, respectively, assign True to $9\%[48\%]$ and $0\%[45\%]$ of the Fours in the training[test] set, accounting for the bulk of the generalization error! In practice, this network would be discarded and retrained from scratch. Since we now have access to the internal logic of the network, we are instead able to surgically replace the memorizing component. The first step we have done implicitly: we use *domain knowledge* to interpret the component function as "excluding Fours". We then train a second network, we call a "prosthetic", learning $[\mathcal{N}'(x) \geq 0]$ (with the same settings and data), to label $4$ as False and $5-9$ as True. We can then excise the memorizing component, replacing its role in the logical circuit with the prosthetic (bottom right) to obtain a new classifier consisting of the three classifiers connected by dotted lines. The classifier we engineer ($[h(x) \geq 0]$ bottom left) does better on Fours, $45\% \rightarrow 9\%$ classified True (dotted rectangle) and has higher test accuracy overall ($.78 \rightarrow .83$).

those circuit components closest to the output are shown. A more involved circuit study is available in Figure A.25.

The dichotomy presented is that Fig 4.3a demonstrates the importance of our method to interpretability, while Fig 4.3b, demonstrates the importance toward improving generalization. Although interpretability and generalization are usually studied separately, understanding "what $\mathcal{N}$ has learned" is actually very closely related to understanding "what $\mathcal{N}$ has memorized". In fact, one of the takeaways from Figure 4.3 is that the *mechanism of memorization* itself can have interpretation. In Figure 4.3b we exploit such an interpretation to improve generalization error by "repairing" the defect.

## 4.5 A Theory of DNNs as Logical Hierarchies

In this section, starting with any fixed DNN classifier, we show how to construct, simplify, measure complexity of, and derive generalization bounds for an equivalent logical circuit. These bounds apply to the original DNN. We show they compare favorably with traditional norm based capacity measures.

### 4.5.1 Boolean Conversion: Notation and Technique

In our theory, we designate $\mu$ and $\tau$ as special characters with dual roles and identical conventions. We consider the symbols, $\mu, \tau$, to represent binary vectors that index by default over all binary vectorsand implicitly promote to diagonal binary matrices, $Diag(\mu), Diag(\tau)$, for purposes of matrix multiplication. For matrices, $M$, we define $(M_{\pm})_{i,j} = \max\{0, \pm M_{i,j}\}$. To demonstrate, we have for $d = 1$: $\mathcal{N}(x) = b^{(1)} + \max_{\mu} A_{+}^{(1)} \mu(b^{(0)} + A^{(0)}x) - \max_{\tau} A_{-}^{(1)} \tau(b^{(0)} + A^{(0)}x)$. In fact, we may write this as a MinMax or a MaxMin formulation by substituting, $-\max_{\tau} = \min -\tau$, and factoring out the Max and Min in either order. Our primary tool to relate to Boolean formulations is the following.

**Proposition 2.** *Let $f : \mathcal{A} \times \mathcal{B} \mapsto \mathbb{R}$. Then we have the following logical equivalence:*

$$\left[\max_{\alpha\in\mathcal{A}} \min_{\beta\in\mathcal{B}} f(\alpha,\beta) \geq 0\right] \iff \bigvee_{\alpha\in\mathcal{A}} \bigwedge_{\beta\in\mathcal{B}} \left[f(\alpha,\beta) \geq 0\right]$$

We classify network states, $\bar{\Sigma} = \bar{\Sigma}_+ \cup \bar{\Sigma}_-$, in terms of the output sign, $\bar{\Sigma}_\pm = \{\bar{\sigma}(x) | \pm\mathcal{N}(x) \geq 0\}$. We use $\bar{\Sigma}_0 = \bar{\Sigma}_+ \cap \bar{\Sigma}_-$ for those states at the boundary. For $J \subset [d]$, we define $\bar{\Sigma}^J[\bar{\Sigma}_0^J]$ to be the projection of $\bar{\Sigma}[\bar{\Sigma}_0]$ onto the coordinates indexed by $J$. As a shorthand, we understand the symbols $\bar{\mu}^{[k]} = (\mu^1, \ldots, \mu^k)$ and $\bar{\mu} = \bar{\mu}^{[d]} = (\mu^1, \ldots, \mu^d)$ to be equivalent in any context they appear together.

Define for every $\tau, \mu$, a linear function of $x$, $P^{(1)}(\mu, \tau, x) = b^{(1)} + A_+^{(1)}\mu^1(b^{(0)} + A^{(0)}x) - A_-^{(1)}\tau^1(b^{(0)} + A^{(0)}x)$, called the "Net Operand". We have $[\mathcal{N}(x) \geq 0] \Leftrightarrow \vee_\mu \wedge_\tau [P^{(1)}(\mu, \tau, x) \geq 0]$. To generalize to more layers, we can recursively define:

$$\begin{aligned} P^{(l+1)}(\bar{\mu}^{[l+1]}, \bar{\tau}^{[l+1]}, x) &= A_+^{(l+1)}\mu^{l+1}P^{(l)}(\bar{\mu}^{[l]}, \bar{\tau}^{[l]}, x) \\ &- A_-^{(l+1)}\tau^{l+1}P^{(l)}(\bar{\tau}^{[l]}, \bar{\mu}^{[l]}, x) + b^{(l+1)}. \end{aligned}$$

One can derive by substitution that $P^{(d)}(\bar{\sigma}(x), \bar{\sigma}(x), x) = \mathcal{N}(x)$. This choice of $\bar{\mu} = \bar{\tau} = \bar{\sigma}(x)$ will always be a saddle point solution to Eqn 4.1 in the following theorem.

**Theorem 4.** *Let $P^{(d)}$ be the net operand for any fully-connected ReLU network, $\mathcal{N}$. Then,*

$$\mathcal{N}(x) = \max_{\mu^d} \min_{\tau^d} \cdots \max_{\mu^1} \min_{\tau^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \tag{4.1}$$

$$\left[\mathcal{N}(x) \geq 0\right] \Leftrightarrow \bigvee_{\mu^d} \bigwedge_{\tau^d} \cdots \bigvee_{\mu^1} \bigwedge_{\tau^1} \left[P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \tag{4.2}$$

Notice that we can derive the second line (4.2) from the first (4.1) by recursive

application of Proposition 2. Since we index over all binary states, the number of terms in our decomposition (Eqn 4.2) is extremely large. Though (it turns out) we may simply matters considerably by indexing instead over network states, $\bar{\Sigma}$. The next Theorem says that when the right hand side(RHS) of Eqn. 4.1 is indexed by only those states realized at the decision boundary, $\bar{\Sigma}_0$, the RHS still agrees with $\mathcal{N}(x)$ in *sign*, but necessarily numerical value. Thus they are equivalent classifiers.

**Theorem 5.** *Let $\mathcal{N}$ be a fully-connected ReLU network with net operand, $P^{(d)}$, and boundary states, $\bar{\Sigma}_0$. Then,*

$$[\mathcal{N}(x) \geq 0] \Leftrightarrow \bigvee_{\mu^d \in \bar{\Sigma}_0^d} \bigwedge_{\tau^d \in \bar{\Sigma}_0^d} \bigvee_{\{\mu^{d-1} | (\mu^{d-1}, \mu^d) \in \bar{\Sigma}_0^{d-1,d}\}} \cdots \bigvee_{\{\mu^1 | \bar{\mu} \in \bar{\Sigma}_0\}} \bigwedge_{\{\tau^1 | \bar{\tau} \in \bar{\Sigma}_0\}} \left[ P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0 \right] \tag{4.3}$$

The proofs for both Theorems 4 and 5 can be found in the Appendix A.3.6. We also include explicit pseudocode, "Network Tree Algorithm" (in Appendix A.3.5) for constructing our Logical Circuit from $\bar{\Sigma}_0$. Somehow, we find the actual python implementation more readable, which we have included in the supplemental named "network_tree_decomposition.py". We use this file to generate the readout in Figure A.29d (Appendix A.3.4) to provide tangible, experimental support for the validity of our conversion algorithm.

### 4.5.2 Formalizing Capacity for Logical Circuits

We repurpose the following theorem used by [Bartlett et al., 2017b] for ReLU networks data-independent VC dimension bounds.

**Theorem 6.** *(Theorem 17 in [Goldberg and Jerrum, 1995]): Let $k$,$n$ be positive integers and*

$f : \mathbb{R}^n \times \mathbb{R}^k \mapsto \{0, 1\}$ *be a function that can be expressed as a Boolean formula containing* $s$ *distinct atomic predicates where each atomic predicate is a polynomial inequality or equality in* $k + n$ *variables of degree at most* $d$*. Let* $\mathcal{F} = \{f(\cdot, w) : w \in \mathbb{R}^k\}$*. Then VCDim*$(\mathcal{F}) \leq 2k \log_2(8eds)$.

As a short hand, we refer to any Boolean formula satisfying the premises in the above theorem as class $(k, s, d)$. If we consider (for fixed $\bar{\Sigma}_0$) the complexity of learning the weights defining the linear maps in Eqn. 5, Jerrum's Theorem tells us that we are primarily concerned with the number of parameters being learned. Fortunately, we only pay a learning penalty for those weights distinguishable by neuron activations in $\bar{\Sigma}_0$. For example, within the same layer, a single neuron is sufficient model any collection of neurons which are always "on" or "off" simultaneously at the decision boundary. In general, we can restrict to a subset of $r_l = rank(\bar{\Sigma}_0^l)$ representative neurons without sacrificing expressivity at the boundary. We can additionally delete entire layers when $r_l = 1$. We use $\bar{r} \triangleq (r_0, r_1, \ldots, r_d)$ to group the dimensions of the reduced architecture into a single vector. Note that when $\mathcal{N}$ is a linear classifier, then $\bar{r}$ is a vector of all 1s. $r_l = 1$ at every layer.

Finally, we define $\Phi(\mathcal{N}) : \mathbb{R}^k \times \mathbb{R}^{\Omega 0}$ to be the Boolean function in Eqn 5 corresponding to the *reduced* network, whose depth we also overload as $d$, and take $k$ to be the number of parameters on which the formula depends. The formula has $s = |\bar{\Sigma}_0|^2$ inequalities. The explicit calculations for determining $k, s, d, \bar{r}$ are described Function $MinimalDescrip$ in the Generalization Bound Calculation, which is the first algorithm of Appendix A.3.5. The following is automatic given the discussion so far.

**Theorem 7.** *Let* $\mathcal{N} : \mathbb{R}^{\Omega 0} \mapsto \mathbb{R}$ *be a fully-connected ReLU network. Suppose the Boolean formula,* $\Phi(\mathcal{N})$*, is of class* $(k, s, d)$*. Define the hypothesis class* $\mathcal{H}_{\Phi(\mathcal{N})} \triangleq \{x \mapsto \Phi(\mathcal{N})(w, x) | w \in \mathbb{R}^k\}$*. Then*

1. $x \mapsto [\mathcal{N}(x) \geq 0] \in \mathcal{H}_{\Phi(\mathcal{N})}$

2. *VCDim*$(\mathcal{H}_{\Phi(\mathcal{N})}) \leq 2k \log_2(8esd)$

Of course, this bound only applies to the learned DNN if the hypothesis class $\mathcal{H}_{\Phi(\mathcal{N})}$ is implied in advance. To address (informally) the capacity for a single classifier, $\mathcal{N}$, we define $VC^{Bool}_{k,d,s}(\mathcal{N}) \triangleq 2k \log_2(8esd)$ to be the complexity of learning the parameters of the $(k, s, d)-$Boolean formula representing $\mathcal{N}$. This is an upper bound for the smallest complexity over formulae $\Phi$ and classes $\mathcal{H}_{\Phi(\mathcal{N})}$ containing $\mathcal{N} \geq 0$ as a member. In Figure 4.4, we train $\mathcal{N}$ to classify samples in DataIII and compare qualitatively our capacity measure, $VC^{Bool}_{k,d,s}(\mathcal{N})$, with those of other well-known approaches as we vary the network size and training duration and depth. We compare with methods which appear at first glance to make use of additional information—that of scale, norm, and margin—which should in principle produce tighter bounds.

And yet, we enjoy a comfortable edge over other comparable methods. Under all conditions, our bound seems to be orders of magnitude smaller than these other (well-respected) bounds. So, what is going on? In fact, it is *our* bound that is advantaged by using more (between-layer) information!.

We revisit the observation that a very deep DNN trained on linearly separable data is a linear classifier. We think that this simple characterization should somehow be accessible to our capacity measure through the weights. Linearly separable data represents, to us, the simplest, plausible, real-world proving ground for models of DNN generalization error. The methods with which we compare bound the distortion applied by each layer in terms of a corresponding weight matrix norm and accumulate the result. We should like our method to "realize" that the DNN classifier is linear, but this can not be discovered by scoring each layer. In fact, having an efficient Boolean representation is a *global* property that is sensitive to the relative configuration of weights across all layers. It is not information that is contained in the weight norms used by other methods, which destroy weight-sign information, among

other properties, on which linearity of the classifier depends. We would even suggest that our notion of regularity is "more nuanced" in the sense that whether a layer is well-behaved only makes sense to talk about within the context of the overall network.

Returning to Figure 4.4b, we observe that we our bound is relatively stable with respect to increasing architecture size and depth. This behavior is instructive in its distinction from that of uniform (data-independent) VC dimension bounds, $VC^{\text{NoData}}$, which depend on the architecture alone. That these bounds produce unreasonably large, vacuous bounds for over-parameterized models is widely known and often recited. Perhaps this notoriety has dissuaded combinatorial analyses of DNN complexity altogether. However, our results demonstrate that the vast majority of the bloat in these $VC^{\text{NoData}}$ bounds can be attributed to a lack of strong data assumptions and not to its combinatorial nature. When we compare against our own (also combinatorial) measure, $VC^{Bool}_{k,d,s}$, in Table A.4 we observe that $VC^{Bool}_{k,d,s}$ produces bounds that are orders of magnitude smaller. We account for this discrepancy as follow: While $VC^{\text{NoData}}$ yields weak bounds on generalization that always apply, $VC^{Bool}_{k,d,s}$ instead produces strong bounds that apply only when the data is nice. These bounds are smaller because the set of DNNs achievable by gradient descent on nice data is much more regular, and of smaller VC dimension. We explore this comparison in more depth in Appendix A.3.1.

Lastly, we offer some perspectives connecting our generalization studies to building better models in the future. There are many descriptions of complexity for DNNs. What makes ours a "good" one? All are equally valid in the sense that *every* one of them can prescribe some sufficiently strong regularity condition that will provably close the gap between training and test error. But, perhaps we should be more ambitious. We actually want to decrease model capacity *while also* retaining the ability to fit those patterns "typical" of real world data. While this second property is critical, it is also completely unclear how to

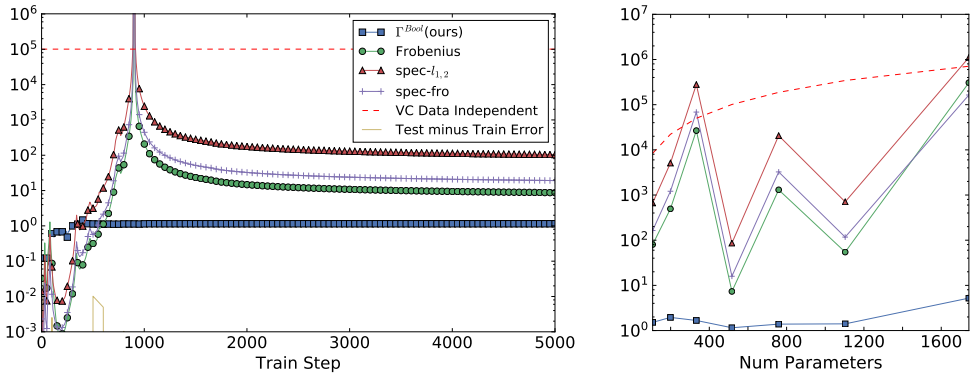
guarantee, even analyze, or even define unambiguously. We surmise that since our capacity measure $VC^{Bool}_{k,d,s}$ seems empirically to be *already* minimized when the data is sufficiently structured, we can hope (and plausibly hypothesize) that those patterns that can be learned efficiently by a function class where $VC^{Bool}_{k,d,s}$ is controlled *explicitly* will not differ from those suited to unregularized DNNs, where we expect the structured nature of real world data to *implicitly* regulate $VC^{Bool}_{k,d,s}$ already.

## 4.6 Related Work

Our discussion of the role the data plays in generalization is perhaps most similar to Arpit et al. [2017]. Many authors have studied the number of linear regions of a DNN before, usually focusing on a 2D slice or path through a the data [Serra et al., 2018, Raghu et al., 2017, Arora et al., 2018b], optionally including study of how these regions change with training [Hanin and Rolnick, 2019, Novak et al., 2018] or an informal proxy for network "complexity" [Zhang et al., 2018, Novak et al., 2018].

Formal approaches to explain generalization of DNNs fall into either "direct" or "indirect" categories. By direct, we mean that the bounds apply exactly to the trained learner, not to an approximation or stochastic counterpart. Ours falls under this category, so these are the bounds we compare to, including [Neyshabur et al., 2015, Bartlett et al., 2017a, Neyshabur et al., 2017b], which we compare to in Fig. 4.4. While our approach relies on bounding possible training labelings (VCdim), these works all rely on having small enough weight norm compared to output margin.

Indirect approaches analyze either a compressed or stochastic version of the DNN function. For example, PAC-Bayes analysis [McAllester, 1999] of neural networks [Langford and Caruana, 2002, Dziugaite and Roy, 2017] produces uniform generalization bounds over distributions of classifiers which scale with the divergence from some prior over classifiers.




(a) Capacity vs Training Step

(b) Capacity vs Num Parameters

Figure 4.4: Qualitative comparisons of bounds on the generalization error for networks trained on DataIII during training (Fig. 4.4a) and as additional layers are added (Fig. 4.4b). Though our bound is in terms of VC dimension only, we compare favorably with other bounds that additionally use margin. Interestingly, the spike in capacity that occurs around 1000 training steps is not reflected in our bound, but captured by others. Thus, our method may be blind to some interesting training dynamics, for example, a massive shift in the relative alignment of weight vectors that leaves the intersection system of neuron state boundaries unchanged. The empirical phenomenon of depth-invariant generalization error is consistent with the behavior of our bound (Fig. 4.4b). These trends are representative of all 9 experiments (Figure A.26).

Frobenius: $\frac{1}{m}\frac{1}{\gamma^2}\prod_{l=1}^{d}\|A^{(l)}\|_F^2$[Neyshabur et al., 2015]

spec-$l_{1,2}$:$\frac{1}{m}\frac{1}{\gamma^2}\prod_{l=1}^{d}\|A^{(l)}\|_2^2\sum_{l=1}^{d}\frac{\|A^{(l)}\|_{1,2}^2}{\|A^{(l)}\|_2^2}$[Bartlett et al., 2017a]

spec-fro: $\frac{1}{m}\frac{1}{\gamma^2}\prod_{l=1}^{d}\|A^{(l)}\|_2^2\sum_{l=1}^{d}h_l\frac{\|A^{(l)}\|_F^2}{\|A^{(l)}\|_2^2}$[Neyshabur et al., 2017b]

$\Gamma^{Bool}$(ours):$\min_{k,s,d}\sqrt{\frac{VC_{k,s,d}^{Bool}(\mathcal{N})}{m}}$ .

Recently, Valle-Pérez et al. [2019] produced promising such PAC-Bayes bounds, but they rely on an assumption that training samples the zero-error region uniformly, as well as some approximations of the prior marginal likelihood. Interestingly, they also touch on descriptional complexity in the appendix, which is thematically similar to our approach, but do not seem to have an algorithm to produce such a description. Another popular approach is to study a DNN through its compression [Arora et al., 2018a][Zhou et al., 2018]. Unlike our approach, which studies an equivalent classifier, these bounds apply only to the compressed version.

## 4.7 Conclusions

The motivation for our investigation was to describe regularity from the viewpoint of "monotonicity". Suppose that during training, the activations of a neuron in a lower layer separate the training data. While the specifics of gradient descent can be messy, there is no "reason" to learn anything other than a monotonic relationship (as we move in the input space) between the activations of that neuron, intermediate neurons in later layers, and the output. Two neurons related in this manner necessarily share discrete information about their state. The same is true of any tuple whose corresponding set of NSBs have empty intersection. We showed that NSBs adopt non-intersecting, onion-like structures, implying that very few measurements of network state are sufficient to determine the output label with a linear classifier. The "reason" $VC^{\text{NoData}}$ produces such pessimistic bounds is because in the worst case, every binary value of $\bar{\sigma}(x)$ is required to determine $\mathcal{N}(x) \geq 0$ up to linear classifier. We expect structure in the data to reduce capacity by excluding these worst cases. For linearly separable data, the the learned DNN classifier depends on *no entry* of $\bar{\sigma}(x)$.

As a result, we have produced a powerful method for analyzing, interpreting, and improving DNNs. A deep network is a black box model for learning, but it need not be treated

as such by those who study it. Our logical circuit formulation requires no assumptions and seems extremely promising for introspection and discussion of DNNs in many applications.

Whether our approach can be extended or adapted to other datasets is an pressing question for future research. An important and particularly difficult open question (precluding such an investigation presently) is the efficient determination of $\bar{\Sigma}_0$ (or even $\bar{\Sigma}$) analytically given the network weights. Such an algorithm seems prerequisite to bring deep logical circuit analysis to bear on datasets of higher dimension where we can no longer grid search.

# Chapter 5

# Interpretable Factorization for Neural Network ECG Models [1].

The ability of deep learning (DL) to improve the practice of medicine and its clinical outcomes faces a looming obstacle: model interpretation. Without description of how outputs are generated, a collaborating physician can neither resolve when the model's conclusions are in conflict with his or her own, nor learn to anticipate model behavior. Current research aims to interpret networks that diagnose ECG recordings, which has great potential impact as recordings become more personalized and widely deployed. A generalizable impact beyond ECGs lies in the ability to provide a rich test-bed for the development of interpretive techniques in medicine. Interpretive techniques for Deep Neural Networks (DNNs), however, tend to be heuristic and observational in nature, lacking the mathematical rigor one might expect in the analysis of math equations. The motivation of

[1] **This chapter is based upon a current submission, in consideration for the 2020 Machine Learning in Healthcare conference. The submission was completed in collaboration with Sriram Vishwanath and Jared Urecheck. The author of this dissertation is first author and responsible for the conceptualization, data generation, and writing of the paper**.

this paper is to offer a third option, a scientific approach. We treat the model output itself as a phenomenon to be explained through component parts and equations governing their behavior. We argue that these component parts should *also* be “black boxes" –additional targets to interpret heuristically with clear functional connection to the original. We show how to rigorously factor a DNN into a hierarchical equation consisting of black box variables. This is not a subdivision into physical parts, like an organism into its cells; it is but one choice of an equation into a collection of abstract functions. Yet, for DNNs trained to identify normal ECG waveforms on PhysioNet 2017 Challenge data, we demonstrate this choice yields interpretable component models identified with visual composite sketches of ECG samples in corresponding input regions. Moreover, the recursion distills this interpretation: additional factorization of component black boxes corresponds to ECG partitions that are more morphologically pure.

## 5.1 Introduction

Deep Neural Networks (DNNs) are a class of general purpose, or black box, models that have immense promise for revolutionizing clinical care [Porumb et al., 2020, Mincholé and Rodriguez, 2019]. Yet, widespread adoption of these high performance black box models has been impeded by decreased understanding of patient level outputs. Although interpretability of these models is a burgeoning area of study, the existing methods of interpreting DNNs for medical predictions still show room for improvement Sethi et al. [2020]. For DNNs, these methods are generally applied to trained models in a post hoc, unprincipled manner. The lack of rigor makes it difficult to predict when they can be relied upon for clinical diagnosis. With this work, we extend DL in the healthcare space by applying our post hoc interpretability method to ECG classification. Numerous studies have shown incremental improvement on the performance of automated DNN ECG classification, so our main focus

is to improve the interpretation of ECG classification outputs. By breaking down a trained neural network into simplified component parts, a causal understanding of why the network predicts a certain outcome can be formed. The ability to quickly interpret why the network outputs its prediction will improve the diagnosis and enhance clinical understanding of the problem.

While early machine learning methods sought to encode logic about the world by hand, it was discovered that many relationships, even ones that humans found trivial, were difficult to interpret and translate explicitly into code. The issue at hand is that we find these black box methods, initially designed to learn formulae too difficult to codify directly, now too complicated to interpret directly. In this case, model explanation becomes quite literally a phenomenological study–one that seeks descriptive generalizations of DNN behavior from (post hoc) experiment and observation. We are simply pointing out this is the scientific process, adapted to explaining phenomena of math instead of nature. Hence, this strange new challenge in data science of providing high level explanations for models we can *define* but struggle to *describe* may be a situation with which clinicians are more familiar. In fact, we can motivate our approach to model interpretation through medical analogy, as indicated in the following section.

**Generalizable Insights about Machine Learning in the Context of Healthcare**

- A counter-intuitive but useful first step to black box model interpretation is increasing the number of black box models requiring interpretation. In medicine, this process is familiar. All of the properties we care about, like the output of neural networks, are emergent features arising from repeated composition of very simple rules. Somehow, a very simple differential equation is sufficient to predict the emergence of lymphoma from DNA sequences using physical laws alone. The challenge of interpreting neural networks is like interpreting this functional relationship without knowing in advance

about all the structure in between. Without knowledge of “cells", “lymph nodes", even certain “viruses", we would simply lack the vocabulary to provide useful interpretation. This is what is currently being attempted. We must instead try to discover this structure, building the interpretation of the whole model on our best understanding of its parts. We propose one such method for for neural networks classifying ECG waveforms.

- When we apply this method experimentally, there are two observations of fundamental interest:
    1. Factorization *as functions* of interpretable DNN models results in component functions that are also interpretable, mapping to abstractions that are components of an explanation. In principle, they could be any strange functions satisfying the same equations.
    2. Repeated factorization produces *cleaner* interpretations: Not only do they remain interpretable, they become easier to interpret. Surely, none of this is guaranteed in the general case, making it important to study which clinical settings qualify.

## 5.2 Related Work

Extensive research has demonstrated the practicality of ECG analysis for various use cases in machine learning (ML) for healthcare. DL has been shown to outperform existing risk metrics for cardiovascular death, as demonstrated by the analysis of long term patient ECGs with a DNN [Shanmugam et al., 2018]. Gupta et al. [2019] finds the most expressive combination of ECG leads, testing combinations from 15 leads and training a convolutional neural network (CNN) for state of the art performance in myocardial infarction detection. Accurate performance has also been achieved for single-lead ECG data; Yıldırım et al. [2018] use CNNs on 10 second ECG fragments for the classification of seventeen types of cardiac arrhythmia. Similar DNNs have also been shown to outperform board-certified

cardiologists in its sensitivity when classifying single-lead ECGs into 12 rhythm classes. [Hannun et al., 2019].

Atrial fibrillation classification in the PhysioNet 2017 Challenge closely resembles our focus for research with ECG signals. Our work extends the ideas present in Goodfellow et al. [2018], who created a high performance model and interpreted its behavior with class activation maps (CAMs). The CAMs visualize typical behavior for the three target labels of an ECG signal: normal rhythm, atrial fibrillation, or other. In order to use CAMs, they first modify a top performing model developed for the original challenge. By removing many of the original max pooling layers, their newer model contains a higher temporal resolution at the layer from which they extract the CAMs. Without this architecture-specific change, the output of the mapping would not be very informative. For DNNs, most of the post hoc methods still require extensive tuning to develop a reasonable understanding of their decision-making [Sethi et al., 2020]. With visual data, these methods provide quick assessment of high-dimensional data but they often highlight fuzzy areas of the input with little pathological importance.

Outside of healthcare, similar visualizations are being used to characterize large networks with intuitive interfaces [Hohman et al., 2020]. We aim to further contribute to interpretable visualizations by applying our method to ECG data. By visualizing the component parts of a classification DNN, we aim to find structure in its intermediate decisions that align with our current diagnostic procedure for ECG signals. The ability to derive phenotypes from machine learning algorithms is unexplored in the clinical landscape, though the importance of explainability and interpretability are becoming crucial for machine learning to be used in the clinical setting [Tonekaboni et al., 2019]. Instead of applying an algorithm to each input, we break down the model into component features that explain the output for clustered input types. For ECG signals, these clusters are directly inspectable and

offer insight into possible phenotypes the model deduces, which further contribute to the clinical understanding of the problem.

## 5.3 MinMax-Representation as a Tool for Interpretation

When we train a DNN model to fit a data pattern, what related high-level concepts does the model learn in the process? Of course, this question makes no sense as stated. The model is just a sequence of math symbols with rules for combination. It does not “know" about abstraction. Yet, in this section we will motivate and propose a mathematically rigorous theory that makes sense of the initial question. We develop formulae relating model outputs to “model concepts".

### 5.3.1 Theory: Motivation, Definitions

For exposition and experimentation, we will use binary ECG classification using a DNN model as a running example. We will use $x$ to denote the input, which is a numeric representation of the ECG signal, and $\mathcal{N}$ to be a trained DNN model with scalar output $\mathcal{N}(x)$. In this context, “trained" means that on some example inputs, the set of positive predictions where $\mathcal{N}(x) > 0$ *more or less* coincides with the cases where “$x$ is a normal ECG". Keeping with the set notation, understanding how our model will perform in the “real-world" is equivalent to understanding the domain of the same set of positive predictions extended now over *all* possible inputs, $\{$all ECGs $x$ such that $\mathcal{N}(x) > 0\}$. In this notation, a valid “model explanation" is simply a concise description of this set for humans.

One possible example model explanation might be that $\mathcal{N}(x) > 0$ (returns normal) if “No ST elevation" “AND" “QT elongation". Here, we would consider "No ST elevation" and “no QT elongation" to both be concepts interpretable to humans since cardiologists can readily evaluate which if either apply to a particular ECG. We also see each concept has a

corresponding input set, e.g., $\{x|x$ has ST elevation$\}$, and that our abstract interpretation is really saying mathematically that $\{x|\mathcal{N} > 0\}$ is an intersection of two sets corresponding to the familiar concepts “ST elevation". In fact all of the ways we combine concepts (AND, OR, NOT, etc.) all have corresponding set operations ($\cap$, $\cup$, complement, etc.).

Therefore, we consider the task of classification model interpretation to be equivalent to finding a combination of *interpretable sets* using set operations that approximates sufficiently $\{x|\mathcal{N}(x) > 0\}$. Here, a concept is just a subset of inputs defined by a property, and that concept is interpretable if a human can reasonably decide whether that property applies. Now, *finding* such a vocabulary of concepts and set operations starting from a given model is in fact *fitting a second model* this time over concept combinations, with under-fitting and over-fitting failure modes. This is a difficult problem under intense study.

Instead of tackling this problem directly, what we propose instead is a method for generating intermediate targets for interpretation, $\{x|\phi(x)_1 > 0\}$, $\{x|\phi(x)_2 > 0\}$, whose intermediate interpretation is related to $\{x|\mathcal{N} > 0\}$ through a closed form, interpretable equation. To discuss this, we need to introduce a definition.

**Definition 3.** *MinMax-Representation:*
*Let integer $k > 0$ be arbitrary and $\mathcal{N}, \phi_1, \ldots, \phi_k$ be a real valued functions of input $x$. We call $\Psi : \mathbb{R}^k \mapsto \mathbb{R}$ a MinMax-Representation if through composition it is generated by a (finite) number of compositions of Max and Min functions applied to subsets of the $k$ scalar inputs. If also $\Psi(\phi_1(x), \ldots, \phi_k(x)) = \mathcal{N}(x)$, then we call $\Psi$ a MinMax-Representation of $\mathcal{N}$ with Character Functions $\phi_1, \ldots, \phi_k$*

The benefit of interpreting MinMax-Representations of Character Functions is that they map directly OR/AND combinations of interpretations of Character Functions. Firstly, this avoids introducing approximation or heuristic in this step. The nature of understanding DNNs probably unavoidably involves both subjectivity *and* approximation at some stage, but

it's helpful to know that this step can be relied upon when analyzing how things go wrong. Secondly, and much more subtly: interpretation of the whole and of the parts give the same answer. One has to decide whether to interpret directly or factor (as functions), interpret, and combine with AND/OR. It is a theoretical point about interpretation-preserving operations, that we will leave here except to say that we need not worry about deriving conflicting interpretations.

It is natural to ask whether there is some correspondence between these subdivisions of the model and subdivisions of the data. After all, a parsimonious model should only apply differing reasoning to differing cases. This correspondence indeed exists.

**Definition 4.** *Attribute Space:*
*Let $\Psi$ be a MinMax-Representation of $\mathcal{N}$ with $CharacterFunctions$ $\phi_1, \ldots, \phi_k$. For each Character Function, $\phi_j$, let $\{x|\phi_j(x) = \mathcal{N}(x)\}$ be the corresponding Attribute Space.*

Note that these spaces partition the input: because Min (resp. Max) agrees with at least one of its inputs at every point, then so does $\Psi$, which is a finite combination of the two. Therefore, each ECG falls into some Attribute Space, and we refer a collection (e.g. the training set) of ECGs all belonging to the same one a model concept. Note also, that on this subset of, the Character Function and $\mathcal{N}$ are the same function, so interpreting the former is equivalent to interpreting the later conditional on this additional information. While, to our knowledge, this section is novel, in the next section we need to briefly dip into the background material to borrow a math technique.

### 5.3.2 Discussion and Approach

The section discusses MinMax representations of a neural network in theory, in the literature, and in our approach. By a neural network, we mean recursive composition of $d$ "layers", each of which is an affine function following a ReLU function, $R(x)_i = \max\{0, x_i\}$, the

output of each usually being referred to as an “activation". To build an example around 1 layer, let us denote by $z(x)$ or simply $z$ the last activation (that is not the output), so that

$$\mathcal{N}(x) = b^{(d)} + A^{(d)} R(z(x)).$$

Here $b^{(d)}$ and $A^{(d)}$ are the bias and linear components affine map in the last of $1, \ldots, d$ layers. A helpful approach is to split the sign components of any vector or matrix, $M$, by using the corresponding subscript, $(M_{\pm})_{i,j} = \max\{0, \pm M_{i,j}\}$. The idea is to organize terms in the optimization so that the *greedy* choice for each $R$ linear component agrees with the one actually realized by the network. Continuing our example we have,

$$\mathcal{N}(x) = b^{(d)} + \max_{\mu} A_{+}^{(d)} \mu(z(x)) - \max_{\tau} A_{-}^{(1)} \tau(z(x))$$

Here, we are considering $\mu$ and $\tau$ to be optimized over binary diagonal matrices– simply enough, they are always driven to “zero out" any negative components. They optimize different variables so, trivially, a difference of maxima can be written equivalently as a MaxMin or a MinMax of the difference, which in this case is a linear function of $z$. All this so far is common to both [Zhang et al., 2018] and [Snyder and Vishwanath, 2020b], but at this point they give qualitatively different approaches to multi-layer networks.

For his original interest in that class of functions, [Zhang et al., 2018] says *any* neural network can be written as a difference of maxima using only linear functions of the input. At first this sounds good. We did not even ask for each Character Function to be interpretable, let alone linear. But, something has to give. If you design your Character Functions to be linear, then it will taken very many of them to represent $\mathcal{N}$. If interpretable functions are “closed under composition", then the MinMax-Representation, $\Psi$, will be too complicated a

to be useful.

As an alternative, we follow the layer-wise approach taken in [Snyder and Vishwanath, 2020b]. Specifically, we are following Algorithm 2 in the appendix. We will give a quick summary in our notation. The idea is to simply recurse the 1 hidden layer expansion we demonstrated. In the first step $\Psi$ has $MaxMin$ structure and arguments $\phi_1, \ldots, \phi_k$ that are $d-1$ layer neural networks. Only the first $d-2$ layers of these networks are identical to the original. If we treat each $d-1$ layered network individually in the same fashion as the original, then we get nested MaxMinMaxMin structure for $\Psi$ which optimizes over terms that are each $d-2$ layer functions. We continue recursively. The indices and number of functions grows like the number of linear regions achieved in the terminal layers.

However, we cannot apply this method exactly because [Snyder and Vishwanath, 2020b] only outline the approach for fully-connected (FC) layers, while in our setting 1D convolutional (Conv) layers and Max Pooling layers (MP) are standard. These layers *can* definitely be used in a similar scheme, but we found it simpler to restrict Conv and MP layers to the initial stages, so that the factorization only "sees" the later FC layers. Because Conv and MP layers can also be represented by FC networks, the algorithm cannot tell which has generated the Character Functions and as such still functions correctly.

## 5.4 Methods

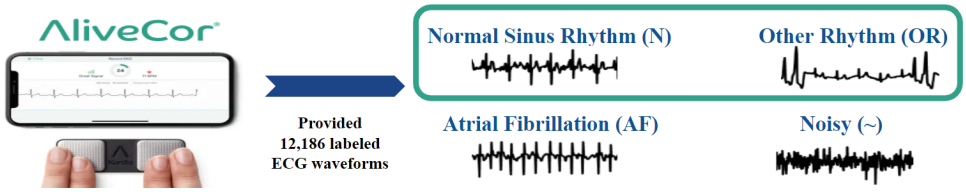



Figure 5.1: The PhysioNet 2017 Dataset.

This section defines an experimental design that reflects the aims, concepts, and

techniques from previous sections. Here the largely theoretical exposition turns sharply practical, as we detail the actual physical steps and procedures to produce our experimental outputs. These include dataset creation, network design, training protocol, as well as our network-MinMax Conversion algorithm and supporting heuristics.

### 5.4.1 Dataset and Data Preprocessing

We used ECG waveform data from the PhysioNet 2017 Computing in Cardiology Challenge [Clifford et al., 2017], which was also a component of Goldberger et al. [2000]. The challenge encouraged development of algorithms that differentiate single-lead ECGs labeled as atrial fibrillation (AF), normal sinus rhythms (N), other rhythms (OR), and rhythms too noisy for classification ($\sim$). While the PhysioNet dataset is often used for bench-marking classification models, we are instead interested in demonstrating the *interpretation* of a classification model. To facilitate this study, several simplifications were made to the original classification task.


Figure 5.2: Dataset Preprocessing.

The PhysioNet dataset was obtained and donated by AliveCor. Lead I (LA-RA) equivalent ECG recordings were generated using an AliveCor hand held device. Each recording ranges from $9$ to $61$ seconds. The complete dataset includes $12,186$ recordings that were partitioned in a $70/30$ split, resulting in a training set of $8,528$ and a test set of $3,658$. For our datasest, the recordings with ($\sim$) and AF labels were removed. As we have access to only the training set, we perform an additional $80/20$ split at random to generate our train and test data. The PhysioNet train/test split was completed along waveform lines to prevent patient data from belonging to both the training and test set. Instead, our model

inputs consist of short snippets of ECGs called “templates". For simplicity here, the patient information was discarded. The $R$ waveform of each template is aligned, and light filtering is performed. Each template “inherits" the label pertaining to the waveform it derived from, as if each ECG complex within the waveform exhibits that labeled morphology.

The data distribution samples uniformly a (waveform,label) pair from either the train or test set, and subsequently samples uniformly a template or ECG complex from that waveform. The DNN model is trained to minimize the negative log likelihood of the label given the template.

### 5.4.2 Architecture Design

We used a convolution layer model roughly based on the one in [Goodfellow et al., 2018] but with some adaptations particular to our setup. Overall, the network consisted of several convolutional alternating convolution and max pooling layers, followed by several fully-connected layers. Layers aside from the max pooling and terminal layers were followed with a ReLU nonlinearity. An illustration is shown in Figure 5.3.

We reduce number of convolutional filters and degree of pooling to reflect the change from whole ECG inputs to shorter waveform inputs. The size of fully-connected filters in the later layers was also reduced (without more than 2-3% change in model accuracy) to reduce the number of linear pieces comprising the terminal 4 layers.

### 5.4.3 Model Training

Neural network training was done with the Tensorflow library. None of the values were tuned, and most were simply inherited from previous reused code. We used the Adam to optimize a sigmoid cross-entropy loss with $1e^{-5}$ learning rate and batch size of $64$. We trained for $80$ training epochs for the models in this paper, but we have no evidence that this

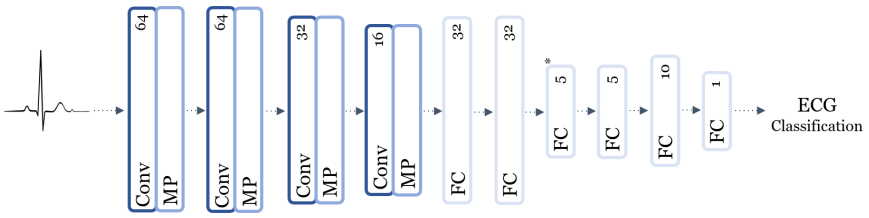



Figure 5.3: Architecture of the network. For the convolutional layers (Conv), we use kernel sizes of 6 and 4 for the first and second halves, respectively, and we use strides of 2. The max pooling (MP) layers all had pool sizes of 2 and strides of 1. Final layers of the neural network were fully-connected (FC).

long length of time is necessary or important.

### 5.4.4 Calculating MinMax-Representation, model concept Partitions

This section covers unique implementation details. Definitions and algorithm for calculating MinMax-Representation are given in Section 5.3.2 and Snyder and Vishwanath [2020b], Algorithm 2. By model concepts, we refer to the Attribute Spaces, defined at the end of Section 5.3.1 and restrict them to training samples.

We apply these algorithms proposing our “input" is actually the embedding output from the first $5$ neuron layer, indicated by the asterisk (Fig. 5.3). The complexity of this approach as implemented grows roughly with the number of linear regions, which is kept reasonably small $(10 - 100)$ by the smaller width. Like Snyder and Vishwanath [2020b], we identify these regions defining MinMax-Representation and Character Functions using a grid search. Ours are unbounded potentially, so we use $99^{\text{th}}$ percentiles.

Min and Max, being differences of maxima, commute and thus provide a choice whether Min or Max should lead each layer representation. We lead with Min. The motivation is that, since we classify Normal (positive) vs Other (negative) rhythms, we want to allow for AND to be the highest level interpretation. The goal is to reach an interpretation like, “$x$ is Normal" iff “$x$ not diagnosis 1", AND “$x$ not diagnosis 2", etc.

The model concepts can be conveniently calculated alongside the recursion building the MinMax-Representation. At each step, simply divide the ECGs associated to the current component to the ArgMax/Min of the substituting representation. An important note is that there were some Character Functions with empty model concept. This is partially because the grid search may explore regions that inputs do not, but also because the MinMax-Representation is only guaranteed to be correct, not minimal. We drop these from the visualization explained next section.

The most obvious way to interpret each Character Function turns out not to work for ECGs. If one views each Character Function as a function on the entire space, as was done in Snyder and Vishwanath [2020b] with MNIST digits, the corresponding interpretation will be correct but perhaps not the simplest correct one. Each Character Function becomes easier to interpret in the context of the others by deriving additional descriptive input classes: Small changes to each Character Function only “cause" the model to change outputs on a subset of inputs we call a model concept.

### 5.4.5 Interpretation through Visualization

While DNN functions can be difficult to visualize directly, we can characterize them through the data partitions associated with their component parts. Ultimately, we want to understand how the classifications boundaries split these characteristic sets. A “tutorial" example of one such model concept visualization is explained below and given in Figure 5.4.

For a waveform of a single class label, we first R-wave peak normalize and align the waveforms. Then we use alpha blending to overlay the waveforms with red and blue corresponding to label, which creates darker areas of the graph where many ECGs have the same normalized potential at the same time point. When plotting a sample of $4000$ waveforms with a small alpha value ($< 0.01$), anomalies plotted by a few waveforms are

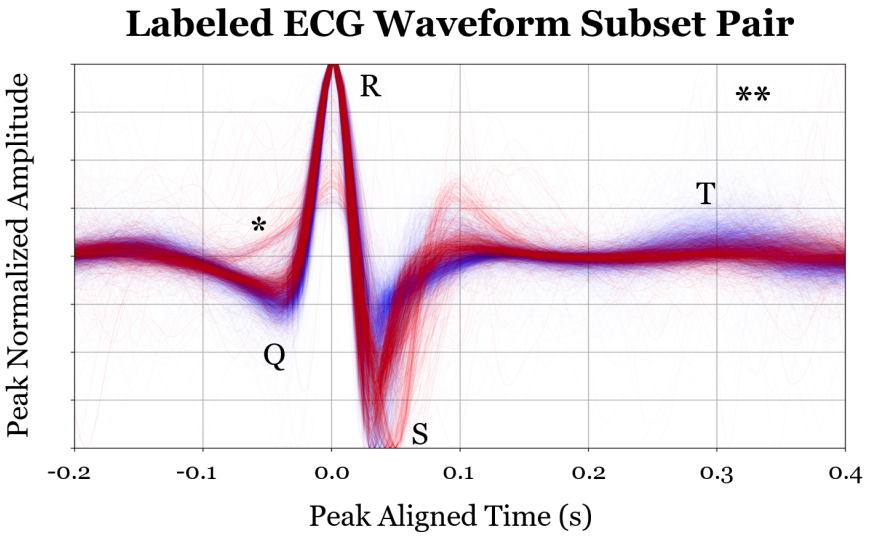



Figure 5.4: ECG plot with peak normalized amplitudes. The figure, a composite visual of individual ECG recordings, quickly conveys variety, distribution, and clustering of trajectories to an observer. Abnormal (negative) classes have a combination of higher **Q** waves, more depressed and extended **S** waves and absent **U** waves. These observations reveal fundamental vibrations of ECG potential that represent class characteristics. To obtain this figure, we align each waveform at $0.0$ seconds in order to closely compare them. Several thousand individual ECG recordings are then drawn with replacement from the abnormal (negative) and normal (positive) classes. The line transparency is adjusted and the ECGs are directly overlaid.

hardly noticeable. Alpha and other parameters such as line thickness were chosen by visual tests to ensure the graph was not over saturated with lines.

## 5.5 Results

Our model achieves $74\%$ accuracy. Other challenge models at the time achieved accuracy in the low $80^{\text{th}}$ percentile. Of course, diagnoses defined in terms of the the R-R interval will be harder without access to whole ECG recordings. Also, the restrictions we placed on the size of the terminal layers may have made it more difficult to classify certain patterns. But, this quite reasonable accuracy indicates our simplified model is *qualitatively representative* of an out-of-the-box ECG model in practice. By extension, we argue our interpretation achieves this level of accuracy as well. When we interpret this model by deriving a component representation of the last $2$ hidden layers, we get a very rich and informative story. Correspondingly, its representation is also very information dense and needs to be digested slowly, at multiple zoom-scales, and with color. By visualizing and arranging the aggregate waveform of each model concept using the technique discussed in Section 5.4.5, we obtain joint interpretation of each component as it relates to the overall model. Refer to Figure 5.5 throughout.

The top row is easiest to understand, and can be viewed independently of the rest. The image 5.5**a.** is a composite of every training sample as labeled by the final trained model. Equally valid would be a representation using test samples; they simply answer different questions. It is useful to compare both but beyond our scope. Instead, we want to follow a relatively simple thread.

The reader may have noticed some of the waveforms plotted are upside-down, having their polarization inverted. The two downward extensions of the Q and S waves (we'll call them *legs*) are present in most images, except some in the last row. What happened was

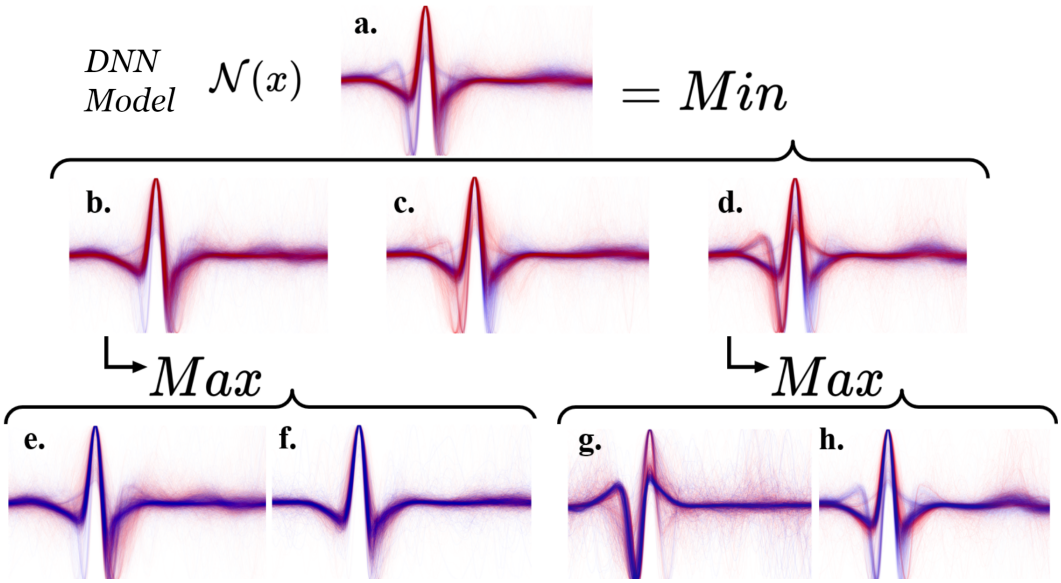



Figure 5.5: A Visual Explanation of a DNN ECG model as emergent from Min,Max equations governing interpretable component parts. The figure is also an equation for the neural network, parameterized by Character Functions represented by the corresponding model concepts. Each waveform sample is drawn in **a.** and once in the ArgMax or ArgMin component visualization following each bracket. Details and analysis in text. But, many interesting features left for the reader to explore. The Character Function in **c.** has no more subdivisions at this depth, so it equally belongs among the third row.

an extremely fortuitous, informative accident. In our attempt to reproduce the code from Goodfellow et al. [2018], we missed the portion that corrects the polarization. Depending on how peak alignment was done, the R wave was sent to either the Q or S leg. The effect of this is to artificially create additional waveform morphologies and phenotypes. This is suboptimal from the point of view of performance maximization. But in fact, it is a wonderful wrinkle–one representative of the realities of clinical modeling–that we can use to demonstrate the potential for our interpretation method.

Surely, in practice similar mistakes occur. One usually cannot easily verify if errors exist in some clinical data samples. Notably, the model behavior does not distinguish between clinical and artificial data structure. So it is extremely important to understand how such mistakes and structures in general become represented in our models. Does the model even identify polarization inverted waveforms as a distinct model concepts? If so, then perhaps further analysis will show it also discovers clinical diagnoses based on morphology. As it turns, the neural network model has three fundamental modes that differ with respect to how they treat inverted Q and S leg waveforms.

In the second row, Figures 5.5**b.**,**c.**,**d.**, we can begin to understand these modes or Character Functions. The combination of the first and second row is also an equation: **a.**=Min(**b.**,**c.**,**d.**) or with Character Functions $\phi_b, \phi_c, \phi_d$ it says $\mathcal{N}(x) = \text{Min}(\phi_b, \phi_c, \phi_d)$. Each sample waveform is represented visually on both sides of this equation (in fact once per row). Because each column is a representation of a model concept, each sample is only drawn in the visualization of the Character Function that achieves the minimum of $\phi_b, \phi_c, \phi_d$, determines the output class label independently of the other two. Therefore, we use the relationship between the two rows to understand the label given to a single sample: A waveform is considered normal iff it is drawn in blue in the top row iff it is drawn in blue in one of the three second row figures. These characterizations also hold between the second

row and bracketed third row Character Functions.

Having defined what a row is, we can see emergent structures in the Figures 5.5**b.**,**c.**,**d.**. We see by the color of each Q and S leg in the second row that alignment direction of the inverted waveform, becoming either the Q or S leg, is a central organizational theme for the neural network. We see 5.5**a.** accounting for about half of the Q leg positive/Normal samples and all of the S leg negatives/Other samples, 5.5**c.** sharing about half of each of the negative Q leg and positive S leg samples. The story with 5.5**d.** is similar to 5.5**c.**, but with mixed Q leg contributions, and complex dynamics overall. Amazingly, this complex structure in 5.5**d.** gets *easier* to interpret the farther we carry the interpretation.

In Figures 5.5**g.**and **h.**, we observe interpretable component representation of the the Character Function in 5.5**d.** We discover that the model *does* treat polarization inverted waveforms as a fundamentally distinct class. The Character Function $\phi_g$ generates the decision boundary for the inverted waveforms where $\phi_d$ is optimal among the first row. The Character Function in 5.5**h.** handles the complement of upright waveforms (along with *some* inverted ones). Observable in both 5.5**d.** and **h.** (perhaps best in 5.5**h.**) are these contrasting red-blue bands contouring the QRS complex. These allow us to interpret how decisions are made among upright ECGs. The red outer R wave detailing in 5.5**h.** suggests a component that labels as negative those samples with R waves that rise too slowly. Likewise, we see abnormal diagnoses associated with P waves that are too deep, and Q waves rising too slowly.

An important point to remember is that these interpretable structures are in no way obligated to manifest. Each sample must be present somewhere in each row, but we they do not also need to be organized and sorted in a way that seems reasonable and interpretable to us. As the complexity of the functions, for example, $\phi_b, \phi_c, \phi_d$, is not controlled, these samples could take any arrangement with sufficiently expressive lower layers. A second

point: even if they stay interpretable, we don't know of any theoretical maxim that says these interpretations should so quickly become simpler.

## 5.6 Discussion

We all have a mental image of what clinical relevance "looks like". Perhaps, one recalls previous papers that tried to solve similar problems. Why does this one introduce so much unorthodoxy and detail so as to obscure that clinical connection? Let us try to motivate why something in the this style of approach is really prerequisite for continuing to make intentional forward progress with Deep Learning, in particular in medicine.

Consider the "stadium wave", in which successive, adjacent groups of seated spectators of sport stand and raise their arms upwards. What neural networks do really well is generate high level concepts by mapping low level inputs, such as pixels, sound amplitudes, heart rates. Interpreting these end-to-end is impossible. Doing so would be analogous to trying to interpret a neural network that predicts the frequency of "stadium wave" behavior from the mere DNA sequences of the sport spectators, without pausing to understand how the model represents "humans" as a concept. It becomes much easier to understand, control, debug, iterate, interpret, and learn from the model if you can consider the wave behavior model from the perspective of changes to stadium members' behavior, rather than individual nucleotide base-pairs. Component interpretations are important for studying modeling with Deep Learning, just as cells are important for studying medicine with humans.

This is especially relevant for medicine where we anticipate these model components can have interpretations that circle back and, in turn, teach us about medicine. We have demonstrated this is exactly what occurs with our approach in Figure 5.5, where phenotype corresponds to morphology. If there are natural clusters that models routinely find useful for modeling vast quantities of data that humans simply do not have the lifespan to access, then

these are useful targets for follow up studies to try to find a common physiologic mechanism.

We would assess this method as not ready for direct use by clinicians but in need of interest cultivation and improvement to supporting algorithms. It works, but it's fragile. One has to properly contextualize: when deep learning paper publishes a model, that work is subsided by *decades* of experience, supporting algorithm development, and standardized libraries. Because our approach is genuinely new, we lack all of that again [2]. But we also have opportunities to improve our results by at really every step.

**Removable Limitations** These are conditions we required experimentally that we believe strongly could be removed with additional theory. For example, we only expanded the fully-connected layers, treating the convolutional ones as an embedding. This is convenient, but unnecessary since they can always be viewed as special cases of fully-connected ones. Generally, one should expect the *theory* to be adaptable to any piece-wise linear operation, including max-pool layers for example. Though, the experimental behavior properties may differ, in part because the parameterization determines the training dynamics and thereby affect the final structure. For now, the trickiest part is keeping the MinMax expansion small enough when the number of neurons in the layers is very large. We accomplished this by having very small width in the later layers. This keeps the expansion small because there are fewer neuron state(on/off) combinations. We suspect that in these cases some small subset is usually sufficient to agree with model behavior with high probability. But, in general when this is possible is determined by the experimental data. We don't expect most data in high dimensions to have a circular decision boundary with small margin, but a fine approximation to a circle with many pieces would break this part.

[2] *Unbelievably*, even the visualization implementation deserves its own further study. To say nothing of the complex subjective human perceptions, we need a custom rendering to blend these better, because existing software will only blend one plot at a time, give a "painted over" feel.

**Intrinsic Limitations** For any of this to work, interpretable component representations (1) have to exist and and (2) have to be representable to humans in some intelligible way. Unfortunately, we don't know how to substantiate either of these with theory. The former seems to happen whenever we can contrive the latter. But, it's just not clear how much we're *really* asking for with that first word "interpretable" components.

# Appendix

## A.1 Appendix for Chapter 2: CausalGAN

### A.1.1 Causality Background

Formally, a structural causal model is a tuple $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathbb{P}_E(.))$ that contains a set of functions $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, a set of random variables $V = \{X_1, X_2, \ldots, X_n\}$, a set of exogenous random variables $\mathcal{E} = \{E_1, E_2, \ldots, E_n\}$, and a probability distribution over the exogenous variables $\mathbb{P}_{\mathcal{E}}$[1]. The set of observable variables $\mathcal{V}$ has a joint distribution implied by the distributions of $\mathcal{E}$, and the functional relations $\mathcal{F}$. This distribution is the projection of $\mathbb{P}_{\mathcal{E}}$ onto the set of variables $\mathcal{V}$ and is shown by $\mathbb{P}_{\mathcal{V}}$. The causal graph $D$ is then the directed acyclic graph on the nodes $\mathcal{V}$, such that a node $X_j$ is a parent of node $X_i$ if and only if $X_j$ is in the domain of $f_i$, i.e., $X_i = f_i(X_j, S, E_i)$, for some $S \subset V$. The set of parents of variable $X_i$ is shown by $Pa_i$. $D$ is then a Bayesian network for the induced joint probability distribution over the observable variables $\mathcal{V}$. We assume causal sufficiency: Every exogenous variable is a direct parent of at most one observable variable.

[1] The definition provided here assumes causal sufficiency, i.e., there are no exogenous variables that affect more than one observable variable. Under causal sufficiency, Pearl's model assumes that the distribution over the exogenous variables is a product distribution, i.e., exogenous variables are mutually independent.

### A.1.2 Proof of Proposition 1

Note that $D_1$ and $D_2$ are the same causal Bayesian networks Pearl [2009]. Under the causal sufficiency assumption, interventional distributions for causal Bayesian networks can be directly calculated from the conditional probabilities and the causal graph. Thus, $\mathcal{M}_1$ and $\mathcal{M}_2$ have the same interventional distributions. □

### A.1.3 Helper Lemmas for CausalGAN

In this section we use $\mathbb{P}_r(l, x)$ for the joint data distribution over a single binary label $l$ and the image $x$. We use $\mathbb{P}_g(l, x)$ for the joint distribution over the binary label $l$ fed to the generator and the image $x$ produced by the generator. Later in Theorem 8, $l$ is generalized to be a vector.

The following restates Proposition 1 from Goodfellow et al. [2014] as it applies to our discriminator:

**Proposition 3** (Goodfellow et al. [2014])**.** *For fixed G, the optimal discriminator D is given by*

$$D^*_G(x) = \frac{\mathbb{P}_r(x)}{\mathbb{P}_r(x) + \mathbb{P}_g(x)}. \tag{A.1}$$

Second, we identify the optimal Labeler and Anti-Labeler. We have the following lemma:

**Lemma 1.** *The optimum Labeler has $D_{LR}(x) = \mathbb{P}_r(l = 1|x)$.*

*Proof.* The proof follows the same lines as in the proof for the optimal discriminator. Consider the objective

$$\begin{aligned}
&\rho\mathbb{E}_{x\sim\mathbb{P}_r(x|l=1)}\left[\log(D_{LR}(x))\right] + (1-\rho)\mathbb{E}_{x\sim\mathbb{P}_r(x|l=0)}\left[\log(1 - D_{LR}(x)\right] \\
&= \int \rho\mathbb{P}_r(x|l=1)\log(D_{LR}(x)) + (1-\rho)\mathbb{P}_r(x|l=0)\log(1 - D_{LR}(x))dx
\end{aligned} \tag{A.2}$$

Since $0 < D_{LR} < 1$, $D_{LR}$ that maximizes (2.3) is given by

$$D^*_{LR}(x) = \frac{\rho \mathbb{P}_r(x|l=1)}{\mathbb{P}_r(x|l=1)\rho + \mathbb{P}_r(x|l=0)(1-\rho)} = \frac{\rho \mathbb{P}_r(x|l=1)}{\mathbb{P}_r(x)} = \mathbb{P}_r(l=1|x) \quad \text{(A.3)}$$

□ □

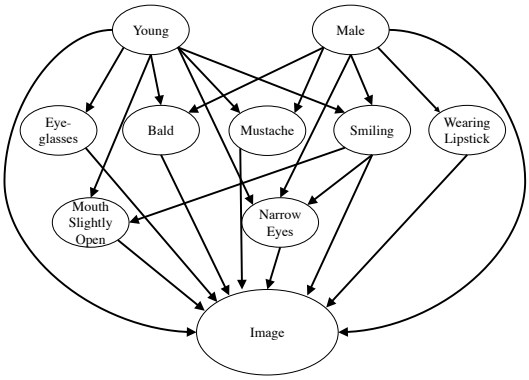



Figure A.1: The causal graph used for simulations for both CausalGAN and CausalBEGAN, called CelebA Causal Graph (G1). We also add edges (see Appendix Section A.1.10) to form the complete graph "cG1". We also make use of the graph rcG1, which is obtained by reversing the direction of every edge in cG1.

Similarly, we have the corresponding lemma for Anti-Labeler:

**Lemma 2.** *For a fixed generator with $x \sim \mathbb{P}_g(x)$, the optimum Anti-Labeler has $D_{LG}(x) = \mathbb{P}_g(l=1|x)$.*

*Proof.* Proof is the same as the proof of Lemma 1. □

### A.1.4 Proof of Theorem 1

**Theorem 1.**

*Define $C(G)$ as the generator loss for when discriminator, Labeler and Anti-Labeler are at their optimum. Assume $\mathbb{P}_g(l) = \mathbb{P}_r(l)$, i.e., the Causal Controller samples from the true label*

*distribution. Then the global minimum of the virtual training criterion $C(G)$ is achieved if and only if $\mathbb{P}_g(l,x) = \mathbb{P}_r(l,x)$, i.e., if and only if given a label $l$, generator output $G(z,l)$ has the same distribution as the class conditional image distribution $\mathbb{P}_r(x|l)$.*

*Proof.* For a fixed generator, the optimum Labeler $D^*_{LR}$, Anti-Labeler $D^*_{LG}$, and discriminator $D^*$ obey the following relations by Prop 3, Lemma 1, and Lemma 2:

$$\begin{aligned}(1 - D^*(x))/D^*(x) &= \mathbb{P}_g(x)/\mathbb{P}_r(x)\\ D^*_{LR}(x) &= \mathbb{P}_r(l = 1|x)\\ D^*_{LG}(x) &= \mathbb{P}_g(l = 1|x).\end{aligned} \tag{A.4}$$

Then substitution into the generator objective in (2.5) yields

$$\begin{aligned}C(G) &= \mathbb{E}_{x\sim p_g(x)}\left[\log\left(\frac{1 - D^*(x)}{D^*(x)}\right)\right]\\ &\quad - \rho\mathbb{E}_{x\sim p^1_g(x)}\left[\log(D^*_{LR}(X))\right] - \bar{\rho}\mathbb{E}_{x\sim p^0_g(x)}\left[\log(1 - D^*_{LR}(X))\right]\\ &\quad + \rho\mathbb{E}_{x\sim p^1_g(x)}\left[\log(D^*_{LG}(X))\right] + \bar{\rho}\mathbb{E}_{x\sim p^0_g(x)}\left[\log(1 - D^*_{LG}(X))\right]\\ &= \mathbb{E}_{x\sim p_g(x)}\left[\log\left(\frac{\mathbb{P}_g(x)}{\mathbb{P}_r(x)}\right)\right] - \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)}\left[\log(\mathbb{P}_r(l|x))\right]\\ &\quad + \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)}\left[\log(\mathbb{P}_g(l|x))\right] \qquad \text{(A.5)}\\ &= \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)}\left[\log\left(\frac{\mathbb{P}_g(x)}{\mathbb{P}_r(x)}\right) + \log(\mathbb{P}_g(l|x)) - \log(\mathbb{P}_r(l|x))\right]\\ &= \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)}\left[\log\left(\frac{\mathbb{P}_g(l,x)}{\mathbb{P}_d(l,x)}\right)\right]\\ &= (\mathbb{P}_g)\,\mathbb{P}_d. \qquad \text{(A.6)}\end{aligned}$$

where KL is the Kullback-Leibler divergence, which is minimized if and only if $\mathbb{P}_g = \mathbb{P}_d$

jointly over labels and images. (A.5) is due to the fact that $\mathbb{P}_r(l = 1) = \mathbb{P}_g(l = 1) = \rho$. $\square$

### A.1.5 Proof of Corollary 1.1

**Corollary 1.** *Suppose $C : \mathcal{Z}_1 \to \mathcal{L}$ is a causal implicit generative model for the causal graph $D = (\mathcal{V}, E)$ where $\mathcal{V}$ is the set of image labels and the observational joint distribution over these labels are strictly positive. Let $G : \mathcal{L} \times \mathcal{Z}_2 \to \mathcal{I}$ be a generator that can sample from the image distribution conditioned on the given label combination $L \in \mathcal{L}$. Then $G(C(Z_1), Z_2)$ is a causal implicit generative model for the causal graph $D' = (\mathcal{V} \cup \{Image\}, E \cup \{(V_1, Image), (V_2, Image), \dots (V_n, Image)\})$.*

*Proof.* Since $C$ is a causal implicit generative model for the causal graph $D$, by definition it is consistent with the causal graph $D$. Since in a conditional GAN, generator $G$ is given the noise terms and the labels, it is easy to see that the concatenated generator neural network $G(C(Z_1), Z_2)$ is consistent with the causal graph $D'$, where $D' = (\mathcal{V} \cup \{Image\}, E \cup \{(V_1, Image), (V_2, Image), \dots (V_n, Image)\})$. Assume that $C$ and $G$ are perfect, i.e., they sample from the true label joint distribution and conditional image distribution. Then the joint distribution over the generated labels and image is the true distribution since $\mathbb{P}(Image, Label) = \mathbb{P}(Image|Label)\mathbb{P}(Label)$. By Proposition 1, the concatenated model can sample from the true observational and interventional distributions. Hence, the concatenated model is a causal implicit generative model for graph $D'$. $\square$

### A.1.6 CausalGAN Analysis for Multiple Labels

In this section, we explain the modifications required to extend the proof to the case with multiple binary labels. The central difficulty with generalizing to a vector of labels $l = (l_j)_{1 \le j \le d}$ is that each labeler can only hope to learn about the posterior $\mathbb{P}(l_j|x)$ *for each j*. This is in general insufficient to characterize $\mathbb{P}_r(l|x)$ and therefore the generator can not

hope to learn the correct joint distribution. We show two solutions to this problem. (1) From a theoretical (but perhaps impractical) perspective each labeler can be made to estimate the probability of each of the $2^d$ label combinations instead of each label. We do not adopt this in practice. (2) If in fact the label vector is a deterministic function of the image (which seems likely for the present application), then using Labelers to estimate the probabilities of each of the $d$ labels is sufficient to assure $\mathbb{P}_g(l_1, l_2, \ldots, l_d, x) = \mathbb{P}_r(l_1, l_2, \ldots, l_d, x)$ at the minimizer of $C(G)$. In this section, we present the extension in (1) and present the results of (2) in Section A.1.7.

Consider Figure 2.3 in the main text. The Labeler outputs the scalar $D_{LR}(x)$ given an image $x$. Previously in Section A.1.3 we showed that the optimum Labeler satisfies $D^*_{LR}(x) = \mathbb{P}_r(l = 1|X = x)$ for a single label. We first extend the Labeler objective as follows: Suppose we have $d$ binary labels. Then we allow the Labeler to output a $2^d$ dimensional vector $D_{LR}(x)$, where $D_{LR}(x)[j]$ is the $j^{th}$ coordinate of this vector. The Labeler then solves the following optimization problem:

$$\max_{D_{LR}} \sum_{j=1}^{2^d} \rho_j \mathbb{E}_{x \sim \mathbb{P}_r(x|l=j)} \log(D_{LR}(x)[j]), \tag{A.7}$$

where $\rho_j = \mathbb{P}_r(l = j)$. We have the following Lemma:

**Lemma 3.** *Consider a Labeler $D_{LR}$ that outputs the $2^d$-dimensional vector $D_{LR}(x)$ such that $\sum_{j=1}^{2^d} D_{LR}(x)[j] = 1$, where $x \sim \mathbb{P}_r(x, l)$. Then the optimum Labeler with respect to the loss in (A.7) has $D^*_{LR}(x)[j] = \mathbb{P}_r(l = j|x)$.*

*Proof.* Suppose $\mathbb{P}_r(l = j|x) = 0$ for a set of (label, image) combinations. Then $\mathbb{P}_r(x, l = j) = 0$, hence these label combinations do not contribute to the expectation. Thus, without loss of generality, we can consider only the combinations with strictly positive probability. We can also restrict our attention to the functions $D_{LR}$ that are strictly positive on these

(label,image) combinations; otherwise, loss becomes infinite, and as we will show we can achieve a finite loss. Consider the vector $D_{LR}(x)$ with coordinates $D_{LR}(x)[j]$ where $j \in [2^d]$. Introduce the discrete random variable $Z_x \in [2^d]$, where $\mathbb{P}(Z_x = j) = D_{LR}(x)[j]$. The Labeler loss can be written as

$$\min -\mathbb{E}_{(x,l)\sim\mathbb{P}_r(x,l)} \log(\mathbb{P}(Z_x = j)) \tag{A.8}$$

$$= \min \mathbb{E}_{x\sim\mathbb{P}_r(x)} (L_x) Z_x - H(L_x), \tag{A.9}$$

where $L_x$ is the discrete random variable such that $\mathbb{P}(L_x = j) = \mathbb{P}_r(l = j|x)$. $H(L_x)$ is the Shannon entropy of $L_x$, and it only depends on the data. Since KL divergence is greater than zero and $p(x)$ is always non-negative, the loss is lower bounded by $-H(L_x)$. Notice that this minimum can be achieved by satisfying $\mathbb{P}(Z_x = j) = \mathbb{P}_r(l = j|x)$. Since KL divergence is minimized if and only if the two random variables have the same distribution, this is the unique optimum, i.e., $D^*_{LR}(x)[j] = \mathbb{P}_r(l = j|x)$.

□

The lemma above simply states that the optimum Labeler network will give the posterior probability of a particular label combination, given the observed image. In practice, the constraint that the coordinates sum to 1 could be satisfied by using a softmax function in the implementation. Next, we have the corresponding loss function and lemma for the Anti-Labeler network. The Anti-Labeler solves the following optimization problem

$$\max_{D_{LG}} \sum_{j=1}^{2^d} \rho_j \mathbb{E}_{\mathbb{P}_g(x|l=j)} \log(D_{LG}(x)[j]), \tag{A.10}$$

where $\mathbb{P}_g(x|l = j) \coloneqq \mathbb{P}(G(z, l) = x|l = j)$ and $\rho_j = \mathbb{P}(l = j)$. We have the following Lemma:

**Lemma 4.** *The optimum Anti-Labeler has* $D^*_{LG}(x)[j] = \mathbb{P}_g(l = j|x)$.

*Proof.* The proof is the same as the proof of Lemma 3, since Anti-Labeler does not have control over the joint distribution between the generated image and the labels given to the generator, and cannot optimize the conditional entropy of labels given the image under this distribution. □

For a fixed discriminator, Labeler and Anti-Labeler, the generator solves the following optimization problem:

$$\begin{aligned}
&\min_G \mathbb{E}_{x\sim p_{\mathrm{g}}(x)}\left[\log\left(\frac{1-D(x)}{D(x)}\right)\right] \\
&-\sum_{j=1}^{2^d}\rho_j \mathbb{E}_{x\sim\mathbb{P}_g(x|l=j)}\left[\log(D_{LR}(X)[j])\right] \\
&+\sum_{j=1}^{2^d}\rho_j \mathbb{E}_{x\sim\mathbb{P}_g(x|l=j)}\left[\log(D_{LG}(X)[j])\right]. \qquad \text{(A.11)}
\end{aligned}$$

We then have the following theorem along the same lines as Theorem 1 showing that the optimal generator samples from the class conditional image distributions given a particular label combination:

**Theorem 8** (Theorem 1 formal for multiple binary labels)**.** *Define* $C(G)$ *as the generator loss as in Eqn. A.11 when discriminator, Labeler and Anti-Labeler are at their optimum. Assume* $\mathbb{P}_g(l) = \mathbb{P}_r(l)$*, i.e., the Causal Controller samples from the true joint label distribution. The global minimum of the virtual training criterion* $C(G)$ *is achieved if and only if* $\mathbb{P}_g(l, x) = \mathbb{P}_r(l, x)$ *for the vector of labels* $l = \{l_i\}_{1\leq i\leq 2^d}$.

*Proof.* For a fixed generator, the optimum Labeler $D^*_{LR}$, Anti-Labeler $D^*_{LG}$, and discriminator $D^*$ obey the following relations by Prop 3, Lemma 3, and Lemma 4:

$$
\begin{aligned}
&(1 - D^*(x))/D^*(x) = \mathbb{P}_g(x)/\mathbb{P}_r(x) \\
&D^*_{LR}(x)[j] = \mathbb{P}_r(l = j|x)\ \forall j \\
&D^*_{LG}(x)[j] = \mathbb{P}_g(l = 1|x)\ \forall j.
\end{aligned}
\tag{A.12}
$$

Then substitution into the generator objective $C(G)$ yields

$$
\begin{aligned}
C(G) &= \sum_{j=1}^{2^d} \rho_j \mathbb{E}_{x\sim\mathbb{P}_g(x|l=j)} \left[ \log\left(\frac{\mathbb{P}_g(x)}{\mathbb{P}_r(x)}\right) + \log(\mathbb{P}_g(l = j|x)) - \log(\mathbb{P}_r(l = j|x)) \right] \\
&= \sum_{j=1}^{2^d} \rho_j \mathbb{E}_{x\sim\mathbb{P}_g(x|l=j)} \left[ \log\left(\frac{\mathbb{P}_g(l = j, x)}{\mathbb{P}_r(l = j, x)}\right) \right] \\
&= \mathbb{E}_{(l,x)\sim\mathbb{P}_g(l,x)} \left[ \log\left(\frac{\mathbb{P}_g(l, x)}{\mathbb{P}_d(l, x)}\right) \right] \\
&= (\mathbb{P}_g)\, \mathbb{P}_d.
\end{aligned}
\tag{A.13}
$$

where KL is the Kullback-Leibler divergence, which is minimized if and only if $\mathbb{P}_g = \mathbb{P}_d$ jointly over labels and images.

□

### A.1.7 CausalGAN Extension to dabels Under Deterministic Labels

While the previous section showed how to ensure $\mathbb{P}_g(l, x) = \mathbb{P}_r(l, x)$ by relabeling combinations of a $d$ binary labels as a $2^d$ label, this may be difficult in practice for a large number of labels and we do not adopt this approach in practice.

Instead, in this section, we provide the theoretical guarantees for the implemented CausalGAN architecture with $d$ labels under the assumption that the relationship between the image and its labels is deterministic in the dataset, i.e., there is a deterministic function that maps an image to the corresponding label vector. Later we show that this assumption is sufficient to guarantee that the global optimal generator samples from the class conditional distributions.

First, let us restate the loss functions more formally. Note that $D_{LR}(x), D_{LG}(x)$ are $d-$dimensional vectors. The Labeler solves the following optimization problem:

$$\max_{D_{LR}} \rho_j \mathbb{E}_{x\sim\mathbb{P}_r(x|l_j=1)} \log(D_{LR}(x)[j]) + (1-\rho_j)\mathbb{E}_{x\sim\mathbb{P}_r(x|l_j=0)} \log(1-D_{LR}(x)[j]). \quad \text{(A.14)}$$

where $\mathbb{P}_r(x|l_j = 0) \coloneqq \mathbb{P}(X = x|l_j = 0)$, $\mathbb{P}_r(x|l_j = 0) \coloneqq \mathbb{P}(X = x|l_j = 0)$ and $\rho_j = \mathbb{P}(l_j = 1)$. For a fixed generator, the Anti-Labeler solves the following optimization problem:

$$\max_{D_{LG}} \rho_j \mathbb{E}_{\mathbb{P}_g(x|l_j=1)} \log(D_{LG}(x)[j]) + (1-\rho_j)\mathbb{E}_{\mathbb{P}_g(x|l_j=0)} \log(1-D_{LG}(x)[j]), \quad \text{(A.15)}$$

where $\mathbb{P}_g(x|l_j = 0) \coloneqq \mathbb{P}_g(x|l_j = 0)$, $\mathbb{P}_g(x|l_j = 0) \coloneqq \mathbb{P}_g(x|l_j = 0)$. For a fixed discriminator, Labeler and Anti-Labeler, the generator solves the following optimization

problem:

$$\min_G \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log(D(x))\right] + \mathbb{E}_{x\sim p_{\text{g}}(x)}\left[\log\left(\frac{1-D(x)}{D(x)}\right)\right]$$
$$-\frac{1}{d}\sum_{j=1}^{d}\rho_j\mathbb{E}_{x\sim\mathbb{P}_g(x|l_j=1)}\left[\log(D_{LR}(X)[j])\right] - (1-\rho_j)\mathbb{E}_{x\sim\mathbb{P}_g(x|l_j=0)}\left[\log(1-D_{LR}(X)[j])\right]$$
$$+\frac{1}{d}\sum_{j=1}^{d}\rho_j\mathbb{E}_{x\sim\mathbb{P}_g(x|l_j=1)}\left[\log(D_{LG}(X)[j])\right] + (1-\rho_j)\mathbb{E}_{x\sim\mathbb{P}_g(x|l_j=0)}\left[\log(1-D_{LG}(X)[j])\right]. \tag{A.16}$$

We have the following proposition, which characterizes the optimum generator for optimum Labeler, Anti-Labeler and Discriminator:

**Proposition 4.** *Define $C(G)$ as the generator loss for when discriminator, Labeler and Anti-Labeler are at their optimum obtained from (A.16). The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $\mathbb{P}_g(x|l_i) = \mathbb{P}_r(x|l_i)\forall i \in [d]$ and $\mathbb{P}_g(x) = \mathbb{P}_r(x)$.*

*Proof.* Proof follows the same lines as in the proof of Theorem 1 and Theorem 8 and is omitted. □

Thus we have

$$\mathbb{P}_r(x, l_i) = \mathbb{P}_g(x, l_i), \forall i \in [d] \text{ and } \mathbb{P}_r(x) = \mathbb{P}_g(x). \tag{A.17}$$

However, this does not in general imply $\mathbb{P}_r(x, l_1, l_2, \ldots, l_d) = \mathbb{P}_g(x, l_1, l_2, \ldots, l_d)$, which is equivalent to saying the generated distribution samples from the class conditional image distributions. To guarantee the correct conditional sampling given all labels, we introduce the following assumption: We assume that the image $x$ determines all the labels. This assumption is very relevant in practice. For example, in the CelebA dataset, which we use,

the label vector, e.g., whether the person is a male or female, with or without a mustache, can be thought of as a deterministic function of the image. When this is true, we can say that $\mathbb{P}_r(l_1, l_2, \ldots, l_n|x) = \mathbb{P}_r(l_1|x)\mathbb{P}_r(l_2|x)\ldots\mathbb{P}_r(l_n|x)$.

We need the following lemma, where kronecker delta function refers to the functions that take the value of 1 only on a single point, and 0 everywhere else:

**Lemma 5.** *Any discrete joint probability distribution, where all the marginal probability distributions are kronecker delta functions is the product of these marginals.*

*Proof.* Let $\delta_{\{x-u\}}$ be the kronecker delta function which is 1 if $x = u$ and is 0 otherwise. Consider a joint distribution $p(X_1, X_2, \ldots, X_n)$, where $p(X_i) = \delta_{\{X_i-u_i\}}, \forall i \in [n]$, for some set of elements $\{u_i\}_{i\in[n]}$. We will show by contradiction that the joint probability distribution is zero everywhere except at $(u_1, u_2, \ldots, u_n)$. Then, for the sake of contradiction, suppose for some $v = (v_1, v_2, \ldots, v_n) \neq (u_1, u_2, \ldots, u_n)$, $p(v_1, v_2, \ldots, v_n) \neq 0$. Then $\exists j \in [n]$ such that $v_j \neq u_j$. Then we can marginalize the joint distribution as

$$p(v_j) = \sum_{X_1,\ldots,X_{j-1},X_j,\ldots,X_n} p(X_1, \ldots, X_{j-1}, v_j, X_{j+1}, \ldots, X_n) > 0, \tag{A.18}$$

where the inequality is due to the fact that the particular configuration $(v_1, v_2, \ldots, v_n)$ must have contributed to the summation. However this contradicts with the fact that $p(X_j) = 0, \forall X_j \neq u_j$. Hence, $p(.)$ is zero everywhere except at $(u_1, u_2, \ldots, u_n)$, where it should be 1. □

We can now simply apply the above lemma on the conditional distribution $\mathbb{P}_g(l_1, l_2, \ldots, l_d|x)$. Proposition 4 shows that the image distributions and the marginals $\mathbb{P}_g(l_i|x)$ are true to the data distribution due to Bayes' rule. Since the vector $(l_1, \ldots, l_n)$ is a deterministic function of $x$ by assumption, $\mathbb{P}_r(l_i|x)$ are kronecker delta functions, and so are $\mathbb{P}_g(l_i|x)$ by Proposition 4. Thus, since the joint $\mathbb{P}_g(x, l_1, l_2, \ldots, l_d)$ satisfies the condition

that every marginal distribution $p(l_i|x)$ is a kronecker delta function, then it must be a product distribution by Lemma 5. Thus we can write

$$\mathbb{P}_g(l_1, l_2, \ldots, l_d|x) = \mathbb{P}_g(l_1|x)\mathbb{P}_g(l_2|x)\ldots\mathbb{P}_g(l_n|x).$$

Then we have the following chain of equalities.

$$\begin{aligned}
\mathbb{P}_r(x, l_1, l_2, \ldots, l_d) &= \mathbb{P}_r(l_1, \ldots, l_n|x)\mathbb{P}_r(x) \\
&= \mathbb{P}_r(l_1|x)\mathbb{P}_r(l_2|x)\ldots\mathbb{P}_r(l_n|x)\mathbb{P}_r(x) \\
&= \mathbb{P}_g(l_1|x)\mathbb{P}_g(l_2|x)\ldots\mathbb{P}_g(l_n|x)\mathbb{P}_g(x) \\
&= \mathbb{P}_g(l_1, l_2, \ldots, l_d|x)\mathbb{P}_g(x) \\
&= \mathbb{P}_g(x, l_1, l_2, \ldots, l_d).
\end{aligned}$$

Thus, we also have $\mathbb{P}_r(x|l_1, l_2, \ldots, l_n) = \mathbb{P}_g(x|l_1, l_2, \ldots, l_n)$ since $\mathbb{P}_r(l_1, l_2, \ldots, l_n)$ $= \mathbb{P}_g(l_1, l_2, \ldots, l_n)$, concluding the proof that the optimum generator samples from the class conditional image distributions.

### A.1.8 CausalBEGAN Architecture

In this section, we propose a simple, but non-trivial extension of BEGAN where we feed image labels to the generator. One of the central contributions of BEGAN (Berthelot et al. [2017]) is a control theory-inspired boundary equilibrium approach that encourages generator training only when the discriminator is near optimum and its gradients are the most informative. The following observation helps us carry the same idea to the case with labels: Label gradients are most informative when the image quality is high. Here, we introduce a new loss and a set of margins that reflect this intuition.

Formally, let $\mathcal{L}(x)$ be the average $L_1$ pixel-wise autoencoder loss for an image $x$, as

in BEGAN. Let $\mathcal{L}_{sq}(u, v)$ be the squared loss term, i.e., $\|u - v\|_2^2$. Let $(x, l_x)$ be a sample from the data distribution, where $x$ is the image and $l_x$ is its corresponding label. Similarly, $G(z, l_g)$ is an image sample from the generator, where $l_g$ is the label used to generate this image. Denoting the space of images by $\mathcal{I}$, let $G : \mathbb{R}^n \times \{0, 1\}^m \mapsto \mathcal{I}$ be the generator. As a naive attempt to extend the original BEGAN loss formulation to include the labels, we can write the following loss functions:

$$Loss_D = \mathcal{L}(x) - \mathcal{L}(Labeler(G(z, l))) + \mathcal{L}_{sq}(l_x, Labeler(x)) - \mathcal{L}_{sq}(l_g, Labeler(G(z, l_g))),$$
$$Loss_G = \mathcal{L}(G(z, l_g)) + \mathcal{L}_{sq}(l_g, Labeler(G(z, l_g))). \tag{A.19}$$

However, this naive formulation does not address the use of margins, which is extremely critical in the BEGAN formulation. Just as a better trained BEGAN discriminator creates more useful gradients for image generation, a better trained Labeler is a prerequisite for meaningful gradients. This motivates an additional margin-coefficient tuple $(b_2, c_2)$, as shown in (A.20,A.21).

The generator tries to jointly minimize the two loss terms in the formulation in (A.19). We empirically observe that occasionally the image quality will suffer because the images that best exploit the Labeler network are often not obliged to be realistic, and can be noisy or misshapen. Based on this, label loss seems unlikely to provide useful gradients unless the image quality remains good. Therefore we encourage the generator to incorporate label loss only when the *image quality margin* $b_1$ is large compared to the *label margin* $b_2$. To achieve this, we introduce a new *margin of margins* term, $b_3$. As a result, the margin equations and update rules are summarized as follows, where $\lambda_1, \lambda_2, \lambda_3$ are learning rates

for the coefficients.

$$
\begin{aligned}
b_1 &= \gamma_1 * \mathcal{L}(x) - \mathcal{L}(G(z, l_g)). \\
b_2 &= \gamma_2 * \mathcal{L}_{sq}(l_x, Labeler(x)) - \mathcal{L}_{sq}(l_g, Labeler(G(z, l_g))). \qquad \text{(A.20)} \\
b_3 &= \gamma_3 * relu(b_1) - relu(b_2). \\
c_1 &\leftarrow clip_{[0,1]}(c_1 + \lambda_1 * b_1). \\
c_2 &\leftarrow clip_{[0,1]}(c_2 + \lambda_2 * b_2). \qquad \text{(A.21)} \\
c_3 &\leftarrow clip_{[0,1]}(c_3 + \lambda_3 * b_3).
\end{aligned}
$$

$$
Loss_D = \mathcal{L}(x) - c_1 * \mathcal{L}(G(z, l_g)) + \mathcal{L}_{sq}(l_x, Labeler(x)) - c_2 * \mathcal{L}_{sq}(l_g, G(z, l_g)). \qquad \text{(A.22)}
$$

$$
Loss_G = \mathcal{L}(G(z, l_g)) + c_3 * \mathcal{L}_{sq}(l_g, Labeler(G(z, l_g))).
$$

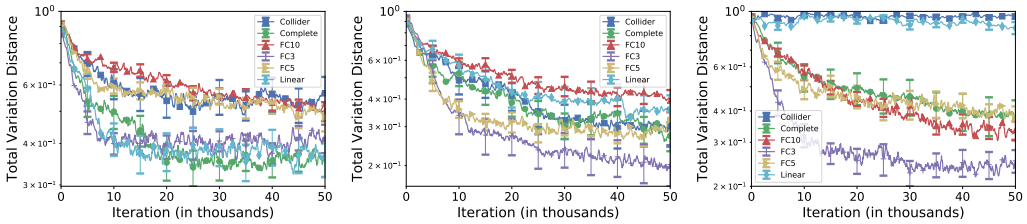
One of the advantages of BEGAN is the existence of a monotonically decreasing scalar which can track the convergence of the gradient descent optimization. Our extension preserves this property as we can define

$$
\mathcal{M}_{complete} = \mathcal{L}(x) + |b_1| + |b_2| + |b_3|, \qquad \text{(A.23)}
$$

and show that $\mathcal{M}_{complete}$ decreases progressively during our optimizations. See Figure A.12.

### A.1.9 Dependence of GAN Behavior on Causal Graph

In Section 2.4 we showed how a GAN could be used to train a causal implicit generative model by incorporating the causal graph into the generator structure. Here we investigate the behavior and convergence of causal implicit generative models when the true data distribution arises from another (possibly distinct) causal graph.




(a) $X \to Y \to Z$ (b) $X \to Y \leftarrow Z$ (c) $X \to Y \to Z, X \to Z$

Figure A.2: Convergence in total variation distance of generated distribution to the true distribution for causal implicit generative model, when the generator is structured based on different causal graphs. (a) Data generated from line graph $X \to Y \to Z$. The best convergence behavior is observed when the true causal graph is used in the generator architecture. (b) Data generated from collider graph $X \to Y \leftarrow Z$. Fully connected layers may perform better than the true graph depending on the number of layers. Collider and complete graphs performs better than the line graph which implies the wrong Bayesian network. (c) Data generated from complete graph $X \to Y \to Z, X \to Z$. Fully connected with 3 layers performs the best, followed by the complete and fully connected with 5 and 10 layers. Line and collider graphs, which implies the wrong Bayesian network does not show convergence behavior.

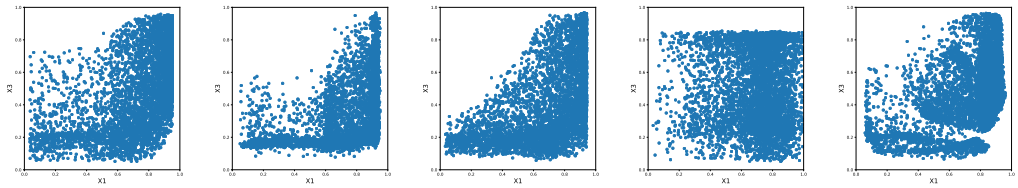
We consider causal implicit generative model convergence on synthetic data whose three features $\{X, Y, Z\}$ arise from one of three causal graphs: "line" $X \to Y \to Z$ , "collider" $X \to Y \leftarrow Z$, and "complete" $X \to Y \to Z, X \to Z$. For each node a (randomly sampled once) cubic polynomial in $n + 1$ variables computes the value of that node given its $n$ parents and 1 uniform exogenous variable. We then repeat, creating a new synthetic dataset in this way for each causal model and report the averaged results of 20 runs for each model.

For each of these data generating graphs, we compare the convergence of the joint distribution to the true joint in terms of the total variation distance, when the generator is structured according to a line, collider, or complete graph. For completeness, we also include generators with no knowledge of causal structure: $\{fc3, fc5, fc10\}$ are fully connected neural networks that map uniform random noise to 3 output variables using either 3,5, or 10 layers respectively.

The results are given in Figure A.2. Data is generated from line causal graph $X \to Y \to Z$ (left panel), collider causal graph $X \to Y \leftarrow$ (middle panel), and complete causal graph $X \to Y \to Z, X \to Z$ (right panel). Each curve shows the convergence behavior of the generator distribution, when generator is structured based on each one of these causal graphs. We expect convergence when the causal graph used to structure the generator is capable of generating the joint distribution due to the true causal graph: as long as we use the correct Bayesian network, we should be able to fit to the true joint. For example, complete graph can encode all joint distributions. Hence, we expect complete graph to work well with all data generation models. Standard fully connected layers correspond to the causal graph with a latent variable causing all the observable variables. Ideally, this model should be able to fit to any causal generative model. However, the convergence behavior of adversarial training across these models is unclear, which is what we are exploring with Figure A.2.

For the line graph data $X \to Y \to Z$, we see that the best convergence behavior is when line graph is used in the generator architecture. As expected, complete graph also converges well, with slight delay. Similarly, fully connected network with 3 layers show good performance, although surprisingly fully connected with 5 and 10 layers perform much worse. It seems that although fully connected can encode the joint distribution in theory, in practice with adversarial training, the number of layers should be tuned to achieve the same performance as using the true causal graph. Using the wrong Bayesian network, the collider, also yields worse performance.

For the collider graph, surprisingly using a fully connected generator with 3 and 5 layers shows the best performance. However, consistent with the previous observation, the number of layers is important, and using 10 layers gives the worst convergence behavior. Using complete and collider graphs achieves the same decent performance, whereas line



(a) $X_1 \to X_2 \to X_3$ (b) $X_1 \to X_2 \to X_3$ (c) $X_1 \to X_2 \to X_3$ $X_1 \to X_3$ (d) $X_1 \to X_2 \leftarrow X_3$ (e) Fully connected

Figure A.3: Synthetic data experiments: (a) Scatter plot for actual data. Data is generated using the causal graph $X_1 \to X_2 \to X_3$. (b) Generated distribution when generator causal graph is $X_1 \to X_2 \to X_3$. (c) Generated distribution when generator causal graph is $X_1 \to X_2 \to X_3 \cup X_1 \to X_3$. (d) Generated distribution when generator causal graph is $X_1 \to X_2 \leftarrow X_3$. (e) Generated distribution when generator is from a fully connected last layer of a 5 layer FF neural net.

graph, a wrong Bayesian network, performs worse than the two.

For the complete graph, fully connected 3 performs the best, followed by fully connected 5, 10 and the complete graph. As we expect, line and collider graphs, which cannot encode all the distributions due to a complete graph, performs the worst and does not actually show any convergence behavior.

### A.1.10 Additional Simulations for Causal Controller

First, we evaluate the effect of using the wrong causal graph on an artificially generated dataset. Figure A.3 shows the scatter plot for the two coordinates of a three dimensional distribution. As we observe, using the correct graph gives the closest scatter plot to the original data, whereas using the wrong Bayesian network, collider graph, results in a very different distribution.

Second, we expand on the causal graphs used for experiments for the CelebA dataset. We use a causal graph on a subset of the image labels of CelebA dataset, which we call CelebA Causal Graph (G1), illustrated in Figure A.1. The graph cG1, which is a completed

| Label Pair | | Male | |
|---|---|---|---|
| | | 0 | 1 |
| Young | 0 | 0.14[0.07](0.07) | 0.09[0.15](0.15) |
| | 1 | 0.47[0.51](0.51) | 0.29[0.27](0.26) |
| Mustache | 0 | 0.61[0.58](0.58) | 0.34[0.38](0.38) |
| | 1 | 0.00[0.00](0.00) | 0.04[0.04](0.04) |

Table A.1: Pairwise marginal distribution for select label pairs when Causal Controller is trained on $G1$ in plain text, its completion $cG1$[square brackets], and the true pairwise distribution(in parentheses). Note that $G1$ treats Male and Young labels as independent, but does not completely fail to generate a reasonable (product of marginals) approximation. Also note that when an edge is added $Young \rightarrow Male$, the learned distribution is nearly exact. Note that both graphs contain the edge $Male \rightarrow Mustache$ and so are able to learn that women have no mustaches.

version of G1, is the complete graph associated with the ordering: Young, Male, Eyeglasses, Bald, Mustache, Smiling, Wearing Lipstick, Mouth Slightly Open, Narrow Eyes. For example, in cG1 Male causes Smiling because Male comes before Smiling in the ordering. The graph rcG1 is formed by reversing every edge in cG1.

Next, we check the effect of using the incorrect Bayesian network for the data. The causal graph G1 generates Male and Young independently, which is incorrect in the data. Comparison of pairwise distributions in Table A.1 demonstrate that for G1 a reasonable approximation to the true distribution is still learned for {Male, Young} jointly. For cG1 a nearly perfect distributional approximation is learned. Furthermore we show that despite this inaccuracy, both graphs G1 and cG1 lead to Causal Controllers that never output the label combination {Female,Mustache}, which will be important later.

Wasserstein GAN in its original form (with Lipshitz discriminator) assures convergence in distribution of the Causal Controller output to the discretely supported distribution of labels. We use a slightly modified version of Wasserstein GAN with a penalized gradient (Gulrajani et al. [2017]). We first demonstrate that learned outputs actually have "approximately discrete" support. In Figure A.4a, we sample the joint label distribution 1000 times,

| **Label, L** | $\mathbb{P}_{G1}(L=1)$ | $\mathbb{P}_{cG1}(L=1)$ | $\mathbb{P}_{D}(L=1)$ |
|---|---|---|---|
| Bald | 0.02244 | 0.02328 | 0.02244 |
| Eyeglasses | 0.06180 | 0.05801 | 0.06406 |
| Male | 0.38446 | 0.41938 | 0.41675 |
| Mouth Slightly Open | 0.49476 | 0.49413 | 0.48343 |
| Mustache | 0.04596 | 0.04231 | 0.04154 |
| Narrow Eyes | 0.12329 | 0.11458 | 0.11515 |
| Smiling | 0.48766 | 0.48730 | 0.48208 |
| Wearing Lipstick | 0.48111 | 0.46789 | 0.47243 |
| Young | 0.76737 | 0.77663 | 0.77362 |

Table A.2: Marginal distribution of pretrained Causal Controller labels when Causal Controller is trained on CelebA Causal Graph ($P_{G1}$) and its completion($P_{cG1}$), where $cG1$ is the (nonunique) largest DAG containing $G1$ (see appendix). The third column lists the actual marginal distributions in the dataset

and make a histogram of the (all) scalar outputs corresponding to any label.

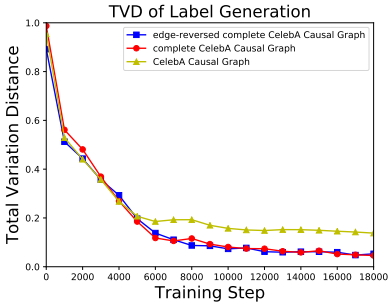
Although Figure A.4b demonstrates conclusively good convergence for both graphs, TVD is not always intuitive. For example, "how much can each marginal be off if there are 9 labels and the TVD is 0.14?". To expand upon Figure A.2 where we showed that the causal controller learns the correct distribution for a pairwise subset of nodes, here we also show that both CelebA Causal Graph (G1) and the completion we define (cG1) allow training of very reasonable marginal distributions for all labels (Table A.1) that are not off by more than 0.03 for the worst label. $\mathbb{P}_D(L=1)$ is the probability that the label is 1 in the dataset, and $\mathbb{P}_G(L=1)$ is the probability that the generated label is (around a small neighborhood of ) 1.

### A.1.11 Wasserstein Causal Controller on CelebA Labels

We test the performance of our Wasserstein Causal Controller on a subset of the binary labels of CelebA datset. We use the causal graph given in Figure A.1.

For causal graph training, first we verify that our Wasserstein training allows the generator to learn a mapping from continuous uniform noise to a discrete distribution. Figure


(a) Essentially Discrete Range of Causal Controller




(b) TVD vs. No. of Iters in CelebA Labels

Figure A.4: (a) A number line of unit length binned into 4 unequal bins along with the percent of Causal Controller ($G1$) samples in each bin. Results are obtained by sampling the joint label distribution 1000 times and forming a histogram of the scalar outputs corresponding to any label. Note that our Causal Controller output labels are approximately discrete even though the input is a continuum (uniform). The 4% between 0.05 and 0.95 is not at all uniform and almost zero near 0.5. (b) Progression of total variation distance between the Causal Controller output with respect to the number of iterations: CelebA Causal Graph is used in the training with Wasserstein loss.

A.4a shows where the samples, averaged over all the labels in CelebA Causal Graph, from this generator appears on the real line. The result emphasizes that the proposed Causal Controller outputs an almost discrete distribution: $96\%$ of the samples appear in $0.05-$neighborhood of $0$ or $1$. Outputs shown are *unrounded* generator outputs.

A stronger measure of convergence is the total variational distance (TVD). For CelebA Causal Graph (G1), our defined completion (cG1), and cG1 with arrows reversed (rcG1), we show convergence of TVD with training (Figure A.4b). Both cG1 and rcG1 have TVD decreasing to 0, and TVD for G1 assymptotes to around 0.14 which corresponds to the incorrect conditional independence assumptions that G1 makes. This suggests that any given complete causal graph will lead to a nearly perfect implicit causal generator over labels and that bayesian partially incorrect causal graphs can still give reasonable convergence.

### A.1.12 More CausalGAN Results

In this section, we present additional CausalGAN results in Figure A.5, A.6.

Intervening vs Conditioning on Wearing Lipstick, Top: Intervene Wearing Lipstick=1, Bottom: Condition Wearing Lipstick=1

Figure A.5: Intervening/Conditioning on Wearing Lipstick label in CelebA Causal Graph. Since $Male \rightarrow WearingLipstick$ in CelebA Causal Graph, we do not expect $do(\textit{Wearing Lipstick} = 1)$ to affect the probability of $Male = 1$, i.e., $\mathbb{P}(Male = 1|do(\textit{Wearing Lipstick} = 1)) = \mathbb{P}(Male = 1) = 0.42$. Accordingly, the top row shows both males and females who are wearing lipstick. However, the bottom row of images sampled from the conditional distribution $\mathbb{P}(.|\textit{Wearing Lipstick} = 1)$ shows only female images because in the dataset $\mathbb{P}(Male = 0|\textit{Wearing Lipstick} = 1) \approx 1$.

Intervening vs Conditioning on Narrow Eyes, Top: Intervene Narrow Eyes=1, Bottom: Condition Narrow Eyes=1

Figure A.6: Intervening/Conditioning on Narrow Eyes label in CelebA Causal Graph. Since $Smiling \rightarrow \textit{Narrow Eyes}$ in CelebA Causal Graph, we do not expect $do(\textit{Narrow Eyes} = 1)$ to affect the probability of $Smiling = 1$, i.e., $\mathbb{P}(Smiling = 1|do(\textit{Narrow Eyes} = 1)) = \mathbb{P}(Smiling = 1) = 0.48$. However on the bottom row, conditioning on *Narrow Eyes* $= 1$ increases the proportion of smiling images (From $0.48$ to $0.59$ in the dataset), although 10 images may not be enough to show this difference statistically.

### A.1.13 More CausalBEGAN Results

In this section, we train CausalBEGAN on CelebA dataset using CelebA Causal Graph. The Causal Controller is pretrained with a Wasserstein loss and used for training the CausalBE-

GAN.

To first empirically justify the need for the margin of margins we introduced in (A.22) ($c_3$ and $b_3$), we train the same CausalBEGAN model setting $c_3 = 1$, removing the effect of this margin. We show that the image quality for rare labels deteriorates. Please see Figure A.11 in the appendix. Then for the labels *Bald*, and *Mouth Slightly Open*, we illustrate the difference between interventional and conditional sampling when the label is 1. (Figures A.7, A.8).

Intervening vs Conditioning on Bald, Top: Intervene Bald=1, Bottom: Condition Bald=1

Figure A.7: Intervening/Conditioning on Bald label in CelebA Causal Graph. Since $Male \rightarrow Bald$ in CelebA Causal Graph, we do not expect $do(Bald = 1)$ to affect the probability of $Male = 1$, i.e., $\mathbb{P}(Male = 1|do(Bald = 1)) = \mathbb{P}(Male = 1) = 0.42$. Accordingly, the top row shows both bald males and bald females. The bottom row of images sampled from the conditional distribution $\mathbb{P}(.|Bald = 1)$ shows only male images because in the dataset $\mathbb{P}(Male = 1|Bald = 1) \approx 1$.

### A.1.14 Label Sweeping and Diversity for CausalGAN

In this section, we provide additional simulations for CausalGAN. In Figures A.9a-A.9d, we show the conditional image generation properties of CausalGAN by sweeping a single label from 0 to 1 while keeping all other inputs/labels fixed. In Figure A.10, to examine the degree of mode collapse and show the image diversity, we show 256 randomly sampled images.

Intervening vs Conditioning on Mouth Slightly Open, Top: Intervene Mouth Slightly Open=1, Bottom: Condition Mouth Slightly Open=1

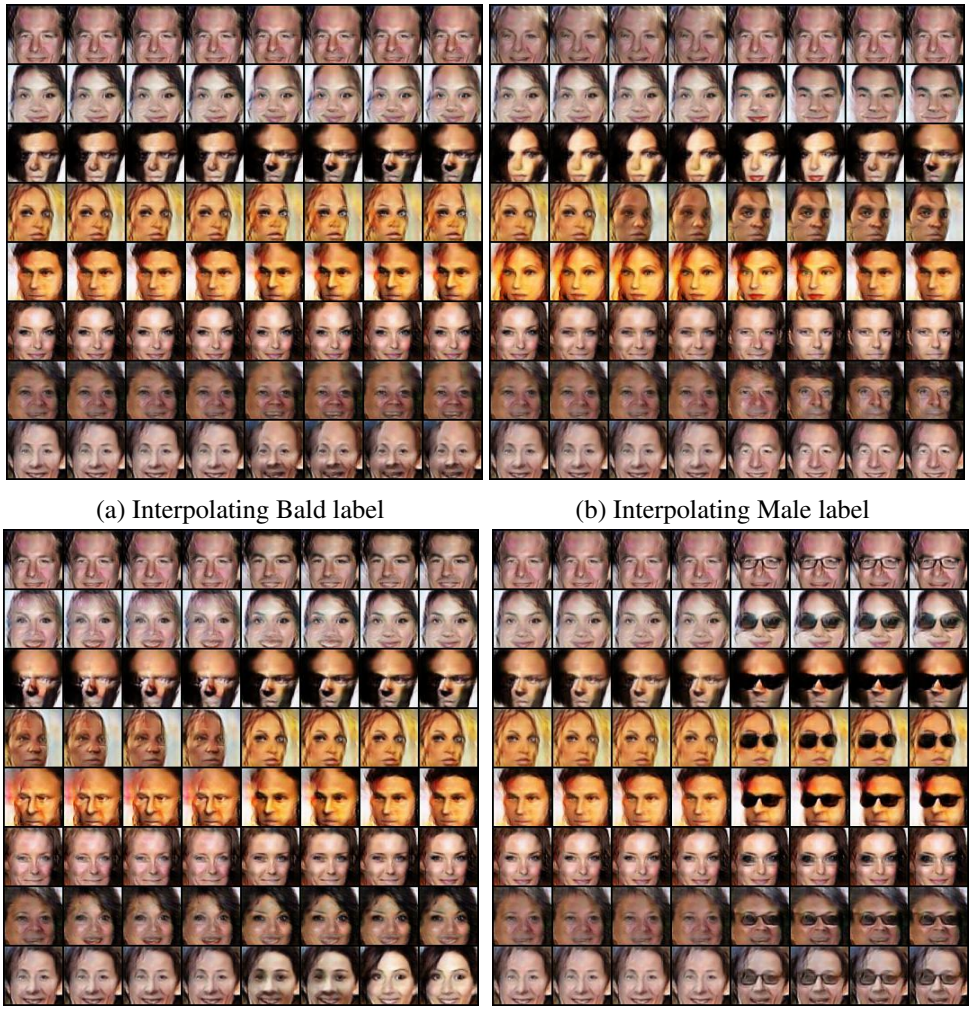
Figure A.8: Intervening/Conditioning on Mouth Slightly Open label in CelebA Causal Graph. Since $Smiling \rightarrow MouthSlightlyOpen$ in CelebA Causal Graph, we do not expect $do(\textbf{\textit{Mouth Slightly Open}} = 1)$ to affect the probability of $Smiling = 1$, i.e., $\mathbb{P}(Smiling = 1|do(\textbf{\textit{Mouth Slightly Open}} = 1)) = \mathbb{P}(Smiling = 1) = 0.48$. However on the bottom row, conditioning on ***Mouth Slightly Open*** $= 1$ increases the proportion of smiling images (From $0.48$ to $0.76$ in the dataset), although 10 images may not be enough to show this difference statistically.

### A.1.15 Additional CausalBEGAN Simulations

In this section, we provide additional simulation results for CausalBEGAN. First we show that although our third margin term $b_3$ introduces complications, it can not be ignored. Figure A.11 demonstrates that omitting the third margin on the image quality of rare labels.

Furthermore just as the setup in BEGAN permitted the definiton of a scalar "$\mathcal{M}$", which was monotonically decreasing during training, our definition permits an obvious extension $\mathcal{M}_{complete}$ (defined in A.23) that preserves these properties. See Figure A.12 to observe $\mathcal{M}_{complete}$ decreaing monotonically during training.

We also show the conditional image generation properties of CausalBEGAN by using "label sweeps" that move a single label input from 0 to 1 while keeping all other inputs fixed (Figures A.13a -A.13d ). It is interesting to note that while generators are often implicitly thought of as continuous functions, the generator in this CausalBEGAN architecture learns a discrete function with respect to its label input parameters. (Initially there is label interpolation, and later in the optimization label interpolation becomes more step function like (not shown)). Finally, to examine the degree of mode collapse and show



(a) Interpolating Bald label

(b) Interpolating Male label

(c) Interpolating Young label

(d) Interpolating Eyeglasses label

Figure A.9: The effect of interpolating a single label for CausalGAN, while keeping the noise terms and other labels fixed.

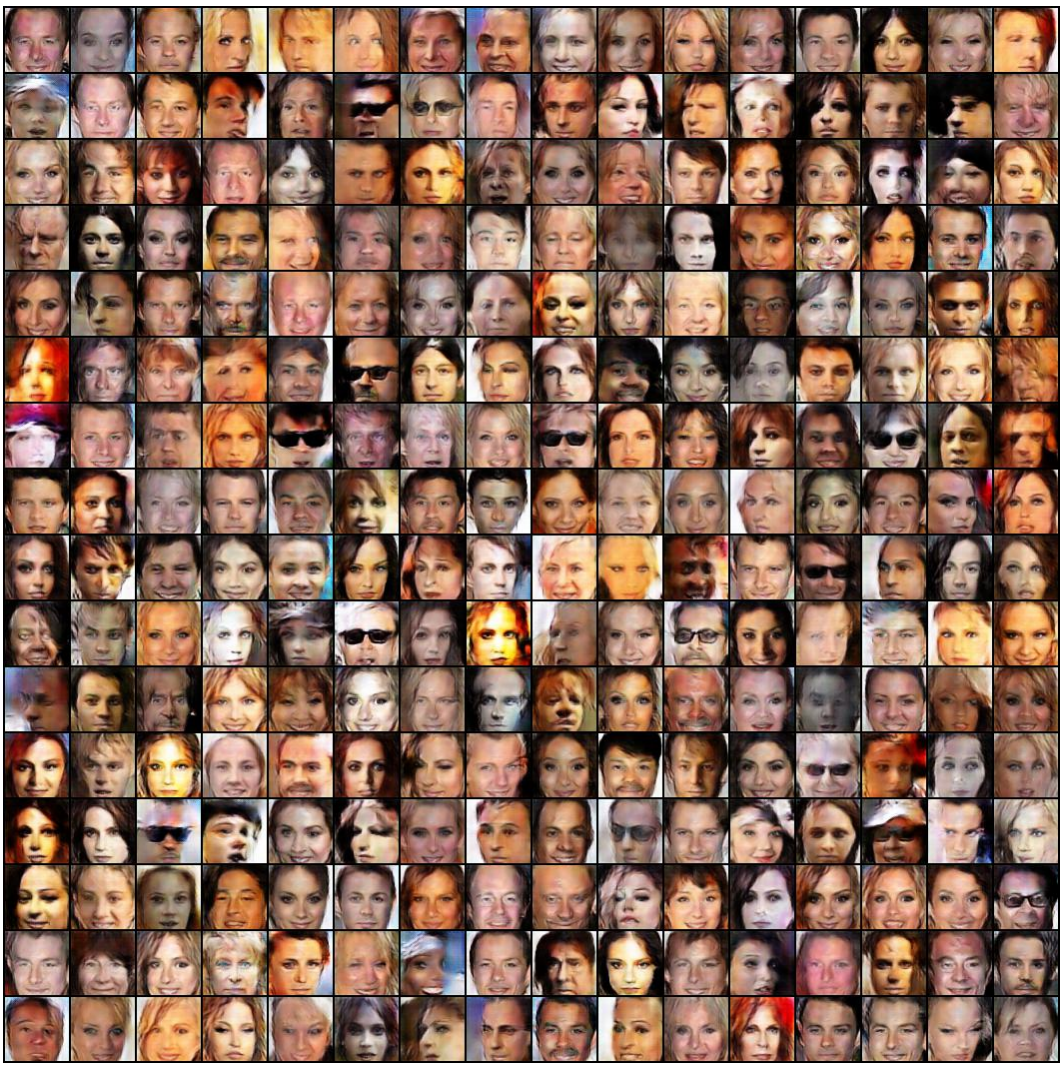


Figure A.10: Diversity of the proposed CausalGAN showcased with 256 samples.

the image diversity, we show a random sampling of 256 images (Figure A.14).

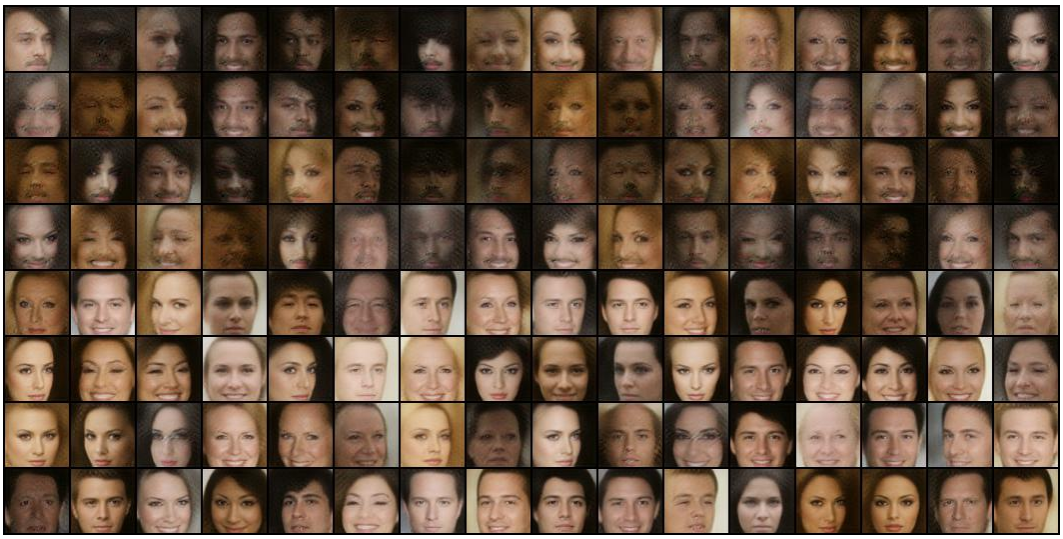


Figure A.11: Omitting the nonobvious margin $b_3 = \gamma_3 * relu(b_1) - relu(b_2)$ results in poorer image quality particularly for rare labels such as mustache. We compare samples from two interventional distributions. Samples from $\mathbb{P}(.|do(Mustache = 1))$ (top) have much poorer image quality compared to those under $\mathbb{P}(.|do(Mustache = 0))$ (bottom).

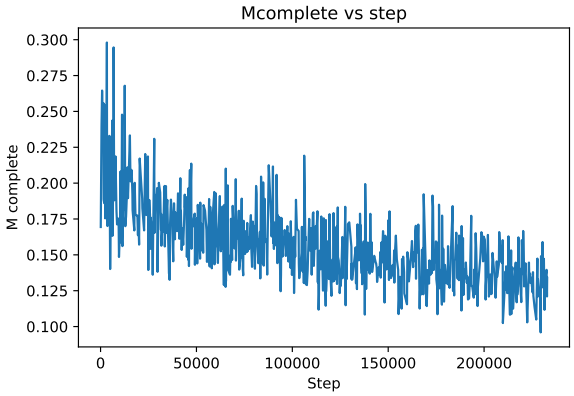
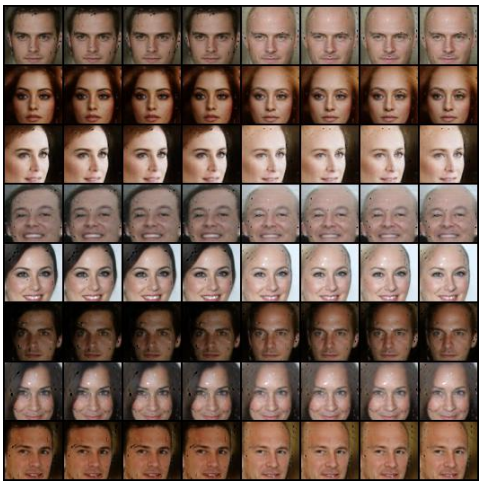
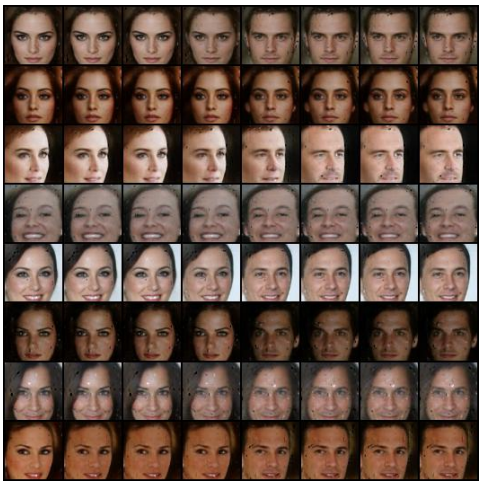


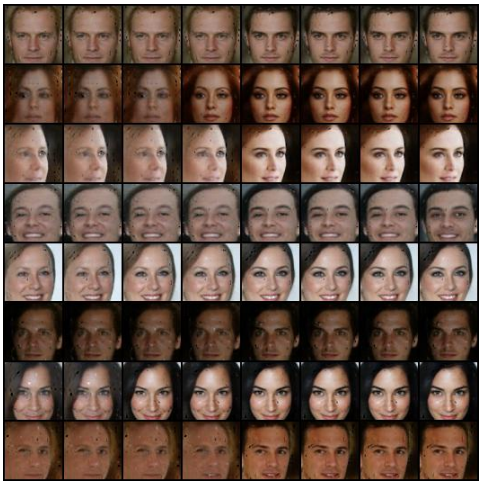

Figure A.12: Convergence of CausalBEGAN captured through the parameter $\mathcal{M}_{complete}$.



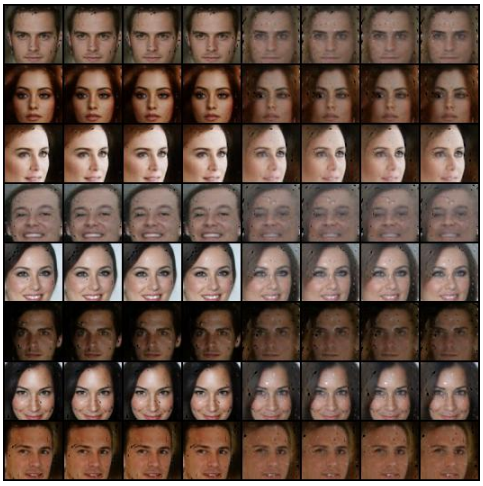
(a) Interpolating Bald label



(b) Interpolating Male label



(c) Interpolating Young label



(d) Interpolating Eyeglasses label

Figure A.13: The effect of interpolating a single label for CausalBEGAN, while keeping the noise terms and other labels fixed. Although most labels are properly captured, we see that eyeglasses label is not.

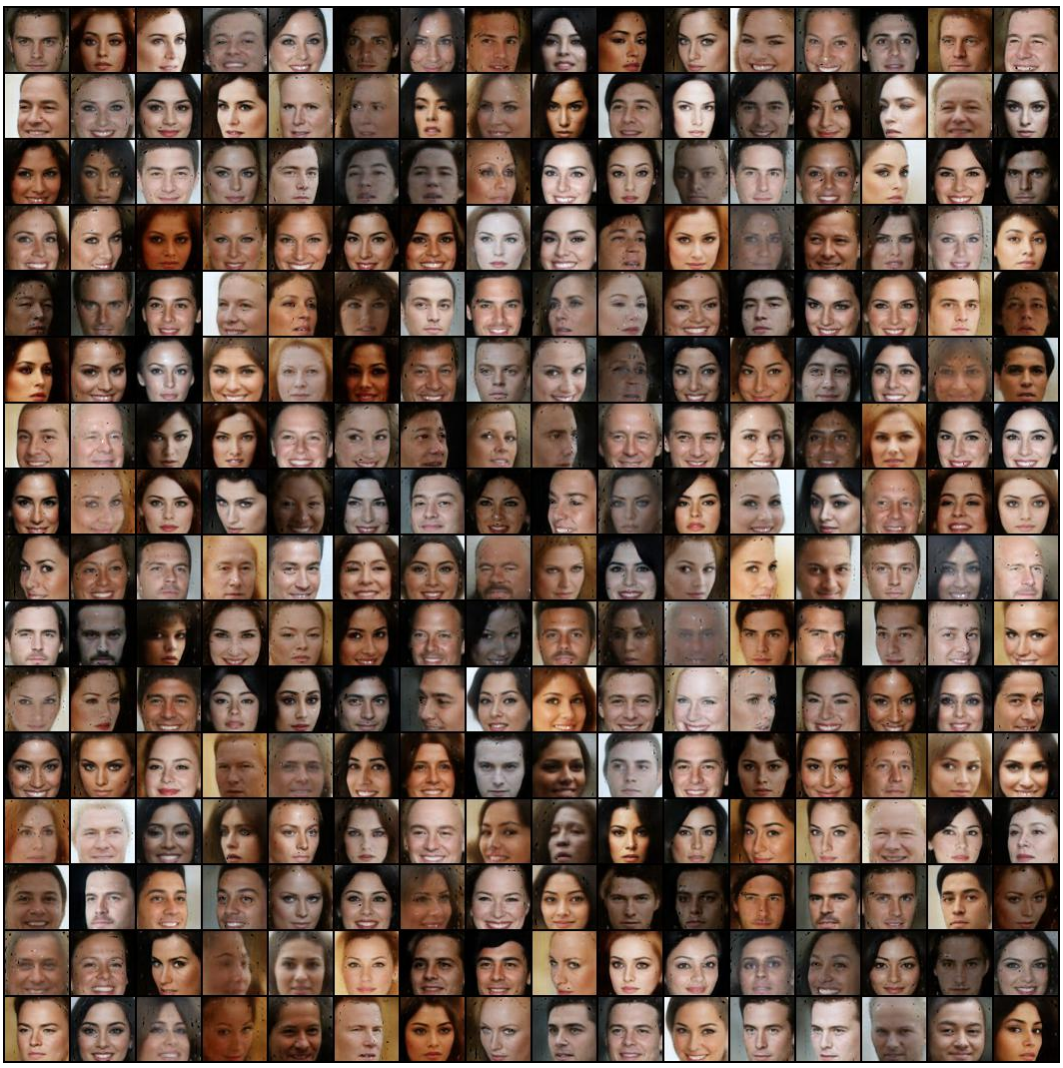


Figure A.14: Diversity of Causal BEGAN showcased with 256 samples.

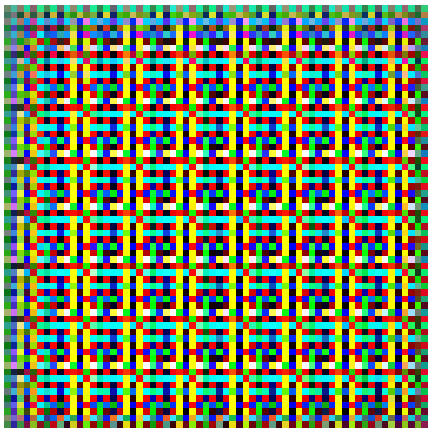


Figure A.15: Failed Image generation for simultaneous label and image generation after 20k steps.

### A.1.16 Directly Training CiGM for Labels+Image Fails

In this section, we present the result of attempting to jointly train an implicit causal generative model for labels and the image. This approach treats the image as part of the causal graph. It is not clear how exactly to feed both labels and image to discriminator, but one way is to simply encode the label as a constant image in an additional channel. We tried this for CelebA Causal Graph and observed that the image generation is not learned (Figure A.15). One hypothesis is that the discriminator focuses on labels without providing useful gradients to the image generation.

### A.1.17 Implementation

The differences between implementation and theory are explained below. Details for both CausalGAN and CausalBEGAN implementation are also explained.

### Pretraining Causal Controller for Face Labels

In this section, we explain the implementation details of the Wasserstein Causal Controller for generating face labels. We used the total variation distance (TVD) between the distribution of generator and data distribution as a metric to decide the success of the models.

The gradient term used as a penalty is estimated by evaluating the gradient at points interpolated between the real and fake batches. Interestingly, this Wasserstein approach gives us the opportunity to train the Causal Controller to output (almost) discrete labels (See Figure A.4a). In practice though, we still found benefit in rounding them before passing them to the generator.

The generator architecture is structured in accordance with Section 2.4 based on the causal graph in Figure A.1, using uniform noise as exogenous variables and 6 layer neural networks as functions mapping parents to children. For the training, we used 25 Wasserstein discriminator (critic) updates per generator update, with a learning rate of 0.0008.

### Implementation Details for CausalGAN

In practice, we use stochastic gradient descent to train our model. We use *DCGAN* Radford et al. [2015], a convolutional neural net-based implementation of generative adversarial networks, and extend it into our Causal GAN framework. We have expanded it by adding our Labeler networks, training a Causal Controller network and modifying the loss functions appropriately. Compared to DCGAN an important distinction is that we make 6 generator updates for each discriminator update on average. The discriminator and labeler networks are concurrently updated in a single iteration.

Notice that the loss terms defined in Section 2.5.2 contain a single binary label. In practice we feed a $d$-dimensional label vector and need a corresponding loss function. We extend the Labeler and Anti-Labeler loss terms by simply averaging the loss terms for every

label. The $i^{th}$ coordinates of the $d$-dimensional vectors given by the labelers determine the loss terms for label $i$. Note that this is different than the architecture given in Section A.1.6, where the discriminator outputs a length-$2^d$ vector and estimates the probabilities of all label combinations given the image. Therefore this approach does not have the guarantee to sample from the class conditional distributions, if the data distribution is not restricted. However, for the type of labeled image dataset we use in this work, where labels seem to be completely determined given an image, this architecture is sufficient to have the same guarantees. For the details, please see Section A.1.7 in the supplementary material.

Compared to the theory we have, another difference in the implementation is that we have swapped the order of the terms in the cross entropy expressions for labeler losses. This has provided sharper images at the end of the training.

**Conditional Image Generation for CausalBEGAN**

The labels input to CausalBEGAN are taken from the Causal Controller. We use very few parameter tunings. We use the same learning rate (0.00008) for both the generator and discriminator and do 1 update of each simultaneously (calculating the for each before applying either). We simply use $\gamma_1 = \gamma_2 = \gamma_3 = 0.5$. We do not expect the model to be very sensitive to these parameter values, as we achieve good performance without hyperparameter tweaking. We do use customized margin learning rates $\lambda_1 = 0.001, \lambda_2 = 0.00008, \lambda_3 = 0.01$, which reflect the asymmetry in how quickly the generator can respond to each margin. For example $c_2$ can have much more "spiky", fast responding behavior compared to others even when paired with a smaller learning rate, although we have not explored this parameter space in depth. In these margin behaviors, we observe that the best performing models have all three margins "active": near $0$ while frequently taking small positive values.

(a) No AL, $t = 20k$ (b) No AL, $t = 30k$ (c) No AL, $t = 40k$ (d) With AL, $t = 20k$

Figure A.16: CausalGAN results with and without Anti-Labeler for the rare label combination *Old males with eyeglasses and mustache and narrow eyes who are not smiling*. (a, b, c) Samples without Anti-Labeler at iterations $20k, 30k, 40k$ respectively. (d) Samples with Anti-Labeler at iteration $20k$. Comparing (a) and (d), we observe that using Anti-Labeler allows for faster convergence. Comparing (c) and (d), we observe that using Anti-Labeler provides more diverse images.

**Role of Anti-Labeler**

In this section, we show results that compare the CausalGAN behavior with and without Anti-Labeler network. In general, using Anti-Labeler allows for faster convergence. For very rare labels, the model with Anti-Labeler provides more diverse images. See Figures A.16, A.17, A.18.

(a) No AL, $t = 20k$ (b) No AL, $t = 30k$ (c) No AL, $t = 40k$ (d) With AL, $t = 20k$

Figure A.17: CausalGAN results with and without Anti-Labeler for the rare label combination *Old bald males who are not smiling but have an open mouth and narrow eyes*. (a, b, c) Samples without Anti-Labeler at iterations $20k, 30k, 40k$ respectively. (d) Samples with Anti-Labeler at iteration $20k$. Comparing (a) and (d), we observe that using Anti-Labeler allows for faster convergence.

(a) No AL, $t = 20k$ (b) No AL, $t = 30k$ (c) No AL, $t = 40k$ (d) With AL, $t = 20k$

Figure A.18: CausalGAN results with and without Anti-Labeler for the common label combination *Young smiling women with lipstick*. (a, b, c) Samples without Anti-Labeler at iterations $20k, 30k, 40k$ respectively. (d) Samples with Anti-Labeler at iteration $20k$. Comparing (a) and (d), we observe that using Anti-Labeler allows for faster convergence.

## A.2 Appendix for Chapter 3: Sample Compression, Support Vectors, and Generalization in Deep Learning

### A.2.1 Accommodation of Biases and Convolutional Layers

This section provides an interpretation of the path space and embedding map in the context of general fully-connected or convolutional Leaky-ReLU (and ReLU) networks. While there is a single canonical way to include biases, multiple methods may be possible for the incorporation of convolutional layers into the theory.

We turn to networks including biases. We now allow $w \in \mathcal{W}$ to represent the choice of biases as well as multiplicative weights, $w = ((A^{(d)}, bs^{(d+1)}), \ldots, (A^{(1)}, bs^{(2)}), (A^{(0)}, bs^{(1)}))$, where each $bs^{(l)} \in \mathbb{R}^{\Omega}$ for $l \in [d]$ (and $bs^{(d+1)} \in \mathbb{R}$) is the bias added to the result of multiplying the activation of layer $l$ by $A^{(l)}$. Our choice of indexing is so that the $bs^{(l)}$ has the same dimension as the width of layer $l$. We have

$$\mathcal{N}(x, w) \triangleq bs^{(d+1)} + A^{(d)}\rho(\ldots(bs^{(2)} + A^{(1)}\rho(bs^{(1)} + A^{(0)}x))\ldots). \tag{A.24}$$

Previously, in Section 3.3.2, it was established that $\mathcal{N}(x, w)$ could be decomposed into a sum of contributions over paths, $p = (i_d, \ldots, i_1, i_0)$. Each path is determined by the choice of a single index per layer, including a "starting" index, $i_0 \in [f]$, "connected by" a sequence of neurons, $i_l \in [\Omega]$ in layer $l = 1, 2, \ldots, d$ to the output. The contribution of this path is "seeded" with value $x_{i_0}$ and is scaled as one "moves" along the path from input to output. The scaling factor for each edge $(i_{l+1}, i_l)$ is $A^{(l)}_{(i_{l+1}, i_l)}$, which amounts to a factor of $\bar{w}_{i_d,\ldots,i_1,i_0}$. The scaling factor of the $i_l^{\text{th}}$ neuron say in layer $l$ is determined by the slope of the nonlinearity of that neuron evaluated at its incoming activation during a forward pass, $\sigma^{(l)}(x, w)_{i_l}$. We grouped these together using the notation $\bar{\sigma}(x, w)_{(i_d,\ldots,i_1)} \triangleq$

$\sigma^{(d)}(x,w)_{i_d} \cdots \sigma^{(1)}(x,w)_{i_1}$ .

With biases, the network output can still be decomposed into contributions across paths by additionally allowing paths to begin at any neuron within the network instead of only at input features:

$$\begin{aligned}\mathcal{N}(x,w) = &\sum_{p=(i_d,\ldots,i_1,i_0)} bs^{(d+1)} + A^{(d)}_{i_d}\sigma^{(d)}(x,w)_{i_d}(\ldots \\ &\sigma^{(1)}(x,w)_{i_1}(bs^{(0)}_{i_0} + A^{(0)}_{i_1,i_0}\sigma^{(0)}(x,w)_{i_0})\ldots) \\ = bs^{(d+1)} &+ \sum_{p=(i_d,\ldots,i_1,i_0)} \bar{w}_p\bar{\sigma}(x,w)_{i_d,\ldots,i_1}x_{i_0} \\ &+ \sum_{k=1}^{d}\sum_{p=(i_d,\ldots,i_k)} \bar{w}_p\bar{\sigma}(x,w)_p bs^{(k)}_{i_k}\end{aligned}$$

The final term consists of a sum over contributions of paths–each can be interpreted as "seed value" of $bs^{(k)}_{i_k}$ which is then scaled by the remaining traversed edges and neurons connecting it to the output. Note that in the above, we have augmented the definition of $\bar{w}$ and $\bar{\sigma}(x,w)$ by allowing additional coordinates corresponding to paths $p = (i_d,\ldots,i_k)$ beginning at some intermediate layer ($k = 1,\ldots,d$) in addition to those beginning at the input ($k = 0$). We have:

$$\begin{aligned}\bar{w}_{i_d,\ldots,i_k} &\triangleq A^{(d)}_{i_d}\cdots A^{(k)}_{i_{k+1},i_k} \\ \bar{\sigma}(x,w)_{i_d,\ldots,i_k} &\triangleq \sigma^{(d)}(x,w)_{i_d}\cdots\sigma^{(k)}(x,w)_{i_k}.\end{aligned}$$

with the convention that $\sigma^{(0)}(x,w)_{i_0} = x_{i_0}$. To round out the notation, if we define a "dummy bias", $bs^{(0)}$, to be a vector of all ones, $\forall i_0\ bs^{(0)}_{i_0} = 1$, then we get a clean formulation for the

network output:

$$\mathcal{N}(x,w) = bs^{(d+1)} + \sum_{\substack{p=(i_d,\ldots,i_k) \\ k=0,\ldots,d}} \bar{w}_p \bar{\sigma}(x,w)_p bs^{(k)}_{i_k}$$
$$\triangleq bs^{(d+1)} + \langle \bar{w}, \phi(x,w) \rangle$$

Turning now to convolutional layers, we seek a simple modification that will allow an analogous max-margin formulation. Consider a network consisting of several convolution layers parameterized by $w^{\text{conv}}$, followed by fully-connected layers parameterized by $w$. To generalize, we simply replace treat the convolution embedding of the inputs $\Psi^{\text{conv}}(x, w^{\text{conv}})$ as if they were the inputs themselves within the SVM formulation:

$$\mathcal{N}(x,w,w^{\text{conv}}) \triangleq bs^{(d+1)} + \langle \bar{w}, \phi(\Psi^{\text{conv}}(x,w^{\text{conv}}),w) \rangle$$

Given that we expect the initial convolutional layers to quickly arrive at certain edge-detecting low level filters that are generically useful, this treatment of the convolutional output as if it were a fixed input may be somewhat justifiable. Most importantly, this simple modification does in fact yield experimental results for convolutional networks that are similar to those we find for fully-connected.

### A.2.2 Relevance of the Max-Margin Assumption

The value of an assumption is in its implications and relevance. If theoretical work in this paper shows the former, this section is aimed at demonstrating the later. There is a bit of nuance in that "relevance" is to be distinguished from "validity". That is, Assumption 2 is not a "conjecture". It is not something that we are supposing applies exactly to unregularized deep learning models as they are. That is unclear. However, this section will show that empirically,

trained deep network models and their max-margin counterparts behave extremely similarly.

What then is the value of analyzing max-margin networks without first establishing the validity of Assumption 2 theoretically? It turns out that analyzing the consequences of Assumption 2 is easier than establishing its validity (if true). Furthermore, it is useful to know ahead of time that Assumption 2 has theoretical consequences before undertaking the task of trying to prove it. Such a study should require additional assumptions about the training data and the initialization, and it is not clear at this time what those should be.

Secondly, we should not fall into the trap of thinking of deep learning as a fixed phenomenon for observational study only. As engineers trying to build better models, we can make it as we like. If it turns out that Assumption 2 is not yet strictly speaking true but has interesting theoretical consequences, then we may modify the training process so that the trained network *is* a max-margin network. It is also not clear right now what the best way to do that is. Though as we shall see, these max-margin models would not represent a huge divergence from current deep learning models. Instead our results indicate the two are quite similar.

Consider a comparison of the two functions $\mathcal{N}_w^{sign}(\cdot, w)$ and the associated max-margin classifier on the same data with feature map $\phi(\cdot, w))$. We compare these functions by comparing the value they return on a finite set of inputs using one of two strategies. The first approach, taken in Figure A.19, is to train each on input data that is merely 2 dimensional so that the decision boundary can actually be visualized by evaluating on a grid of input points. The second approach, taken in Table A.3, is to use more realistic input data for training, such as CIFAR-10, but to compare outputs on validation data instead. Though this will not imply that the two functions are equal everywhere, if we are interested using the max-margin assumption for generalization theory, then high probability agreement on support of the data distribution is sufficient.

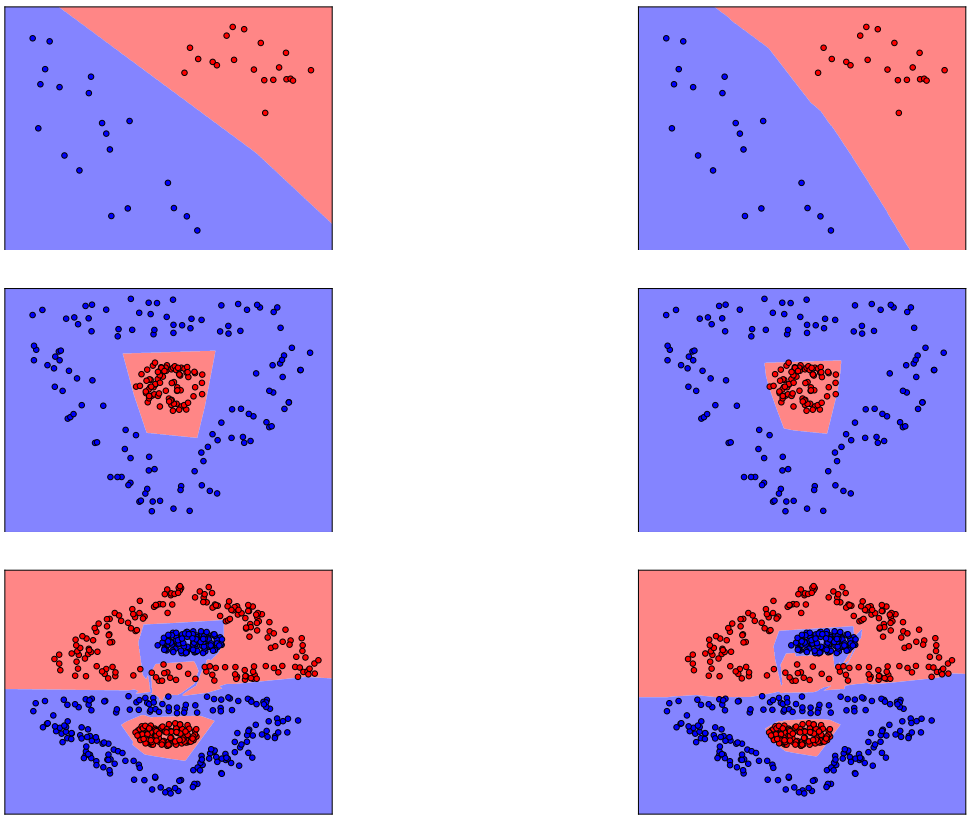
For the first approach in Figure A.19, we designed 3 toy data sets and trained a fixed 9 layer fully-connected (FC) network on each of them, obtaining weights $w$ and classifier $\mathcal{N}_w^{sign}(\cdot, w)$. Then we used the scikit-learn library to train a max-margin linear classifier on the image of the same training data under the embedding map $\phi(\cdot, w)$ for the same weights $w$. More details available in the appendix A.2.3. Optically, the decision boundaries of the DNNs the left column A.19a trained by back propagation and their max-margin counter parts in the right column A.19b are quite similar. Where the decisions of the two classifiers differ, the data samples with very low probability, suggesting that the two have very similar generalization error.

In the second approach, we classify Frogs vs Ships on a binarized CIFAR-10 dataset using a convolutional layer network. We varied the number of fully-connected(FC) layers following the initial 5 layers of alternating convolution and max pooling. The range of depths we chose was determined by technical constraints and more details are available in A.2.3. By design, the DNN had perfect training accuracy, and therefore, so did the max-margin classifier. When we compared the two classifiers across a range of depths (Table A.3), not only were the validation accuracies very similar, but also the *same* samples were misclassified by both models, whose predictions agreed more than $99\%$ of the time.

Table A.3: DNN vs Max-Margin on CIFAR Validation Data

| **(FC) Depth** | **Score Net** | **Score Max-Margin** | **Prob Agreement** |
|---|---|---|---|
| 3 | 0.968 | 0.964 | **0.990** |
| 4 | 0.976 | 0.973 | **0.992** |
| 5 | 0.972 | 0.972 | **0.993** |
| 6 | 0.974 | 0.974 | **0.995** |
| 7 | 0.971 | 0.969 | **0.994** |

We can make a final, clever attempt to empirically invalidate Assumption 2. (One can



(a) Learned DNN Classifier

(b) Max-Margin Classifier

Figure A.19: Displaced in each row is a visual comparisons between the DNN classifier(Column A.19a) and the max-margin classifier(Column A.19b). Each row corresponds to a different training dataset, and each image corresponds to a different classifier. Blue[red] circles represent training data with positive[negative] labels. Blue[red] regions in a particular image represent inputs assigned positive[negative] label by the corresponding classifier. The DNN classifiers, $\mathcal{N}_w^{sign}$ in the left column are obtained by gradient descent on the displayed training data, $\{(x^j, y^j)\}_{j=1}^m$, to learn a set of weights $w$. These weights are fixed for the entire row and define our feature embedding $\phi(\cdot, w)$. This embedding map is used to train a max-margin classifier on the training data $(\phi(x^j, w), y^j)_{j=1}^m$, which is displayed in the right column. Further experimental details can be found in Section A.2.3. Although a 2 dimensional input space is far from a general setup, at least in this setting that we are able to visualize the max-margin classifier and $\mathcal{N}_w^{sign}$ models appear visually very similar. Where there are differences in the decision boundary, those differences do not appear in the vicinity of the training data (and perhaps, the data distribution).

never empirically *validate* a hypothesis). If the max-margin assumption were actually true, what else would we expect to see? We seek to exploit the fact that for every $x$, $\bar{\sigma}(x, w)$ is coordinate-wise positive, since each entry is a product of $\beta = 0.1$ and $\gamma = 1.0$, each raised to various powers that depend on $x$ and $w$. We observe that when $y^j x^j$ is also coordinate-wise positive for each training sample, $(x^j, y^j)$, so too is each embedded, $y^j \phi(x^j, w)$. Suddenly, we have a seemingly strong conclusion: since the max-margin classifier is in the positive linear combination of the $\{y^j \phi(x^j, w)\}_{j=1}^m$, we see that Assumption 2 implies that $\bar{w}$ is coordinate-wise positive.

Yet, experimentally we can reproduce this theoretical implication. We consider training data $S^{(m)} \subset \mathbb{R}^2$ organized by label into the $1^{st}$ and $3^{rd}$ quadrants so that for each training datum $(x^j, y^j)$ is coordinate-wise positive. The weights obtained from training (without biases) with Leaky-ReLU nonlinearity on the described data are shown graphically in Figure A.20 (More details and training data can be found in appendix. The decision boundary of this network is shown in Figure A.21.

Through close inspection of Figure A.20, we see that every path from the input to output traverses an even number of negative weights, which is of course equivalent to $\bar{w}$ being coordinate-wise positive. Not only is this an immediate consequence of Assumption 2, it is not clear to us through any other theoretical lens that we are aware of. Notice also that this regularizing structure of the weights is not nearly so apparent when the weights are conceptually grouped by layer instead of across paths.

### A.2.3 Experiment Details

Concerning the experiment described in Figures A.20 and A.21, we use a truncated normal weight initialization centered around $0$ with $0.025$ standard deviation. We train with gradient descent for $15000$ iterations with a learning rate of $0.005$. Our nonlinearity, Leaky-ReLU,

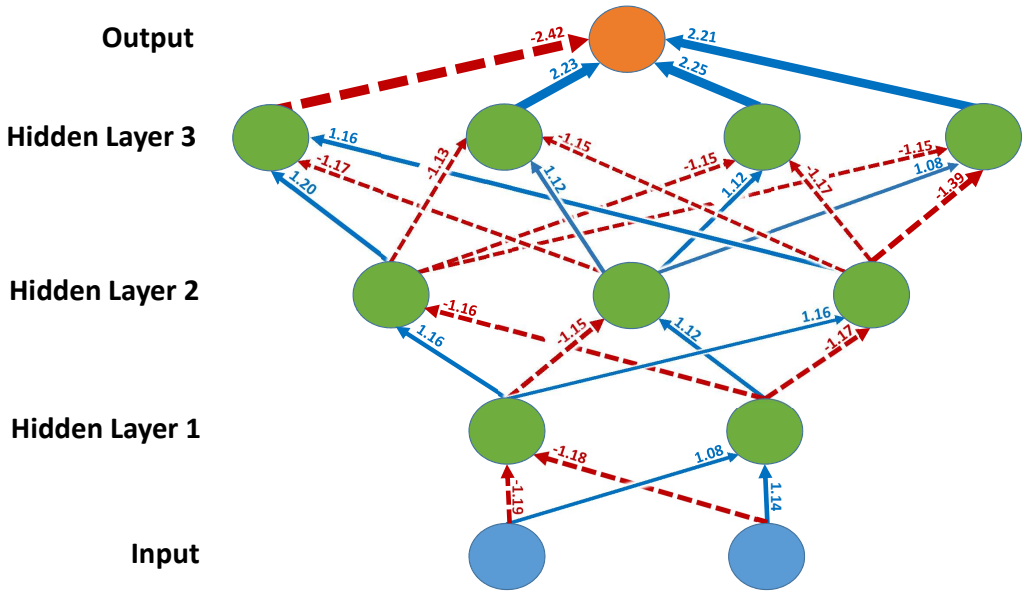



Figure A.20: Learned network weights after training on data with positive samples in the $1^{st}$ quadrant and negative samples in the $3^{rd}$ quadrant. Negative [positive] weights are represented by red dotted [blue solid] arrows [respectively]. Thicker arrows correspond to weights of larger magnitude. The finding is that each path from any input feature to the output contains an even number of red arrows (negative weights). This coordination of weight signs across layers is a striking feature of training that is implied by Assumption 2, but is not readily explained otherwise.

has slopes $\beta = 0.1$ and $\gamma = 1.0$.

The primary finding from the experiment, $\bar{w} > 0$, happens reliably as long as the weight initialization and learning rate are suitably small. Just as we are not claiming Assumption 2 always holds, we are also not claiming that $\bar{w} > 0$ always holds exactly under all related circumstances. For example, if the weight initialization is too large, it is possible to have some few very small weights with signs that do not agree with $\bar{w} > 0$, though the

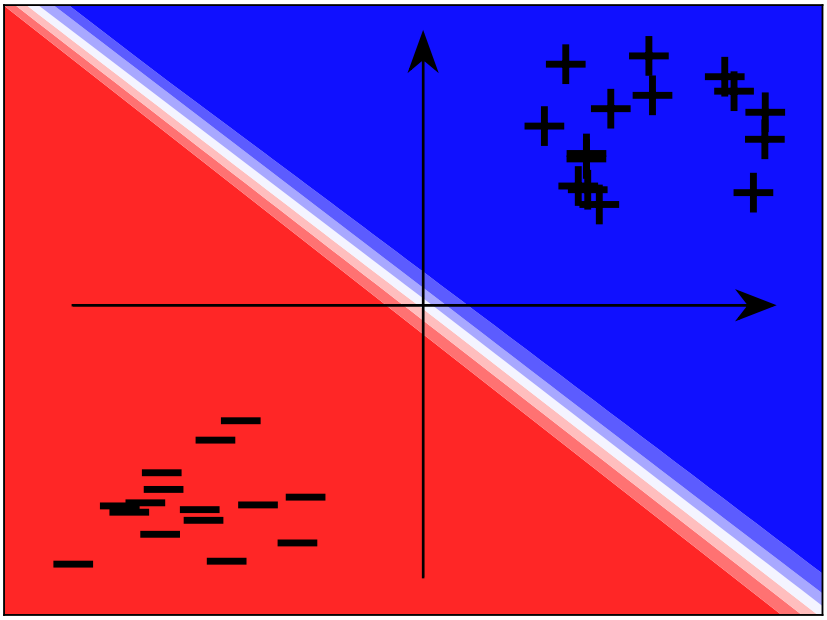


Figure A.21: Learned decision boundary and training data corresponding to the learned weights in Figure A.20. "Black plus [minus] signs correspond to locations of positively [negatively] labeled training data. Blue [red] regions correspond to positive [negative] evaluations by the network.

entries of $\bar{w}$ with largest magnitude will all have the same positive sign. Optically, it seems like the gradient can become small too quickly to overcome a large initialization of a given weight with the "wrong" sign. Though, a complete analysis of this phenomenon is not part of the scope of this work.

For Figures 3.1, 3.2, and 3.3, the setup is slightly different. We train a fully connected neural network with nonlinearity $\rho(x) = ReLU(x)$ ($ReLU(x) = \max\{0, x\}$) using SGD with momentum parameter $0.05$, learning rate $0.01$, and batch size $100$. We train on "flattened" MNIST images ($f = 28 \times 28$) with labels grouped into the binary classes

$\{0, 1, 2, 3, 4\}$ and $\{5, 6, 7, 8, 9\}$. Unless explicitly varied in the figure, we use a fixed, random subset of $m = 20000$ training samples and an architecture consisting of $d = 3$ hidden layers of uniform width, $\Omega = 16$. All experiments displayed actually achieved $0$ training error. The reason we use only $2/5^{ths}$ of the training data is because achieving *exactly* $0$ training error with every architecture considered is necessary to compare the number of support vectors and difficult to do with the entire training set.

Once we train the network to learn $w$, we experimentally determine the set of network support vectors by running a SVM classifier on the embedded data defined by the *fixed* feature map $x \mapsto \phi(x, w)$. To match the constraints.svm.SVC function in the scikit-learn library, we use hinge loss and regularization constant $C = 1e^{-5}$. We argue though that when the training error is identically $0$ and the data is linearly separable, the SVC model with hinge loss will return the maximum margin classifier independently of the value of $C$. This is because for any $C$, the weights are eventually near the optimum where none of the constraints are active. This agrees with what we see experimentally when we varied $C$ (not shown).

The data points in Figures 3.2 and 3.3 representing the number of support vectors vs width and depth are all averages of 3 trials. One tricky experimental detail is that neural network models have to be trained for a very long time, sometimes upwards of $100$ epochs, in order to get *exactly* $0$ training error needed to guarantee linear separability. This is especially true for the larger width and larger depth runs.

When we randomize the labels, as in Figure A.22, we are determining *every* sample label by a fair coin flip once before training starts, then fixing that label during training.

For the comparison of max-margin classifier in Figure A.19, three different synthetic datasets were generated by random sampling. The idea in the choice of distributions was to give a variety of both "easy" and "difficulty" 2-dimensional classification tasks. The network

used was a $9$ hidden layer fully connected network of widths $4, 6, 8, 10, 12, 14, 16, 20, 30$. The training parameters used to obtain weights $w$ were identical to above.

To produce the max-margin classifiers, we used $\phi(\cdot, w)$ as a fixed embedding (corresponding to the learned parameters $w$, and trained a max-margin classifier using the sci-kit learn library. Specifically, we first calculated the kernel matrix for all training samples. Then we trained a linear classifier using $C = 1e - 5$ and tolerance $1e - 5$ without the shrinking heuristic available to the SVC solver.

For the convolutional experiments in Figure 3.4 and Table A.3, we used a convolutional network. The first $5$ layers consisted of $3$ convolutional layers of $64$ filters each, interleaved by $2$ max pooling layers. The convolutional layers used $3x3$ kernels with a stride of $1$, and the max pooling layers took the maximum over $2x2$ regions. This convolutional embedding was flattened. Experimentally, we varied "FC depth", or the number of subsequent fully-connected $64$ neuron layers between this flattened output and the network output.

The dataset, CIFAR-10, was chosen based on suggestion by a reviewer. Because we only study binary classifiers, we restricted ourselves to discriminating "Frogs" from "Ships". This was also simply the first binarization that we tried. A foreseen benefit was also that there would be only $10k$ training samples had either of these labels, which makes running in-memory SVM classification easier.

There were upper and lower constraints on the ranges of the FC depth explored. On the higher end, we found that it was impossible to use a *fixed* learning rate of $0.005$ across a huge range of depths. For large FC depth, the training would become unstable in a way that could be mitigated by decreasing the learning rate.

The lower constraint limiting FC depth $\geq 3$ is slightly curious. Though the network would still have perfect training accuracy, the SVM solver would struggle to find any linear separator. We know theoretically that one must exist (since $\bar{w}$ is one), but it seems in practice

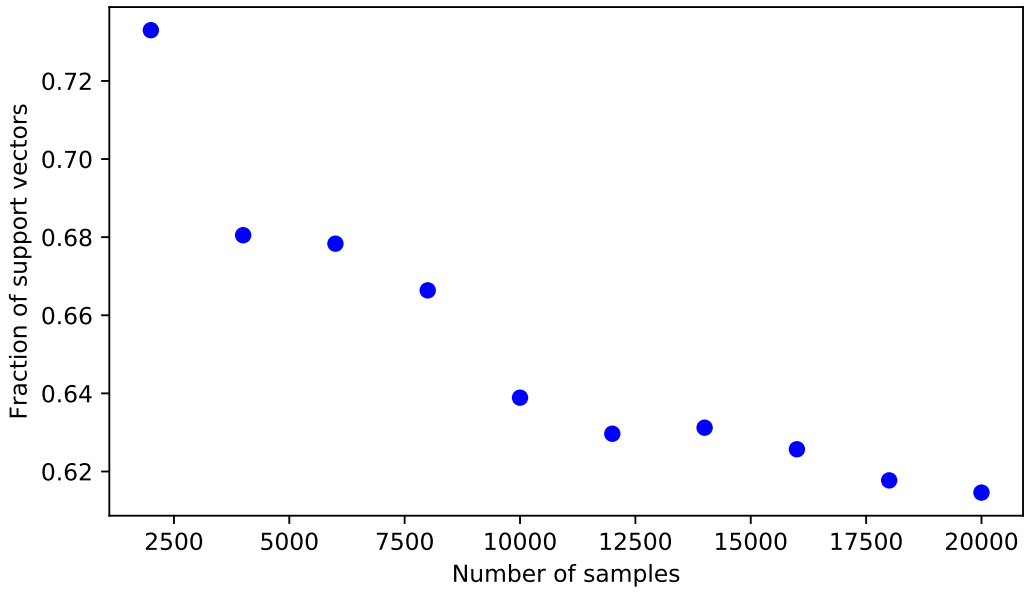



Figure A.22: Fraction Network Support Vectors $(s/m)$ vs $m$ under Randomized Labels: Once before training, the label of each training datum is replaced by a sample drawn uniformly from $\mathcal{Y}$. Compared to the setting with true labels (Figure 3.1), the data appear shifted up vertically by $0.5$.

that the SVM solver has trouble finding it for shallow networks.

### A.2.4 Theorem 3: The Skeleton and NN Recovery

**Theorem 3.** *For $P \subset \mathcal{F}$, define $\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(P)) \triangleq \{\mathcal{N}^{sign}(\cdot, w) : w \in \mathcal{W}, \Lambda(w) \in P\}$. For $\bar{w} \in \mathcal{F}$, define $\mathbb{R}^+\bar{w} \triangleq \{\alpha\bar{w} : \alpha > 0\}$.*

*Then*

$$|\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(\mathbb{R}^+\bar{w}))| \leq 2^n \tag{3.5}$$

*where $n = d\Omega$ is the number of neurons in the Leaky-ReLU network.*

*Proof.* Suppose we are given a positive multiple of $\bar{w}$. We may assume without loss of generality that every neuron $\eta$ belongs to at least some path, $p(\eta)$, with $\bar{w}_{p(\eta)} \neq 0$. If some

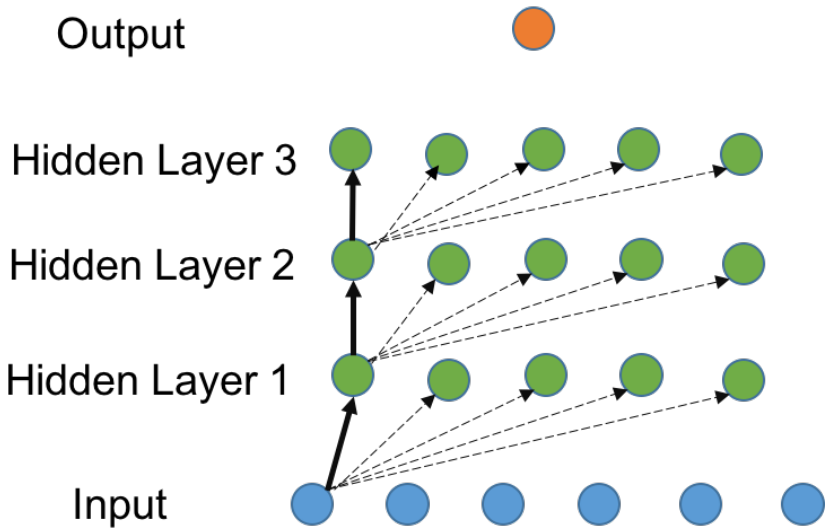



Figure A.23: An illustration of one possible collection of edges corresponding to a skeleton (the key ingredient in the proof of Theorem 3). A "skeleton" is a collection of $n$ edges, $Skel$, with corresponding network weights, $SkelW$, containing for each neuron one path from some input feature to that neuron. For each $\bar{w}$, $SkelW$ is in bijection with $\Lambda^{-1}(\bar{w})$. We may imagine the solid black lines to be the "spine" and the dotted lines to be the "ribs", though there are valid configurations that are less anatomic.

neuron is not a member of any such path, then it makes no contribution to the function $x \mapsto \mathcal{N}(x, w) = \sum_{p \in Paths} \bar{w}_p \phi(x, w)_p$. Thus we may drop all such neurons without affecting which functions $\mathcal{N}_w^{sign}$ are feasible given $\bar{w}/\|\bar{w}\|$. If by removing neurons in this manner we run out of neurons in a single hidden layer, then the bound is trivially true since $\mathcal{N}_w$ must be the zero function.

Thus, for all $1 \leq l \leq d$, every neuron $i_1$ in layer 1 has at least some corresponding index $s_0(i_1)$ in layer 0 such that $A^{(0)}_{i_1, s_0(i_1)} \neq 0$. Because Leaky-ReLU commutes with positive diagonal matrices, we can rescale column $i_1$ of $A^{(1)}$ by $|A^{(0)}_{i_1, s_0(i_1)}|$ and row $s_0(i_1)$ of $A^{(0)}$ by $|A^{(0)}_{i_1, s_0(i_1)}|^{-1}$.

We continue renormalizing this way until every row of each of the weight matrices $A^{(0)}$ through $A^{(d-1)}$ has at least one weight in $\{-1, 1\}$. For each neuron not corre-

sponding to the input or output, fix a particular choice of indices in the previous layer, $Skel \triangleq \{(l, i_l, s_{l-1})\}_{l,i_l}$, so that the corresponding weights $SkelW \triangleq \{A^{(l-1)}_{i_l,s_{l-1}}\}_{l,i_l}$ are all in $\{-1, 1\}$. Call these indices, $Skel$, a "skeleton" of the network, and the corresponding weights $SkelW \subset \{-1, 1\}^n$ "skeleton weights". A path $p = (i_d, \ldots, i_0)$ will be said to be "in the skeleton" if $\forall l$ $A^{(l)}_{i_{l+1},i_l}$ is a skeleton weight. We have shown that as long as every neuron is along some path $p$ with $\bar{w}_p \neq 0$, then the network has a skeleton. (Figure A.23 illustrates one such configuration of weights, but is not explicitly used in this proof).

We will show that given $\alpha\bar{w}$ and a skeleton $Skel$, every choice of skeleton weights determines a different set of weights $w = (\alpha A^{(d)}, A^{(d-1)}, \ldots, A^{(0)})$ with $\Lambda(w) = \alpha\bar{w}$. Thus we will show that the set of weights compatible with $\bar{w}$ are in bijection with the set of $2^n$ possible skeleton weights, up to rescaling of $A^{(d)}$. Since scaling $A^{(d)}$ by $\alpha > 0$ doesn't change the sign of the network output, we will have at most $2^n$ distinct possible classification functions compatible with some scaling of $\bar{w}$.

Fix a choice of skeleton $Skel$ and skeleton weights $SkelW$. Then for every layer $l$, for every neuron $i_l$ in layer $l$, there is a path $(j_{l-1}, j_{l-2}, \ldots, j_0)$ from the input to that neuron which stays in the skeleton and for which the product of weights is $\pm 1 \neq 0$. We introduce the notation $\bar{b}$ by using $\bar{b}_{j_{l-1},j_{l-2},\ldots,j_0}$ to mean "the product of weights along a particular path within $Skel$ from the input to a particular neuron":

$$\bar{b}_{i_l,j_{l-1},j_{l-2},\ldots,j_0} \triangleq A^{(l-1)}_{i_l,j_{l-1}} A^{(l-2)}_{j_{l-1},j_{l-2}} \cdots A^{(0)}_{j_1,j_0}. \tag{A.25}$$

First we show that all of the projection weights, $A^{(d)}$, are determined up to scale by $\alpha$. For neuron $i_d$ in layer $d$, get a path $j_{d-1}, j_{d-2}, \ldots, j_0$ from the input to neuron $i_d$ within

the skeleton so that $\bar{b}_{j_{d-1},j_{d-2},\ldots,j_0} \neq 0$. Then simply solve

$$\frac{\alpha \bar{w}_{i_d,j_{d-1},j_{d-2},\ldots,j_0}}{\bar{b}_{i_d,j_{d-1},j_{d-2},\ldots,j_0}} = \frac{\alpha A^{(d)}_{i_d} A^{(d-1)}_{i_d,j_{d-1}} \cdots A^{(0)}_{j_1,j_0}}{A^{(d-1)}_{i_d,j_{d-1}} \cdots A^{(0)}_{j_1,j_0}} = \alpha A^{(d)}_{i_d}. \tag{A.26}$$

We next show how to find any weight. let $l < d$,$i_{l+1}, i_l$ be arbitrary. From neuron $i_{l+1}$ in layer $l+1$ and neuron $i_l$ in layer $l$ get paths $(i_{l+1}, j_l, j_{l-1}, \ldots, j_0)$ and $(i_l, k_{l-1}, \ldots, k_0)$ within $Skel$ with $\bar{b}_{i_{l+1},j_l,j_{l-1},\ldots,j_0}$ and $\bar{b}_{i_l,k_{l-1},\ldots,k_0}$ nonzero. Furthermore, we are guaranteed some indices $e_d, e_{d-1}, \ldots, e_{l+2}$ such that $\bar{w}_{e_d,e_{d-1},\ldots,e_{l+2},i_{l+1},j_l,j_{l-1},\ldots,j_0} \neq 0$ since every neuron is connected to the output through at least some path with nonzero weights. Then we simply solve

$$\frac{\alpha \bar{w}_{e_d,e_{d-1},\ldots,i_{l+1},i_l,k_{l-1},\ldots,k_0}}{\bar{b}_{i_l,k_{l-1},\ldots,k_0}} \frac{\bar{b}_{i_{l+1},j_l,j_{l-1},\ldots,j_0}}{\alpha \bar{w}_{e_d,e_{d-1},\ldots,i_{l+1},j_l,j_{l-1},\ldots,j_0}} \tag{A.27}$$

$$= \frac{\alpha A^{(d)}_{e_d} A^{(d-1)}_{e_d,e_{d-1}} \cdots A^{(l+1)}_{e_{l+2},i_{l+1}} A^{(l)}_{i_{l+1},i_l}}{\alpha A^{(d)}_{e_d} A^{(d-1)}_{e_d,e_{d-1}} \cdots A^{(l+1)}_{e_{l+2},i_{l+1}}} \tag{A.28}$$

$$= A^{(l)}_{i_{l+1},i_l}. \tag{A.29}$$

□

### A.2.5 Broader Significance Theorem 3 Discussion

While, numerically, $n$ is equal to the number of neurons, in this setting one should think of it as the smallest number of weighted edges in the network graph needed to connect every neuron to the output. Such a subset, we called a skeleton. Similarly, $2^n$ refers not to the number of neuron state configurations, but to the number of sign configurations of the n weights whose edges are in the skeleton.

One nice perspective afforded by the exposition in our paper is that while a Support

Vector Machine(SVM) learns a classifier in a feature space, a Neural Network(NN) learns both a classifier, $\Lambda(w)$, and an embedding map, $\phi(x, w)$. The flexibility to learn the embedding can then be seen as an additional mode of expressivity, loosely speaking of course, that is available to NNs but not to SVMs. However, the extent of this flexibility is unclear because both the classifier, $\Lambda(w)$, and the embedding map, $\phi(x, w)$, depend on the weights, i.e., they are entangled. What is the nature of this dependence?

Theorem 3 answers that question completely. Fix any skeleton subgraph. It says each classifier, $\Lambda(w)$, corresponds to only finitely many embedding maps, with one map corresponding to each one of the $2^n$ configurations of weight-signs in that skeleton.

### A.2.6 PAC-Bayes Background

In this section, we review the sample compression version of PAC-Bayes bounds, which we will invoke to prove Theorem 3.3.5. We are largely following Laviolette and Marchand [2007].

In the PAC-Bayes framework (without sample compression), one typically works with a distribution over classifiers that is updated after seeing the training set $S^{(m)}$. A prior $P$ over $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ is declared before training, without reference to the specific samples in $S^{(m)}$. Then consider another distribution over classifiers, $Q$, called a "posterior" to reflect that it is allowed to depend on $S^{(m)}$. Each posterior $Q$ defines a Gibbs classifier $G_Q$ that makes predictions stochastically by sampling classifiers according to $Q$. Similarly, we define the true risk $R(G_Q)$ and empirical risk $R_S(G_Q)$ of the Gibbs classifier $G_Q$ as

$$R_{\mathcal{D}}(G_Q) = \mathop{\mathbb{E}}_{h\sim Q}[R_{\mathcal{D}}(h)] \qquad\qquad R_{S^{(m)}}(G_Q) = \mathop{\mathbb{E}}_{h\sim Q}[R_{S^{(m)}}(h)]$$

PAC-Bayes gives a very elegant characterization of the relationship between the true

and empirical risks of Gibbs classifiers. Let $KL(Q||P)$ be the Kullback-Leibler divergence between distributions $Q$ and $P$. For scalars $q, p$, define $kl(q||p)$ to be the Kullback-Leibler divergence between Bernoulli $q$ and $p$ distributions. We have the following classical uniform bound over posteriors $Q$ $\forall \delta \in (0, 1]$ (Theorem 1 in Laviolette and Marchand [2007])

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall Q \quad \Phi(Q, P, m, \delta)] \geq 1 - \delta$$

where $\Phi(Q, P, m, \delta)$ is the event

$$kl(R_{S^{(m)}}(G_Q)||R_{\mathcal{D}}(G_Q)) \leq \frac{KL(Q||P) + \ln \frac{m+1}{\delta}}{m} \tag{A.30}$$

To relate this to classical bounds for finite hypothesis sets, notice that if $P = Unif(\mathcal{H})$ and $Q$ is a delta distribution on $\xi_0 \in \mathcal{H}$, then the PAC-Bayes bound is governed by the ratio $KL(Q||P) = \ln |\mathcal{H}|$ to $m$. When $\mathcal{H}$ is not finite, one can still get bounds for stochastic neural network classifiers as in Dziugaite and Roy [2017], or one can convert these bounds into bounds for deterministic classifiers by considering the risk of the classifier which outputs the majority vote over $Q$ (see Laviolette and Marchand [2007] section 3 for example). We take neither of these approaches, but just mention them for the reader's interest.

Thus far we have discussed "data-independent" priors. We now turn to Laviolette and Marchand [2007] to discuss priors $P_{S^{(m)}}$ over hypotheses that depend on the training set through a "reconstruction function", $\mathcal{R}$, mapping subsets of the training data and some "side-information" to a hypothesis.

The idea is to describe classifiers in terms of a subset of training samples, called a "compression sequence", and an element from some auxiliary set, called a "message". For

the moment, consider any arbitrary sequence[2] , $T \subset (\mathcal{X} \times \mathcal{Y})^m$. Given $T$, define a set of "allowable messages" $\mathcal{M}(T)$ so that we have a "reconstruction function", $R : T \times \mathcal{M}(T) \mapsto \mathcal{Y}^{\mathcal{X}}$, which sends arbitrary sets of samples $T$, and optionally some side information in $\mathcal{M}(T)$, to a classifier mapping $\mathcal{X}$ to $\mathcal{Y}$.

Let $I$ be the set of subsets of $[m]$. Considering now our training set $S^{(m)}$, for $\mathbf{i} \in I$ define $S^{(m)}_{\mathbf{i}}$ to be the subset of training points at indices $\mathbf{i}$. We now introduce a single (data-dependent) set for the support of our prior and posterior. Define

$$\mathcal{M}_{S^{(m)}} \triangleq \bigcup_{\mathbf{i} \in I} \mathcal{M}(S^{(m)}_{\mathbf{i}}). \tag{A.31}$$

In the sample compression setting, we sample hypotheses in $\mathcal{Y}^{\mathcal{X}}$ by sampling $(\mathbf{i}, z)$ from $I \times \mathcal{M}_{S^{(m)}}$ according to either our ($S^{(m)}$-dependent) prior $P_{S^{(m)}}(\mathbf{i}, z)$ or our posterior $Q(\mathbf{i}, z)$ and passing $(\mathbf{i}, z)$ to our reconstruction function $R$ to obtain the hypothesis $R(\mathbf{i}, z) : \mathcal{X} \mapsto \mathcal{Y}$.

For the results to follow, we require our prior and posterior to factorize accordingly:

$$P_{S^{(m)}}(\mathbf{i}, z) = P_I(\mathbf{i}) P_{\mathcal{M}(S^{(m)}_{\mathbf{i}})}(z)$$
$$Q(\mathbf{i}, z) = Q_I(\mathbf{i}) Q_{\mathcal{M}(S^{(m)}_{\mathbf{i}})}(z)$$

That is, though the prior does depend on the training set, the marginal prior $P_I$ over subsets $\mathbf{i} \in I$ does *not*. Also, conditioned on $\mathbf{i} \in I$, the prior on messages $z \in \mathcal{M}(S^{(m)}_{idx})$ only depends on those training samples, $S^{(m)}_{\mathbf{i}} \subset S^{(m)}$, indexed by $\mathbf{i}$ and *not* the whole training set. The same factorization is likewise required of $Q$. In fact, we will assume throughout

[2]The terminology "sequence" is used here to highlight situations which apply to *any* set of inputs, not just probable ones.

that any distribution on $I \times \mathcal{M}_{S^{(m)}}$ has this factorization.

Given training set $S^{(m)}$ and posterior $Q = Q_I Q_{\mathcal{M}(S^{(m)})}$ (possibly depending on $S^{(m)}$) the Gibbs classifier $G_Q$ classifies new $x$ by sampling $\mathbf{i} \sim Q_I(\mathbf{i})$, $z \sim Q_{\mathcal{M}(S^{(m)}_{\mathbf{i}})}(z)$, setting $\xi = R(\mathbf{i}, z)$, and returning the label $\xi(x)$.

In analogy with the data independent setting, the goal is again to claim that the empirical Gibbs risk $R_{S^{(m)}}(G_Q)$ is close to the true Gibbs risk $R_{\mathcal{D}}(G_Q)$ when $KL(Q||P)$ is small compared to the number of samples $m$. This is the content of Theorem 3 in Laviolette and Marchand [2007], which, though more general than we require, we cite verbatim for reference. For example, the theorem uses notation $\bar{Q}$ and $d_{\bar{Q}}$, which will simplify to $\bar{Q} = Q$ and $d_{\bar{Q}} = s$ in our more specialized setting where $P, Q$ have nonzero weight only for $|\mathbf{i}| = s$. A specialized version to follow:

**Theorem 9.** *(Theorem 3 in Laviolette and Marchand [2007])*
*For any $\delta \in (0, 1]$, for any reconstruction function mapping compression sequences and messages to classifiers, for any $S^{(m)} \in (\mathcal{X} \times \mathcal{Y})^m$ and for any prior $P_{S^{(m)}}$ on $I \times \mathcal{M}_{S^{(m)}}$, we have*

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall Q \quad \Phi(Q, P, m, \delta)] \geq 1 - \delta$$

*where $\Phi(Q, P, m, \delta)$ is the event*

$$kl(R_{S^{(m)}}(G_Q)||R_{\mathcal{D}}(G_Q)) \leq \frac{KL(\bar{Q}||P) + \ln \frac{m+1}{\delta}}{m - d_{\bar{Q}}}$$

In the special case we consider where $P, Q$ have nonzero weight only for $|\mathbf{i}| = s$, we have the reduction $\bar{Q} = Q$ and $d_{\bar{Q}} = s$. More specialized still, we consider Laviolette and Marchand [2007] Theorem 9, which specializes to the case where $G_Q$ achieves zero

training error. It is *slightly* tighter than simply plugging in 0 for $R_{S^{(m)}}(G_Q)$ by an additive factor of $\ln(m+1)/m$. Notice in the following that the form of the bound arises because $kl(0||R) = -\ln(1-R)$:

**Theorem 10.** *(Special case of Theorem 9 in Laviolette and Marchand [2007])*
*Fix $s \leq m$. Let $I_s \subset I$ be the set of $s$-sized subsets of indices $[m]$. For any $\delta \in (0,1]$, for any reconstruction function mapping compression sequences and messages to classifiers, for any fixed prior $P_T$ that defines for every arbitrary sequence $T \in \mathcal{X} \times \mathcal{Y}^m$ a distribution on $I_s \times \mathcal{M}_{S^{(m)}}$ , we have*

$$\begin{aligned} &\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall \{Q : R_{S^{(m)}}(G_Q) = 0\} \quad \Phi(Q, P, m, \delta, s)] \\ &\geq 1 - \delta \end{aligned}$$

*where $\Phi(Q, P, m, \delta, s)$ is the event*

$$R_{\mathcal{D}}(G_Q) \leq 1 - exp\left[ -\frac{KL(Q||P_{S^{(m)}}) + \ln\left(\frac{1}{\delta}\right)}{m-s} \right] \tag{A.32}$$

Notice though that

$$0 \leq -\ln(1 - R_{\mathcal{D}}(\mathcal{N}_w^{sign})) - R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq \epsilon(R_{\mathcal{D}}(\mathcal{N}_w^{sign})) \tag{A.33}$$

, where $\epsilon(R_{\mathcal{D}}(\mathcal{N}_w^{sign})) \leq 0.03$ for the reasonable operating range, $R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq 0.2$. Therefore, as a matter of taste, in place of Equation A.32 in the above theorem we can claim the (very slightly) weaker but notationally more compact bound:

$$R_{\mathcal{D}}(G_Q) \leq \frac{KL(Q||P_{S^{(m)}}) + \ln\left(\frac{1}{\delta}\right)}{m-s} \tag{A.34}$$

In fact as long as for in the range of $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$, we would Now we are ready to prove our main theorem.

### A.2.7 Theorem 3.3.5:A Neural Network Sample Compression Bound

We restate and prove Theorem 3.3.5 from Section 3.4.

**Theorem 2.** *Let $\mathcal{N}$ refer to a Leaky-ReLU neural network with $d$ hidden layers each consisting of width $\Omega$ neurons so that we have $n = d\Omega$ neurons total. Let the weights $w$ be deterministic functions of $S^{(m)}$, which is a set of $m$ i.i.d. data samples from $\mathcal{D}$. Let $s < m$ be a fixed integer which does not depend on $S^{(m)}$. Supposing that:*

1. *Assumption 1 (Zero training error): $\mathcal{N}_w^{sign}(x) = y \; \forall (x, y) \in S^{(m)}$,*
2. *Assumption 2 (Max-margin): $\Lambda(w)$ is some positively scaled version of the max-margin classifier for $\{(\phi(x, w), y) : (x, y) \in S^{(m)}\}$, and*
3. *(At most $s$ support vectors): $\Lambda(w) = \sum_{k=1}^{m} \alpha_k y^k \phi(x^k, w)$ for some set of coefficients $\alpha_k$, at most $s$ of which are nonzero.*

*then we have, $\forall \delta \in (0, 1]$*

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} \left[ R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq \mathcal{F}(m, n, s, \delta) \right] \geq 1 - \delta$$

*where*

$$\begin{aligned} \mathcal{F}(m, n, s, \delta) &= \frac{n + ns + s + s \ln\left(\frac{m}{s}\right) + \ln\left(\frac{1}{\delta}\right)}{m - s} \\ &\approx \frac{ns + \ln\left(\frac{1}{\delta}\right)}{m} \end{aligned} \tag{3.4}$$

*Proof.* We start by defining without reference to a training set: our reconstruction function, our base space, and a fixed prior $P_T$ for every possible sequence $T \subset (\mathcal{X} \times \mathcal{Y})^m$.

Let $T \in (\mathcal{X} \times \mathcal{Y})^m$ be arbitrary. Let $I^{(s)}$ be the set of subsets of $s$ elements from $[m]$. Let $\mathcal{M}^\sigma(T)$ be the set of tuples of neuron states for inputs $T$ that are achievable with at least some network weights: $\mathcal{M}^\sigma(T) \triangleq \{(\bar{\sigma}(x, v))_{(x,y)\in T} : v \in \mathcal{W}\}$.

For future convenience, define a "max-margin conditional", $C_{MM}(T_\mathbf{i}, \Sigma)$, to be "True" iff there exist a nonempty set of weights $\mathcal{W}(T_\mathbf{i}, \Sigma) \subset \mathcal{W}$ such that $\forall v \in \mathcal{W}(T_\mathbf{i}, \Sigma)$: (1) $\Sigma = (\bar{\sigma}(x, v))_{(x,y)\in T}$ and (2) $\Lambda(v)$ is the max-margin classifier for $\{(\phi(x, v), y)\}_{(x,y)\in T}$. Put $\kappa(T_\mathbf{i}, \Sigma) = |\{\mathcal{N}_v^{sign} : v \in \mathcal{W}(T_\mathbf{i}, \Sigma)\}|$ to be the number of neural network classifiers obtained from some model parameter in $\mathcal{W}(T_\mathbf{i}, \Sigma)$. Note that $\kappa(T_\mathbf{i}, \Sigma) \leq 2^n$ by Theorem 3.

For $\Sigma \in \mathcal{M}^\sigma(T)$, if $C_{MM}(T_\mathbf{i}, \Sigma)$ is True, put $\mathcal{M}^\pi(T, \Sigma) = [\kappa(T_\mathbf{i}, \Sigma)]$ and put $\mathcal{M}^\pi(T, \Sigma) = [1]$ otherwise.

Let our prior $P_T(\mathbf{i}, \Sigma, j)$ have support on $I^{(s)} \times \mathcal{M}_T^\sigma \times \mathcal{M}_T^\pi$, where the component spaces are defined as

$$\mathcal{M}_T^\sigma \triangleq \bigcup_{\mathbf{i}\in I^{(s)}} \mathcal{M}^\sigma(T_\mathbf{i}).$$
$$\mathcal{M}_T^\pi \triangleq \bigcup_{\mathbf{i}\in I^{(s)} \Sigma\in\mathcal{M}^\sigma(T_\mathbf{i})} \mathcal{M}^\pi(T_\mathbf{i}, \Sigma)$$

where the prior $P_T$ has the factorization $P_T(\mathbf{i}, \Sigma, j) = P^I(\mathbf{i})P^\sigma_{T_\mathbf{i}}(\Sigma)P^\pi_{(T_\mathbf{i},\Sigma)}(j)$, where $P^I$ is not allowed to depend on the sequence $T$, and each factor distribution is uniform on the corresponding set of allowable messages:

$$P^I = Uniform(I^{(s)})$$
$$P^\sigma = Uniform(\mathcal{M}^\sigma(T_\mathbf{i}))$$
$$P^\pi = Uniform(\mathcal{M}^\pi(T_\mathbf{i}, \Sigma)) \tag{A.35}$$

Our reconstruction function maps each $(\mathbf{i}, \Sigma, j) \in I^{(s)} \times \mathcal{M}_T^\sigma \times \mathcal{M}_T^\pi$ to a classifier as follows: if $C_{MM}(T_\mathbf{i}, \Sigma)$ is True, $\mathcal{R}(T_\mathbf{i}, \Sigma, j)$ returns the $j^{\text{th}}$ classifier, in $\{\mathcal{N}_w^{sign} : w \in \mathcal{W}(T_\mathbf{i}, \Sigma)\}$ (any total ordering on network classifiers $\{\mathcal{N}_v^{sign} : v \in \mathcal{W}\}$, can be used to clarify the meaning of $j^{\text{th}}$). Else if $C_{MM}(T_\mathbf{i}, \Sigma)$ is False, $\mathcal{R}(T_\mathbf{i}, \Sigma, j)$ returns a "dummy" classifier. To make a concrete choice, return the constant classifying function: $\mathcal{R}(T_\mathbf{i}, \Sigma, j) = (x \mapsto +1)$ if $C_{MM}(T_\mathbf{i}, \Sigma)$ False.

Only now, let $S^{(m)}$ be a training set sampled from $\mathcal{D}^m$. Consider now only the "posterior" distributions $Q$ on $I^{(s)} \times \mathcal{M}_{S^{(m)}}^\sigma \times \mathcal{M}_{S^{(m)}}^\pi$ that satisfy $\operatorname{supp} Q \subset \operatorname{supp} P_{S^{(m)}}$ and factorize according to $Q(\mathbf{i}, \Sigma, j) = Q^I(\mathbf{i}) Q^\sigma_{S^{(m)}_\mathbf{i}}(\Sigma) Q^\pi_{(S^{(m)}_\mathbf{i}, \Sigma)}(j)$. Note, in contrast with the prior, here each of $Q^I, Q^\sigma, Q^\pi$ are allowed to depend on the samples $S^{(m)}$. Let $G_Q$ be the Gibbs classifier which classifies $x$ stochastically by sampling $(\mathbf{i}, \Sigma, j) \sim Q(\mathbf{i}, \Sigma, j)$ and returning $\mathcal{R}(\mathbf{i}, \Sigma, j)(x)$.

Then, from Theorem 10 and Equation A.34, we know that $\forall \delta \in (0, 1]$,

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall \{Q : R_{S^{(m)}}(G_Q) = 0\} \Phi(Q, P, m, \delta, s)] \geq 1 - \delta$$

where $\Phi(Q, P, m, \delta, s)$ is the event

$$R_\mathcal{D}(G_Q) \leq \frac{KL(Q||P_{S^{(m)}}) + \ln\left(\frac{1}{\delta}\right)}{m - s} \tag{A.36}$$

Consider the weights $w$ classifier $\mathcal{N}_w^{sign}$ we obtain from training the neural network $\mathcal{N}$ on $S^{(m)}$. Since the above bound is uniform over all $Q$, if we can find a posterior $Q_\mathcal{N}$ such that $G_{Q_\mathcal{N}} = \mathcal{N}_w^{sign}$, then we can use Equation A.36 to bound the true risk $R_\mathcal{D}(\mathcal{N}_w^{sign})$ of our neural network. Else if we cannot, then Equation A.36 does not comment on the risk $R_\mathcal{D}(\mathcal{N}_w^{sign})$. However, we will show that whenever the three assumptions of the theorem hold, we can find such a posterior, and the bound will hold.

Well, by Assumption 2, we know that $\Lambda(w)$ is the unique max-margin classifier for $(\phi(x^j, w), y^j)_{j=1}^m$. But, since we have also assumed at most $s$ network support vectors, we know that $\Lambda(w)$ is *also* the unique max-margin classifier for some subset of support vectors $S^{(s)} \subset S^{(m)}$. Since $|S^{(s)}| \leq s$, we can get $\mathbf{i}_\mathcal{N} \in I^{(s)}$ such that $S^{(s)} \subset S^{(m)}_{\mathbf{i}_\mathcal{N}}$. Furthermore, there is at least one value $\Sigma_\mathcal{N} \in \mathcal{M}^\sigma_{S^{(m)}}$, namely $\Sigma_\mathcal{N} \triangleq (\bar{\sigma}(x, w))_{(x,y)\in S^{(m)}_{\mathbf{i}_\mathcal{N}}}$, for which $C_{MM}(S^{(m)}_{\mathbf{i}_\mathcal{N}}, \Sigma_\mathcal{N})$ is True and $\mathcal{W}(S^{(m)}_{\mathbf{i}_\mathcal{N}}, \Sigma_\mathcal{N}) \ni w$ is nonempty. Hence, for some $j_\mathcal{N} \in \mathcal{M}^\pi_{S^{(m)}}$, $\mathcal{R}(\mathbf{i}_\mathcal{N}, \Sigma_\mathcal{N}, j_\mathcal{N}) = \mathcal{N}^{sign}_w$ as desired.

Let $Q^I$ be a distribution on $I^{(s)}$ which samples the index set $\mathbf{i}_\mathcal{N}$ with probability 1. Let $Q^\sigma$ be a distribution on $\mathcal{M}^\sigma_{S^{(m)}}$ which is uniform over the set of activations consistent with $\mathcal{N}^{sign}_w$:

$$Sym(w, S^{(m)}_{\mathbf{i}_\mathcal{N}}) \triangleq \{\Sigma : \exists v \in \mathcal{W}(S^{(m)}_{\mathbf{i}_\mathcal{N}}, \Sigma) \text{ with } \mathcal{N}^{sign}_v = \mathcal{N}^{sign}_w\} \tag{A.37}$$

$$Q^\sigma = Uniform(Sym(w, S^{(m)}_{\mathbf{i}_\mathcal{N}})). \tag{A.38}$$

For example, within-layer neuron permutations yield different $\Sigma$ but the same classifier. At last, for each $\Sigma \sim Q^\sigma$, let $Q^\pi_{(S^{(m)}_\mathbf{i}, \Sigma)}$ be a distribution on $\mathcal{M}^\pi_{S^{(m)}}$ placing all of its mass on the (unique) index $j_\Sigma$ such that $\mathcal{R}(\mathbf{i}_\mathcal{N}, \Sigma, j_\Sigma) = \mathcal{N}^{sign}_w$ as functions. Therefore, $Q_\mathcal{N} \triangleq Q^I Q^\sigma Q^\pi$ is a posterior distribution returning $\mathcal{N}^{sign}_w$ with probability one. Thus the Gibbs classifier $G_{Q_\mathcal{N}}$ is a deterministic classifier and is equal to $\mathcal{N}^{sign}_w$.

There, we may claim Equation A.36 holds for posterior $Q_\mathcal{N}$ with probability at least $1 - \delta$.

To conclude our theorem, we simply expand and upper bound $KL(Q_{\mathcal{N}}, P_{S^{(m)}})$:

$$
\begin{aligned}
&KL(Q_{\mathcal{N}}||P_{S^{(m)}}) = \\
&= \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \mathop{\mathbb{E}}_{j\sim Q^\pi} \ln\left(\frac{Q^I(\mathbf{i})Q^\sigma(\Sigma)Q^\pi(j)}{P^I(\mathbf{i})P^\sigma(\Sigma)P^\pi(j)}\right) \\
&= \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \ln\left(\frac{Q^I(\mathbf{i})}{P^I(\mathbf{i})}\right) + \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \ln\left(\frac{Q^\sigma(\Sigma)}{P^\sigma(\Sigma)}\right) \\
&\quad + \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \mathop{\mathbb{E}}_{j\sim Q^\pi} \ln\left(\frac{Q^\pi(j)}{P^\pi(j)}\right) \\
&= \ln\left(\binom{m}{s}\right) + \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \ln\left(\frac{|\mathcal{M}^\sigma(S^{(m)}_{\mathbf{i}_{\mathcal{N}}})|}{|Sym(w, S^{(m)}_{\mathbf{i}_{\mathcal{N}}})|}\right) \\
&+ \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \mathop{\mathbb{E}}_{j\sim Q^\pi} \ln\left(|\mathcal{M}^\pi(S^{(m)}_{\mathbf{i}_{\mathcal{N}}}, \Sigma)|\right). \qquad \text{(A.39)}
\end{aligned}
$$

To conclude the proof, we crudely upper bound $\forall \mathbf{i}\ |\mathcal{M}^\sigma(S^{(m)}_{\mathbf{i}})| \leq 2^{ns}$, which follows because at each of $s$ samples $(x, y) \in S^{(m)}_{\mathbf{i}_{\mathcal{N}}}$ and at each neuron, $(l, i_l)$, of $n$ possible neurons, $\sigma^{(l)}(x, w)_{i_l}$ can take one of two values. We also have $|\mathcal{M}^\sigma(T_{\mathbf{i}})| \leq 2^n$ by Theorem 3. Clearly, $|Sym(w, S^{(m)}_{\mathbf{i}_{\mathcal{N}}})| \geq 1$. Combining this with Equation A.39, we have

$$KL(Q_{\mathcal{N}}||P_{S^{(m)}}) \leq \ln\left(\binom{m}{s}\right) + \ln(2^{ns}2^n)$$

where we simply drop $\ln(2) < 1$, and approximate $\binom{m}{s} \leq (\frac{me}{s})^s$ to arrive at

$$KL(Q_{\mathcal{N}}||P_{S^{(m)}}) \leq s\ln\left(\frac{m}{s}\right) + s + ns + n$$

Substituting the above into Equation A.36 finishes the proof. □

## A.3 Appendix for Chapter 4: Deep Logical Circuits as Neural Networks

### A.3.1 A Comparison of VC Dimension Bounds: Why Can We Get Away with Less?

The generalization bounds we presented can be extremely small, despite thematically similar to traditional VC dimension uniform bounds, denoted $VC^{\text{NoData}}$), which by contrast produce some of the largest bounds. That these measures differ by orders of magnitude (refer again to Table A.4) becomes more notable still once one realizes that "under the hood" they apply exactly the same theoretical machinery. Our workhorse theorem 6 can also be applied directly to derive tight bounds on $VC^{\text{NoData}}$ as in Bartlett et al. [2017b]. We reproduce it below for convenience.

**Theorem 6.** *(Theorem 17 in [Goldberg and Jerrum, 1995]): Let $k$,$n$ be positive integers and $f : \mathbb{R}^n \times \mathbb{R}^k \mapsto \{0, 1\}$ be a function that can be expressed as a Boolean formula containing $s$ distinct atomic predicates where each atomic predicate is a polynomial inequality or equality in $k + n$ variables of degree at most $d$. Let $\mathcal{F} = \{f(\cdot, w) : w \in \mathbb{R}^k\}$. Then VCDim$(\mathcal{F}) \leq 2k \log_2(8eds)$.*

In this section, we dissect the proof of the data independent VC dimension bound to understand more concretely what allows ours to be so much smaller. To rephrase, we want to understand through what mechanism the data-independent VC bound could potentially be improved if allowed additional assumptions on $\mathcal{N}$ of the form supported by our empirical observations of networks training on nice data. We will show that the notorious numerical bloat characteristic of $VC^{\text{NoData}}$ bounds can be localized to a particular proof step, and that this step can be greatly accelerated when the set system of neuron state boundaries organized to minimize intersections.

In addition to Figure 4.1 and the surrounding discussion, we also provide in the Appendix a more complete catalogue of similar experiments in Figure A.24, which may be a useful reference for the following discussion.

We will summarize briefly Theorem $8$ in Bartlett et al. [2017b]. The theorem orders the neurons in the network so that every neuron in layer $l$ comes before every neuron in layer $l+1$, with the output neuron last, whose state is the label by convention. Moving from beginning to end of this list of neurons, one asks how many additional polynomial [3] queries must be answered to determine the state of the $i+1^{\text{th}}$ neuron given the state of the first $1, \ldots, i$. The total resources in terms of the number of parameters $k$, polynomial inequalities $s$, and polynomial degree $d$ required to determine all the states is plugged into the VC bound in Theorem 6.

Perhaps much of the gap is attributable to neuron state coupling. Beyond the fact that many neurons are simply always on or always off, those with a nonempty neuron state boundary (NSB) share *a lot* of state information. A subset of $k$ neurons whose NSBs have empty intersection share state information, as not all $2^k$ states are possible. When they form parallel structures, as in Figure A.24, often the state of one will imply immediately the state of the other. Furthermore, we can infer by analyzing our Boolean formula (Eqn 5) that to determine the output label, we *do not need to determine* the state of those neurons whose NSB does not intersect the DB. To In our experiments, it is rare for even two such neuron boundaries to cross, and all seem repulsed from the decision boundary, implying that gradient descent has strong *combinatorial* regularizing properties.

Note that strong coupling of neuron states in a single hidden layer does *not* require linear dependence of the corresponding weight matrix rows: strong coupling is possible even for weight vectors in (linear) general position. For example, consider that for ReLU,

[3] With respect to the input, these are all linear inequalities. But, the weights parameterizing these linear functionals are composed by multiplication, so they are polynomial of degree bounded by the depth.

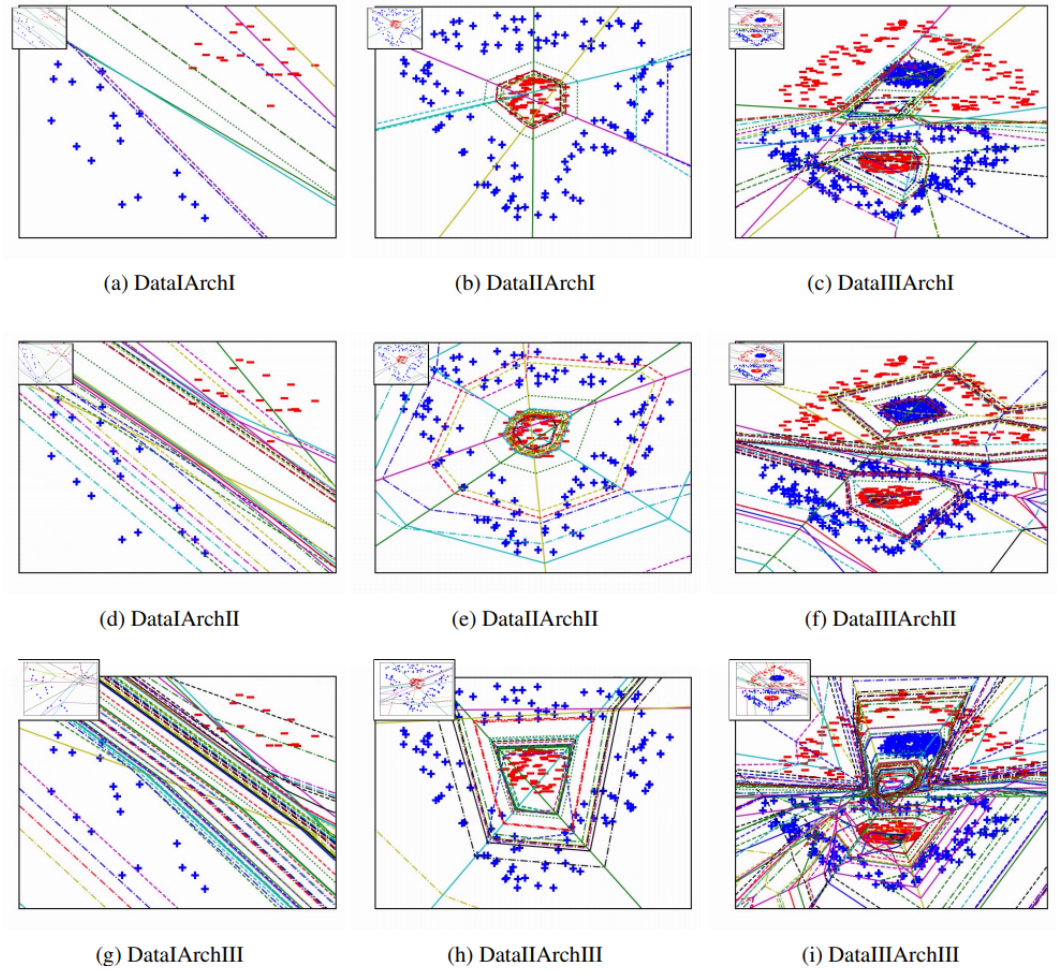
the activations from the previous layer are always in the (closed) positive orthant, $\mathbb{R}^n_+$. General (linear) dependence means there is no nonzero linear combination of weight vectors reaching the origin,$\{0\}$, which it will be instructive to consider as the dual of the entire space, $\{0\} = (\mathbb{R}^n)^*$. We propose that a more suitable notion of "general position" is instead the absence of nonzero linear combination of weights reaching the dual of some set containing the activation-image of the input. For simplicity, consider the positive orthant, $\mathbb{R}^n_+ = (\mathbb{R}^n_+)^*$. Our new notion of dependence coincides with neuron state dependence in the next hidden layer. For example, suppose for $\gamma_i > 0, \eta \in \mathbb{R}^n_+ \setminus \{0\}$, we have $\gamma_1 w_1 + \gamma_2 w_2 - \gamma_3 w_3 - \gamma_4 w_4 = \eta$. Then because $\forall x \in \mathbb{R}^n_+$, we have $\eta^T x > 0$, we cannot simultaneously have $w_1^T x, w_2^T x < 0$ while both $w_3^T x, w_4^T x > 0$. Conversely if for all $x \in \mathbb{R}^n$, the states $w_1^T x, w_2^T x < 0$ and $w_3^T x > 0$ imply $w_4^T x > 0$, then apply Farkas' lemma to get a positive linear combination of $-w_1, -w_2, w_3, w_4$ in $\mathbb{R}^n_+$.

These observations about dependencies between these neuron states illustrate nicely how the combinatorial capacity of networks trained on structured data diverges from the worst case theoretical analysis. Rather than handle one neuron at a time, we found it more useful to shift from a neuron-level to a network-level analysis by introducing network states $\bar{\sigma}(x)$, which we found to be significantly cleaner for interpretation and generalization bounds.

Table A.4: Measures of Combinatorial Capacity:

| | | $VC^{Bool}_{k,d,s}(\mathcal{N})(\lvert\bar{\Sigma}_0\rvert)[\lvert\bar{\Sigma}\rvert]$ | | |
|---|---|---|---|---|
| Architecture | $VC^{\text{NoData}}(\text{Arch})$ | DataI | DataII | DataIII |
| ArchI | 8,400 | 18(1)[36] | 380(10)[121] | 1870(38)[247] |
| ArchII | 101,000 | 74(2)[80] | 252(8)[166] | 806(19)[507] |
| ArchIII | 710,000 | 74(2)[167] | 335(6)[622] | 22,000(144)[2884] |

### A.3.2 Further Discussion on Simple Observations



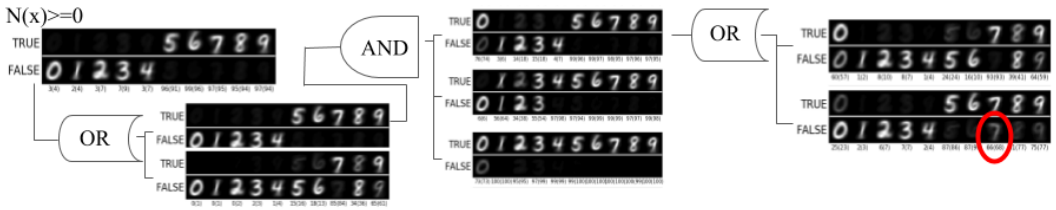
(a) DataIArchI (b) DataIIArchI (c) DataIIIArchI

(d) DataIArchII (e) DataIIArchII (f) DataIIIArchII

(g) DataIArchIII (h) DataIIArchIII (i) DataIIIArchIII

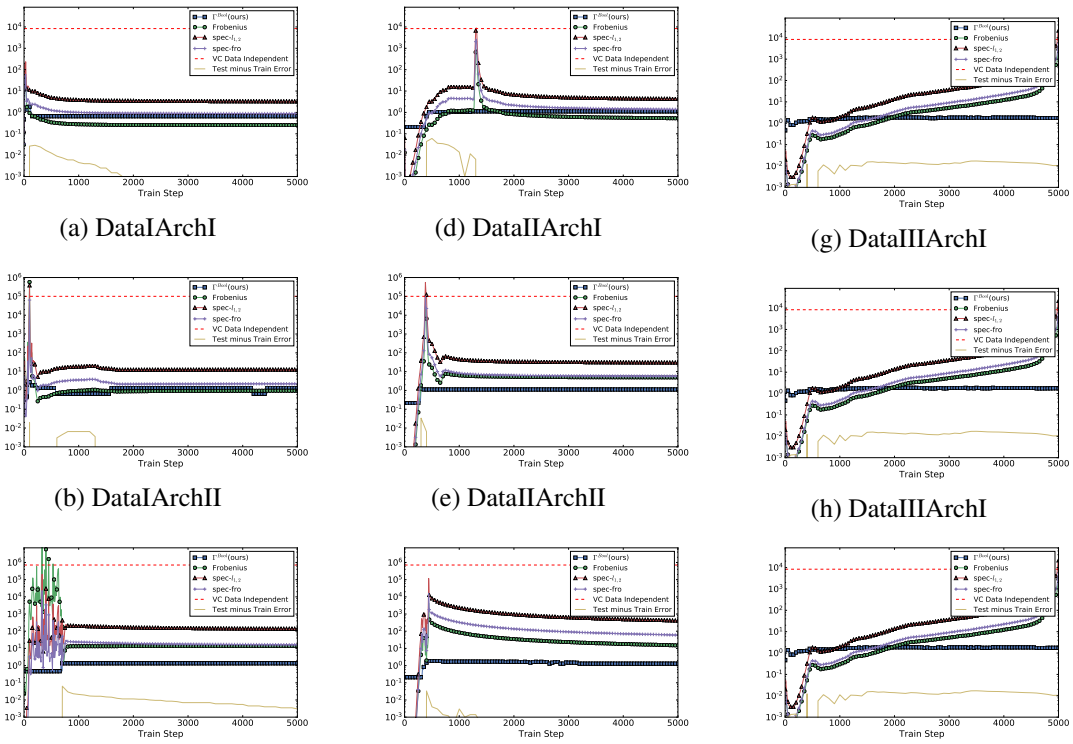
Figure A.24

Figure A.24: All neuron state boundaries (NSBs) of $9$ networks after training (large) and near the start of training (each top left insert) with varying data and architecture complexity. Animations of this process for all $9$ experiments are available in the supplemental material. For ArchI (top row), the $4$ line styles in order of increasing dotted-ness, solid, dashed, dash-dot, and dotted, correspond to increasing layer numbers. For all networks, we reserve the most dotted line for the decision boundary. We make the following observations: For fixed data, increasing the architecture size increases the number of linear regions, $|\bar{\Sigma}|$, but not necessarily the number of linear decision boundary pieces, $|\bar{\Sigma}_0|$, making it a more plausible candidate to relate to generalization error, which is also architecture size invariant. For fixed architecture, varying the data complexity controls the number of boundary pieces, which seem to be *around* as few as needed to separate the regions of positive and negative density. There is the appearance of a sort of "repulsive force", most readily apparent in the ArchII figures, between the decision boundary and the other NSBs. Recall that the decision boundary only "bends" when it intersects other NSBs. Such a force would then have a regularizing effect that minimizes the number of boundary pieces. Regions where individual neurons are active tend to nest hierarchically with somewhat parallel boundaries, producing fewer possible network states.




(a) A Strange Way to Learn 7s

Figure A.25: This is a companion figure to Figure 4.3, which contains generic instructions on how to read this type of diagram. Depicted is a selected subset of the logical circuit of a network trained on MNIST. This circuit is particularly large, and all depicted classifiers are Boolean combinations of other classifiers that are not shown. Similarly to the redundancy of Figure 4.2, many of the omitted are similar to those shown. In contrast to the very sensible organization of circuit Fig4.3a, the point of this figure is to show that these intermediate logical steps, though interpretable, can also be counter-intuitive, even circuitous seeming. We proceed from right to left. Already two logical operations before the terminal network output, one of the classifiers (row 2 far right) has *nearly* separated the data (images of training data shown here), except that it has difficulty with many 7s (red circle). Curiously, this is amended by combining using OR with a classifier that labels *both* 7 and 0 as True. This is curious. It is not clear why including 0 should be necessary or useful when including 7. To remedy the new problem of returning True for 0, it uses an AND junction to combine with another intermediate classifier that outputs False for $0, 1, 2, 3$. .




(a) DataIArchI (d) DataIIArchI (g) DataIIIArchI

(b) DataIArchII (e) DataIIArchII (h) DataIIIArchI

(c) DataIArchIII (f) DataIIArchIII (i) DataIIIArchI

Figure A.26: Generalization bounds for all experiments. See Figure 4.4 for additional details.

### A.3.3 Experimental Conditions

This training process can be seen as a function mapping a training set, network architecture pair to the DNN classifier and thus also to the generalization error we aim to study. As such, we are interested in observing how the complexity of our learned classifier depends on the "architecture complexity" and "data complexity", which we treat as independent variables. For this purpose, we designate 3 different network architectures, ArchI, ArchII, and ArchIII, of increasing size and three different datasets, DataI, DataII, and DataIII, of increasing "complexity" (both pictured in Appendix Figure A.27,A.28). Together, these comprise 9 experiments total. Rather than define explicitly what makes a dataset complex, to justify

DataI, DataII, and DataIII are of "increasing complexity", we simply note that these datasets are nested by construction, and, as a result, so too are the sets of classifiers that fit the data [4].

The three architecture sizes were chosen as our best guesses for the widest range of sizes our grid search algorithm would support comfortably. We do not recall changing them thereafter. The datasets were the the simplest interesting trio with the nesting property. The specific scaling and shifting configuration hard-coded into DataI,DataII,DataIII was simply the first one we found (after not much search) that allowed the shallowest network to achieve $0$ training error on the most difficult dataset. It is somewhat important for the data to be centered. Also, using Adam rather than gradient descent made this much easier, so we decided to standardize all our experiments to Adam(beta1=0.9,beta2=0.999) (the Tensorflow default configuration).

All experiments had a learning rate of $0.0005$, biases initialized to $0$, multiplicative weights initialized with samples from a mean, $0$, standard deviation, $0.05$, truncated normal distribution. These were originally set before the lifetime of this project in some code we re-purposed. We don't recall ever changing these thereafter.

Unlike the experiments in the rest of the paper, the deep logical circuits displayed in the MNIST Figures 4.3,A.25, are not identical from run to run. Instead, it seems almost every run gives a circuit that is at least slightly different. Initially, our intention was to train networks to label just a few digits, (instead of separating $0-4$ from $5-9$, but these circuits all turned out to be too trivial to be interesting. Our experimental design was to vary the architecture and the number of training samples, $m$. Our implementation in network_tree_decompositionṗy not optimized, so we only had time for $21$ of these runs. Most runs were interesting, so the real criteria was whether they could fit on a page cleanly.

In order to determine $\bar{\Sigma}_0$ for the MNIST experiments, we use a trick. Instead of

[4]There are many ways to define data complexity, but it's not clear which one should apply here. The ability to convincingly vary the data complexity in spite of this is in fact a key advantage of using synthetic data.

Table A.5: Architecture Properties

| Architecture | Depth (d) | Parameters | VCdim |
|---|---|---|---|
| ArchI | 3 | 107 | 8376 |
| ArchII | 6 | 517 | 101110 |
| ArchIII | 9 | 1743 | 709558 |

performing a grid search over the input, which is $784$ dimensional, we learn and additional $4$ to $6$ width *linear* layer before the first hidden layer of each architecture. We then do grid search over the first linear layer and compose with the projection map in post-processing.

We now cover some specific details of the circuits that were shown. Figure 4.3a is $0.98(0.94)$ training(validation) accuracy ArchI network with a $6$ neuron linear layer, trained with $m = 50$k samples. The circuit in Figure A.25 has the same settings and has $0.96(0.93)$ training(test) accuracy.

The overfit circuit with $1.0(0.78)$ train(test) accuracy in Figure 4.3b has a $4$ neuron linear layer, uses ArchIII, and has only 1k training points. The prosthetic network that we trained, also used ArchIII and used a subset of the *same* training data. Since we trained it to label $4$ as False and $5 - 9$ as True, we had a class imbalance issue, so we subsampled $30\%$ of the True labels. Therefore, the "prosthetic network" was trained with only $m = 250$ training samples.

All experiments were run comfortably on a TitanX GPU.

### A.3.4 Experimental Support for Theoretical Results

In this section we use the included file "network_tree_decomposition.py", which implements Algorithm A.3.1, to show experimental support for Theorems 4 and 5. Specifically we show that Equation 5 holds everywhere in the input space. Not mentioned in the theory section, we also show that the same indexing trick can be applied to Eqn 4.1 of Theorem 4: if one replaces $\bar{\Sigma}_0$ with $\bar{\Sigma}$ then one recovers a hierarchical MinMax. . .MinMax formulation that is

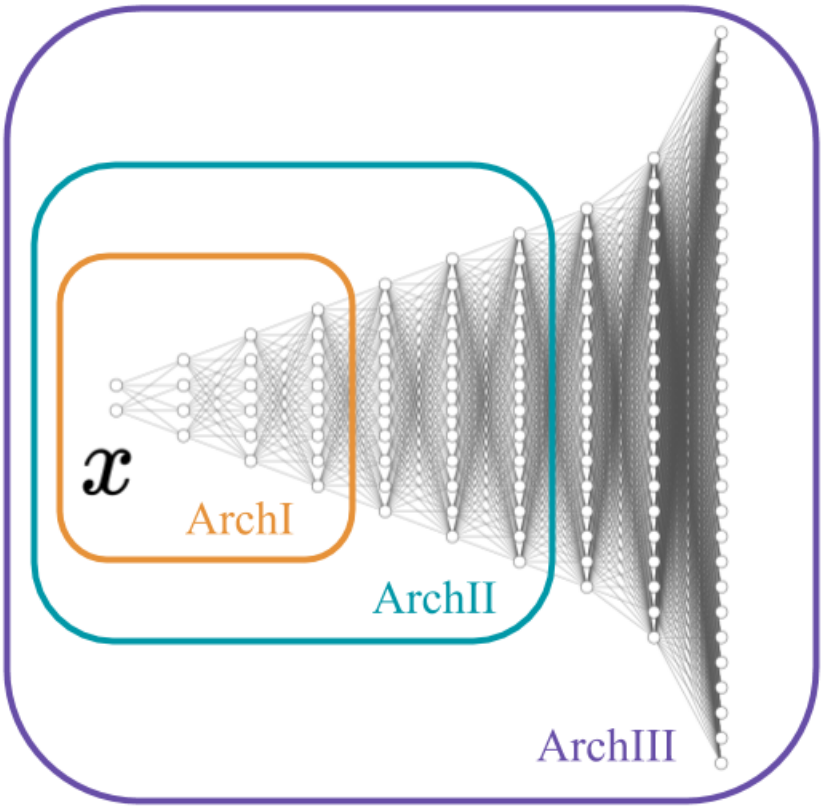



Figure A.27: This figure defines the network architectures, ArchI, ArchII, and ArchIII, used for binary classification experiments in this work. The two leftmost neurons represent 2-dimensional input data, while the remainder are hidden. Lines correspond to multiplicative weights between neurons. Biases are used in experiments but not shown in this figure. There are additional multiplicative weights not shown that connect the final hidden (equiv. rightmost) layer of each network to a scalar output. The architectures are nested, e.g., the structure of the first 3 hidden layers is constant across all experiments. The first 6 hidden layers have the same structure for ArchII and ArchIII.

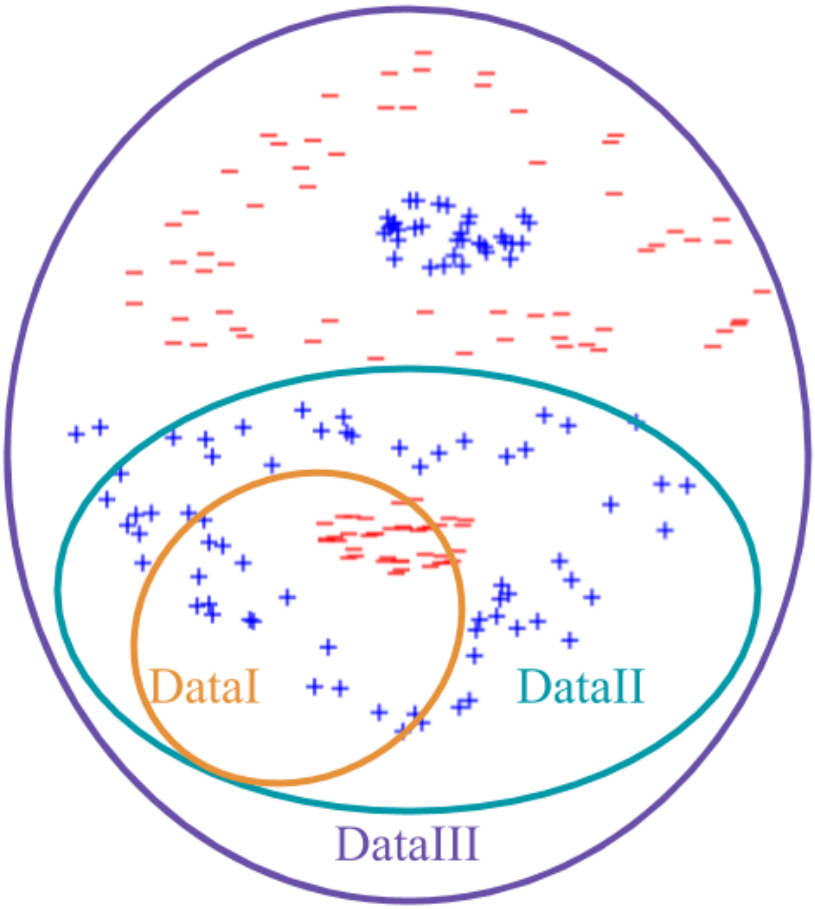
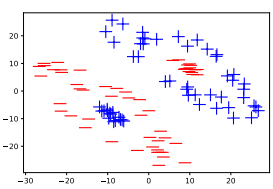
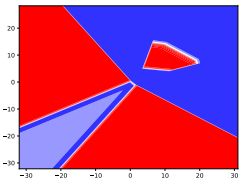



Figure A.28: Displayed is one possible sampling of the three data distributions used for experiments in this paper. Blue plus[red minus] signs correspond to training data from the positive[negative] class. We designed our datasets to explore the effect of classification difficulty/complexity on neural network VC dimension. Although we do not yet understand exactly what properties make a dataset "more difficult", we can reasonably expect datasets ordered by inclusion to also be ordered in complexity, e.g., every hypothesis that correctly classifies DataII also correctly classifies DataI. It should be noted that this inclusion ordering displayed in this figure is up to affine transformation. For example, the figure should be interpreted to mean that there is some affine transformation so that transformed samples from DataII follow the same distribution of samples from DataIII that lie lower than average. In fact, DataI, DataII, and DataIII all have roughly the same center of mass in experiments.

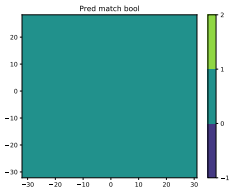
numerically equal to $\mathcal{N}(x)$.



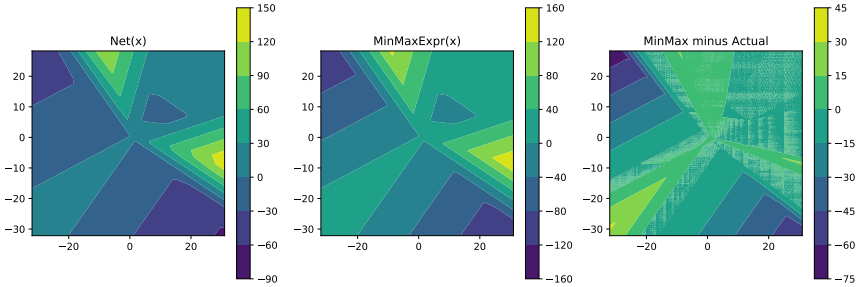
(a) The training data



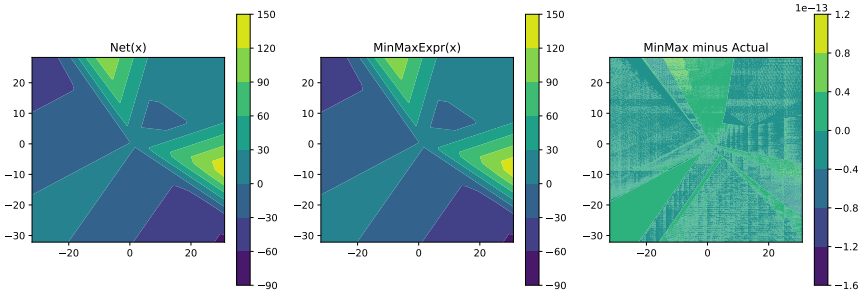
(b) Learned Classification




(c) Validate $\mathcal{N}$pred==Bool




(d) Algorithm A.3.1 (Mode=Logical) Output Comparison




(e) Algorithm A.3.1 (Mode=Numeric) Output Comparison

Figure A.29: (Caption on next page.)

Figure A.29: (Previous page.) Experimental readout from network_tree_decomposition.py, which implements Algorithm A.3.1, confirming the validity of our theoretical claims. Pictured is the classification boundary A.29b of a DNN trained on A.29a. The second[third] row (Fig. A.29d [Fig. A.29e]) plots the numeric value of the MinMax tree when indexed over $\bar{\Sigma}_0[\bar{\Sigma}]$ corresponding to mode=Logical [mode=Numeric] in Algorithm A.3.1. From left to right, rows 2 and 3 depict the network output, MinMax tree output, and their difference. For mode=Numeric, their difference is within machine precision of 0 (row 3 column 3). For the second row, mode=Logical uses the same MinMax formulation, but indexes over $\bar{\Sigma}_0$ instead (It is Thm 5 but with MinMax instead of $\wedge\vee$). We can see the numeric relation to $\mathcal{N}(x)$ is lost (row 2 column 3). However, the *sign* of this output still agrees with $\mathcal{N}(x)$ everywhere. One can check this visually by comparing row 2 columns 1 and 2. Or, one can refer to Fig. A.29c, where we plot 1 everywhere the two are equivalent. Because we see a constant image that is 1 everywhere, we can infer our predictions are the same for all labels.

### A.3.5 Algorithms: Definitions and Pseudocode

Algorithm: Generalization Bound Calculation

---

**Function** `BdryStates` $(x \mapsto \bar{\sigma}(x), \mathcal{N}(x), \mathcal{X}^{compact}, \delta)$ **:**

Grid=MakeGrid($\mathcal{X}$,$\mathcal{X}^{\text{compact}}$,spacing=$\delta$)

$\bar{\Sigma}_-, \bar{\Sigma}_+ = \varnothing, \varnothing$

**for** $x$ *in Grid* **do**

  **if** $\mathcal{N}(x) \geq 0$ **then**

    $\bar{\Sigma}_+ \leftarrow \bar{\Sigma}_+ \cup \{x\}$

  **else if** $\mathcal{N}(x) < 0$ **then**

    $\bar{\Sigma}_- \leftarrow \bar{\Sigma}_- \cup \{x\}$

**end**

$\bar{\Sigma}_0 = \bar{\Sigma}_+ \cap \bar{\Sigma}_-$

**return** $\bar{\Sigma}_0$

**Function** `MinimalDescrip` $(\bar{n}, \bar{\Sigma}_0)$ **:**

$s = |\bar{\Sigma}_0|^2$

$r_{d+1} = 1$

**for** $l \in d, \ldots, 1$ **do**

  $t_l = rank(\bar{\Sigma}_0^l)$

  $m_l = |\{i \in n_l | \exists \sigma \in \bar{\Sigma}_0^l \text{ with } \sigma_i \neq 0\}$

  $r_l = \min\{m_l, t_l r_{l+1}\}$

**end**

$k = \sum_{l=0}^{d} (r_l - 1) r_{l+1}$

$\bar{r} = (r_0, r_1, \ldots, r_d)$%effective widths

$d = |\{l | r_l > 1\}|$%effective depth

**return** $k, s, d, \bar{r}$

**return**

**Function** `Thm3Bound` $(k, s, d, m)$ **:**

$VC = 2k \log_2(8esd)$

**return** $\sqrt{VC/m}$

**return**

---

Algorithm: Network Tree Decomposition

**Data:** $n_{total} = \sum_{l=1}^{d} n_l$, $\Sigma \in \{\bar{\Sigma}, \bar{\Sigma}_0\}$, $Mode \in \{Numeric, Logical\}$.

$Split\Sigma \subset \bigtimes_{l=1}^{d} \{0,1\}^{|\Sigma| \times n_l}$ is $\Sigma$ split along last into a layer-indexed list of arrays of shape.

network weights $A^{(d)}, b^{(d)}, \ldots, A^{(0)}, b^{(0)}$

**Result:** If $\Sigma == \bar{\Sigma}$ is the set of network states, return a tree whose terminal leaves are affine functions, whose nodes are Min or Max compositions of children, and whose root computation is numerically equivalent to $\mathcal{N}(x)$. If $\Sigma == \bar{\Sigma}$ is the set of network states or $\Sigma == \bar{\Sigma}_0$ is the set of network states at the boundary, return a tree whose leaves are affine classifiers, whose nodes are And or Or compositions of children, and whose root is logically equivalent to the statement $\mathcal{N}(x) \geq 0$.

**Let:**

| | | |
|---|---|---|
| OpTree | =ROOT | % accumulates symbolic rep |
| LeafInfo.$\mu Index$ | $[1, 2, \ldots, \lvert\Sigma\rvert]$ | |
| LeafInfo.$\tau Index$ | $[1, 2, \ldots, \lvert\Sigma\rvert]$ | |
| LeafInfo.$HiddenLayer$ | $d$ | %depth, |
| LeafInfo.$Affine.(\alpha, \beta)$ | $[A^{(d)}, b^{(d)}]$ | $\%(z \mapsto \alpha z + \beta)$ |
| LeafStack | {ROOT:LeafInfo} | %leaf indexed lookup |
| TerminalLeafs | {} | %empty accumulator |

**if** *Mode is Numeric* **then**
  MeetOp,JoinOp=Min,Max;
**else if** *Mode is Logicial* **then**
  MeetOp,JoinOp=And,Or;
**while** *LeafStack* **do**
  LeafSymbol,LeafInfo=LeafStack.pop();
  $lyr$=LeafInfo.$HiddenLayer$;
  RemainPosSig=$Split\Sigma[lyr]$[LeafInfo.$\mu Index$];
  RemainNegSig=$Split\Sigma[lyr]$[LeafInfo.$\tau Index$];
  $\alpha, \beta$=LeafInfo.$Affine$;
  %Group remaining neuron states that are same on positive/negative support of $\alpha$
  JoinSymbols=[];
  **for** *$\mu linear, \mu idx$ in Unique(RemainPosSig$\odot\alpha_+$)* **do**
    MeetSymbols=[];
    **for** *$\tau linear, \tau idx$ in Unique(RemainNegSig$\odot\alpha_-$)* **do**
      NewInfo.$Affine$=ComposeAffine([[$A^{(lyr-1)}, b^{(lyr-1)}$],[$\mu linear + \tau linear, \beta$]];
      NewInfo.$\mu Index = \mu idx$ %tuple of indices from LeafInfo.$\mu Index$;
      NewInfo.$\tau Index = \tau idx$;
      NewInfo.$HiddenLayer = lyr - 1$;
      NewSymbol=MakeSymbol(LeafSymbol,$\tau^{lyr}, \mu^{lyr}$);
      **if** $lyr - 1 > 0$ **then**
        LeafStack.update({NewSymbol:NewInfo});
      **else**
        TerminalLeafs.update({NewSymbol:NewInfo});
      **end**
      MeetSymbols.append(NewSymbol);
    **end**
    JoinSymbols.append( Reduce(MeetSymbols, MeetOP) );
  **end**
  NewBranch=Reduce(JoinSymbols, JoinOP );
  OpTree.subs( LeafSymbol, NewBranch );
**end**

### A.3.6 Supporting Theoretical Exposition

This section contains the proofs for the theorems laid out in the paper. All previous results are restated for convenience. Some new results are added to facilitate exposition.

**Proposition 2.** *Let $f : \mathcal{A} \times \mathcal{B} \mapsto \mathbb{R}$. Then we have the following logical equivalence:*

$$\left[\max_{\alpha\in\mathcal{A}} \min_{\beta\in\mathcal{B}} f(\alpha,\beta) \geq 0\right] \iff \bigvee_{\alpha\in\mathcal{A}} \bigwedge_{\beta\in\mathcal{B}} \left[f(\alpha,\beta) \geq 0\right]$$

*Proof.* There is essentially nothing to prove as both statements are equivalent to $\exists\alpha\forall\beta f(\alpha,\beta) \geq 0$. □

**Definition 5.** *(Network Operand, $P^{(d)}$)*

*For a ReLU DNN composed of weights, $A^{(l)}$, and biases, $b^{(l)}$, for $l = 0,\ldots,d$, mapping inputs $x$ to $\mathbb{R}$, we define the Network Operand, $P^{(l)}$ (at layer $l$), as follows:*

$$P^{(0)}(x) = b^{(0)} + A^{(0)}x$$
$$P^{(1)}(\mu^1,\tau^1,x) = b^{(1)} + A^{(1)}_{+}\mu^1(b^{(0)} + A^{(0)}x) - A^{(1)}_{-}\tau^1(b^{(0)} + A^{(0)}x)$$

*Given $P^{(l)}$, define $P^{(l+1)}$ by*

$$\begin{aligned} P^{(l+1)}(\mu^1,\ldots\mu^{l+1},\tau^1,\ldots,\tau^{l+1},x) = b^{(l+1)} + \\ A^{(l+1)}_{+}\mu^{l+1}P^{(l)}(\mu^1,\ldots\mu^l,\tau^1,\ldots,\tau^l,x) - \\ A^{(l+1)}_{-}\tau^{l+1}P^{(l)}(\tau^1,\ldots\tau^l,\mu^1,\ldots,\mu^l,x) \end{aligned} \tag{A.40}$$

*taking note that the roles of $\tau$ and $\mu$ are switched in the last term.*

For induction purposes, we define $\mathcal{N}^{(l)}$ to be the vector valued ReLU network

consisting of hidden layers $1, \ldots, l$ of $\mathcal{N}$:

$$\mathcal{N}^{(l)}(x) \triangleq b^{(l)} + A^{(l)}R(b^{(l-1)} + A^{(l-1)}R(b^{(l-2)} + \ldots + A^{(1)}R(b^{(0)} + A^{(0)}x)\ldots)).$$

so that $\mathcal{N}^{(d)} = \mathcal{N}$ and $\mathcal{N}^{(l+1)}(x) = b^{(l+1)} + A^{(l+1)}R(\mathcal{N}^{(l)}(x))$.

**Theorem 4.** *Let $P^{(d)}$ be the net operand for any fully-connected ReLU network, $\mathcal{N}$. Then,*

$$\mathcal{N}(x) = \max_{\mu^d} \min_{\tau^d} \cdots \max_{\mu^1} \min_{\tau^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \tag{4.1}$$

$$\Big[\mathcal{N}(x) \geq 0\Big] \Leftrightarrow \bigvee_{\mu^d} \bigwedge_{\tau^d} \cdots \bigvee_{\mu^1} \bigwedge_{\tau^1} \Big[P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\Big] \tag{4.2}$$

*Proof.* We first prove Eqn 4.1 by induction on $d$. Eqn 3 will follow directly from 2 through repeated application of Prop 2. Clearly, $\mathcal{N}^{(0)}(x) = b^{(0)} + A^{(0)}x = P^{(0)}(x)$, so Eqn 4.1 holds for $k = 0$ hidden layers. Assume Eqn 4.1 is true for $1, \ldots, k$.

By definition of $\mathcal{N}^{(k+1)}$ and the inductive assumption,

$$
\begin{aligned}
\mathcal{N}^{(k+1)}(x) &= b^{(k+1)} + A^{(k+1)} R(\mathcal{N}^{(k)}(x)) \\
&= b^{(k+1)} + \max_{\mu^{k+1}} \min_{\tau^{k+1}} (A_+^{(k+1)} \mu^{k+1} - A_-^{(k+1)} \tau^{k+1})(\mathcal{N}^{(k)}(x)) \\
&= b^{(k+1)} + \max_{\mu^{k+1}} \min_{\tau^{k+1}} (A_+^{(k+1)} \mu^{k+1} - A_-^{(k+1)} \tau^{k+1})(\max_{\mu^k} \min_{\tau^k} \cdots \\
&\quad \max_{\mu^1} \min_{\tau^1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&= b^{(k+1)} + \max_{\mu^{k+1}} \min_{\tau^{k+1}} \Bigg[ A_+^{(k+1)} \mu^{k+1} (\max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&\quad - (\max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} A_-^{(k+1)} \tau^{k+1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&= b^{(k+1)} + \max_{\mu^{k+1}} \min_{\tau^{k+1}} \Bigg[ A_+^{(k+1)} \mu^{k+1} (\max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&\quad (\min_{\mu^k} \max_{\tau^k} \cdots \min_{\mu^1} \max_{\tau^1} -A_-^{(k+1)} \tau^{k+1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&= b^{(k+1)} + \max_{\mu^{k+1}} \min_{\tau^{k+1}} \Bigg[ (\max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} A_+^{(k+1)} \mu^{k+1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&\quad (\max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} -A_-^{(k+1)} \tau^{k+1} P^{(k)}(\tau^1, \ldots \tau^k, \mu^1, \ldots, \mu^k, x)) \Bigg] \\
&= \max_{\mu^{k+1}} \min_{\tau^{k+1}} \max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} \Bigg[ b^{(k+1)} + A_+^{(k+1)} \mu^{k+1} P^{(k)}(\mu^1, \ldots \mu^k, \tau^1, \ldots, \tau^k, x)) \\
&\quad - A_-^{(k+1)} \tau^{k+1} P^{(k)}(\tau^1, \ldots \tau^k, \mu^1, \ldots, \mu^k, x) \Bigg] \\
&= \max_{\mu^{k+1}} \min_{\tau^{k+1}} \max_{\mu^k} \min_{\tau^k} \cdots \max_{\mu^1} \min_{\tau^1} \Big[ P^{(k+1)}(\mu^1, \ldots \mu^k, \mu^{k+1}, \tau^1, \ldots, \tau^k, \tau^{k+1}, x) \Big]
\end{aligned}
$$

Thus by induction Eqn 4.1 holds for any depth. Now we can apply Prop 2 recursively to derive Eqn 4.2.

$$\begin{aligned}
\left[\mathcal{N}(x) \geq 0\right] &\iff \left[\max_{\mu^d}\min_{\tau^d}\max_{\mu^{d-1}}\min_{\tau^{d-1}}\cdots\max_{\mu^1}\min_{\tau^1} P^{(d)}(\mu^1,\dots\mu^d,\tau^1,\dots,\tau^d,x) \geq 0\right] \\
&\iff \bigvee_{\mu^d}\bigwedge_{\tau^d}\left[\max_{\mu^{d-1}}\min_{\tau^{d-1}}\cdots\max_{\mu^1}\min_{\tau^1} P^{(d)}(\mu^1,\dots\mu^d,\tau^1,\dots,\tau^d,x) \geq 0\right] \\
&\iff \bigvee_{\mu^d}\bigwedge_{\tau^d}\bigvee_{\mu^{d-1}}\bigwedge_{\tau^{d-1}}\left[\cdots\max_{\mu^1}\min_{\tau^1} P^{(d)}(\mu^1,\dots\mu^d,\tau^1,\dots,\tau^d,x) \geq 0\right] \\
&\vdots \\
&\iff \bigvee_{\mu^d}\bigwedge_{\tau^d}\bigvee_{\mu^{d-1}}\bigwedge_{\tau^{d-1}}\cdots\bigvee_{\mu^1}\bigwedge_{\tau^1}\left[P^{(d)}(\mu^1,\dots\mu^d,\tau^1,\dots,\tau^d,x) \geq 0\right]
\end{aligned}$$

□

For the following, we recall the following notation: For $J \subset [d]$, $\bar{\Sigma}^J[\bar{\Sigma}_0^J]$ is the projection of $\bar{\Sigma}[\bar{\Sigma}_0]$ onto the coordinates indexed by $J$. The symbols $\bar{\mu}^{[l]} = (\mu^1,\dots,\mu^l)$ for $l \leq d$ and $\bar{\mu} = \bar{\mu}^{[d]} = (\mu^1,\dots,\mu^d)$ to be equivalent (representing the same quantity) in any context they appear together. For example, if $\bar{\mu} \in \bar{\Sigma}$ then $\bar{\mu}^{[l]} \in \bar{\Sigma}^{[l]}$. We will here consider $\bar{\mu}^{[l]}$ as a concatenated vector in $\mathbb{R}^{\sum_{i=1}^{l}\Omega i}$.

**Lemma 6.** *The Fundamental Lemma of the Net Operand*

*Let $\mathcal{N}$ be a ReLU network with net operand, $P^{(d)}$, and $\bar{\sigma} : \mathbb{R}^{\Omega 0} \mapsto \bar{\Sigma}$, mapping inputs, $x$, to network states, $\bar{\sigma}(x)$. Then for arbitrary binary vectors $\hat{\mu}, \hat{\tau}$ the following relations hold*

$$\min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x),\bar{\tau},x) = \mathcal{N}(x) = \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu},\bar{\sigma}(x),x) \tag{A.41}$$

$$\min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\hat{\mu},\bar{\tau},x) \leq P^{(d)}(\hat{\mu},\bar{\sigma}(x),x) \leq \mathcal{N}(x) \leq P^{(d)}(\bar{\sigma}(x),\hat{\tau},x) \leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu},\hat{\tau},x). \tag{A.42}$$

It's worth pointing out that Lemma 6 implies $\mathcal{N}(x) = P^{(d)}(\bar{\sigma}(x),\bar{\sigma}(x),x)$, as was

claimed in the text. This is in fact how numerical equality is achieved in Equation 4.1.

*Proof.* We claim it is sufficient to show

$$\max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x) \leq \mathcal{N}(x) \leq \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x), \bar{\tau}, x). \tag{A.43}$$

To see this, note the following chain of inequalities. They make use of the fact that the maximum[minimum] is always greater[less] than or equal to any particular fixed value.

$$\begin{aligned}
&\min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\hat{\mu}, \bar{\tau}, x) \leq P^{(d)}(\hat{\mu}, \bar{\sigma}(x), x) \leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x) \leq \mathcal{N}(x) \\
&\leq \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x), \bar{\tau}, x) \leq P^{(d)}(\bar{\sigma}(x), \hat{\tau}, x) \leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \hat{\tau}, x).
\end{aligned}$$

Thus the we obtain the inequalities in Eqn A.42. Analyzing the special case that $\hat{\mu} = \hat{\tau} = \bar{\sigma}(x)$, we also obtain

$$\min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x), \bar{\tau}, x) \leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x) \leq \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x), \bar{\tau}, x) \leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x).$$

Thus we obtain the equalities in Eqn. A.41. Now we turn to proving Equation A.43.

Consider the term-wise expansion of $P^{(d)}$. We claim every term containing $\mu^1$ has a leading $+$ and every term containing $\tau^1$ has a leading $-$. When $d = 1$, this is obvious. And, if it is true in the expansion of $P^{(l)}(\mu^{[l]}, \tau^{[l]}, x)$, then by definition (Eqn. A.40) it is true in the expansion of $P^{(l+1)}(\mu^{[l+1]}, \tau^{[l+1]}, x)$, since the minus sign in front of $P^{(l)}(\tau^{[l]}, \mu^{[l]}, x)$ also features $\mu^1$ and $\tau^1$ switching roles.

Since every matrix in the expansion of $P^{(d)}$ is entry-wise nonnegative, for every fixed $\mu^2, \ldots, \mu^d$ and $\tau^2, \ldots, \tau^d$, we can by combining terms obtain some bias $\beta$ and nonnegative

vectors $v_+^a, v_+^b \in \mathbb{R}_+^{\Omega 1}$ so that

$$\begin{aligned} P^{(d)}(\bar{\mu}, \bar{\tau}, x) &= \beta + v_+^a \mu^1 (b^{(0)} + A^{(0)} x) - v_+^b \tau^1 (b^{(0)} + A^{(0)} x) \\ &= \beta + v_+^a \mu^1 \mathcal{N}^{(0)}(x) - v_+^b \tau^1 \mathcal{N}^{(0)}(x) \end{aligned}$$

Therefore, for every fixed $\mu^2, \ldots, \mu^d$ and $\tau^2, \ldots, \tau^d$, we may consider $\tau^1 = \sigma^{(l)} 1(x)$ to be an optimal choice (for any $\mu^1$) and $\mu^1 = \sigma^{(l)} 1(x)$ to be an optimal choice (for any $\tau^1$). We have shown that for any $\mu^2, \ldots, \mu^d$ and $\tau^2, \ldots, \tau^d$, that

$$\begin{aligned} &\max_{\mu^1} P^{(d)}(\mu^1, \mu^2, \ldots, \mu^d, \sigma^{(l)} 1(x), \tau^2, \ldots, \tau^d, x) \\ &\leq P^{(d)}(\sigma^{(l)} 1(x), \mu^2, \ldots, \mu^d, \sigma^{(l)} 1(x), \tau^2, \ldots, \tau^d, x) \\ &\leq \min_{\tau^1} P^{(d)}(\sigma^{(l)} 1(x), \mu^2, \ldots, \mu^d, \tau^1, \tau^2, \ldots, \tau^d, x) \end{aligned}$$

Now suppose for some $k \leq d$ that $\mu^1 = \tau^1 = \sigma^{(l)} 1(x), \ldots, \mu^{k-1} = \tau^{k-1} = \sigma^{(l)} k-1(x)$. Let $\mu^{k+1}, \ldots, \mu^d$ and $\tau^{k+1}, \ldots, \tau^d$ be arbitrary and fixed. It is quite clear by substitution that $P^{(k-1)}(\bar{\sigma}^{[k-1]}, \bar{\sigma}[k-1], x) = \mathcal{N}^{(k-1)}(x)$. Then we may substitute $\mathcal{N}^{(k-1)}(x)$ for every $P^{(k-1)}$ in the expansion of $P^{(d)}$. We are left with the same expansion as before but with $(\mu^k, \tau^k)$ in place of $(\mu^1, \tau^1)$, $d-k+1$ in place of $d$, and $\mathcal{N}^{(k-1)}(x)$ in place of $\mathcal{N}^{(0)}(x)$. Accordingly, we can conclude by the same logic that $\tau^k = \sigma^{(l)} k(x)$ minimizes $P^{(d)}$, and that $\mu^k = \sigma^{(l)} k(x)$ maximizes $P^{(d)}$, as soon as $\mu^l = \tau^l = \sigma^{(l)} l(x)$ for $l < k$.

If we apply this reasoning recursively, we can see

$$
\begin{aligned}
\max_{\bar{\mu}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x) &= \max_{\mu^d,\ldots,\mu^1} P^{(d)}(\mu^1, \ldots \mu^d, \bar{\sigma}(x), x) \\
&\leq \max_{\mu^d,\ldots,\mu^2} P^{(d)}(\sigma^{(l)}1(x), \mu^2, \ldots \mu^d, \bar{\sigma}(x), x) \\
&\leq \max_{\mu^d,\ldots,\mu^3} P^{(d)}(\sigma^{(l)}1(x), \sigma^{(l)}2(x), \mu^3, \ldots \mu^d, \bar{\sigma}(x), x) \\
&\vdots \\
&\leq P^{(d)}(\bar{\sigma}(x), \bar{\sigma}(x), x) = \mathcal{N}(x) \\
&\vdots \\
&\leq \min_{\tau^d,\ldots,\tau^3} P^{(d)}(\bar{\sigma}(x), \sigma^{(l)}1(x), \sigma^{(l)}2(x), \tau^3, \ldots \tau^d, x) \\
&\leq \min_{\tau^d,\ldots,\tau^2} P^{(d)}(\bar{\sigma}(x), \sigma^{(l)}1(x), \tau^2, \ldots \tau^d, x) \\
&\leq \min_{\tau^d,\ldots,\tau^1} P^{(d)}(\bar{\sigma}(x), \tau^1, \ldots \tau^d, x) \\
&= \min_{\bar{\mu}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x)
\end{aligned}
$$

Since we have shown Equation A.43, this concludes the proof.

□

This Theorem is not essential to the main story, but the last line (Eqn. A.46) is referenced as "Mode=Numeric" in Algorithm A.3.1. It can be thought of as a numeric analogue of Theorem 5 that models the *function* $\mathcal{N}(x)$ using Min and Max. Of course, this model requires that all network states, $\bar{\Sigma}$, participate. Not just those at the boundary, $\bar{\Sigma}_0$.

**Theorem 11.** *The order of the operands in Eqn 4.1 may be switched as*

$$\begin{aligned}
\mathcal{N}(x) &= \max_{\mu^d} \min_{\tau^d} \cdots \max_{\mu^1} \min_{\tau^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \\
&= \max_{\bar{\mu}\in\bar{\Sigma}} \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) && \text{(A.44)} \\
&= \min_{\bar{\tau}\in\bar{\Sigma}} \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) && \text{(A.45)} \\
&= \max_{\mu^d\in\bar{\Sigma}^d} \min_{\tau^d\in\bar{\Sigma}^d} \max_{\{\mu^{d-1}|(\mu^{d-1},\mu^d)\in\bar{\Sigma}^{d-1,d}\}} \min_{\{\tau^{d-1}|(\tau^{d-1},\tau^d)\in\bar{\Sigma}^{d-1,d}\}} \cdots \\
&\quad \max_{\{\mu^1|(\mu^1,\ldots,\mu^d)\in\bar{\Sigma}\}} \min_{\{\tau^1|(\tau^1,\ldots,\tau^d)\in\bar{\Sigma}\}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) && \text{(A.46)}
\end{aligned}$$

*Proof.* To show Equations A.44 and A.45, simply use the equality relations in Lemma 6 and the minmax inequality:

$$\begin{aligned}
\mathcal{N}(x) &= \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x), \bar{\tau}, x) \\
&\leq \max_{\bar{\mu}\in\bar{\Sigma}} \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \leq \min_{\bar{\tau}\in\bar{\Sigma}} \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \\
&\leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x) \\
&= \mathcal{N}(x)
\end{aligned}$$

□

The proof of Eqn A.46 could have been incorporated into the minmax inequality step above, but instead we treat it separately for notational clarity. In the next chain of equations we suppress the index set notation for a few lines so as not to obscure the shuffling of Min and Max operators.

$$\mathcal{N}(x) = \min_{\bar{\tau}} \max_{\bar{\mu}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \tag{A.47}$$

$$= \min_{\tau^d \in \bar{\Sigma}^d} \min_{\{\tau^{d-1} | (\tau^{d-1}, \tau^d) \in \bar{\Sigma}^{d-1,d}\}} \cdots \min_{\{\tau^1 | (\tau^1, \ldots, \tau^d) \in \bar{\Sigma}\}}$$

$$\max_{\mu^d \in \bar{\Sigma}^d} \max_{\{\mu^{d-1} | (\mu^{d-1}, \mu^d) \in \bar{\Sigma}^{d-1,d}\}} \max_{\{\mu^1 | (\mu^1, \ldots, \mu^d) \in \bar{\Sigma}\}} P^{(d)}(\bar{\mu}, \bar{\tau}, x)$$

$$= \min_{\tau^d} \cdots \min_{\tau^1} \max_{\mu^d} \cdots \max_{\mu^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x)$$

$$\geq \min_{\tau^d} \cdots \min_{\tau^2} \max_{\mu^d} \cdots \max_{\mu^1} \min_{\tau^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x)$$

$$\geq \min_{\tau^d} \cdots \min_{\tau^3} \max_{\mu^d} \cdots \max_{\mu^2} \min_{\tau^2} \max_{\mu^1} \min_{\tau^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x)$$

$$\vdots$$

$$\geq \max_{\mu^d \in \bar{\Sigma}^d} \min_{\tau^d \in \bar{\Sigma}^d} \cdots \max_{\{\mu^1 | (\mu^1, \ldots, \mu^d) \in \bar{\Sigma}\}} \min_{\{\tau^1 | (\tau^1, \ldots, \tau^d) \in \bar{\Sigma}\}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) = \text{Eqn A.46}$$

$$\geq$$

$$\vdots$$

$$\geq \max_{\bar{\mu}} \min_{\bar{\tau}} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \tag{A.48}$$

$$= \mathcal{N}(x)$$

**Proposition 5.** *Let $\mathcal{N}$ be a ReLU network with net operand, $P^{(d)}$, and states $\bar{\Sigma} = \bar{\Sigma}_+ \cup \bar{\Sigma}_-$. Then*

$$\left[\mathcal{N}(x) \geq 0\right] \Leftrightarrow \left[\max_{\bar{\mu} \in \bar{\Sigma}_+} \min_{\bar{\tau} \in \bar{\Sigma}_-} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \Leftrightarrow \left[\min_{\bar{\tau} \in \bar{\Sigma}_-} \max_{\bar{\mu} \in \bar{\Sigma}_+} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \tag{A.49}$$

*Proof.* If $\mathcal{N}(x) \geq 0$, then $\bar{\sigma}(x) \in \bar{\Sigma}_+$. Therefore

$$\max_{\bar{\mu}\in\bar{\Sigma}_+} \min_{\bar{\tau}\in\bar{\Sigma}_-} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq \min_{\bar{\tau}\in\bar{\Sigma}} P^{(d)}(\bar{\sigma}(x), \bar{\tau}, x) = \mathcal{N}(x) \geq 0.$$

Conversely, if $\mathcal{N}(x) < 0$, then $\bar{\sigma}(x) \in \bar{\Sigma}_-$ and

$$\min_{\bar{\tau}\in\bar{\Sigma}_-} \max_{\bar{\mu}\in\bar{\Sigma}_+} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \leq \max_{\bar{\mu}\in\bar{\Sigma}} P^{(d)}(\bar{\mu}, \bar{\sigma}(x), x) = \mathcal{N}(x) < 0.$$

We have proved the first and third arrows of the below sequence of implications,

$$\left[\mathcal{N}(x) \geq 0\right] \Rightarrow \left[\max_{\bar{\mu}\in\bar{\Sigma}_+} \min_{\bar{\tau}\in\bar{\Sigma}_-} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \Rightarrow \left[\min_{\bar{\tau}\in\bar{\Sigma}_-} \max_{\bar{\mu}\in\bar{\Sigma}_+} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \Rightarrow \left[\mathcal{N}(x) \geq 0\right].$$

The middle one is a consequence of the minmax inequality. □

We are almost done with our theoretical development, and we have not yet used the fact that $P^{(d)}$ is a linear (affine) function of $x$. That changes with the next Theorem, which requires Farkas' Lemma. To linearize, we define $\tilde{x} = \binom{x}{1} \in \mathbb{R}^{\Omega 0+1}$ to be the embedding of our inputs in the affine plane.

**Theorem 5.** *Let $\mathcal{N}$ be a fully-connected ReLU network with net operand, $P^{(d)}$, and boundary states, $\bar{\Sigma}_0$. Then,*

$$\begin{aligned}[\mathcal{N}(x) \geq 0] \Leftrightarrow \bigvee_{\mu^d\in\bar{\Sigma}_0^d} \bigwedge_{\tau^d\in\bar{\Sigma}_0^d} \bigvee_{\{\mu^{d-1}|(\mu^{d-1},\mu^d)\in\bar{\Sigma}_0^{d-1,d}\}} \cdots \\ \bigvee_{\{\mu^1|\bar{\mu}\in\bar{\Sigma}_0\}} \bigwedge_{\{\tau^1|\bar{\tau}\in\bar{\Sigma}_0\}} \left[P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right]\end{aligned} \tag{4.3}$$

We will reuse the same trick operator shuffling technique from the proof of Theorem 11 (From Eqn. A.47 to Eqn. A.48). If we can prove equivalence of the form in Equation A.49,

but with $(\bar{\Sigma}_0, \bar{\Sigma}_0)$ in place of $(\bar{\Sigma}_+, \bar{\Sigma}_-)$ for the index sets, then we can use $2d$ applications of the minmax inequality to sandwich the actual term we care about. The reader should refer to these manipulations, as it is a very handy trick!

*Proof.* Let $X_{\bar{\tau}} = \{x | \bar{\sigma}(x) = \bar{\tau}\}$. Each pair of binary vectors, $\bar{\mu}, \bar{\tau}$, corresponds to a vector $v_{\bar{\mu},\bar{\tau}} \in \mathbb{R}^{\Omega 0+1}$ so that $P^{(d)}(\bar{\mu}, \bar{\tau}, x) = v_{\bar{\mu},\bar{\tau}} \cdot \tilde{x}$. Consider $\bar{\mu}_+ \in \bar{\Sigma}_+ \setminus \bar{\Sigma}_-$. For $\bar{\tau}_- \in \bar{\Sigma}_-$, let $K^{\bar{\mu}_+}_{\bar{\tau}_-}$ be the (convex) region of the input where $v_{\bar{\mu}_+,\bar{\tau}_-} \cdot \tilde{x} = \min_{\bar{\tau} \in \bar{\Sigma}_-} v_{\bar{\mu}_+,\bar{\tau}} \cdot \tilde{x}$.

Suppose that $\mathcal{N}(x) \geq 0$. Let $x$ be an arbitrary input. It must, for some $\bar{\tau}_-$, be in one of the convex sets, say $K^{\bar{\mu}_+}_{\bar{\tau}_-}$. Now suppose further that for this binary vector, $\bar{\tau}_-$, we have $v_{\bar{\mu}_+,\bar{\tau}_-} \cdot \tilde{x} \geq v_{\bar{\mu},\bar{\tau}_-} \cdot \tilde{x}$ for all $\bar{\mu} \in \bar{\Sigma}_+$. Suppose that for $\forall \bar{\mu}$. In this case we have

$$\begin{aligned} v_{\bar{\mu}_+,\bar{\tau}_-} &\geq \max_{\bar{\mu} \in \bar{\Sigma}_+} v_{\bar{\mu},\bar{\tau}_-} \geq 0 \\ &\geq \max_{\bar{\mu} \in \bar{\Sigma}_+} \min_{\bar{\tau} \in \bar{\Sigma}_-} v_{\bar{\mu},\bar{\tau}} \geq 0 \end{aligned}$$

To rephrase, let $A$ be a matrix with the vectors $\{v_{\bar{\mu},\bar{\tau}_-}\}_{\bar{\mu} \in \bar{\Sigma}_+}$ for columns. We showed $\nexists \tilde{x} \in K^{\bar{\mu}_+}_{\bar{\tau}_-}$ such that $A^T \tilde{x} \succeq 0$ and $v_{\bar{\mu}_+,\bar{\tau}_-,} . \tilde{x} < 0$. Therefore, Farkas' Lemma tells us that we can find positive nonnegative scalars, $\{\alpha_{\bar{\mu}} | \bar{\mu} \in \bar{\Sigma}_+\}$, and some dual vector $\eta \in (K^{\bar{\mu}_+}_{\bar{\tau}_-})^*$ such that

$$\begin{aligned} v_{\bar{\mu}_+,\bar{\tau}_-} &= \eta + \sum_{\bar{\mu} \in \bar{\Sigma}_+} \alpha_{\bar{\mu}} (v_{\bar{\mu}_+,\bar{\tau}_-,-} v_{\bar{\mu},\bar{\tau}_-}) \\ \Leftrightarrow \sum_{\bar{\mu} \in \bar{\Sigma}_+} \alpha_{\bar{\mu}} v_{\bar{\mu},\bar{\tau}_-} &+ \left(1 - \left(\sum_{\bar{\mu} \in \bar{\Sigma}_+} \alpha_{\bar{\mu}}\right)\right)(v_{\bar{\mu}_+,\bar{\tau}_-}) = \eta \qquad \text{(A.50)} \\ \Leftrightarrow \sum_{\bar{\mu} \in \bar{\Sigma}_+} \alpha_{\bar{\mu}} v_{\bar{\mu},\bar{\tau}_-} &= \eta + \left(\left(\sum_{\bar{\mu} \in \bar{\Sigma}_+} \alpha_{\bar{\mu}}\right) - 1\right)(v_{\bar{\mu}_+,\bar{\tau}_-}) \qquad \text{(A.51)} \end{aligned}$$

Here, we point out that $\sum_{\bar{\mu} \in \bar{\Sigma}_+} \alpha_{\bar{\mu}} < 1$ is not possible. Since the RHS of Equation A.50 has nonnegative dot product with every $\tilde{x} \in K^{\bar{\mu}_+}_{\bar{\tau}_-}$, we know that at least one of the

vectors on the LHS must as well. This would imply nonnegativity of maximum over all of $\bar{\Sigma}_+$, given by, $\max_{\bar{\mu}\in\bar{\Sigma}_+} v_{\bar{\mu},\bar{\tau}_-}\tilde{x} \geq 0\ \forall \tilde{x} \in K^{\bar{\mu}_+}_{\bar{\tau}_-}$. But, we know this to be false, since $\bar{\tau}_- \in \bar{\Sigma}_-$, which guarantees the existence of at least some $\tilde{x}$ with $\max_{\bar{\mu}} v_{\bar{\mu},\bar{\tau}_-} \cdot \tilde{x} < 0$. But now, consider the RHS of Eqn A.51, which is nonnegative. We see that every time $v_{\bar{\mu}_+,\bar{\tau}_-}\tilde{x} \geq 0$, then also we are guaranteed some term in the left summation, say indexed by $\mu' \in \bar{\Sigma}_+$, such that $v_{\mu',\tau^*}\bar{y} \geq 0$ (we may assume not all $\alpha_{\bar{\mu}} = 0$). In other words, we have the first arrow of

$$\left[\mathcal{N}(x) \geq 0\right] \Rightarrow \left[\max_{\bar{\mu}\in\bar{\Sigma}_0}\min_{\bar{\tau}\in\bar{\Sigma}_-} P^{(d)}(\bar{\mu},\bar{\tau},x) \geq 0\right] \Rightarrow \left[\max_{\bar{\mu}\in\bar{\Sigma}_0}\min_{\bar{\tau}\in\bar{\Sigma}_0} P^{(d)}(\bar{\mu},\bar{\tau},x) \geq 0\right] \Rightarrow \text{Eqn.5}$$
$$\Rightarrow \left[\min_{\bar{\tau}\in\bar{\Sigma}_0}\max_{\bar{\mu}\in\bar{\Sigma}_0} P^{(d)}(\bar{\mu},\bar{\tau},x) \geq 0\right] \Rightarrow \left[\min_{\bar{\tau}\in\bar{\Sigma}_0}\max_{\bar{\mu}\in\bar{\Sigma}_+} P^{(d)}(\bar{\mu},\bar{\tau},x) \geq 0\right] \Rightarrow \left[\mathcal{N}(x) \geq 0\right]$$

The contrapositive of the very last arrow (starting with $\mathcal{N}(x) < 0$) is extremely similar to the first, so we omit it. The remaining arrows are all either minmax inequality or obtained shrinking or growing the set being optimized over. Thus completing the proof. □

# Bibliography

Murat Kocaoglu, Christopher Snyder, Alexandros G. Dimakis, and Sriram Vishwanath. CausalGAN: Learning Causal Implicit Generative Models with Adversarial Training. In *Proceedings of International Conference on Learning Representations*, pages 1–37, 2018.

Shakir Mohamed and Balaji Lakshminarayanan. Learning in Implicit Generative Models. In *arXiv*, 2016.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Proceedings of Neural Information Processing Systems*, Montreal, Canada, dec 2014.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating Videos with Scene Dynamics. In *Proceedings of Neural Information Processing Systems*, Barcelona, Spain, dec 2016.

Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. In *arXiv*, nov 2016. URL `http://arxiv.org/abs/1411.1784`.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs. *Proceedings of International Conference on Machine Learning*, oct 2017. URL `http://arxiv.org/abs/1610.09585`.

Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face Aging With Conditional Generative Adversarial Networks. In *arXiv*, 2017. URL https://arxiv.org/abs/1702.01983.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv*, 2017. URL https://arxiv.org/pdf/1701.07875.pdf.

David Berthelot, Thomas Schumm, and Luke Metz. BEGAN: Boundary Equilibrium Generative Adversarial Networks. In *arXiv*, 2017. URL https://arxiv.org/pdf/1703.10717.pdf.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision*, Washington, DC, 2015.

Kumar Sricharan, Raja Bala, Matthew Shreve, Hui Ding, Kumar Saketh, and Jin Sun. Semi-Supervised Conditional GANs. *arXiv*, aug 2017. URL http://arxiv.org/abs/1708.05789.

Yan Chen Xi Duan, Houthooft Rein, John Schulman, Sutskever Ilya, Abbeel Pieter, Xi Chen, Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Proceedings of Neural Information Processing Systems*, pages 2172–2180, Barcelona, Spain, dec 2016.

Kumar Sricharan. Personal communication, 2017.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial Feature Learning. *Proceedings of International Conference on Learning Representations*, apr 2017. URL http://arxiv.org/abs/1605.09782.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adverserially Learned Inference. *Proceedings of International Conference on Learning Representations*, 2017. URL https://arxiv.org/pdf/1606.00704.pdf.

Qiang Liu and Dilin Wang. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. *arXiv*, pages 4–7, 2016. URL https://arxiv.org/abs/1608.04471.

Chris Donahue, Akshay Balsubramani, Julian Mcauley, and Zachary C Lipton. Semantically Decomposing the Latent Spaces of Generative Adversarial Networks. *Proceedings of International Conference on Learning Representations*, 2018. URL https://arxiv.org/pdf/1705.07904.pdf.

Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed Sensing Using Generative Models. *arXiv*, 2017. URL https://arxiv.org/pdf/1703.03208.pdf.

David Lopez-Paz and Maxime Oquab. Revisiting Classifier Two-Sample Tests. In *Proceedings of International Conference on Learning Representations*, Toulon, France, 2016. URL https://arxiv.org/abs/1610.06545.

David Lopez-Paz, Robert Nishihara, Soumith Chintala, Bernhard Schölkopf, and Léon Bottou. Discovering Causal Signals in Images. In *Proceedings of Computer Vision and Pattern Recognition*, Honolulu, CA, may 2016. URL http://arxiv.org/abs/1605.08179.

Mohammad Taha Bahadori, Krzysztof Chalupka, Edward Choi, Robert Chen, Walter F Stewart, and Jimeng Sun. Causal Regularization. In *arXiv*, 2017.

Michel Besserve, Naji Shajarisales, Bernhard Schölkopf, and Dominik Janzing. Group invariance principles for causal generative models. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, Lanzarote, Spain, 2018. URL `http://proceedings.mlr.press/v84/besserve18a/besserve18a.pdf`.

Olivier Goudet, Diviyan Kalainathan, Philippe Caillou, David Lopez-Paz, Isabelle Guyon, Mic Ele Sebag, Aris Tritas, and Paola Tubaro. Learning Functional Causal Models with Generative Neural Networks. *Explainable and Interpretable Models in Computer Vision and Machine Learning*, 2018. URL `https://arxiv.org/pdf/1709.05321.pdf`.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2009.

Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. A Bradford Book, 2001.

David Heckerman. A Tutorial on Learning with Bayesian networks. *Tech. Rep. MSR–TR–95–06, Microsoft Research*, 1995.

David Maxwell Chickering. Optimal Structure Identification With Greedy Search. *Journal of Machine Learning Research*, 3:507–554, 2002.

Patrik O Hoyer, Dominik Janzing, Joris Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear Causal Discovery with Additive Noise Models. In *Proceedings of Neural Information Processing Systems*, Vancouver, Canada, 2008.

Antti Hyttinen, Frederick Eberhardt, and Patrik Hoyer. Experiment Selection for Causal Discovery. *Journal of Machine Learning Research*, 14:3041–3071, 2013.

Alain Hauser and Peter Bühlmann. Two Optimal Strategies for Active Learning of Causal Networks from Interventional Data. *International Journal of Approximate Reasoning*, 55(4):926–939, 2014. URL https://www.sciencedirect.com/science/article/pii/S0888613X13002879.

Karthikeyan Shanmugam, Murat Kocaoglu, Alexandros G Dimakis, and Sriram Vishwanath. Learning Causal Graphs with Small Interventions. *Proceedings of Neural Information Processing Systems*, pages 1–9, 2015. URL http://arxiv.org/abs/1511.00041.

David Lopez-Paz, Krikamol Muandet, Bernhard Schölkopf, and Ilya Tolstikhin. Towards a Learning Theory of Cause-Effect Inference. In *Proceedings of International Conference on Machine Learning*, Lille, France, 2015.

Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal Inference Using Invariant Prediction: Identification and Confidence Intervals. *Statistical Methodology, Series B*, 78: 947–1012, 2016.

Jalal Etesami and Negar Kiyavash. Discovering Influence Structure. In *Proceedings of IEEE International Symposium on Information Theory*, Los Angeles, CA, 2016.

Christopher Quinn, Negar Kiyavash, and Todd Coleman. Directed Information Graphs. *Proceedings of IEEE Transactions on Information Theory*, 61:6887–6909, 2015. URL https://arxiv.org/pdf/1204.2003.pdf.

Murat Kocaoglu, Alexandros G Dimakis, Sriram Vishwanath, and Babak Hassibi. Entropic Causal Inference. In *Proceedings of Association for the Advancement of Artificial Intelligence*, San Francisco, CA, 2017a.

Murat Kocaoglu, Alexandros G Dimakis, and Sriram Vishwanath. Cost-Optimal Learning

of Causal Graphs. In *Proceedings of International Conference on Machine Learning*, Sydney, Australia, 2017b.

Joris M Mooij, Oliver Stegle, Dominik Janzing, Kun Zhang, and Bernhard Schölkopf. Probabilistic latent variable models for distinguishing between cause and effect. In *Proceedings of Neural Information Processing Systems*, Vancouver, Canada, 2010.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. In *arXiv*, 2017.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. *Proceedings of Neural Information Processing Systems*, pages 1–10, 2016. doi: arXiv:1504.01391.

Christian Naesseth, Francisco Ruiz, Scott Linderman, and David Blei. Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 489–498, Fort Lauderdale, FL, USA, 2017. PMLR.

Matthew Graham and Amos Storkey. Asymptotically Exact Inference in Differentiable Generative Models. In Aarti Singh and Jerry Zhu, editors, *Proceedings of International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 499–508, Fort Lauderdale, FL, 2017. PMLR.

Christopher Snyder and Sriram Vishwanath. Sample Compression, Support Vectors, and Generalization in Deep Learning. *IEEE Journal on Selected Areas in Information Theory*, 2020a. URL `http://arxiv.org/abs/1811.02067`.

Warren S McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous

Activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, dec 1943. ISSN 0007-4985. doi: 10.1007/BF02478259. URL http://link.springer.com/10.1007/BF02478259.

Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and Generalization in Neural Networks: An Empirical Study. *Proceedings of International Conference on Learning Representations*, 2018. URL https://arxiv.org/pdf/1802.08760.pdf.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann Lecun, and Nathan Srebro. Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks. *Proceedings of International Conference on Learning Representations*, 2019. URL https://arxiv.org/pdf/1805.12076.pdf.

Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-Independent Sample Complexity of Neural Networks. *Proceedings of the Conference On Learning Theory*, dec 2018. URL http://arxiv.org/abs/1712.06541.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-Based Capacity Control in Neural Networks. *Proceedings of the 28th Conference on Learning Theory*, 40:1–26, 2015. URL http://proceedings.mlr.press/v40/Neyshabur15.pdf.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Proceedings of Neural Information Processing Systems*, pages 6241–6250, 2017a. URL http://papers.nips.cc/paper/7204-spectrally-normalized-margin-bounds-for-neural-networks.

Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Robust Large Margin Deep Neural Networks. *IEEE Transactions on Signal Processing*, 65(16):4265–

4280, aug 2017. ISSN 1053-587X. doi: 10.1109/TSP.2017.2708039. URL http://ieeexplore.ieee.org/document/7934087/.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks. *arXiv*, jul 2017a. URL http://arxiv.org/abs/1707.09564.

David A McAllester. Some PAC-Bayesian Theorems. *Machine Learning*, 37(3):355–363, 1999. ISSN 08856125. doi: 10.1023/A:1007618624809. URL http://link.springer.com/10.1023/A:1007618624809.

David McAllester. A PAC-Bayesian Tutorial with A Dropout Bound. *arXiv*, jul 2013. URL http://arxiv.org/abs/1307.2118.

John Langford and Rich Caruana. (Not) Bounding the True Error. In T G Dietterich, S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 809–816. MIT Press, 2002. URL http://papers.nips.cc/paper/1968-not-bounding-the-true-error.pdfhttps://papers.nips.cc/paper/1968-not-bounding-the-true-error.

Gintare Karolina Dziugaite and Daniel M Roy. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. *Proceedings of Uncertainty in Artificial Intelligence*, mar 2017. URL http://arxiv.org/abs/1703.11008.

Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nathan Srebro. Exploring Generalization in Deep Learning. Technical report, Long Beach, CA, USA, 2017b. URL https://arxiv.org/pdf/1706.08947.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Flat Minima. *Neural Computation*, 9(1):1–42, jan 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.1.1. URL http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.1.1.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in Deep Learning. *arXiv*, 2017. URL https://arxiv.org/pdf/1710.05468.pdf.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning. *Computing Research Repository*, dec 2014. URL http://arxiv.org/abs/1412.6614.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger Generalization Bounds for Deep Nets via a Compression Approach. *arXiv*, feb 2018a. URL http://arxiv.org/abs/1802.05296.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Compressibility and Generalization in Large-Scale Deep Learning. *arXiv*, 2018. URL https://arxiv.org/pdf/1804.05862.pdf.

Nick Littlestone and Manfred K Warmuth. Relating Data Compression and Learnability. Technical report, Univeristy of California Santa Cruz, 1986. URL https://users.soe.ucsc.edu/{%}7B{~}{%}7Dmanfred/pubs/T1.pdf.

Sally Floyd and Manfred Warmuth. Sample Compression, Learnability, and the Vapnik-Chervonenkis Dimension. *Machine Learning*, 21(3):269–304, dec 1995. ISSN 0885-6125. doi: 10.1007/BF00993593. URL http://link.springer.com/10.1007/BF00993593.

Pascal Germain, Alexandre Lacoste, François Laviolette, Mario Marchand, and Sara Shanian. A PAC-Bayes Sample Compression Approach to Kernel Methods. *Pro-*

*ceedings of the International Conference on Machine Learning*, 2011. URL http://www.icml-2011.org/papers/218{_}icmlpaper.pdf.

François Laviolette and Mario Marchand. PAC-Bayes Risk Bounds for Stochastic Averages and Majority Votes of Sample-Compressed Classifiers. *Journal of Machine Learning Research*, 8:1461–1487, 2007. ISSN ISSN 1533-7928. URL http://www.jmlr.org/papers/v8/laviolette07a.html.

Daniel Soudry, Elad Hoffer, and Nathan Srebro. The Implicit Bias of Gradient Descent on Separable Data. *Journal of Machine Learning Research*, 19:1–57, 2017. URL https://arxiv.org/pdf/1710.10345.pdf.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the Black Box of Deep Neural Networks via Information. *Computing Research Repository*, mar 2017. URL http://arxiv.org/abs/1703.00810.

Daniel Soudry and Yair Carmon. No Bad Local Minima: Data Independent Training Error Guarantees for Multilayer Neural Networks. *arXiv*, 2016. URL https://arxiv.org/pdf/1605.08361.pdf.

Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A Generalized Representer Theorem. In *Proceedings of Conference on Computational Learning Theory*, pages 416–426, Amsterdam, Netherlands, 2001. Springer, Berlin, Heidelberg. doi: 10.1007/3-540-44581-1_27. URL http://link.springer.com/10.1007/3-540-44581-1{_}27.

Ali Rahimi. Machine learning has become alchemy., 2017. URL https://www.youtube.com/watch?v=x7psGHgatGM.

Chiyuan Zhang, Samy Bengio, Google Brain, Moritz Hardt, Benjamin Recht, Oriol Vinyals, and Google Deepmind. Understanding Deep Learning Requires Rethinking Generalization.

*Proceedings of International Conference on Learning Representations*, 2017. URL https://arxiv.org/pdf/1611.03530.pdf.

Peter L Bartlett, Nick Harvey, Chris Liaw, and Abbas Mehrabian. Nearly-Tight VC-Dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks. *Proceedings of Machine Learning Research*, 65:1–5, mar 2017b. URL http://arxiv.org/abs/1703.02930.

Paul W Goldberg and Mark R Jerrum. Bounding the Vapnik-Chervonenkis Dimension of Concept Classes Parameterized by Real Numbers. *Machine Learning*, 18:131–148, 1995. URL https://link.springer.com/content/pdf/10.1007/BF00993408.pdf.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A Closer Look at Memorization in Deep Networks, 2017. URL https://dl.acm.org/citation.cfm?id=3305406.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and Counting Linear Regions of Deep Neural Networks. *Proceedings of International Conference on Machine Learning*, 2018. URL https://arxiv.org/pdf/1711.02114.pdf.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the Expressive Power of Deep Neural Networks. *Proceedings of the International Conference on Machine Learning*, 2017. URL https://arxiv.org/pdf/1606.05336.pdf.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding Deep Neural Networks with Rectified Linear Units. *Proceedings of International Conference on Learning Representations*, 2018b. URL https://arxiv.org/pdf/1611.01491.pdf.

Boris Hanin and David Rolnick. Complexity of Linear Regions in Deep Networks. *Proceedings of International Conference on Machine Learning*, 2019. URL https://arxiv.org/pdf/1901.09021.pdf.

Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical Geometry of Deep Neural Networks. *Proceedings of International Conference on Machine Learning*, 2018. URL https://arxiv.org/pdf/1805.07091v1.pdf.

Guillermo Valle-Pérez, Chico Q Camargo, and Ard A Louis. Deep Learning Generalizes Because the Parameter-Function Map is Biased Towards Simple Functions. *Proceedings of International Conference on Learning Representations*, may 2019. URL http://arxiv.org/abs/1805.08522.

Mihaela Porumb, Saverio Stranges, Antonio Pescapè, and Leandro Pecchia. Precision Medicine and Artificial Intelligence: A Pilot Study on Deep Learning for Hypoglycemic Events Detection based on ECG. *Scientific Reports*, 10(1), dec 2020. ISSN 20452322. doi: 10.1038/s41598-019-56927-5.

Ana Mincholé and Blanca Rodriguez. Artificial intelligence for the electrocardiogram. *Nature Medicine*, 25(1):22–23, 2019. ISSN 1546170X. doi: 10.1038/s41591-018-0306-1.

Tavpritesh Sethi, Anushtha Kalia, Arjun Sharma, and Aditya Nagori. Interpretable artificial intelligence: Closing the adoption gap in healthcare. *Artificial Intelligence in Precision Health*, pages 3–29, jan 2020. doi: 10.1016/B978-0-12-817133-2.00001-X. URL https://www.sciencedirect.com/science/article/pii/B978012817133200001X.

Divya Shanmugam, Davis Blalock, and John Guttag. Multiple Instance Learning for ECG Risk Stratification. In *Proceedings of Machine Learning Research*, pages 1–15, dec 2018. URL http://arxiv.org/abs/1812.00475https://bit.ly/2UoDmDN.

Arjun Gupta, E. A. Huerta, Zhizhen Zhao, and Issam Moussa. Deep Learning for Cardiologist-level Myocardial Infarction Detection in Electrocardiograms. *ArXiv e-prints*, dec 2019. URL http://arxiv.org/abs/1912.07618.

Özal Yıldırım, Paweł Pławiak, Ru-San Tan, and Rajendra Acharya. Arrhythmia Detection Using Deep Convolutional Neural Network With Long Duration ECG Signals. *Article in Computers in Biology and Medicine*, 102:411–420, 2018. doi: 10.1016/j.compbiomed.2018.09.009. URL https://www.researchgate.net/publication/327602644.

Awni Y. Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H. Tison, Codie Bourn, Mintu P. Turakhia, and Andrew Y. Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine*, 25(1):65–69, jan 2019. ISSN 1546170X. doi: 10.1038/s41591-018-0268-3.

Sebastian D Goodfellow, Andrew Goodwin, Robert Greer, Peter C Laussen, Danny Eytan, S D Goodfellow, A Goodwin, R Greer, P C Laussen, M Mazwi, and D Eytan. Towards Understanding ECG Rhythm Classification Using Convolutional Neural Networks and Attention Mappings. In *Proceedings of Machine Learning Research*, volume 85, pages 1–18, 2018.

Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Polo Chau. Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1096–1106, 2020. ISSN 19410506. doi: 10.1109/TVCG.2019.2934659.

Sana Tonekaboni, Shalmali Joshi, Melissa D McCradden, and Anna Goldenberg. What Clinicians Want: Contextualizing Explainable Machine Learning for Clinical End Use. In *Proceedings of Machine Learning Research*, pages 1–21, may 2019. URL http://arxiv.org/abs/1905.05134.

Christopher Snyder and Sriram Vishwanath. Deep Networks as Logical Circuits: Generalization and Interpretation. *arxiv e-prints*, mar 2020b. URL `http://arxiv.org/abs/2003.11619`.

Gari D. Clifford, Chengyu Liu, Benjamin Moody, Liwei H. Lehman, Ikaro Silva, Qiao Li, A. E. Johnson, and Roger G. Mark. AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017. In *Computing in Cardiology*, volume 44, pages 1–4. IEEE Computer Society, 2017. doi: 10.22489/CinC.2017.065-469.

A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation*, 101(23), 2000. ISSN 15244539. doi: 10.1161/01.cir.101.23.e215.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *arXiv*, 2015.